JXTA 2.8x Proposal (First Draft)

After three years of almost complete silence in project JXTA, a small group of individuals have taken it up on them to propose a 2.8 release of the JXTA code. This document outlines the proposed changes, the strategic choices and the vision behind them.

Background

Ever since Sun, and later Oracle, reduced their commitment to project JXTA, without withdrawing their rights to JXTA as trademark, it has become difficult for developers to be committed to any initiative for improving the P2P framework. The intended continuation of project JXTA, Chaupal, is paralised by the implications of their ambition to replace JXTA with an improved alternative that is aimed to be available under the AFS (Apache Foundation License), and by lack of developers who are willing to commit themselves to this ambitions of this project. The JXTA 2.8x proposal that is drawn here aims to bridge the gap between the current status and the ambitions of project Chaupal by proposing to start along three development lines, which the intention of eventually making project Chaupal an OSGI implementation of the JXTA specs. This approach will allow a gradual phasing out of JXTA code, while retaining the benefits of the current code. These development lines are:

1. (Minimal) adjustments to the JXTA code to allow support for OSGI declarative services
2. Develop an OSGI-based 'exo-skeleton' around the current code which falls under the AFS license, ECL or Apache. This part can be hosted under the Chaupal project
3. Develop additional tooling around the exo-skeleton which aims to replace the long overdue Swing-based examples which JXTA currently supports.

The proposed upgrade of JXTA 2.8 focuses on the first part of this development, and will be the focus of this document.

The first ideas of this proposal were developed early 2013 by the author of this document, when he revived his efforts to implement JXTA for one the the projects he carries out for his company Condast. A comprehensive search for good P2P replacements suchs as JGroups, proved that the current JXTA implementation is still one of the best options once one tries to move beyond the Intranet. However, new developments such as RabbitMQ, JMS and, in general, in social media, suggest that the *comprehensive* approach to JXTA (which currently includes a CMS, messaging services and so on) might better be replaced by a minimal approach that focuses on P2P specific services (in effect, peer group registration and discovery), and integration of these other developments as (OSGI) services. A first attempt to develop JXTA in this direction is currently hosted in the Eclipselabs incubator project JXTA-Eclipse, but this project mainly focuses on the third aspect of the development lines discussed earlier; the 'exo-skeleton' is minimally implemented, and poorly thought through with respect to the aims of providing a P2P *framework*. However, the project did revive the interest of some other individuals in JXTA, and currently three individuals have expressed interest in putting effort in making a 2.8x release:

- Kees Pieters: OSGI and JXTA from an end-user perspective
- Matt Gumbley: P2P Architecture and Messaging
- Wu Tao: Eclipse Tooling

Jérôme Verstrynge, the lead of the JXSE 2.6x and 2.7x release has kindly offered to take up an advisory role in the preparations of this roadmap.

Ideally, we could use some expertise on the P2P internal architecture by long-term committers, but considering the scope and aims of the 2.8 release, we think that the current expertise could suffice.

The next section will cover the proposed changes in more detail.

Detailed Technical Changes

The concrete changes on the existing JXTA code are aimed to enable the bare-bone changes for an OSGI-implementation. This means that the current dependencies on often outdated third-party libraries should be changed in favour of declarative services wherever possible:

- Jetty 4.2x : The dependency with the 4.2x version of `org.mortbay.jetty` should be modified to the OSGI version of Jetty (version 6.x and higher). This has currently be largely implemented in jxta-eclipse, but a few improvements should be considered. Especially the option to register the JXTA HTTP servlet through the standard OSGI servlet bundle should be looked in to. One problem here is that JXTA registers both a servlet and a connector, so the latter has to be provided as well. Another issue is that JXTA currently allows to specify a specific port that the server will listen to has to be thought through. The current implementation in jxta-eclipse seems to work but needs more testing. The affected package is `net.jxta.impl.endpoint.servlet.http.`
- Derby and H2. JXTA currently has two dependencies on database clients which offer the possibility of registration through `javax.jdbc` datasources. It would be a great advantage to change the current code to use this mechanism as well. In itself the required code changes are quite small, but the network configurator should be extended with a property that allows selection of the desired database client, e.g a datasourceHint option. If the desired datasource isn't available, JXTA should fall back to a default implementation and give a warning message, or stop. The affected package is: `net.jxta.impl.cm.sql`.
- Bouncycastle: the dependencies on bouncycastle are currently carried out in jxta-eclipse through a *fragment* bundle, which are convenient for the current purposes. Version 1.4.x is currently used, as newer implementations no longer offer seamless integration with JXTA. It could be an option to offer the required functionality through declarative services, if needed by implementing a bundle that serves as a bridge between JXTA and bouncycastle. The main package that is affected is `net.jxta.impl.membership.pse` and `net.jxta.impl.protocol.`
- Netty: The current implementation of JXTA works with current implementations of the netty bundles. It could be considered to implement this as a declarative service.

