Database Components

16 Aug 2007

1 Introduction

The sf-database JAR contains the components needed to deploy and configure databases under SmartFrog. The key use cases were

- Deploy a large MySQL database/Java application server system, cross-linked.
- Create a new database for further expansion.
- Run a pure Java database for lightweight system testing.

Having deployed MySQL under SmartFrog for the SE3D project, we knew one important aspect of MySQL is that its time to start up depends upon the size of the database, and the state it was last in. Small databases with no transactions to reply are near-instantaneous, but large databases cause MySQL to be slow to start. There is a simple test for availability: can you open up a JDBC connection to the port.

The design goals were

1. A way to describe a database binding that could be enhanced in future for JDBC connection pooling.
2. A workflow component to issue SQL commands to a database specified by a database binding.
3. Support SQL commands inline, and in .sql files on the classpath or in files.
4. A component to issue SQL commands on termination.
5. Run MySQL in a separate process, as the active user, on Linux/Unix systems, and possibly on Windows too.
6. Tests to run against both HSQLDB and MySQL, with Apache Derby a future possibility
7. Do not depend upon the MySQL client libraries, and do not distribute them.
   We wish to remain an LGPL product, but do not wish to break the GPL terms and conditions. GPL compliance will be a requirement of anyone using the MySQL components, and (implictly) our test deployments.
8. Liveness test component to check that a JDBC connection can be made.
9. Liveness test component to check that the result of a SELECT statement has the expected value (or a number of results within a defined range)
10. Provide a way to get data from a SELECT statement into SmartFrog attributes, for reference by other components.

This led to some design decisions.

- Use the `RunShell` component to provide a component to run MySQL [5].
- Target general purpose JDBC drivers, with extra support for any driver that implements a method called ping(). The MySQL driver does this, but so may others. [7]
- Have a `Binding` component separate from any JDBC operation components. The bindings are deployed outside of a workflow, and provide connections for all database components *running in the same process.* [1]
- Have a hierarchy of JDBC components [2,3,4], for:
  - SQL transactions on startup
  - SQL transactions on termination

– SQL on both startup and termination

– A SELECT operation for liveness tests

– An `SqlSelect` component to issue a SELECT command and attach the results as properties.

The base class will be `AbstractJdbcOperation`; this class will contain the logic to talk to the Binding to get the connection, to issue requests against a server, etc.

For ease of implementation, we adopted the Apache Ant `<sql>` task as the foundation for the JDBC operation classes; this provided the code to read SQL statements from text files or inline, to parse it and extract JDBC commands. This code is fairly brittle, as it attempts to support SQL files from a number of vendors, each of who have a different commenting policy. To keep the number of support calls down, the SmartFrog parser supports a subset of the SQL that the Ant tool (unreliably) supports. The most reliable way to specify a list of commands is in a vector, with one SQL command per element. However commands are specified, each command executes as a self-contained transaction. There is currently no support for aggregating a series of operations into a single transaction.

1.1 Status and Roadmap

The SmartFrog 3.11.00 release is the first SmartFrog release containing components to deploy and configure databases. The sf-database.jar is built for Java 5+; it contains Java 5 language constructs and has not been tested on prior Java versions.

The first release offers the ability to deploy databases, including MySQL, and to issue SQL commands to a database when a component is started or stopped. This enables a database to be set up when an application is deployed, and (possibly) cleaned up when it is terminated. The condition components enable other parts of a deployment to be delayed until the database is live, and to detect a failure of the database to start up.

