

## CP5046 : Project

# SAL-T : Project Documentation

Jochen Braun, Andreas Knirsch, Andreas Seemann

November 16, 2008

*This document is intended to give the reader an understanding about the SAL-T project. It describes how the project was managed and provides the documentation of the resulting software, which has been developed. The project itself is part of subject CP5046 and performed in semester 2-2008 at the James Cook University (JCU), Townsville. The accompanying lecture was hold by Prof. Ian Atkinson. The project was supervised by Chris Christensen.*

## 1 Project Management

### 1.1 Project Plan

The initial project plan was not changed much during the project. The only task that was postponed from the first cycle to the second cycle was testing the performance of DataTurbine [1]. During the first cycle, we realized that it takes more time than expected to set up the DataTurbine and connect a data producer, which sends the data. Therefore the time in the beginning of the project was rather used to evaluate the features of the software. For the performance tests we needed a data producer, which should be the prototype of our sal-DataTurbine connector. We did not have it at that time, so we decided to do performance measuring at the end of the second cycle with real data out of SAL [2].

There were no further changes to the initial project plan and we were able to fulfill all functional requirements as outlined in section 3.

### 1.2 Risks

The identified risks from the beginning of the project did change a little during the project. While the “requirement understanding“ could be set down to medium the ”project scope“ got a more severe. During the project it turned out, that the scope of the resulting software could be extended a lot. What we did, was to stick on the requirements, which were agreed with the customers in the beginning of the project. Some features which would be important to have in the software product were explained to be still open at the final project presentation to the customer. But as they were not formally required they are not part of our resulting software product. In section 6 the potential improvements are described.

## 2 Software Architecture

The SAL-T software can be categorized into a central transmission part, the glue code to the consumer (the SAL) and the glue code to the producer (the SRB [3]). This section outlines those parts in order to give the reader a brief overview of the internal architecture, supported by UML diagrams.

