JrsProposal

Java Resource Simulator (JRS) Proposal

The following is a proposal for a new top-level project within the ASF.

Abstract

The Java Resource Simulator (JRS) is a Java-based interface response capture/playback tool for testing purposes, intended for use in development or testing environments.

Proposal

JRS will provide a lightweight (meaning no or very little code changes to the application under test) utility for Java applications that simulates interactions with remote services, without having to have an active connection to those remote services. JRS is currently a community source, internal project at American Express. This proposal is to develop the JRS code and community at the ASF.

Background

A big problem that legacy-dependent development groups have is that when developing new applications, the legacy systems that they depend on move relatively slowly from a development and integration standpoint, are often unavailable during development and test and require long lead times to set up test data sufficient to give good path and data coverage for front end testing. For this reason, people tend to develop ad hoc "simulators" or "smart stubs" over and over again to enable modularity in development. JRS aims to provide a generic solution to this problem, or at least to minimize the work required to set up and maintain effective resource simulators.

Rationale

Increasingly, systems are being built by aggregating services from multiple sources. While there are a number of advantages to this approach, testing these service-client systems can be a challenge. Test services have to be deployed and maintained, test data must be managed, test service nodes must be up and running and accessible to clients. Every additional service used adds an additional dependency that makes testing harder to coordinate, execute, and manage in a repeatable fashion.

Core use cases that JRS aims to address

1. *Mock Design*. When building a system, external resources are not yet available, so those interfaces can be simulated by manually writing hard-coded responses which are played back through JRS. The manually-written responses may allow more experimentation in new development situations where the interactions are not clear, and when established these test cases may help drive the final definition of the interfaces.
2. *Code Coverage*. When attempting to achieve high code coverage in a set of tests, it may be easier to artificially hard code responses from external systems rather than attempting to prod the desired behavior out of those external systems. The responses are then played back through JRS.
3. *Test Without Dependence*. When building a system, it may be problematic to repeatedly execute tests against external resources, because they are unstable, intermittently available, inaccessible, or just difficult to repeatedly populate with test data sets. The external interactions are recorded once, possibly edited, and played back multiple times as needed without requiring that the external resources be available. In this way, regression tests can be run against the system-under-test without having to coordinate with external systems.
4. *Stress Test*. JRS can be used for pre-production stress testing. This is not a substitute for production stress testing since it is the interaction with external resources themselves where bottlenecks often occur and JRS is only simulating these interactions. Stress testing using JRS can help identify bottlenecks within the system under test itself, and offers the advantage that delay factors can be artificially varied in order to observe the effect of changing remote system latencies.

JRS will provide a framework for simulating remote systems in each of the following ways:

- Interactions can be recorded (this requires the remote service to be connected), and later played back (the remote service does not have to be connected).
- Request/response pairs can be manually added or edited (e.g. after having been recorded).
- Custom Java handlers can be defined

JRS will operate as a library (jar file) that is included with the application that intercepts calls to supported protocols. The persistence component ("JRS server" – for record/playback) can run in the same JVM as the system under test, or it can be run remotely as indicated in a JRS configuration file. The current (working) version supports JMS, Session EJBs, and web services. Support for JDBC is in early stages of development.

Why ASF?

The developers of JRS are interested in joining the Apache Software Foundation for several reasons:

- We would like to see the framework extended and improved via open, standards-based development.
- We would like open development of JRS to take place within the ASF legal, licensing and oversight structure.
- As indicated in the "Alignment" section below, JRS has dependencies on and natural connections with several ASF projects. We would like to establish and leverage cross-pollinating relationships with these communities to improve the design of JRS, its standards-compliance, and ease of use.

Initial Goals

In addition to establishing an open development community, the immediate goal is to continue the development of the framework. Main areas of effort:

- Add JDBC support.
- Provide JRS server real-time console.
- Implement additional mechanisms to reduce record conflicts (same recorded input generates different responses at different times). Options include understanding ordered sequences or more generally support for stateful client-server dialogs, as well as techniques to map test sequences to separate namespaces.
- Implement multi-user JRS server support, enabling separate projects to share a common test server.
- Implement more robust JRS configuration management for n client x m server scenarios.
- Full JMS support (currently only pseudo synchronous supported).
- Add import/export to facilitate early stage test-driven development.

Current Status

JRS is currently a community source, internal project at American Express. The code base has been in development for 18 months and an initial internal release is being used by American Express development projects. Along with several other internally maintained components, JRS was opened internally as a community source project earlier this year.

Meritocracy

The JRS project will be meritocractic open development. The purpose of this submission is to make a quality product that will be used by many users, and the only way to achieve that is by recognizing the value of community. There are many difficult design and implementation problems to solve in JRS and we feel strongly that open, meritocratic development will result in a substantially better, more extensible and fully-featured product than we could develop internally.

Community

JRS has been developed over the last couple of years as a corporate internal project, with most of the work done by three developers and one architect. Nevertheless, there has been from the beginning an interest to eventually open source the project, as such the project has been opened as an internal "community source" project at American Express. While this clearly does not equate to an open source community development, we believe it gives us a strong base to build upon.