Some possible future enhancements are:

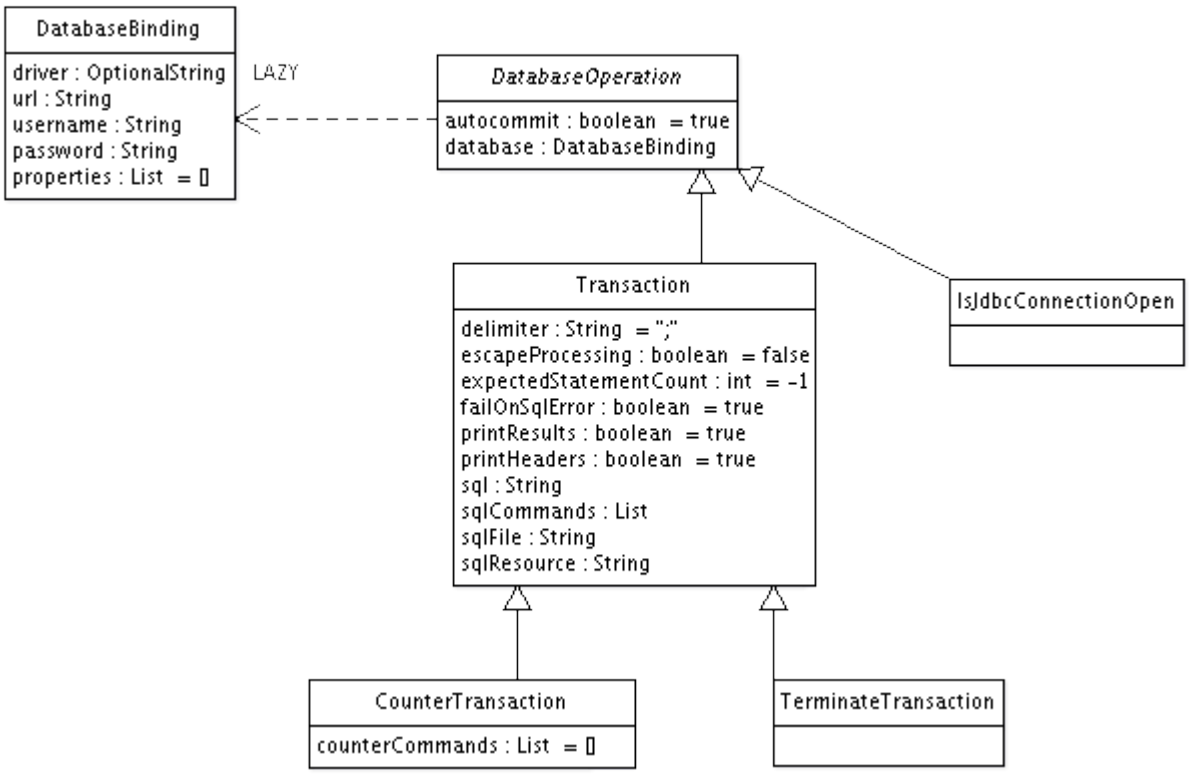
- Add the ability to read the results of transactions (especially SELECT statements) as SmartFrog attributes.
- Add RMI methods to the `Transaction` component, to enable it to issue commands on behalf of other components.
- Use the sf-jmx module to work with the MySQL Connector/MXJ library, for better MySQL deployment.
- Add Apache Derby support.
- More, better tests.
- Caching of connection details from the `DatabaseBinding`, so that those components that run at shutdown don't depend on the `DatabaseBinding` still being live.
- Possibly, pooling in the `DatabaseBinding` component. I'm unsure about this as I'm not sure how much load you should be generating during deployment, and to support pooling means that we'd need to deploy a new `DatabaseBinding` in every process. We may make it an optional extra; a `DatabaseConnectionPool` component that implements the relevant binding attributes/RMI interface, but also offers a non-exported pool API for stuff in the same JVM to use.
- Better Windows support.

There is no timetable for these enhancements. SmartFrog users with a pressing need for these or other database-related features are strongly encouraged to work with the team to implement the features they need, and the tests to accompany the features.

2 Components

2.1 Database Components

Here are the core SmartFrog components for manipulating a database. These are all database neutral, though of course the SQL commands issued by the components will have to be targeted at specific databases.