Besides these core dependencies, there are some other developments that could be integrated in the 2.8x release, and which result in dependencies with third party libraries (bundles):

- Base64: currently JXTA has its own implementation. It could be a consideration to implement a Base64 interface that allows coupling with the relevant `apache.commons` bundle, and leave the current code as a fall-back when the bundle is not found.
- JSON. One of the nice-to-haves on the Chaupal wishlist is to change the current XML-based implementation of advertisements and messages into an implementation that is based on JSON. This will be a major operation, as it may affect code throughout the implementation. If this requirement is taken up in the 2.8x release, then a builder service should be implemented, which offers an interface that provides a builder for advertisements, messages and so on.
- There are some rumours (I haven't found them yet) that there are some dependencies with proprietary libraries, such as `com.sun.xxx`. This should be investigated and open source alternatives should be considered.

There is one more addition to the current code that the author would like to implement, and that is a JxseContainer implementation that has similar functionality as a Servlet container. The current architecture is fairly straightforward and consists of a network manager with a network configurator, and a collection of services (Modules) that are started. Some of these services themselves can host services in a tree-like structure. One of the problems with the current implementation is that it is hard to differentiate between services that have to be started and stopped, and when other services can be connected to parent services. A servlet container kind of approach may prove more stable and powerful; ideally it should be possible to dynamically register, start and stop services while JXTA is running.

Another issue, related to this, is the fact that most services mainly consist of configuration activities. These configurations can feed factories and builders that create the service (modules) and register them, after which the container takes care of starting, initialising and stopping the services. This approach is currently being implemented in `jxta-eclipse` in a rudimentary fashion, but already it allows very flexible ways of creating JXTA applications. It also paves the way for a radical implementation of IoC principles á la Gemini Blueprint. The current network configurator is a rather problematic concept for these ambitions, as it tries to centralise a number of configuration properties of relevant JXTA modules. It would have author's preference to keep configuration activities closely connected to the relevant services. As an example, the current TCP (or HTTP, Multicast or Security) configuration can be implemented as distinct services with a small configuration file (e.g in XML):

```
<jxta:tcp id="org.mybundle.myp2p.container.service.http"
          container="org.mybundle.myp2p.container"
          auto-start="true">
  <enabled>true</enabled>
  <port>
    <start>9700</start>
    <active>9715</active>
    <end>9730</end>
  </port>
</jxta-tcp>
```

Instead of overloading the network configurator with an increasing number of settings, the OSGI approach would be to create a bunle that contains this file in a pre-defined place (e.g a JXSE-INF folder), after which the bundle register itself with the specified jxse container and the functionality is automatically registered and started.

In effect, the result of the modifcations proposed above is that creating a JXTA application follows a distinct work cycle:

- Create a configuration and fill it
- Offer this to a factory or builder and create the service.
- register it with the container, where it is started when appropriate.

This approach reflects the way the majority of the JXTA code is implemented by end users, but instead of hardcoding this, this approach becomes largely automated and regulated through, for instance, an xml configuration file.

Approach

Depending on the ambitions and the availability of developers, three scenarios can be followed to

achieve the proposed roadmap:

1. Minimal changes are made in the JXTA code for the 2.8x release. The actual OSGI application is developed through Chaupal, which includes a layer of bundles that serve as bridges between the raw JXTA jars and the declarative service mechanism. Dependencies with third party libraries will all be managed through interfaces, either in the jxta code, or in the bridge bundles, so that no code in JXTA directly depends on these libraries.
2. Bullet one is extended with the implementation of the jxse-container for the 2.8x release, which is achieved by defining an interface that wraps the current network container, the network configurator and the modules. The container is exportable to external code through an interface.
3. The 2.8x release will be fully OSGI ready. In this case it may be wise to consider offering a JXTA 3.0 version.

The interfaces can all be easily wrapped in the declarative service mechanism that OSGI offers, which then can be further developed through the Chaupal project, which will also be the home project of the tooling. The Chaupal project will then complete the goal of defining bundles that contain the services in an IoC-like fashion. The main question is which efforts can best be done in the current JXTA code, and which can best be performed in Chaupal.