### 2.1 Transmission of data

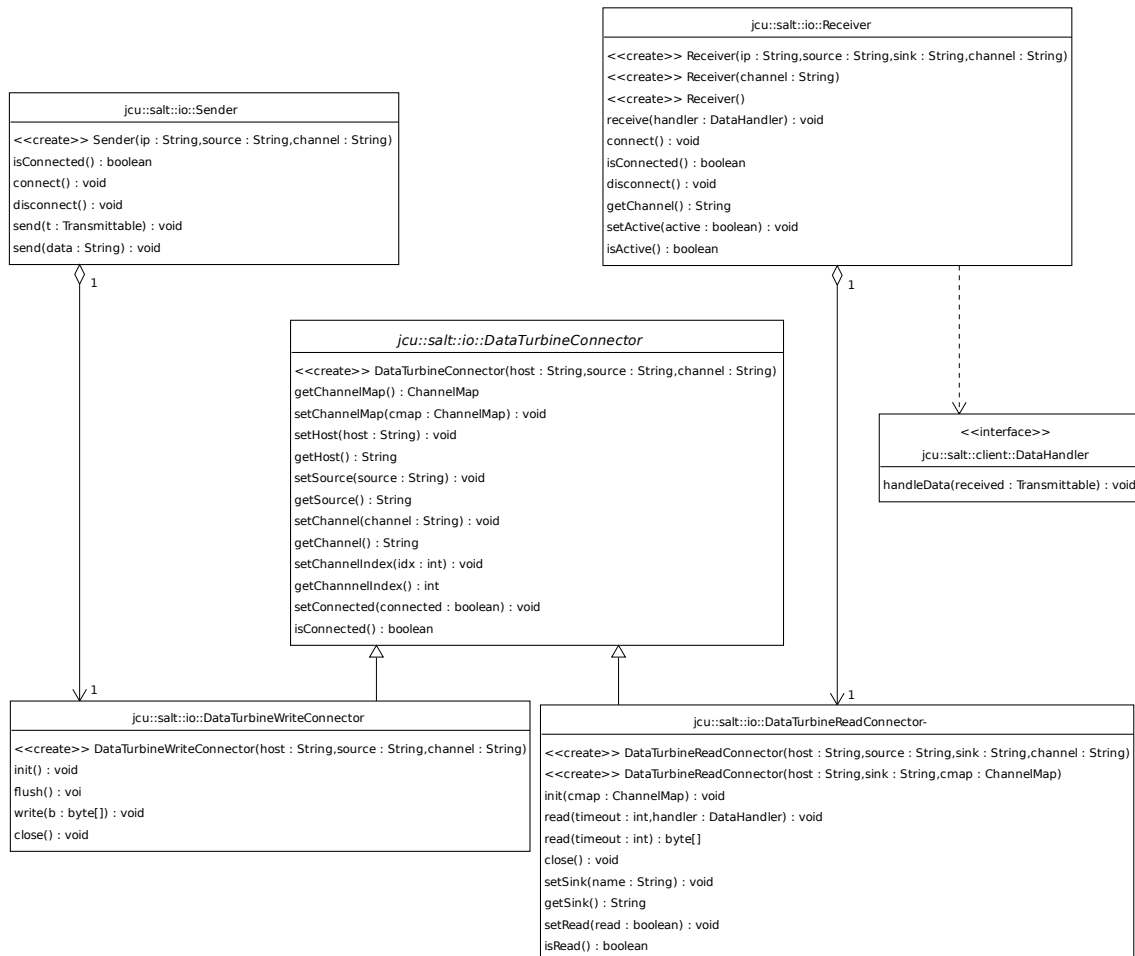


Figure 1: Transmission

The transmission of data into and out of DataTurbine is handled by the five classes shown in Figure 1. The central class is the `DataTurbineConnector`. It defines general methods to access DataTurbine and the specific channels within it. The two classes for read and write access - `DataTurbineWriteConnector` and `DataTurbineReadConnector` - are derived from this class. They are used by other parts of the software to have easy use of read and write functionality to DataTurbine. For example, the `Receiver` and `Sender` classes use their respective counterparts to profit from these simple interfaces to DataTurbine.

| jcu::salt::io::Transmittable  |
|---|
| <pre> serialVersionUID : long timestamp : long dataType : String sensorID : int salID : int data : byte[] </pre>  |
| <pre> Transmittable() Transmittable(salID : int,sensorID : int) Transmittable(salID : int,sensorID : int,dataType : String) Transmittable(salID : int,sensorID : int,dataType : String,data : byte[]) Transmittable(salID : int,sensorID : int,dataType : String,data : byte[],timestamp : long) equals(obj : Object) : boolean setTimestamp() : void setTimestamp(timestamp : long) : void getTimestamp() : long setDataType(dataType : String) : void getDataType() : String setSensorID(sensorID : int) : void getSensorID() : int setSalID(salID : int) : void getSalID() : int setData(data : byte[]) : void getData() : byte[] toString() : String </pre> |

Figure 2: Transmittable Object

## 2.2 Transmittable Object

All communication through DataTurbine is based on sending and receiving Transmittable objects (see Figure 2). The Transmittable class focuses on enhancability to provide better maintainability as a non functional quality for software (see [4]). To do so, it provides the capability to attach metadata information to the actual sensor data.

## 2.3 The Producer classes

The producer classes (see Figure 3) are those classes, which take the main part in handling the connection between SAL and DataTurbine. On one side, the SaltRmiClient establishes the connection to SAL, while opening a SensorThread for each sensor type. The SensorThreads use the Sender class to then push the data into the DataTurbine.

## 2.4 The Consumer classes

Within the consumer classes (see Figure 4) most of the logic for the connection between DataTurbine and SRB is implemented. While the SaltSRBClient uses the SRBConnection class to connect to SRB, it opens multiple TurbineReceiverThreads to read from all SAL-T related channels inside the DataTurbine.