SmartFrog components for database manipulation

The conceptual model is that in a deployment, one or more `DataBaseBinding` components are deployed, each of which describe how to connect to a database. These are then referenced in other components that actually talk to the database. The `Transaction`, `TerminateTransaction` and `CounterTransaction` components can issue a series of SQL commands to the database identified by their specific database bindings; these commands can be issued when the components are started, when stopped, or when started and stopped, respectively.

There is also a condition, `IsJdbcConnectionOpen`, which can test for the ability to connect to a database. This can be used to test that a database is live, and that the correct JDBC drivers are on the classpath.

2.1.1 DatabaseBinding

The `DatabaseBinding` component is used to declare a JDBC binding to a database. *It does not deploy the database*.

```
DatabaseBinding extends Prim {
  sfClass "org.smartfrog.services.database.core.JdbcBindingImpl";
  //this is not exported across processes, as it does VM-specific setup, and
  //will in future export non-Remoteable methods for pooling
  sfExport false;
  jdbcBindingSchema extends Schema {

      driver extends OptionalString {
```

```
      description "the classname of the driver, which must be in the current classpath";
    }

    url extends String {
      description "URL of the jdbc connection";
    }

    username extends OptionalString {
      description "username for the database";
    };

    password extends OptionalString{
      description "password for the database";
    };

    properties extends OptionalVector {
      description "list of name,value pairs for the JDBC connection. Not yet functional";
    };

  }
}
```

The component is deliberately not exported (`sfExport false`), so that only components in the same process can share the same binding. This is to permit a future option for database pooling; the database binding would become the source of connections.

Example:

```
InMemoryBinding extends DatabaseBinding {
  driver "org.hsqldb.jdbcDriver";
  url "jdbc:hsqldb:mem:test";
  username "sa";
  password "";
}
```

This example is an HSQLDB binding to an in-memory (i.e. non-persistent) database called "test".

2.1.2 Transaction

The `Transaction` component lets you run a series of SQL commands against a database identified by a database binding.

```
Transaction extends DatabaseOperation {
  transactionSchema extends Schema {

    delimiter extends String {
      description "delimiter char between SQL statements, usually ;";
    }
    escapeProcessing extends Boolean {
      description "should SQL commands be escaped before execution?";
    }
    expectedStatementCount extends Integer {
      description ##
        this is primarily for internal testing, but can be used for debugging
        statement parsing problems. It is the count of statements expected
        after the string/file/resource is parsed. If there is a mismatch,
        an exception is thrown. Omit or use -1 for do not care' #;
    }
    failOnSqlError extends Boolean {
      description "should we fail on an SQL error?";
    }
    printResults extends Boolean {
      description "should the results be printed?";
    }
    printHeaders extends Boolean {
      description "should the column headers be printed?";
    }
    sql extends OptionalString {
      description "inline commands seperated by the delimiter";
    }
    sqlCommands extends OptionalVector {
        description "a list of commands to execute, each command in a separate element";
      };
    sqlFile extends OptionalFilenameType {
```

```
      description "UTF-8 encoded file containing SQL commands to run";
    }
    sqlResource extends OptionalString {
      description "name of UTF-8 encoded text resource containing SQL commands to run";
    }
  }

  sfClass "org.smartfrog.services.database.core.TransactionImpl";
  sfShouldTerminate true;
  delimiter ";";
  escapeProcessing false;
  expectedStatementCount -1;
  failOnSqlError true;
  printResults true;
  printHeaders true;
}
```

The component sets `sfShouldTerminate true`, to indicate that as soon as it executes, it should terminate -it is intended for use in workflows. Set this attribute to false to revert to a component whose lifecycle is managed by its parent component.

Example: SQL to create a diary

```
      CreateDiary extends Transaction {
          sql ##
            CREATE DATABASE diary;
            GRANT ALL PRIVILEGES ON diary.* TO 'diary'@'localhost';
            SET PASSWORD FOR 'diary'@'localhost' = PASSWORD('secret');
            #;
      }
```

This component needs its database attribute set before it can be deployed; when it is deployed it will run a series of SQL commands against the database to create a diary. It is likely the commands only work on some databases, SQL being so inconsistent across implementations.

