Towards a Common Java LDAP API

Emmanuel Lecharny
Apache Software Foundation
elecharny@apache.org

Ludovic Poitou
Sun Microsystems
ludovic.poitou@sun.com

Abstract.

When it comes to access LDAP from the Java platform, developers are facing a poor choice: either use JNDI which suffers from being too generic or use Netscape LDAP Java SDK (or JLDAP) which was designed many years ago and lacks the new features of the Java language. Both Apache Directory Server and OpenDS projects are developing LDAP based servers and tools in Java, and both have the same need: a client API that can be used in their server and for the client tools, so they each are in the process of finishing their client API for Java.

Realizing that LDAP developers are not so numerous and a good single API is more than needed, the Apache Directory Server and OpenDS developers started to discuss on the path to define this single standard LDAP API for Java programmers. This paper looks at the current LDAP APIs and their deficiencies, discusses a list of requirements and needs for a better LDAP API. Then it dives in specific areas of the proposed interfaces and discusses the standardization path.

Today's Java developer's choice

For the past few years, the choice of an LDAP library for Java developers was very limited: it was either JNDI, or the Netscape LDAP Java SDK or JLDAP.

The de facto standard API for dealing with LDAP directory services is the Java Naming and Directory Interface (JNDI). It's part of the Java platform since Java 1.3 and is leveraged by many Enterprise Java features. However JNDI has some major deficiencies. First it is a very generic interface for Naming and Directory, so its semantic often differs from the LDAP one. Even some LDAP operations are not supported. JNDI offers only APIs to issue synchronous requests. JNDI has a limited support for LDAP extension like Controls and Extended requests. A JNDI LDAP extension library (the LDAP Booster Pack) could be downloaded from Sun. Alas, this library is no longer available.

The Netscape LDAP Java SDK and JLDAP libraries both implement a set of interfaces that matches closely the LDAP specifications but has been defined several years ago in an expired

Internet-Draft, with some minor differences. As a result, both libraries lack support of the new Java language features such as Collections and Generics. Moreover the development of both libraries have stalled and even maintenance releases are irregular and unfrequent.

Where are we now ?

About two years ago, Sun Microsystems along with OpenLDAP, Apache Directory Server and other experts in LDAP, initiated an effort to submit a Java Specification Request to define a pure Java API to communicate with LDAP directory servers. For various reasons, the effort didn't go through and the various project teams started to work independently.

The Apache Directory Server team started to work on the design of a pure Java LDAP server in 2001 and was accepted in the ASF incubator in 2003. From day one, the need for a full set of common objects was obvious, so it has been aggregated in what has become our own LDAP API. Later on, as the team was working on tooling (Studio), the question arose about the need of better API than JNDI (Tools and Server were all based on JNDI back then). Also, the tests were developer with either JNDI or LdapSDK, which started to create maintenance problems. When the Apache development team met Sun's team at the Austin Apache Conference, we shortly discussed about the idea of a replacement API for JNDI, and 6 months later, Sun came back with the idea of a new JSR.

Sadly, for many reasons, the JSR never came to light. The need of an extended API was now pretty urgent for the Apache Directory Server project, as it was decided to remove all reference to JNDI into the server, substituting it with the internal set of API. Lately, developers also needed to define the client side for replication and tooling. It was then obvious that this will become a complete API, and, as collaboration is the essence of the ASF work, the team thought that it would be a waste to work in isolation, without trying to collaborate with other teams. Some of the Apache Directory Server project members were contacted by OpenLDAP who looking for a maintainer of the JLdap API, and around the same time Sun team came back to see if collaboration on API was still doable, which it is.

We, Apache Directory Server team, strongly believe that users don't need 5 or 6 different APIs. There is no value added to such a duplication of efforts. Moreover, the API is totally defined, and there is no reason to see a divergence on the main elements. The core API should be common, and if we look at it, the only difference is probably on the name selection (LdapDN, LDAPDN, DN,...). Our goal is clear : we want a JSR to be created, have a LDAP API defined, have a RI and other implementations if needed, so that developers don't waste their precious time switching from one API to another one. In any case, there is no competitive advantage in defining a new isolated API.

Along the same time, in the OpenDS project, as the directory server is getting mature and new services are being built to extend the current capabilities, the development team faced the need to have a common set of interfaces for both incoming and outgoing LDAP connections. As the interfaces are being defined mostly for use on the server side, with high performance in mind, a

strong focus was put on asynchronous methods to be used in a highly threaded program. During the extracting of the code from the server side to turn it into a client API, more importance was given on the ability to write simple code while maintaining the requirements of power users.

Also a few months ago, UnboundID, an Austin based startup, released a preview version of a new LDAP SDK for Java, and since then has released the 1.0 version and made it's code available at SourceForge.