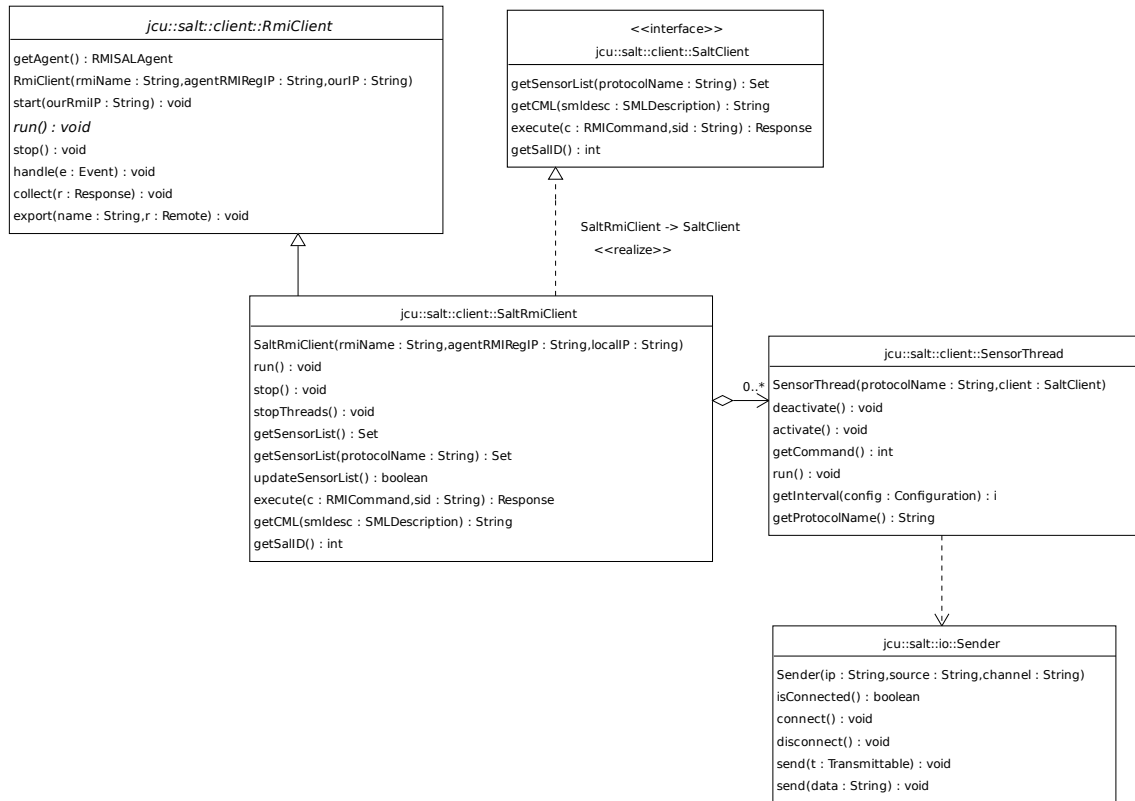


Figure 3: Producer

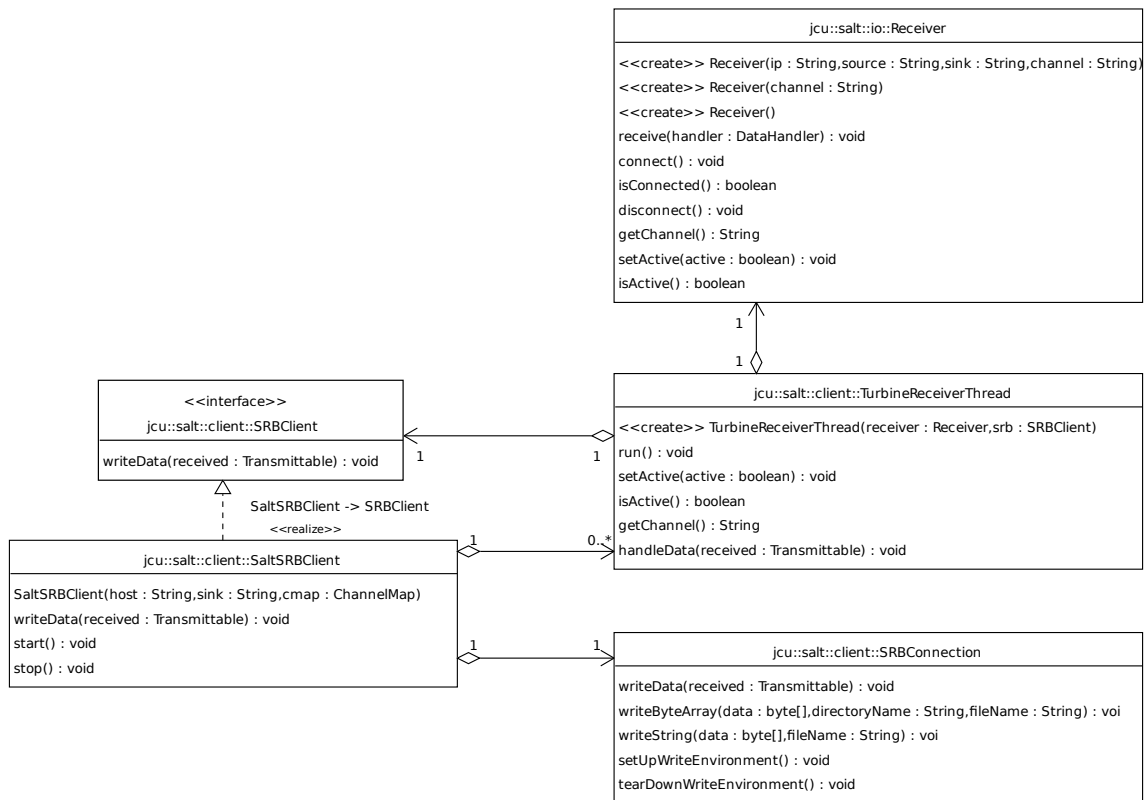


Figure 4: Consumer

### 3 Requirements

This section lists the customer approved requirements and outlines how they have been fulfilled by the project. The sorting reflects the weight of importance appointed to the requirements by the client.

| Requirement  | Fulfilment description   |
|--|--|
| <b>The performance and configuration of Data Turbine shall be quickly evaluated.</b>   | In the beginning of the project the features of DataTurbine were examined. Configuration options can be found in the configuration file of SAL-T. Performance values can be found in section 4.    |
| <b>The data related to each sensor connected to SAL shall be read and sent to Data Turbine. A sensor can be either one of the following: video, 1-wire sensor, snmp or os data</b> | All these sensors are supported.   |
| <b>The connection to SAL shall use RMI.</b>  | The SALRMIClient class is used to connect to SAL. It was provided by SAL.  |
| <b>The data sent to DataTurbine shall include metadata, which is in first place the timestamp and the sensor-id.</b>   | Beside the data itself, the timestamp and sensor id provided by the sensor are transmitted through DataTurbine. Furthermore a SAL identifier is added, which can be set in the configuration file. |
| <b>Sampling rate for sensor data reading shall be configurable depending on the sensor type.</b>   | For each sensor type a thread is being started, which enables the software to poll different sensor types in different intervals. The sampling rate is configurable in the configuration file.     |
| <b>(optional) The connection to SAL shall use local objects/direct communication.</b>  | Because we had to concentrate on the core requirements, this optional requirement was not covered.   |