Example: drop a table.

```
DropTable extends Transaction {
   table TBD;
   sqlCommands [
    ("DROP TABLE " ++ table)
   ];
}
```

Here we have used the vector of commands, rather than relying on the Ant-derived code to parse our statements. We also add a new attribute, table, which must be defined at deployment time. This attribute is appended to the "DROP TABLE " string to create the complete operation.

2.1.3 TerminateTransaction

A `TerminateTransaction` is a `Transaction` component that runs its operations when a component is terminated, rather than when it is deployed.

```
TerminateTransaction extends Transaction {
  sfShouldTerminate false;
  sfClass "org.smartfrog.services.database.core.TerminationTransactionImpl";
}
```

When deployed, this component remains live (`sfShouldTerminate false`) until terminated by its parent. At which point, it will attempt to run its SQL operations.

Some warnings

1. SQL errors will not propagate. They will be logged.
2. The component will only terminate if SmartFrog itself shuts down cleanly. Unplanned system failures can kill the host or the SmartFrog processes without warning, in which case the transactions will not get

executed.

3. The transactions will not operate if the referenced `DatabaseBinding` has already terminated, or the database itself is already shut down.

Because of these warnings, this component (and all others that attempt to clean up during termination) have to be viewed as a best-effort operation. You cannot rely on the components running. If you need a database to be in a known (clean) state before every deployment, consider deploying `Transaction` components that put it in that state, rather than simply relying on cleanup from the previous run.

Here is an example of its use:

```
DropTableOnTermination extends TerminationTransaction {
   table TBD;
   sqlCommands [
    ("DROP TABLE " ++ table)
   ];
}
```

This component will drop the specified table when it is terminated.

2.1.4 CounterTransaction

This component extends the Transaction component with a new attribute, `counterCommands`, which contains a list of SQL commands to issue on cleanup. This lets a single component contain both a list of transactions to run when started, and a matching set of transactions to run when terminating, to roll back the previous work.

```
CounterTransaction extends Transaction {
  counterTransactionSchema extends Schema {
    counterCommands extends Vector {
      description "a list of commands to execute when undeploying ";
    };
  }
  //we do not terminate
  sfShouldTerminate false;
  counterCommands [];
}
```

Here is an example of the component first creating, then dropping a user.

```
  CreateAndDropUser extends CounterTransaction {
    username TBD;
    password TBD;
    sqlCommands [
     (("CREATE USER " ++ username ) ++ (" PASSWORD " ++ password))
    ];
    counterCommands [
     ("DROP USER " ++ username )
    ];
  }
```

The warnings associated with `TerminationTransaction` also apply: there is no guarantee that the termination command can be issued reliably.

2.1.5 IsJdbcConnectionOpen

This component is a *Condition*; it implements an RMI interface whose `evaluate()` method returns true when the condition is met.

```
IsJdbcConnectionOpen extends DatabaseOperation {
  sfClass "org.smartfrog.services.database.core.ConnectionOpenCondition";
}
```

This specific component tests that

1. The JDBC driver can be loaded.
2. The database connection can be made with the URL and properties supplied by the database binding.

Subclasses may override the empty `boolean ping(Connection connection)` method of the component to perform more testing on the opened connection.

It can be used with any workflow components that support conditional classes, including blocking a sequence until a condition is met, or failing a deployment if it is not met within a bounded period of time:

```
BlockForJdbcConnection extends FailingWaitFor {
 database TBD;
 condition extends IsJdbcConnectionOpen {
   database LAZY PARENT:database;
 }
  interval 200;
  timeout 10000;
 }
```

This example waits for ten seconds for a database to start up, polling five times a second. If it is not responding within the time given, the component terminates abnormally, an event which can be handled by the parent.

