[TopCoder SOFTWARE]

LDAP SDK 1.0.1 Component Specification

1. Design

The LDAP SDK Interface component provides a means to perform standard LDAP functions through a consistent API regardless of the underlying SDK. Initially, only an interface to the Netscape Directory SDK is provided, but it is possible to develop plugins for other SDKs and configure the LDAP SDK Interface without changing client code.

To enable new plugins to be used without changing the client code, the plugins must be instantiated without knowing the class name at compile time. This can be done using reflection.
There is a set of operations that each plugin must provide (connect, authenticate, add entry, etc), that will be defined in an Interface and each plugin will implement it.
There are basically two approaches for dealing with entries: wrapping each "concrete" (for example Nestcape Directory) SDK class and let them store the data, or building classes that will store the data and feed "concrete" SDK class just before doing an operation. Even if the structure of an entry is well defined by standards and should be the same for every LDAP implementation, the way of loading the entry with data can be different from an implementation to another. For instance, Netscape Directory uses an LDAPAttributeSet class to store a set of attributes, while another implementation might choose to use a container in the entry. If the wrapper structure (that has to be the same for all the plugins) doesn't fit well with the concrete SDK classes, the plugin code will no more be clean and straightforward, and may have to appeal to store itself all the attributes.
Building the classes that store entry data in the LDAP SDK will require more work in the LDAP SDK itself, but it can avoid doing that work many times in the plugins, and the code will be cleaner and easier to maintain.
Also, because the structure of an entry is independent of the LDAP implementation, it is reasonable to think about the Entry object as an entry in no matter which LDAP implementation.

1.1 Design Patterns

- A modified **Factory Method** pattern is used to choose one of the SDK implementations, probably using reflection. This makes possible to add new plugins without changing the client code.
    - Creator and ConcreteCreator: `LDAPSDK`
    - Product: `LDAPSDKFactory`
    - Concrete Product: `NetscapeFactory` (more can be added)

- The **Abstract Factory** pattern is used to create a connection from the SDK implementation chosen. There are two products: a connection and an SSL Connection. In the case of Netscape Directory, and probably in many other cases, both products return an instance of the same class but initialized differently. This pattern will be helpful in the future if other implementation dependent classes are needed, because they could be added without changing the core design.
    - Abstract Product: `LDAPSDKConnection`
    - Products: different instances of `NetscapeConnection` (more can be added)
    - Abstract Factory: `LDAPSDKFactory`
    - Concrete Factory: `NetscapeFactory` (more can be added)

1.2 Industry Standards

LDAP Protocol

1.3 Required Algorithms

None

1.4 Component Class Overview

The classes at `com.topcoder.util.net.ldap.sdkinterface` are the base for the LDAP SDK Interface. The classes at `com.topcoder.util.net.ldap.sdkinterface.netscape` implement the Netscape Directory Plugin.

1.4.1 LDAP SDK Interface classes

LDAPSDKConnection (interface)

As the *Abstract Product* of the Abstract Factory Pattern, it represents a connection to an LDAP. It has all the operations (connect, search, add entry, etc) that the implementing classes (plugins for the different SDK) must override.

LDAPSDKFactory (**interface)**

As the *Abstract Factory* of the Abstract Factory Pattern, it defines the products that each *Concrete Factory* (for example `NetscapeFactory`) must provide. The defined products are connections and SSL connections.

As the *Product* of the Factory Method Pattern, it defines the interface for the type of objects that the factory method produces.

In other words, the Factory Method Pattern is used to produce different factories depending on the LDAP implementation. Then, the factory can produce different types of connections.

LDAPSDK:

As the *Creator* in the Factory Method Pattern, this class creates a concrete product (for example a `NetscapeFactory`) and stores the instance in a private member variable. Note that because this class implements `LDAPSDKFactory`, it provides a set of common methods with the instantiated factory (methods `createConnection ()` and `createSSLConnection ()`)

As a *Concrete Creator* in the Factory Method Pattern, this class provides concrete products from the factory chosen.

This design makes the use of these classes quite easy, saving the user to deal with intermediate factory classes.

