ROSS Software Architecture Document

This document defines the ROSS software architectural design. The software architecture will be the basis for an iterative software development process comprising several phases of prototyping and review.

Summary

ROSS aims to provide a single tool that will allow the RISM to determine the status of the main components of the Radionuclide (RN) monitoring network at any given time, and integrate the station status data and the IDC processing results into a single view within that tool.
# Document History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>2011-03-21</td>
<td>Mark Harrison</td>
<td>Started Initial draft</td>
</tr>
<tr>
<td>0.2</td>
<td>2011-04-19</td>
<td>Mark Harrison</td>
<td>Completed initial draft and ready for peer review.</td>
</tr>
<tr>
<td>0.3</td>
<td>2011-04-19</td>
<td>Marian Harustak</td>
<td>Peer Review</td>
</tr>
<tr>
<td>0.4</td>
<td>2011-04-19</td>
<td>Mark Harrison</td>
<td>Implemented Review feedback from JW and AB</td>
</tr>
<tr>
<td>0.5</td>
<td>2011-04-20</td>
<td>Jan Wüster</td>
<td>Management Review</td>
</tr>
</tbody>
</table>
## Project Summary

<table>
<thead>
<tr>
<th>Project Name</th>
<th>Radionuclide Operations Support System (ROSS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Number</td>
<td>C1103</td>
</tr>
<tr>
<td>Customer/User</td>
<td>Dongmei Han</td>
</tr>
<tr>
<td>Room</td>
<td>E0644</td>
</tr>
<tr>
<td>Contractor</td>
<td>AWST</td>
</tr>
<tr>
<td>Contractor Project Manager</td>
<td>Alex Boresch</td>
</tr>
<tr>
<td>Project Dates</td>
<td>Start (2011-03-01) ➔ End (2012-02-29)</td>
</tr>
</tbody>
</table>

**Contact Information**

- **Customer/User**: Dongmei Han  
  - **Phone**: x6249
- **Room**: E0644
- **Email**: Dongmei.Han@ctbto.org
- **Contractor**: AWST
  - **Address**: AWST GmbH, Mariahilfer Str. 47/3/1, A-1060 Wien, Austria
- **Contractor Project Manager**: Alex Boresch  
  - **Phone**: + 43-1-586 13 14 16
  - **Email**: boresch@awst.at

**Angewandte Wissenschaft, Software und Technologie GmbH**

Mariahilfer Str. 47/3  
1060 Wien, Austria
Contents

Summary ................................................................................................................. 1
Document History .................................................................................................. i
Project Summary .................................................................................................... ii
Contents .................................................................................................................. iii
List of Figures .......................................................................................................... v
List of Tables .......................................................................................................... v

1 Introduction ......................................................................................................... 1
  1.1 Document Definition and Scope ...................................................................... 1
  1.2 Document Objectives ....................................................................................... 1
  1.3 Project Identification ....................................................................................... 2
  1.4 Document Overview ....................................................................................... 2
  1.5 Conventions ..................................................................................................... 3
  1.6 References ...................................................................................................... 3

2 System Overview ................................................................................................ 5
  2.1 System Purpose ................................................................................................ 5
  2.2 System Background & History ........................................................................ 5
  2.3 System Functionality ...................................................................................... 8
  2.4 Overview of ROSS Technical Platform ............................................................ 9
  2.5 Stakeholders .................................................................................................... 9

3 Design Considerations ....................................................................................... 11
  3.1 ROSS Requirements ..................................................................................... 11
  3.2 Technical Platform ....................................................................................... 11
  3.3 Goals and Guidelines ................................................................................... 14

4 Architectural Representation & Methods ............................................................ 15
  4.1 Definition of Software Architecture ................................................................ 15
  4.2 Use of Architectural Views ........................................................................... 15

5 Logical View ......................................................................................................... 17
  5.1 Overview ........................................................................................................ 17
  5.2 ROSS Client Tier ........................................................................................... 17
  5.3 Presentation Layer ......................................................................................... 20
  5.4 Application Service Layer Architecture .......................................................... 22
  5.5 Domain Layer ................................................................................................ 22
  5.6 Utility Layer Architecture .............................................................................. 24
  5.7 Data Layer Architecture ................................................................................ 24
  5.8 Packages and Package Dependencies .............................................................. 25
  5.9 ROSS Domain Model ..................................................................................... 25