2.2 HSQLDB Support

HQLDB is a lightweight, pure Java database that is implemented entirely within its own JDBC driver. It is usable for single-application use, and used in many products (such as OpenOffice.org itself). Because it is so self-contained, it is ideal for embedding and testing, which is where it is used in the SmartFrog components.

```
#import "/org/smartfrog/services/database/hsqldb/components.sf"
```

The main component an `HsqldbBinding` extension of the database binding:

```
HsqldbBinding extends DatabaseBinding {
  driver "org.hsqldb.jdbcDriver";
  username "sa";
  password "";
}
```

This can be used to simplify bindings

```
TestDB extends HsqldbBinding {
  url "jdbc:hsqldb:mem:test";
}
```

The file also contains a component to start up a self contained server in a separate process.

```
HsqldbServerProcess extends Java {
    database TBD;
    database.name TBD;
    address "any";
    port "9001";
    processName "hsqldb";
    classname "org.hsqldb.Server";
    arguments [
        "-address",address,
        "-port", port,
        "-database.0",database,
        "-dbname.0",database.name
    ];
}
```

A URL to this deployed database would be `"jdbc:hsqldb:hsql://localhost:9001/dbname"`, if `dbname` was the name given in the `database.name` attribute.

A web server process supports URLs such as `"jdbc:hsqldb:http://host:8080/dbname"`, if `dbname` was the name given in the `database.name` attribute:

```
HsqldbWebServerProcess extends Java {
    database TBD;
    database.name TBD;
    address "any";
    port "8080";
    processName "hsqldb";
    classname "org.hsqldb.Server";
```

```
    arguments [
        "-address",address,
        "-port", port,
        "-database.0",database,
        "-dbname.0",database.name
    ];
}
```

There are also some Transactions to shut down the database

```
ShutdownCommand extends Transaction {
    sqlCommands [
        "SHUTDOWN"
    ];
}

ShutdownAndCompactCommand extends Transaction {
    sqlCommands [
        "SHUTDOWN COMPACT"
    ];
}
```

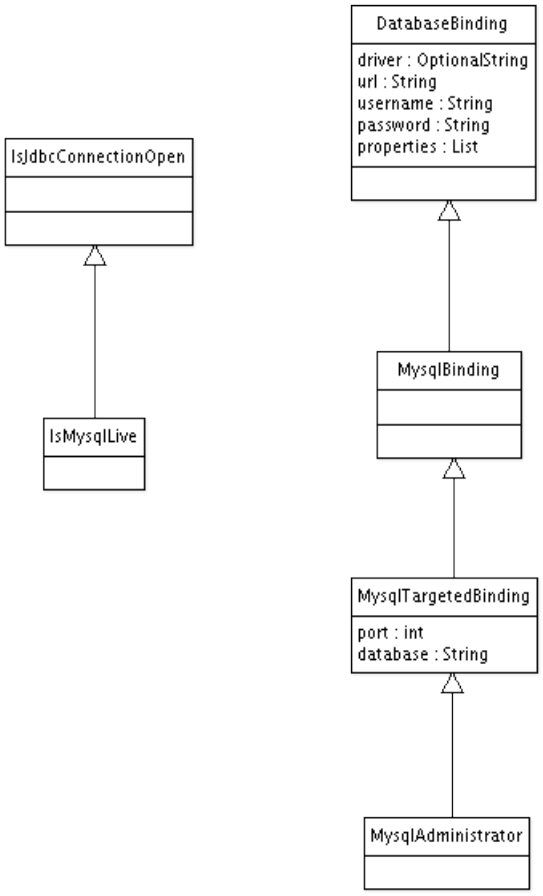
Consult the HSQDB web site for more SQL commands: http://hsqldb.org/.

2.3 MySQL Support

MySQL support comes in the its own deployment descriptor

```
#import "/org/smartfrog/services/database/mysql/components.sf"
```

This file imports the base database components, so there is no need to import them explicitly. In the MySQL package, there are two groups of components. One set extends the core database components with some MySQL specific definitions. Alongside this is a set of component templates that cover starting, stopping and administrating a MySQL installation.