Table 1: Requirements related to the data producer

| Requirement  | Fulfilment description  |
|--|---|
| <b>Sensor data shall be dumped into the SRB.</b>   | All the data, which is written to DataTurbine is read and dumped to SRB.                                      |
| <b>The format of the sensor data stored into SRB shall be specified; it shall be optimized for data retrieval.</b> | As agreed with the customer a structure based on the timestamps and SAL identifier is used to store the data. |
| <b>The metadata which comes with the sensor data itself shall be stored into SRB.</b>                              | The metadata of the received object is used to create the directory and file name of the data                 |
| <b>To access SRB the jargon shall be used.</b>   | Due to a customer wish jargon is used to access SRB.  |

Table 2: Requirements related to the data consumer

## 4 Performance

This chapter describes three performance test, which are exemplary for the system running on its limits. The hardware and the SAL-T testing parameters for each of the test cases were the same. It is listed in the following two tables. The DataTurbine Parameters changed for the three tests and are therefore responsible for the outcomes. Before each test the software was restarted to get an equal environment. For each test case the java virtual machine performance statistics are provided in a screenshot.

Only the data producer was tested, because it is the critical part of the software. Once data is in the turbine it may take a while to read it and store it in SRB.

| Hardware  |                        |
|-----------|------------------------|
| RAM       | 1 GByte                |
| Harddrive | 64 GByte               |
| Processor | Pentium4 3GHz DualCore |

| SAL-T     |                                    |
|-----------|------------------------------------|
| Channels  | 5000                               |
| Packets   | 25000000 each 174 kByte            |
| Data Size | 25000000 x 174 Bytes = 4.051 GByte |

### 4.1 TEST1

| Data Turbine  |         |
|---------------|---------|
| JVM Heap Size | 1 GByte |
| CACHESIZE     | 5000    |
| ARCHIVESIZE   | 20000   |
| ARCHIVEMODE   | append  |

After half an hour we terminated the test, because no more packets were sent to the DataTurbine for the last 14 minutes. The hard drive was extremely busy, even minutes after we stopped the test case and finally the DataTurbine. The hard drive seemed to be the bottleneck.

Packets sent: 692709 (2,8%)

It turned out that the archive mechanism stored files only some kBytes in size. Altogether 439 MB were archived - the 2,8% of the packets needed more than 10% of the overall data! This would mean the data size is more than trippled, when it is archived in the DataTurbine.

The hard drive was extensively used by the archive mechanism, because very many but small files were written to disk.

### 4.2 TEST2

The Archiving was turned off to see if the archiving caused the trouble with the hard drive.

| Data Turbine  |         |
|---------------|---------|
| JVM Heap Size | 1 GByte |
| ARCHIVEMODE   | none    |

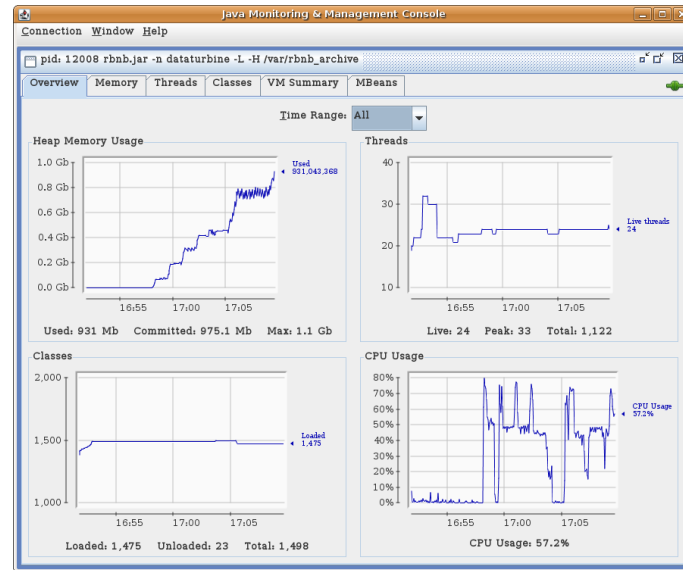


Figure 5: Test1

After three Minutes no more packets were sent, the test case was cancelled after six minutes. The hard drive was very busy again.