This piece of code demonstrates how to work with it:

```
LDAPSDK ldap=new LDAPSDK (className);
LDAPSDKConnection con=ldap.createConnection ();
```

Values:

This class is used to store the values of an Entry.

The values can be text, stored in `String`, or binary, stored in `byte []`.

There are zero or more text values and zero or more binary values.

Entry:

This class represents an entry in the LDAP structure, which is uniquelly identified by its Distinguished Name. It has zero or more attributes, each one being a pair <attributeName, Values> in the `attributes` Map.

Modification:

This class represents a modification to be done in an entry.

This class is not public because the user doesn't need to use it directly. It is used as an intermediate class to store one modification.

Update:
This class represents an update in an LDAP entry, consisting in many modifications, each one represented by a Modification object.

LDAPSDKExceptionFactory:
This class is used to instantiate LDAP exceptions. Given an error code, this class is responsible for instantiate the right exception class.

1.4.2 Netscape Plugin Classes

NetscapeFactory:
As a *Concrete Product* from the Method Factory Pattern, it is one of the classes that the *Creator* (LDAPSDK) can instantiate.
Once this *Concrete Product* is chosen, it serves as a *Concrete Factory* from the Abstract Factory Pattern. This *Concrete Factory* is used to create the corresponding *Products* (connections and SSL connections), that are actually instances of *Netscape Connection.*

NetscapeConnection:
As a *Product* from the Abstract Factory Pattern, this class does the actual work of executing the operations in the Netscape Directory LDAP.

1.5 Component Exception Definitions

LDAPSDKException:
This exception represents an LDAP error. There are many classes inheriting from this class that provide a more specific error. If the error is unknow, an object of this class will be thrown, in other case it will be one of its descendants who will be thrown.

The class extends the Base Exception to provide chained exceptions. Depending on the LDAP implementation, when an operation is executed, an exception can be thrown, or may be just a return code indicating error is given. In both cases, or any other variation, the result code must be retrieved, the message error retrieved or generated, and an exception of this type or a subclass must be thrown. Selecting which exception class has to be thrown is the task of LDAPSDKExceptionFactory, who decides it based on the error code. The following functions throw a LDAPSDKException or a class derived from it:

- `connect`
- `disconnect`
- `addEntry`
- `deleteEntry`
- `renameEntry`
- `updateEntry`
- `readEntry`
- `search`

- `authenticate`
- `authenticateAnonymous`
- `authenticateSASL`

1.5.1 LDAPSDKException inherited classes

Classes inherited from LDAPSDKException provide a more specific error. Each exception represents one or more error codes from the LDAP specification. The list of the error codes that each exception covers is in the class documentation of each class in poseidon and in the javadoc for each class.

LDAPSDKCommunicationException
This exception is thrown when an operation was unsuccesfull because of an error in the communication with the server or in the server.

LDAPSDKAuthenticationMethodException
The server is expecting a different authentication method or there is an error in the way of authenticating**.**

LDAPSDKAccesDeniedException
The operation could not be completed due insuficient access rights.

LDAPSDKLimitsExceededException
Processing time or maximum entries found in the server exceeded its limits.

LDAPSDKSchemaViolationException
The operation could no be completed because it would violate the LDAP schema.

LDAPSDKInvalidFilterException
The filter for the search is invalid.

LDAPSDKNoSuchAttributeException
The attribute was not found.

LDAPSDKNoSuchObjectException
The entry was not found.

LDAPSDKInvalidAttributeSyntaxException
The attribute sytnax is invalid

LDAPSDKInvalidDNSyntaxException
The DN sytnax is invalid

LDAPSDKCallErrorException
The call to an LDAP service was unsuccesfull due to a protocol difference, a bad parameter or not supported functionality.

LDAPSDKOperationNotAllowedException
The operation is not allowed in the entry specified.