MySQL binding components

Most of these components are merely extended templates to set up the driver, URL and password of a database. The `IsMysqlLive` component is special, in that it has a new implementation class behind it, one that calls extra methods on the JDBC connection to check that the connection with the database is live.

```
IsMysqlLive extends IsJdbcConnectionOpen {
  sfClass "org.smartfrog.services.database.mysql.IsMysqlLive";
}
```

Only use this to check MySQL databases or other databases whose JDBC driver has a method called `ping()` on every connection it returns:

```
BlockForMysqlLive extends BlockForJdbcConnection {
  condition extends IsMysqlLive {
    database LAZY PARENT:database;
  }
}
```

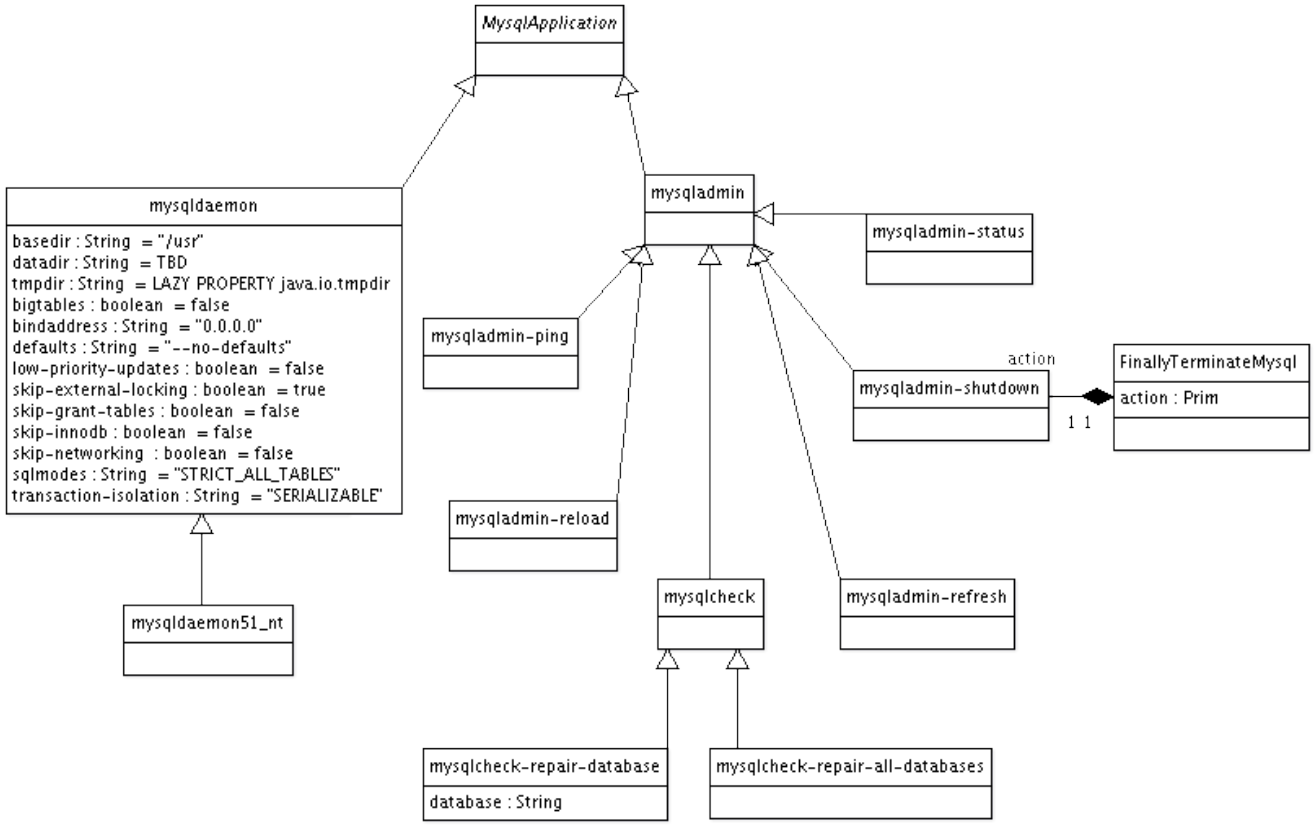
This example has extended the `BlockForJdbcConnection` template defined in section 2.1.5, replacing its condition with the MySQL-specific one.