6 Process View ....................................................................................................... 26
  6.1 ROSS Client Application ............................................................................... 26
  6.2 ROSS Relational Database Cluster .................................................................. 27
  6.3 Web Server ...................................................................................................... 27

7 Component View ................................................................................................ 29
  7.1 Client Tier Components ................................................................................. 29
  7.2 Database Server Tier Components .................................................................. 29
  7.3 Web Server Tier Components ......................................................................... 30

8 Deployment View ................................................................................................ 31
  8.1 Processing Nodes ............................................................................................ 31
  8.2 Cross-Process Connectors ................................................................................ 32
  8.3 UML Deployment Diagram ............................................................................ 32

9 Technology and Vendor Selections ..................................................................... 33

Software Architecture Document for ROSS  iii
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1 Development Languages</td>
<td>33</td>
</tr>
<tr>
<td>9.2 Development Tools</td>
<td>33</td>
</tr>
<tr>
<td>9.3 Code Quality Tools</td>
<td>33</td>
</tr>
<tr>
<td>9.4 Test, Build and Source Control Tools</td>
<td>33</td>
</tr>
<tr>
<td>9.5 Software Licensing</td>
<td>34</td>
</tr>
<tr>
<td>10 Architectural Factors and Decisions</td>
<td>35</td>
</tr>
<tr>
<td>Appendix 1: Acronyms</td>
<td>44</td>
</tr>
<tr>
<td>Appendix 2: Noble Gas Application Code Review Findings</td>
<td>45</td>
</tr>
<tr>
<td>Introduction</td>
<td>45</td>
</tr>
<tr>
<td>Summary of Static Analysis of ROSS Java Code</td>
<td>45</td>
</tr>
<tr>
<td>General Observations on ROSS Java Code</td>
<td>45</td>
</tr>
<tr>
<td>Appendix 3: ROSS storyboard</td>
<td>49</td>
</tr>
<tr>
<td>Appendix 4: ROSS Requirements Traceability Matrix</td>
<td>50</td>
</tr>
<tr>
<td>Key</td>
<td>50</td>
</tr>
<tr>
<td>Requirements</td>
<td>51</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1: Kruchten’s 4+1 View Model .................................................. 16
Figure 2: The ROSS Plug-in Architecture ........................................... 18
Figure 3: ROSS Deployment Diagram .................................................. 32

List of Tables

Table 1: Typographical Conventions used in this document ...................... 3
Table 2: ROSS software plug-ins ..................................................... 19
Table 3: Main ROSS Java Packages .................................................... 25
1 Introduction

1.1 Document Definition and Scope

This Software Architecture Document (SAD) describes the software architecture for version 1.0 of the Radionuclide Operations Support System (ROSS), for the CTBTO International Data Center. ROSS is being developed by AWST.

All information regarding the software architecture that is available at the time of writing may be found in this document, although much information is incorporated by reference to other documents. In particular, the software architecture is conceived to meet the specified user requirements for the ROSS system, as documented in the Software Requirements [ROSS-SR].

1.2 Document Objectives

The objectives of this document are to:

1. Formally document the architecture of ROSS, including:
   a. An overview of the ROSS application and its architectural drivers.
   b. An overview of the overall architecture of ROSS.
   c. Relevant views of the ROSS architecture.
   d. Dependencies between ROSS and other RN systems.

2. Improve stakeholder understanding of the most important strategic decisions concerning:
   a. The major components of ROSS and their relationships.
   b. How these components will collaborate to fulfil the architecturally-significant requirements and constraints.

3. Provide a foundation for project management and control to:
   a. Plan the further design, implementation, integration testing, deployment, and maintenance of ROSS.
   b. Manage the system’s size, cost, and schedule.
   c. Develop work orders, and work order tasks for the development of the system.
   d. Support the analysis and verification of the ROSS architecture.

4. Improve scope for future extensibility by documenting a flexible architectural foundation on which to extend the system as requirements change.

5. Ensure that the rationales for architectural decisions are not lost, therefore lowering the risk of:
   a. Rejected architectural decisions being revisited in the future.
   b. Accepted architectural decisions being misunderstood in the future.
1.3 Project Identification

The official name of the system described in this document is the *Radionuclide Operations Support System (ROSS)*.

In documentation produced prior to the Kick-Off meeting (e.g. TOR) the software is referred to as RN-OPS.

1.4 Document Overview

In order to fully document all the aspects of the architecture, this SAD contains the following subsections:

**Section 1**: this section, introduces the document.

**Section 2**: describes the ROSS system and the place of ROSS within RN-OPS.

**Section 3**: discusses the design considerations, assumptions and constraints that influence the software architecture.