Packets sent: 1160492 (4,6%)

Also after three minutes the max heap size was reached. As in Test one the Heap size was equal to the physical RAM size, which means that the harddrive could have been very busy swapping system memory while dataturbine extensively the Java memory. The hard drive was still extremely busy, even minutes after we stopped salt and the Data Turbine.

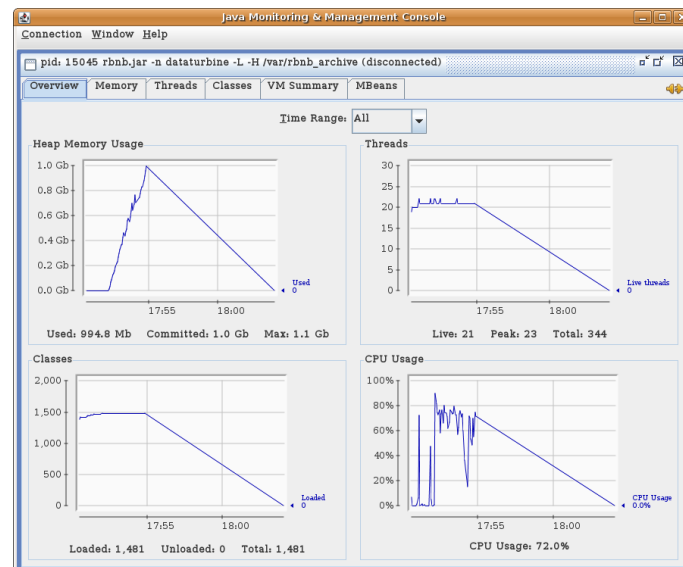


Figure 6: Test2

### 4.3 TEST3

In this test archiving was turned off and the heap size of the java virtual machine was reduced to 512 MBytes, so that there should be no problem with swapping or archiving.

| Data Turbine  |           |
|---------------|-----------|
| JVM Heap Size | 512 MByte |
| ARCHIVEMODE   | none      |

after a few minutes when the maximum heap size was almost reached, the throughput became very slow, just a few hundred packets per minute were sent, after 10 minutes it finally stopped.

Packets sent: 685645

The hard drive was not busy at all at this point, but no more packets were sent and no further connections to the Data Turbine could be established.

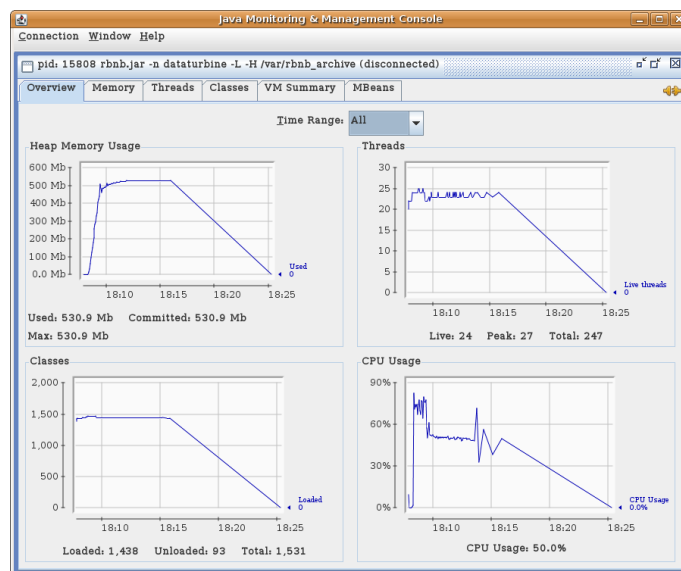


Figure 7: Test3

The tests dumped just as many packets to the turbine as possible. That way the effect on the DataTurbine could be determined in case the system is overloaded. The parameter CACHESIZE would be interesting to test, which was not done yet. To get meaningful test results, a real world scenario would have to be set up. Then the needed hardware and software environment can be adjusted to the needs of this particular case.

