JBoss ESB Service Test UI

John Graham (jgraham@redhat.com)

Use Case

The ability to test services easily is important during the service development and when simply learning about how particular services work. Since JBoss ESB is a uniform messaging paradigm, the method of invoking services is (close to) boilerplate code and hence it is a good candidate to support with a UI.

Task Flow

Context: The user's workspace has one or more ESB projects present, each defining service content.

1. The user opens the “Service Test” view (assuming an Eclipse view for the moment; other options might be possible)
2. The view filters the workspace projects to show only ESB projects
3. Each ESB project whose archives are current with the server (that is, “synchronized” not “republish” or “publish”) is enabled for testing
4. The user selects from the enabled service archives
5. The testing view inspects the jboss-esb.xml file for the selected service archive to determine:
    1. What services are exposed
    2. What the call options are for each service (InVM always available; others, such as JMS, might be)
6. The user selects the service to test
7. (Optional) The default call mechanism is InVM. The user might choose an alternative call mechanism if available
8. The user enters the test data:
    1. By typing directly into a text box in the test UI
    2. By selecting a file in the workspace/file system as input test data
9. (Optional) The user changes various configuration parameters as available
10. The user clicks on a “Test” button to send the test data to the chosen service
11. Any errors in calling the service are displayed in the test UI

Note that due to the nature of services, the user will have to look for the effects of testing (for example, JBoss ESB console output) elsewhere.

Considerations

In typical distributed object (for example usual web services) tests, the invocation are requests/response, so the results can be easily displayed in the same test UI. JBoss ESB does not make this assumption, however, and it is hard to predict where/if output is expected. In many of the ESB Quick Starts, test output is shown in the JBoss ESB console, so having the JBT console view open while testing shows that output. In other cases, response messages might be sent to a variety of endpoints (such as a JMS queue or a FTP site). In these cases it would be nice if the test UI could also be configured to connect to the output endpoint and listen for the result message.

In the future we'd like to be able to discover services deployed on a JBoss ESB instance and test call them using the same UI. The ability to do this, however, assumes the ability to discover and display deployed services and to read the configuration file (jboss-esb.xml) associated with the deployed service. Since we do not have these capabilities, we will need to limit the service test UI to local archives in ESB projects for now.

Design Notes

- Refer to chapter 3 “Services and Messages” in the JBoss ESB Programmers' Guide for details about service invocation. (From the JBoss ESB distribution.)
- The classes SendJMSMessage.java and SendESBMessage.java appear in many of the ESB Quick Start tests. These classes give an indication of how tests can be run.
- Since the tester UI will refer to ESB projects in the workspace, and since each ESB project that can be tested needs to be (a) associated with a JBoss ESB instance, (b) have updated content on the server, then the tester UI should use the Server API to determine details about the JBoss ESB run time, such as host name, port numbers, and so on.
- If the target server is not running, the test UI probably should not enable associated services for testing. In this cases, and when workspace content is not in synch with server content (publish or republish), the tester UI should display text indicating why the particular service(s) are not available for testing.
- When the user changes server state – start, stop, publish, republish, etc. -- the tester UI needs to update the list of services enabled for testing based on this new information. If the JBT Servers view/framework does not have API to support this currently, we should work with Rob Stryker to find a solution.
- The tester UI should not require that the JBT Servers view be open. Ideally, this means that the tester UI should be able to get the required information about target servers from the same API that the JBT Servers view does. If the current API set requires the JBT Servers view to be open, then we should work with Rob Stryker to find a solution.
- It would be useful to optionally save the test as an Eclipse launch configuration, so it could be re-run with same data and parameter settings quickly. Creating a launch configuration every time a service is tested, however, is too much and quickly clogs the launch configuration list with extra data. Having an option to save a test to a launch configuration would mean integration with Eclipse run/debug, but would give the best user experience.