**Section 4**: describes the models and views used to create and represent the software architecture.

**Section 5**: presents the Logical View of the software architecture.

**Section 6**: presents the Process View of the software architecture.

**Section 7**: presents the Component View of the software architecture.

**Section 8**: presents the Deployment View of the software architecture.

**Section 9**: summarizes technology and vendor selections relevant to the software architecture.

**Section 10**: reviews key architectural factors and decisions in the form of a set of Technical Memos.

In addition to the above there are four appendices:

**Appendix 1**: a list of acronyms.

**Appendix 2**: a code review of the Noble Gas Application.

**Appendix 3**: the storyboard for ROSS v0.1 & 0.2

**Appendix 4**: the user requirements for ROSS
1.5 Conventions

<table>
<thead>
<tr>
<th></th>
<th>Times Roman, italics, capitalized, Pascal case</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ClassName</strong></td>
<td>Times Roman, italics, not capitalized, lower camel case</td>
</tr>
<tr>
<td><strong>methodName</strong></td>
<td>Arial, lower case</td>
</tr>
<tr>
<td><strong>packagename</strong></td>
<td>Courier</td>
</tr>
<tr>
<td><strong>ross_configuration_parameter</strong></td>
<td>Courier</td>
</tr>
<tr>
<td><strong>filename.doc</strong></td>
<td>Courier</td>
</tr>
<tr>
<td><strong>DBID_DataBaseTableName</strong></td>
<td>Times Roman, bold</td>
</tr>
<tr>
<td><strong>ComponentName</strong></td>
<td>Times Roman, italics, capitalized, Pascal case</td>
</tr>
</tbody>
</table>

Table 1: Typographical Conventions used in this document

1.6 References

1.6.1 CTBTO References

[ROSS-SR], *ROSS Software Requirements*, https://www.awst.at/intranet/display/ROSS/Requirements

[ROSS_STORYBOARD], AWST ROSS v0.1 Storyboard, https://www.awst.at/intranet/display/ROSS/Storyboard

[ROSS-TOR], CTBTO IDC, Terms of Reference for Development of Radionuclide Operations Software, 2010

1.6.2 External References


[GOF95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.


2 System Overview

This section provides an overview of ROSS and how it will support the work of the Responsible IMS/IDC radionuclide Staff Members (RISM). [ROSS-TOR] provides a description of the organisational objectives that ROSS must support.

2.1 System Purpose

ROSS aims to provide a single tool that will allow the RISM to determine the status of the main components of the Radionuclide (RN) monitoring network at any given time, and integrate the station status data and the IDC processing results into a single view within that tool.

ROSS will realise this aim by:

- Developing a single tool that replaces the functionality currently provided by the existing tools that are listed in section 2.2.1
- Provide better integration between the various modules.
- Collate data from various sources (Station status data, processing results and products, raw data files) and display them in a single view.
- Provide additional features that are currently not available in the existing tools e.g. a feature to “Show My Stations”. The additional features required are listed in the [ROSS-SR]

2.2 System Background & History

IDC/IMS already have in place a number of tools to monitor the status of the main components of the RN monitoring network, to view station data and processing results. These systems have often been developed in isolation, having very little to no integration at all with one another, have known shortcomings and are using a variety of different technologies and targeting various IDC databases. ROSS will replace these legacy systems with a single tool that is extendable, maintainable and targets the IDC-OPS database.

2.2.1 Existing Systems

ROSS will eventually replace the following legacy systems:

1. Workflow Tools

The Workflow tools are designed to provide a high-level view of the timeliness of messages sent from stations, including if a message was sent at all, or/and the processing status of those messages.

There are a several workflow tools in use at the CTBTO providing different types of information and features. It is desired that all of the required information and required features that these tools currently provide are merged into a single workflow tool:

1. Robert Werzi’s Workflow Tool
This is the main workflow tool that is used by RISM. It is developed by Robert Werzi using HTML and CGI scripts. This workflow tool shows the timeliness of messages sent from stations. RISM like how the data is presented in this tool and want this format to be reused in ROSS. However RISM have identified a number of shortcomings with this tool:

a) It’s doesn’t scale. The tool only shows the last 48 hours worth of data. This can cause problems when reviewing data generated during a weekend because the data generated on a Friday night is not visible.

b) There is no visible link between this workflow tool and the raw data.

c) Once a message has been received there is no indication if a message arrived on time or was late.

d) There is no link between this tool and the tools used to look at the message data in more detail. E.g. SOH Application and Noble Noble Gas Application.