The author has a preference for either option one or two, with a slight preference for the latter. Even though relatively more code will be developed under the current JXTA license, it is my opinion that it will make the current code more robust for developing and more OSGI-ready. This means that a JXTA 3.0 will not be considered, rather we will aim for Chaupal 1.0 release in due time. This means that all the developments on the non-JXTA 2.8x code, such as the tooling, will be hosted under `net.chaupal` or `net.p2p.chaupal`. The latter is a consideration to make the goals of Chaupal more intuitive to third parties.
Even though the tooling project is geared towards Eclipse, the architecture should not be overly dependent on this framework; it should be possible to offer, say, the JXTA shell for Netbeans as well.

Either way, the consequences for the JXTA 2.8x release will mean, amongst others, of keeping the code in the current repository (SVN) and in project Kenai. New code will only be added sparsely, and wherever possible be made available under ASF. This may mean that project Kenai may host two jars (or more) that in its entirety make up JXTA 2.8x.

Extending Previous Developments

According to Jérôme, JXTA 2.7x was one of the most stable releases yet. There were some patches made to the 2.6x branch that reflected ongoing developments with Netty / Http tunneling, but all in all most of the goals of the two releases were met in the consecutive releases. For the 2.8x release this means that we mainly need to assess which developments branches are worth porting to the master branch. From what we understand this will be a minor operation, especially if the developers of the patches are willing to lend their expertise to this initiative.

Improving the Membership Service

Another possible major item to address for 2.8 is a better membership service.

“Many newcomers to JXTA are often disappointed by the currently available implementations of the membership service. Their expectations are high and not met. This situation will surely improve in the future releases of JXSE as there is a growing demand in the community." (Practical JXTA II, p 46)

New developments in distributed consensus algorithms, such as Paxos or Raft may help in providing a better implementation.

Functionally speaking, this revamping operation will involve authentication and identity too. Right now, anyone can intercept and spam communication between peers. This is not a small architectural job. It goes to the very core of what a peer group is (or should be). There is currently no resistance to route poisoning, no solution for CA, and self-certification is an issue. PSE needs to be reviewed from scratch. It is theoretically possible to encrypt communications passing via 'central' peers to prevent MITM attacks.

Other Running Issues

Further proposed improvements which are targeted for, or may affect the JXTA 2.8x release include:

- Establish a new continuous integration system: the system used in earlier releases - MikeCI - is no longer in service. Chaupal will be built up with such a system from scratch, but the relevant consequences for the kenai, the current repository for JXTA, should be investigated
- Set up a new build/release infrastructure, documentation and blogs. Improve the 'Mavenisation' efforts in JXTA where necessary.
- Establish a new public relay/rendezvous server. Especially the options to use free, or low-cost VPS services, or maybe Google App Engine, offer new means to make and maintain a new public seeds. However, we need to consider the consequences for malicious use, su ch as DoS atttacks on these public seeds.
- Ensure that all dependencies are available from the Central Maven repository, rather than adding links to personal repositories.
- Update all dependencies to their most recent versions, as far as is practical.
- Remove the options to load byte code in classloaders-per-peergroup. It is better to offer this at the level of OSGI services, such as currently is already possible in Equinox. As a default, instead of throwing an exception when multiple network managers try to start the WorldPeergroup, it should be an option to register to the running instance.
- Switch to SLF4J API for logging, removing the dependency on Log4J, allowing framework users to choose their logging framework.
- Ports to other languages that were not as up-to-date as the JXSE implementation may be worth reviving. Perhaps a JavaScript port would be of use - and JVM-language-centric wrappers, to provide more idiomatic APIs for e.g. Scala, Clojure, Jruby.
- Integrate the functionality of the JXTA Shell in Chaupal's tooling project. The current implementation has not been maintained for years, and can better be developed from scratch.
- Improve the build/quality infrastructure for running load tests, profiling, etc. As extensive testing was done for the 2.6.X and 2.7x releases, the implications for the 2.8x code should be minimal.

Miscellaneous

There is a preference amongst the current task group to perform the activities for Chaupal with Git or Mercurial. The JXTA 2.8 release will be carried out in the current SVN environment.

It is the intention of the proposal to respect the choices made by the previous committers of project JXTA and Chaupal and continue with the goals and aims defined in the Chaupal roadmap, wherever it does not conflict with the issues outlined above.

Some other initiatives have been made by individuals to develop JXTA tooling for Eclipse. If willing, we would like to integrate these efforts in the tooling sub-project hosted in Chaupal.

We would also like to try to salvage the useful documentation on JXTA and collect them in a central repository. Currently a lot of documentation is outdated, hard to find and conflicting. Many useful posts are being removed.

12. Initial Committers

Kees Pieters (keesp), Matt Gumbley, Wu Tao

13. Sponsors

We don't have a sponsor yet.