1.5.2 Other exceptions

ClassNotFoundException

When a class is created using reflection, it is possible that the class doesn't exist, may be because is misspelled or it is not correctly installed. This exception is thrown in that case. The method that throws this exception is:

```
- <<create>> LDAPSDK (String className)
```

IllegalArgumentException

This exception is used to indicate that a method has been passed an illegal or inappropriate argument. The methods that throw this exception are:

```
- LDAPSDKConnection.search (String base, int scope, String
filter)

- LDAPSDKConnection.search (String base, int scope, String
filter, String []attrs)

- NetscapeConnection.search (String base, int scope, String
filter)

- NetscapeConnection.search (String base, int scope, String
filter, String []attrs)
```

Also any plugin will have in his search method this exception thrown if the scope is not one of the constants defined for that purpose.

1.6 Component Benchmark and Stress Tests

This component adds a layer of abstraction to accessing any SDK, so any operation will be slower than executing it directly using the corresponding libraries. Anyway, in most cases the overhead should be negligible, because the time that takes all the LDAP execution, including sending data through a network, is usually big in comparison to the time it takes to call some methods and to copy some data in memory.

The time of executing operations with the component can be compared with the time to do the same thing using the corresponding libraries. This can be done reading entries with different sizes to test if bigger entries produce more overhead. Although it should take longer to do the translation between the specific LDAP objects and the component object, also the operation will be slower, so it's very likely for the overhead to be still negligible.

Also some other times for operations like deleting and updating entries should be measured. The expected result is a very little overhead. For a normal situation with a not very slow computer as the client and not an ideal network and server, less than 1% of overhead will be very good, between 1% and 5% will be good, and more than 5% will be too much.

2. Environment Requirements

2.1 Environment

- Development language: Java 1.3
- Compile Targets: Java 1.3, Java 1.4

2.2 TopCoder Software Components

- BaseException 2.0

NOTE: The default location for TopCoder Software component jars is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.

2.3 Third Party Components

None

3. Installation and Configuration

3.1 Package Name

com.topcoder.util.net.ldap.sdkinterface

3.2 Configuration Parameters

None

3.3 Dependencies Configuration

In order to use this component the JARs with `Base Exception 1.0` should be accessible via CLASSPATH. See documentation for this component to determine the steps needed to configure it (if any).

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute 'ant test' within the directory that the distribution was extracted to.

4.2 Required steps to use the component

1. Configure the plugin
2. Connect to LDAP
3. Authenticate
4. Work with entries
5. Disconnect

Examples of all steps are shown in 4.3

4.3 Demo

This piece of code shows some operations that can be done using this component, including basic exception handling:

```
try {

        // Configure the plugin
        LDAPSDK ldap=new LDAPSDK
("com.topcoder.util.net.ldap.sdkinterface.netscape.netscapeFactory");
        LDAPSDKConnection con=ldap.createConnection ();
```

```
            // Connect
            con.connect ("localhost",389);

            // Authenticate
            con.authenticate ("cn=TCSDESIGNER, ou=Components,
o=TopCoder","SecretPassword");

            // Add an Entry
            Entry entry=new Entry ("cn=Simpson, ou=Components, o=TopCoder");
            entry.setAttribute ("sn",new Values ("Homer"));
            String [] phones={"1234-5678","1234-5679"};
            entry.setAttribute ("telephonenumber",new Values (phones));
            con.addEntry (entry);

            // Delete one of the pone numbers of the entry just added
            Update update= new Update ();
            update.delete ("telephonenumber",new Values ("1234-5679"));
            con.updateEntry ("cn=Simpson, ou=Components, o=TopCoder", update);

            // Delete an entry
            con.deleteEntry ("cn=user1, ou=Components, o=TopCoder");

            // Read an entry
            Entry e= con.readEntry ("cn=Simpson, ou=Components, o=TopCoder");
            System.out.println ("phones:");
            List list = e.getValues ("telephonenumber").getTextValues ();
            for (Iterator it=list.iterator (); it.hasNext ();)
                  System.out.println ((String)  it.next ());

            // disconnect from LDAP
            con.disconnect ();

      } catch (LDAPSDKException e) {
            System.out.println (e.getMessage ());
      } catch (Exception e) {
            e.printStackTrace ();
      }
```

Future Enhancements

LDAP protocol has lots of features that this first version does not implement, but they could be added in future versions.

- Work with schema information
- Configure search preferences
- Handle referrals
- Asynchronous operations
- Extended operations
- LDIF files