2. Dennis Giordanino’s Workflow Tool

The NG software developed by Dennis Giordanino also contains a workflow tool that shows the timeliness of messages sent from the stations. This workflow tool can only show data for a single station at any one time and RISM do not like how the data is presented in this tool but they do like the zooming feature that is available.

3. Shaban Laban’s Workflow Tool
(http://kuredu.idc.ctbto.org:8000/WebFlow/RnSamples.html)

This workflow tool was developed by Shaban Laban as a Java applet application and shows the processing status of messages. This information is of secondary importance to RISM and is rarely used.

4. IDC Workflow Tool

Though not used by RISM, the processing engineers at IDC have their own IDC Workflow tool that shows the processing status of messages. Two features of this tool are of particular interest to RISM:

a) Input Failure & Input Process Workflow Lines

In this workflow tool there are two additional lines (Input Failure & Input Process). These lines show messages that could not be inputted or processed and therefore could not be associated with a station. The Input Failure line shows messages that failed to be inputted into IDC database and Input Process shows all messages received.

b) Processing Statuses

It has been decided that ROSS will mimic the processing statuses that are used in the IDC Workflow Tool instead of the Shaban Laban’s Workflow Tool. The reason for this decision is primarily to introduce consistency between the various workflow tools in use at the IDC.

It should be noted that ROSS will not replace the IDC Workflow Tool. The IDC Workflow Tool will remain the tool used by the IDC processing engineers.
2. SOH Application

The SOH application was developed by Robert Werzi as a thick client application in Visual Basic 6.0. Its sole focus is on presenting particulate sample data including processing results in graph format (81 different graph types are supported) and the reviewing of particulate sample data.

3. Noble Gas Application

The Noble Gas application was developed by Dennis Giordanino as a thick client application in Java. Its focus is on presenting Noble gas sample data including processing results in various different formats (graph, table, raw, workflow and flowchart). Some work was also invested in supporting particulate sample data but this was at an early stage in development and has since been commented out.

Dennis has also developed two small applications to support the NG software:

- an application to create new NG users and configure their access permissions
- an application to configure the flow chart layouts and the database queries that are executed when an item is clicked in the flowchart.

4. Test & Evaluation Report

The Test & Evaluation Report (TER) is a web-based report developed in HTML and CGI scripts by Robert Werzi that is used for the testing and evaluation of new stations that are being introduced into the IMS RN monitoring network. The high-level report shows a summary of the sample messages (e.g. Acquisition and Collection Start Time) received for a particular station and a certain month. Clicking on a single sample allows a user to drill down into a detailed report for that particular sample. This functionality will be integrated into ROSS. Dennis Giordanino had started developing a TER for Noble gas station but this work was never completed.

5. Command Tool

RISM are able to issue commands to a station to perform certain operations (e.g. Send Sample) through a web tool developed in HTML and a CGI script developed by Robert Werzi. The commands are sent via email in an agreed format. This tool is rarely used by RISM. In addition to replacing this tool they would like to see known information (e.g. Station Name) preselected/entered and to be able to view the replies that are sent in response to their commands in the same view. Currently they can only be viewed through an email client (e.g. Outlook or Firebird) and by navigating through the responses until the appropriate response is found.

6. Raw Data Viewer

RISM are able to navigate and view the raw data messages through a web based raw data viewer. The messages are organized in hierarchical format by Station, Year, Message Type and Month. RISM do not wish to replace this web viewer but
want better integration between ROSS and the raw data. They have identified several places where they wish to be able to view the Raw data in ROSS:

a) Clicking on a message in the workflow
b) Clicking on a point in a graph.
c) Clicking on an item in the table format when viewing the data used in the various graphs.
d) Clicking on an item in the list of alerts and messages that failed to be processed.

RISM want the raw data viewer in ROSS to provide additional features such as Searching, Filtering, Highlighting, and Navigating to previous/next message.

2.3 System Functionality

The scope of ROSS version 1.0 has been restricted to providing the functionality listed in this section. The need for additional functionality is foreseen post ROSS v1.0, e.g. the migration of the current laboratory operations software to ROSS.

2.3.1 Operations Functions

In this section we list the day-to-day operational functions of ROSS:

1. **Workflow**: Provides a general overview of the RN monitoring network by plotting incoming messages and IDC processing status to a workflow showing the status of the RN stations, with a direct link to the raw data stored in the file system.