## 5 SAL-T Manual

This section is intended to provide information related to the setup and operation of the project result.



## 5.1 Build / Installation

Create a directory where all the components reside. It will be the <BASEDIR>. Copy the SALT directory as well as the SAL directory into this BASEDIR<sup>1</sup>.

```
cd <BASEDIR>
svn checkout http://www.hpc.jcu.edu.au/projects/SAL/svn/SAL/trunk SAL
svn checkout http://sal-t.googlecode.com/svn/trunk/ SALT
cd SALT
ant
```

## 5.2 Configuration

All parts of SALT are highly adaptable to any environment. This is valid for the SAL, the DataTurbine and the SRB connections which have to be established, as well as certain attributes defining the behavior. Therefore the configuration file <BASEDIR>/SALT/salt.properties has to be edited. The comments in the file explain the meaning of the key-value pairs.

## 5.3 Dependencies / Preconditions

- SAL has to be up and running (default is the same machine as the SAL-GLUE is started - configurable within salt.properties (see subsection 5.2))
- DataTurbine should be available (make use of the one we set up in our project room - configurable within salt.properties (see subsection 5.2))
- SRB should be accessible (make use of the one provided by the client - configurable within salt.properties (see subsection 5.2))

## 5.4 Operation

To start the producer component:

```
cd <BASEDIR>/SALT
./run-salt.sh sal
```

To start the consumer component:

```
cd <BASEDIR>/SALT
./run-salt.sh srb
```

There are some more parameters available, which can be used for testing purposes. The parameters "send" and "recv" just send a simple text message through the turbine and receive it on the console. Just run `./run-salt.sh` to see the options.

---

<sup>1</sup>The reason for the need of SAL located in the same directory is related to the argument `java.rmi.server.codebase`, which should be resolved in a further development phase.

## 6 Potential Improvements

Despite the fact that all functional requirements were fulfilled, there is left some decent space for improvements. The current project results should be seen as a starting point for achieving a comprehensive system usable within a operational sensor networks environment. To give the reader some impression where to proceed within a further development phase this section mentions selected limitations which should be improved to increase the softwares usability as well as reliability [4].

### 6.1 Reconnect to SAL

If the connection to SAL is lost during runtime, the connection will not be recovered automatically. The SALT glue has to be restarted in this case. To circumvent this, an exception handling for connection losses plus a polling loop in which reconnects are attempted should be implemented.

### 6.2 Reconnect to SRB / Write errors

Same as with connection losses to SAL. Connections will not be picked up again automatically, so an exception handling and reconnect polling has to be implemented. Errors during the write process to SRB are not handled and will result in data loss. The consumer glue should therefore be enhanced to remember which data has not been written and attempt to write this data in appropriate time intervals.

### 6.3 Refresh of SAL sensor list and protocols

If sensors or protocols are added or removed from a SAL instance, the SALT glue is not updated accordingly. SAL already provides functionality to give callback notices on these events, so these callbacks should be used to update the SALT glue.

## References

- [1] The Open Source DataTurbine Initiative, *Data Turbine Home*. [Online]. Available: <http://http://www.dataturbine.org/>
- [2] G. Gigan, *SAL project overview*. [Online]. Available: <http://plone.jcu.edu.au/dimsim/Members/gillesgigan/sal/sal-overview>
- [3] San Diego Supercomputing Center, *Storage Resource Broker Home*. [Online]. Available: <http://www.sdsc.edu/srb/index.php>
- [4] International Organization for Standardisation (ed.), *Software Engineering - Product Quality - Part 1: Quality Model*, 9126th ed., ISO, Geneva, Switzerland, 2001, last checked 28/10/2008. [Online]. Available: [http://www.iso.org/iso/iso\\_catalogue/catalogue\\_tc/catalogue\\_detail.htm?csnumber=22749](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=22749)