2.3.1 MySQL Application Components

The MySQL Application Components are a set of components that provide commands to run `mysql` or issue commands against it.

1. These components have only been tested on Linux, not Windows.
2. The MySQL tests all run with security off (`mysqldaemon:skip-grant-tables true`)

3. Many more tests are needed, particularly of shutting down the MySQL program without warning, mid-way through other operations. Or trying to run `mysqlcheck-repair-database` against a live database. Users are encouraged to contribute such test cases, and to avoid creating such conditions in production systems until the tests are completed.



MySQL application components.

One area that has caused trouble in tests is MySQL logging and setting up the database.

1. The mysql database must first be set up, “installed”. Any warning message such as “Table 'mysql.proc' doesn't exist” is a warning that this has not happened. The `mysql_install_db` component can set up the database, running the appropriate `mysql_install_db` shell script for the local platform.
2. Some installations (such as those that come with Ubuntu), set up the binary logging such that no mysqld operations can take place except as root. Changing the permissions to make in `/var/log/mysql` world-writeable corrects this, at the risk of making database security logs vulnerable to tampering by a local user.

2.4 MySQL Licensing

MySQL is a GPL application; its JDBC and JMX components are also GPL licensed. Consult the licensing terms and conditions before deploying a component that loads the MySQL drivers, to make sure that the specific deployment is compliant with MySQL AB's licensing terms:

- http://www.mysql.com/company/legal/licensing/opensource-license.html
- http://www.mysql.com/company/legal/licensing/foss-exception.html

Even if a deployment descriptor is constructed that dynamically downloads the specific JDBC driver at run time, any deployment that is designed to work only with a MySQL database, is implicitly bound to the GPL driver, so must be compliant with the licensing conditions, or have a commercial license to use the software.

SmartFrog does not distribute MySQL, and does not contain any hard-coded requirements for MySQL's presence, except in the unit test suite, which runs MySQL-related tests if `mysqld` is found on the path.