2. **Station flowchart**: A station flowchart for different types of Radionuclide particulate and Noble Gas stations, which shows colour-coded status of sensors at that station.

3. **SOH interactive review**: An Interactive review tool that displays all SOH parameters and sample results for each RN station (particulate and Noble Gas), insert comments, link to IMS Reporting System to open reports if needed, and link to the raw data stored in the file system.

4. **Commands**: A tool for accessing an email account for receiving and sending plain text messages to issue commands to station operator for Special measurements, sending samples, Change decay, Update Calibration pairs, etc.

2.3.2 Evaluation Functions

In this section we list the functions that are related to the evaluation of stations performance and the evaluation of IDC results that are produced from station samples.
1. **Station sample report**: A summary report of station samples, including SOH parameter check, reporting time check, sample categorization result, radionuclides identified by IDC, operation and analyst comments, etc.

2. **Station performance**: A report evaluating station performance by means of Key Performance Indicators over a period of time (daily, monthly, quarterly, yearly and a period according to specified evaluation time).

3. **Statistics Report**: Generation of statistical distribution of SOH parameters, as well as statistical report for IDC result regarding sample categorization and radionuclide concentrations.

### 2.3.3 Software Administration

In this section we list the administration functions of ROSS:

1. **Station/user Configuration**: add/delete stations, users, managing user access levels, Customize station groups according to user’s responsibilities.

2. **Configuration**: Possibility to customize database queries e.g. Flowchart queries; Configure database connections;

3. **User Preferences**: Possibility for a user to customize ROSS. e.g., allow a user to set default settings for auto refresh, date range filters, start mode.

### 2.4 Overview of ROSS Technical Platform

ROSS will be developed as a client-server application, with a GUI client executing on the user’s workstation. The client application is a traditional “thick” client with the business logic being executed on the client workstation. The client tier is developed in Java (Swing UI libraries) and uses the NetBeans Platform [NETBEANS_PLATFORM]. Other open source components will be utilised in ROSS. These include: JFreeChart [JFREECHART] for graphing requirements, Jasper Reports [JAPSER_REPORTS] for reporting requirements and Log4J [LOG4J] for logging requirements. The server tier is an Oracle database cluster. Access to the database cluster is via the root cluster instance using the Hibernate Framework [HIBERNATE_FRAMEWORK].

Deployment of the application is achieved using Java Web Start [JAVA-WEBSTART], eliminating the usual problems of installing and maintaining client-server application across an organisation.

More detail on the chosen technical platform for ROSS is provided in the later sections of this document (see Chapter3: Design Considerations and Chapter4: Architectural Representation & Methods).

### 2.5 Stakeholders

Stakeholder analysis has identified the following ROSS stakeholders:

- **IDC Management**: Project Sponsor, Overview and responsibility for IDC activities.
**IDC End User Roles**

- **Responsible IMS/IDC Radionuclide Staff (RISM):** the day-to-day user of ROSS. This will be their main tool for reviewing the status of the main components of the RN monitoring network and the data quality of station data.

- **IDC Analysts:** responsible for the analysis of samples. They often communicate with RISM when discussing data quality issues.

- **OPS Centre duty staff:** responsible for the day-to-day running of the OPS centre and first POC for station data issues. They communicate with RISM when issues regarding data quality issues occur in the OPS centre.

- **IDC Processing Engineers:** responsible for the processing of station messages.

**IDC SA Section:** responsible for administering and maintaining ROSS.
3 Design Considerations

3.1 ROSS Requirements

During the initial handover phase of the project, requirements were gathered from several different sources:

- TOR
- Technical interviews with key users
- Examining the legacy tools including their source code.
- Technical interview with Robert Werzi
- Storyboard workshop with key users

The requirements that have been gathered during these interviews and that should be addressed under the ROSS project are listed in Appendix 4.

3.2 Technical Platform

3.2.1 Client Tier

The client tier is a rich thick Windows/Linux GUI client that executes locally on the user’s workstation. The client tier will be developed in Java SE6. The visual interface of the client is developed using the Swing UI libraries and the NetBeans Platform. ROSS will also utilize a number of other open source components in the client tier. These include: JFreeChart [JFREECHART] for graphing requirements, Jasper Reports [JASPER_REPORTS] for reporting requirements and Log4J [LOG4J] for logging requirements.