Core Developers

The initial developers for the project are associated with the donating company. Two of the developers have worked on open source projects before (one is an ASF member) and have experience with open development principles. There are a number of other strong developers in the internal community that have expressed interest and we expect will prove themselves worthy committers in a short period of time. In addition, there are some current ASF committers interested in participating in the project and we have included them in the list of initial committers.

Alignment

The initial code has been implemented in Java and uses a number of Apache and other open source components, such Maven, Log4J, XStream, JUnit, etc. It is expected that further integration may happen with Apache projects such as Oro (regular expressions processing of requests), Axis/CXFire/Mina (alternative JRS client-server protocol and server implementation), Derby (for DBMS persistence of messages), Struts (for JRS server console), and ActiveMQ (for processing JMS selectors during playback).

Known Risks

Orphaned products

The initial committers are users of this package and have a long-term interest in use and maintenance of the code. All of the active developers would like to become JRS Committers and plan to remain active in the project.

Inexperience with open source

The initial committers have varying degrees of experience with open source projects as users, participants or committers, with one being an active ASF member. All have been involved with source code that has been released under an open source license and with internal open source projects.

Homogeneous developers

Since the Java Resource Simulator has been developed to date by American Express, the initial contributors to the project are associated with that corporation, though not all are employees of American Express. They are experienced working in a geographically distributed and diverse team and they have a broad range of experiences with open source, industry standards, emerging technologies and product development. Furthermore, our strong intention is to attract a diverse set of additional committers, beyond the initial contributors and current Apache committers listed below.

Reliance on salaried developers

It is expected that, at the beginning, JRS development will occur on both salaried time and on volunteer time, after hours. While there is reliance on developers associated with American Express, through the incubation process, we expect the independent Community to become actively involved in the project.

No ties to other Apache products

JRS currently uses or is planned to use a number of Apache and other open source projects. These have been outlined above.

A fascination with the Apache brand

JRS has been started as a response to real and critical needs of development projects over many years. The originating environment has been IT internal of a non-software company, as such there was/is no need to associate the Apache brand with JRS. We believe that JRS will solve in an elegant and lightweight manner development lifecycle problems and, as such, we are interested in the best way for the project to develop and flourish. We have no interest or intention of "productizing" JRS for commercial purposes or offering paid services associated with its use; though part of our motivation for pursuing open development of JRS under the ASL is that this will not prevent others from doing so.

Scope of the project

The scope of the JRS project at ASF would be to continue the product development and would include adding new features and improving performance, scalability, quality, and extensibility.

Documentation

Documentation is available on request. See below.

Initial source

Initial source code for the project will be granted (see next section) to the ASF on acceptance of this proposal and once granted it will be made publicly available and the normal Incubator IP Clearance process will be followed to approve inclusion of the initial sources in the project source repository.

Source and Intellectual Property Submission Plan

American Express is prepared to submit a code grant and a CCLA and to license all JRS code under the ASL. All rights to the current codebase are owned by American Express. The initial committers have or will all submit ICLAs.

External Dependencies

The current implementation depends on the following components:

- XStream- BSD- http://xstream.codehaus.org/license.html
- JUnit- CPL - http://www.opensource.org/licenses/cpl.php
- Maven - ASF
- Log4J - ASF
- J2EE API - CDDL

- XPP3 - http://www.extreme.indiana.edu/viewcvs/~checkout~/XPP3/java/LICENSE.txt

All dependencies have ASL or ASL-compatible licenses.

Cryptography

JRS currently makes no direct use of cryptographic functions.

Required resources

Mailing lists

- jrs-private (with moderated subscriptions)
- jrs-dev
- jrs-commits
- jrs-user

Subversion or CVS repositories

JRS would like to use a Subversion repository: [WWW] https://svn.apache.org/repos/asf/incubator/jrs

Issue tracking

Since JRS would have its own release cycle, it should have its own JIRA project

- Project Name: JRS
- Project Key: JRS

Initial set of committers

- Brendan McCarthy (brendan_dot_mccarthy_at_ gorillalogic_dot_com)
- Tony Ambrozie (tony_dot_a_dot_ambrozie_at_aexp_dot_com)
- Phil Steitz (psteitz_at_apache_dot_org)
- Ian Gray (ian_dot_d_dot_gray_at_aexp_dot_com)
- Rahul Akolkar (rahul_at_apache_dot_org)
- Sebastian Bazley (sebb_at_apache_dot_org)
- Martin van den Bemt (mvdb_at_apache_dot_org)
- Henri Yandell (bayard_at_apache_dot_org)

Affiliations

Tony Ambrozie, Phil Steitz and Ian Gray are employees of American Express. Brendan McCarthy is an employee of Gorilla Logic, working under contract to American Express. The rest are ASF committers working for distinct companies. As individuals, none of the ASF committers have any contract or employment relationship with American Express.

Sponsors

Champion

- Phil Steitz

Nominated Mentors

- Phil Steitz
- Sebastian Bazley
- Martin van den Bemt
- Henri Yandell

Sponsoring Entity

We are asking the Incubator PMC to sponsor this proposal.