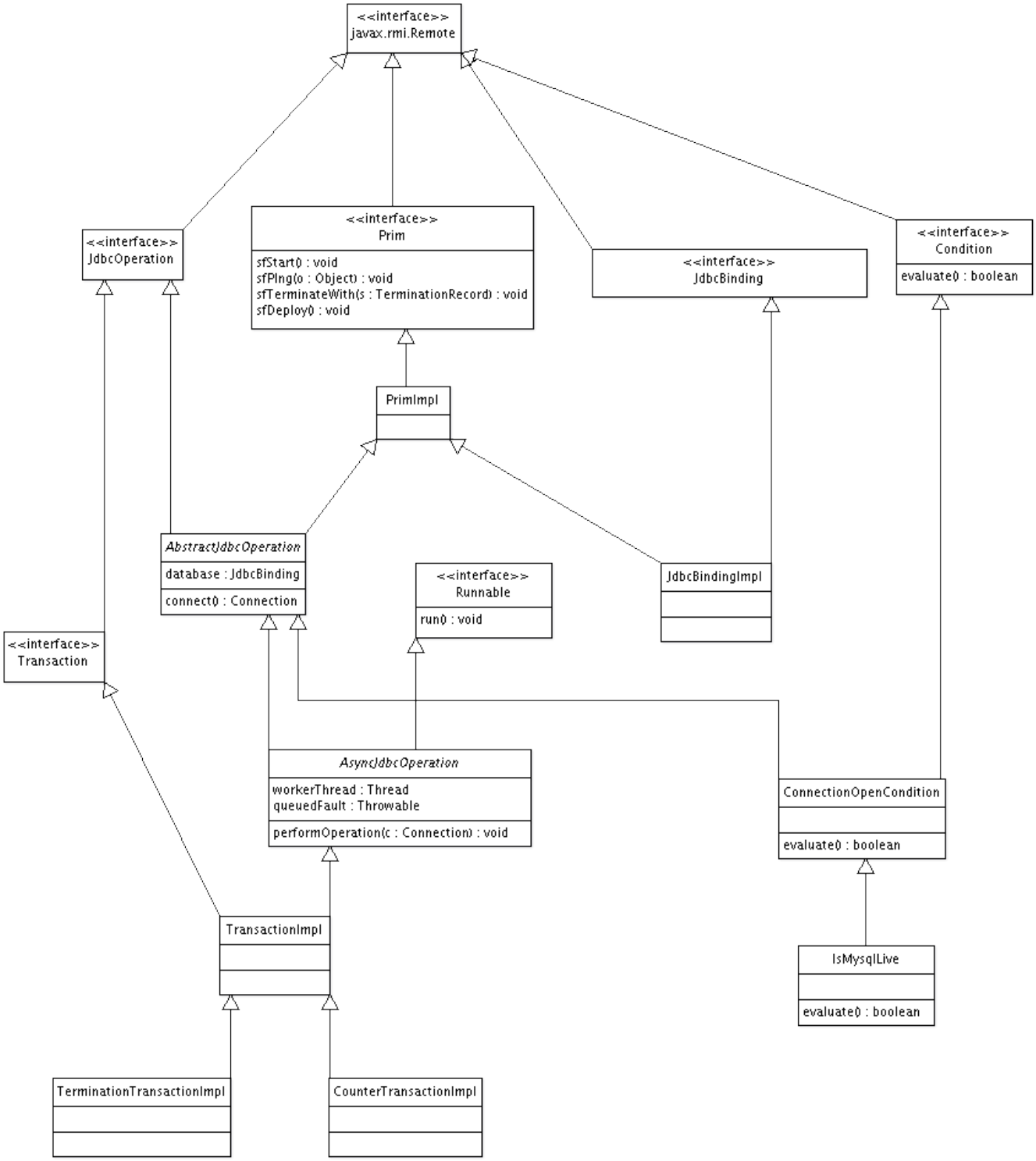
3 Implementation

The UML design of the application is shown below. Many of the Remote interface provide no operations, they only serve to provide information about the type of the component at the far end, and host the attribute names of the component descriptors. All these components are under the `org.smartfrog.services.database` package, in the sf-database library.

The `AbstractJdbcOperation` component contains the code to bind and talk to the database; the `AsyncJdbcOperation` class extends this with a worker thread to run operations in the background; any failure in this thread is propagated to the main thread when the component's `sfPing()` method is called. Non-abstract children of this class must implement the `performOperation()` method; this is called in the worker thread to do the actual database work.

The `TransactionImpl` class builds a list of SQL transactions to run at startup; the `TerminationTransactionImpl` component runs the transactions on termination, rather than startup, so can clean up other component's work. The `CounterTransactionImpl` class runs a different list of transactions on both startup *and* termination, so can be used to create components that change the state of a database on deployment, and roll it back when terminated.

Alongside the components to change the state of the database, come the conditional components, which can be used in workflows to test for the availability of a database, and to block the deployment of components until a database is considered functional. The simplest test, `ConnectionOpenCondition`, tries to open a JDBC connection according to the specified `JdbcBinding`; it returns false if a connection cannot be made. This is adequate for testing that a binding is set up to talk to a database, and, if a shared database, that a connection can be made to it.



The `IsMysqlLive` component goes one step further, and uses introspection to locate any method called `ping()` on the connection. This matches up to recent MySQL JDBC drivers, or with any other JDBC driver that implements an operation with a matching signature.

4 Usage Notes

- Oracle JDBC talks over HTTP, and so if the http proxy settings are wrong, it cannot connect to the database. We recommend deploying SmartFrog with the Java proxy settings set up to bypass any proxy when talking to the database.
- SmartFrog can only start `mysqld` or other database as the user that it itself is deployed as.