3.2.2 Database Tier

The database tier is an Oracle database cluster. ROSS should connect to the Real Application Cluster (RAC), not to individual nodes in the cluster. Access to the database cluster is via the Hibernate platform [HIBERNATE_FRAMEWORK]. Database connections are standard Oracle connections over TCP/IP.

ROSS will target the IDC OPS database but will be developed against the current schema from the IDC Testbed database because this schema will be promoted to the IDC OPS database by the time ROSS is ready for production.

3.2.3 Web Server Tier

The [JAVA-WEBSTART]–deployed client is hosted on a Web Server. This web server is solely used as a deployment platform for the client binaries; it does not host any active content that accesses the CTBTO databases.

3.2.4 Email Server Tier

ROSS will connect to the CTBTO email server for retrieving and sending command messages. Access to the Email tier will be via SMTP and IMAP.
3.2.5 File Store Tier
ROSS will connect to the CTBTO-File-Store for retrieving the raw data messages. Access to the FileStore tier will be via SSH.

3.2.6 Network
The client workstations and the database server will run on the same TCP/IP Local Area Network at the CTBTO.

3.2.7 Reuse of the existing software
After examining the existing applications we foresee being able to reuse some of the source code from the Noble Gas application. The source code that can be reused will be migrated to the NetBeans Platform early in the first development phase. A code review of the NG software is available in Appendix 2. It shows that significant refactoring will be necessary.

The other applications listed in section 2.2.1 will be used as references to identify how functional behavior was implemented and identify database queries used for retrieving specific data.

3.2.8 Plug-in Architecture
ROSS will be developed has a set of plug-ins [PLUGIN_ARCHITECTURE] that plug into the NetBeans platform and work and interact with one another to form the ROSS application. Each ROSS plug-in will be given a defined responsibility, (See Table 2) and interactions with that plug-in will be through defined extension points. A plug-in architecture has been chosen for ROSS for following reasons:

1. **Extensibility**: ROSS can be easily and dynamically extended in the future to include new features.
2. **Reuse**: The plug-ins developed during ROSS have the potential to be reused in other CTBTO projects.
3. **Simplicity**: Each Plug-in will have a defined responsibility and focus.
4. **Independent Development**: Plug-ins could be developed in the future for ROSS without requiring intimate knowledge of other ROSS plug-ins.

One of the reasons for building ROSS on top of the NetBeans Platform was because of the flexible “plug-in” architecture offered by the NetBeans Platform.

3.2.9 Security requirements and regulations
Access to ROSS should be restricted to authorized personal only. Furthermore the identity of the authenticated ROSS users should be used to provide a fine-grained access permissions to ROSS features. E.g., Administration features should be restricted to certain administration users only.

Authorisation requirements for the current release of ROSS have not yet been fully defined. Furthermore a request to provide a single sign-on solution has been requested by Mika Nikkinen at the kick-off meeting and a cross platform authentication solution is desired. Implementations of a single sign-on solution using
Active Directory for Windows have already been implemented in software applications at the CTBTO. Systems that have implemented single sign-on authorisation at the CTBTO will be examined before a design is defined and implemented for ROSS. AWST’s POC for discussing single sign-on implementations at the CTBTO is Enrique Castillo, Ops Center Manager.

3.2.10 Network communications

Network communication within the secure network at CTBTO is via TCP/IP. Communication with the ROSS web server is over HTTP. Database connectivity between the client and server tiers will be via Oracle OCI library.

3.2.11 Look and Feel

The new ROSS Graphical User Interface (GUI) will not conform to any specific guidelines or specifications on look and feel. It will however aim to:

1. Provide a standard Java on Windows/Linux user interface, using standard controls, dialogs, menus and other cues and features in order to maximise familiarity for the users.
2. Provide an ergonomic user interface experience that follows the basic principles of software GUI usability.
3. Improves upon the user experience of the existing applications.

3.2.12 Testing and Deployment

AWST will be responsible for unit testing and system testing of ROSS. Unit testing will be carried out in the AWST development and build environment. Integration and functional testing will be carried out in the AWST test environment.

CTBTO are assumed to be responsible for acceptance testing of ROSS (on the IDC Test System) and the deploying of the final version into IDC OPS.

3.2.13 Deployment

Deployment of the ROSS client must be simple and straightforward. There are many workstations within the CTBTO and executing and maintaining many local installations of the software should be avoided.