While the choice offered to developers was sparse, having too many competing projects is not good either. So the Apache Directory Server team and the OpenDS team resumed the collaboration with the intend to compare their respective interfaces, identify common interfaces, identify the points of divergence and produce the ground work for restarting a standardization effort for the Java API to access LDAP directories.
The remaining of this paper is a summary of the current state of the discussion.

The beginning of work

Although the Netscape LDAP Java SDK and JLDAP suffer from some deficiencies, they are well structured, organized. So they are used as models for defining the new interfaces.

The major changes we see in all current developments are :
- Use of Generics
- Use of Collections
- Use of Futures to deal with asynchronous interfaces
- Simplification of the API and most importantly of the code required to make use of them

The clear sketch of the API is now appearing.
A set of APIs to handle the connections to the directory servers. A set of interfaces for all LDAP requests and responses. A set of interfaces for all objects composing entries and parameters of the requests: Distinguished Names, Relative Distinguished Names, Entries, Attributes, Filters, Controls, Changes...

The longest work to reach agreement on a standard API will definitely be in the details for each of the objects and methods. But first some large roadblocks must be addressed :
- The connection factory API
- The Asynchronous vs Synchronous operation calls
- Schema aware API vs non-schema aware
- Representation of values in the API

Let's discuss each of these issues.

Identified Roadblocks

The Connection Factory

The underlying implementation for the transport, IO handling must be transparent to client applications, and should be configurable so that the client SDK can be used within Java based servers.

We propose to keep the model of JLDAP and Netscape LDAP Java SDK: the LDAPConnection is an interface, which can only be instantiated by a factory, the LDAPConnectionFactory. When retrieving an LDAPConnection, in addition to the connection information such as the host and port to connect to, some options may be specified. We propose to have an LDAPConnectionOptions object that contains a set of common options to all implementations, but could be extended. Common options include whether to use a TLS secure connection, key managers, trust managers. Optional options could be whether the Connection is from a pool or standalone (if the Factory supports pool).

As more people will join and help define the API, there will be more questions to answer: how to plug a specific "transport", whether there should be a single LDAPConnectionFactory, i.e. a static one, or should applications obtained a new LDAPConnectionFactory, and thus there might be different factories, like one for simple connection, one for pool-able connections.

Asynchronous vs Synchronous API

By looking at the 3 current APIs in progress, we see 3 different approaches in the way to deal with Synchronous vs Asynchronous operations.

OpenDS favors the asynchronous operations. All operations return a Future. A synchronous call can be done by simply calling the get() method of the Future.

```
ModifyResult response = connection.modify(modifyRequest, null).get();
```

Apache Directory Server defaults to the synchronous operations and the method return directly a Response which is blocking. But if a "Listener" is passed as parameter, the method is asynchronous and returns the Future.

The 3rd option and the one selected by UnboundID, is to provide different methods for the synchronous operations and for the asynchronous operations, for example add() vs asyncAdd().

We will need more feedback from the developers community to make a choice on this subject.

Schema Aware vs schema agnostic APIs

When preparing LDAP requests such as Add or Modify or for properly processing Search

results,
some specific interfaces do need to know elements of the server's schema. Building entries to add or modifications without knowledge of the LDAP schema can lead to server errors that are then reported directly to the end user who has no help for solving them.
A typical example is the Add request and how to construct an Entry to be added to the server. When building an entry, sets of attributes and values are added. A schema aware API is able to detect duplicated values (even if they don't have the same representation), group attributes even if different names were used (2.5.4.3 and cn are the same attribute).
Similarly, without having an understanding of LDAP schema, the API cannot properly find attributes in returned entries, cannot make any choice on how to display or copy values (string or array of bytes).

We believe that to be really useful, the LDAP API should be schema aware, or at least know how to deal with standard object classes, attributes and syntaxes. Methods should be provided for the library to enrich its schema definitions, by looking up on a server or by reading local files.

Representation of values in the interface.

This seems a minor aspect, but it has deep implications. If we look at the way LDAP deals with values they are either a string or octets. But looking at the various existing LDAP Java libraries, there is not a single agreement on how to represent values. Apache Directory Server has a Value object which wraps a value in either a String or a bytes array. UnboundID SDK uses ASN1OctetString and OpenDS, ByteString which mostly extends bytes array. Also each project has defined its client API from its server code, which has been optimized to avoid conversions and unnecessary coping.

It looks like a solution would be to multiply methods in the client API to deal as well with String and bytes array. Special consideration should be taken on how to deal with retrieving set of values and iterating through them.

Conclusion

Defining a new API for standardization is a long process and should involve all experts in the area. We do not pretend to be smart enough to be able to conclude it on our own, and there are many areas that have not been considered. But we've compared our respective approaches, compared our own interfaces. After long discussions we've started to sketch a general direction, we've identified some big rocks to address first and prepared the path for an expert group to resume the standardization effort for an LDAP API for Java.

We now would like to invite all volunteers to participate.