For this reason the client tier of ROSS will be distributed as a Rich Client using the Java Web Start platform [JAVA-WEBSTART]. The components of the ROSS client will be hosted on an Apache Web Server within the CTBTO network and launched by the users via an HTTP URL. Updates to the ROSS software across the entire user base can then be achieved by updating the published version hosted on the web server.

During development, user trialling and formal testing of the ROSS software it is expected that many prototype versions of the software will be directly deployed onto the AWST publication server using the build management tools.

Production deployment will be more controlled. Distribution of new versions of the ROSS to the users will only take place after acceptance testing.
3.3 Goals and Guidelines

Other goals and guidelines for the architecture and development of ROSS are listed below:

- Building a single ROSS application
- Functional integration of the ROSS modules to provide a richer user experience
- Making ROSS easily extendable and maintainable
- General compliance with relevant “Patterns and Practises” for Java, including validation by PMD, FindBugs and CheckStyle.
- Usage of a layered, domain-driven application design.
- Usage of an object-relational mapping-based data-access layer.
- Use of Unit Tests (using NB jUnit, Jemmy) during development.
- Documentation of variables, procedures and algorithms directly in the source code using the standard JavaDoc syntax.
4 Architectural Representation & Methods

4.1 Definition of Software Architecture

The “software architecture” for a system is the structure of that system, comprising all of its software elements, the properties of those elements, and the relationships between them. The IEEE defines software architecture as:

“...the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution”

[IEEE-1471]

The definitive software architecture for a system ultimately resides in executable code; however a system’s architecture may be visualized and planned using models. This SAD provides such a model.

A software architecture model describes the public interfaces of software elements, how these elements are used, how they relate to each other and how they interact. Low-level specifications for the internal implementation details of software elements that are irrelevant to their interaction and usage (e.g. algorithms) are generally omitted from a software architecture. This division between high-level architecture and low-level design and implementation is reflected in this SAD. Private details of software elements solely relevant to internal implementation are not architectural and will not be documented in the SAD.

4.2 Use of Architectural Views

Software systems are complex and can be difficult to grasp all at once. Moreover, the multiple stakeholders in a system have differing concerns, which leads to a need for multiple blueprints of the system. “Architectural Views” are a standard means of dealing with this complexity by restricting attention at any one moment to a small number of the system’s structures [IEEE-1471], [IEEE-1016], [Larman02]. A view is a representation of a system from the perspective of a particular set of concerns. This SAD follows the principle that documenting a software architecture is a matter of documenting the relevant views.

There are many possible classifications of architectural views. The view model used in this document is the classic “4+1 View Model” originally defined by Philippe Kruchten in 1995 for the Rational Unified Process [Kruchten]. The 4+1 View Model describes software architecture using four basic concurrent views, each of which addresses a specific set of concerns:

1. The **Logical** or **Design View** describes the design's object model and packaging, and its decomposition into logical layers;

2. The **Process View** describes the design's concurrency and synchronization aspects;

3. The **Component, Implementation** or **Development View** describes the software's static organization in the development environment. The system is arranged into layers or components (physical layers, as opposed to the logical layers in the Logical View) and the communication lines between those layers are defined.
4. The **Deployment** or **Physical View** describes the mapping of the software onto the hardware and shows the system’s distributed aspects;

These four basic views can then be illustrated with a fifth view, the **Use Case View**, which contains selected usage scenarios as demonstrators of the architecture.

In addition to the standard 4+1 Views defined by Kruchten, it is sometimes common to also include an optional **Data View** which describes the major data stores of the system. A Data View is not included in this document as the Data Layer is covered in other Views.

In addition to simplifying the depiction of complex software architectures, the 4+1 View Model also allows various stakeholders to quickly find what they need in the software architecture. System engineers can approach it first from the physical view, then the process view; end users, customers, and data specialists can approach it from the Logical View; and project managers and software-configuration staff members can approach it from the Component View.

The standard 4+1 Views are summarized in the diagram below:

![Diagram of Kruchten’s 4+1 View Model](image)

**Figure 1: Kruchten’s 4+1 View Model**

Logical View, Component View, Process View and Deployment View are provided in the following sections. Given the graphical nature of the required ROSS features, a storyboard has been elaborated rather than use cases. The storyboard is attached to this document as Appendix 3: ROSS storyboard,
5 Logical View

Under the 4+1 View Model the “Logical View” models the translation of the system requirements into functional aspects of the system. It identifies the main classes, subsystems and packages and applies a separation of concerns and distribution of responsibilities. The main focus in this view is on how specific functionality is satisfied by architectural abstractions (i.e. use case realizations and the domain object model). As such this is the main view for modeling of the problem domain.

5.1 Overview

The ROSS application is decomposed into several distinct layers, which are in turn distributed across a standard 2-Tier client server model. This layering model defines each layer according to a particular set of responsibilities. This model has been chosen because it encapsulates and isolates the various system responsibilities from one another, simplifying both system development and maintenance.

The high-level ROSS tiers are:

- ROSS Client Tier: the ROSS Java application including GUI and business logic.
- IDC OPS Database Tier: the Oracle database server, containing both the IDC data store and any new tables added by ROSS.

These are supplemented by a secondary web server tier which is not strictly a part of the ROSS client-server application.

- ROSS Web Server Tier: the Apache publication server.

These deployment tiers are described in more detail in the Deployment View chapter of this document.

5.2 ROSS Client Tier

As already mentioned in Section 3.2.8, ROSS will be developed as a series of plug-ins that plug into the NetBeans platform. Figure 2 shows the ROSS plug-ins that have been identified for development and are contained within the client tier. Table 2 shows what functionality each plug-in will be responsible for providing and in Appendix 4 we have a traceability matrix that traces the user requirements to the plug-in modules.
Figure 2: The ROSS Plug-in Architecture
<table>
<thead>
<tr>
<th>Module Name</th>
<th>Description of the module’s functionality</th>
<th>Layer</th>
<th>Storyboard Ref.</th>
<th>TOR Ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROSS Workflow View</td>
<td>This plug-in will be responsible for the workflow view.</td>
<td>Presentation Layer</td>
<td>Screen 2</td>
<td>5.2.2. Sub-module 1</td>
</tr>
<tr>
<td>ROSS Station</td>
<td>This plug-in will be responsible for presenting the data for a specific station in various formats (Graphs, Data, Flowchart).</td>
<td>Presentation Layer</td>
<td>Screens 8 - 13</td>
<td>5.2.2. Sub-module 2 &amp; 3</td>
</tr>
<tr>
<td>ROSS Commands</td>
<td>This plug-in will be responsible for the Command view. The Command view will retrieve and send commands.</td>
<td>Presentation Layer</td>
<td></td>
<td>5.2.2. Sub-module 5</td>
</tr>
<tr>
<td>ROSS Administration</td>
<td>This plug-in will be responsible for the administration functionality. This functionality could be separated out into three separate plug-ins, Station/user Configuration Administration plug-in, Database Configuration plug-in and a Software Configuration plug-in.</td>
<td>Presentation Layer</td>
<td></td>
<td>5.2.4.</td>
</tr>
<tr>
<td>ROSS Raw Data Viewer</td>
<td>This plug-in will be responsible for the functionality for viewing the raw data messages.</td>
<td>Presentation Layer</td>
<td>Screens 4 &amp; 5</td>
<td>5.2.2. Sub-module 4</td>
</tr>
<tr>
<td>ROSS Reports</td>
<td>This plug-in will contain the ROSS reporting functionality.</td>
<td>Presentation Interface</td>
<td></td>
<td>5.2.3.</td>
</tr>
<tr>
<td>ROSS Common GUI</td>
<td>This plug-in will contain common GUI components that can be reused throughout ROSS and in other RN applications. Some examples include station hierarchy, date filter and message properties.</td>
<td>Presentation Interface</td>
<td>Screens 1 &amp; 2</td>
<td>N/A</td>
</tr>
<tr>
<td>ROSS App Services</td>
<td>This plug-in is responsible for presenting a number of use-case driven Façade services to the Presentation Layer that simplify access to the Domain Layer.</td>
<td>App Service</td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td>ROSS Domain</td>
<td>This plug-in is responsible for the core business logic of the ROSS application</td>
<td>Domain</td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td>ROSS Utils</td>
<td>This plug-in is responsible for providing non-domain specific utility functions. An example would be a utility class to identify if the user has admin security privileges.</td>
<td>Utility Layer</td>
<td></td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Table 2: ROSS software plug-ins**

The ROSS plug-ins described in Figure 2 and Table 2 fall into one of the following architecture layers:
• **Presentation Layer**: responsible for the displaying of the data in the ROSS graphical user interface and handling user events such as mouse clicks.

• **Application Service Layer**: responsible for presenting a number of use-case driven Façade services to the **Presentation Layer** that simplify access to the **Domain Layer**. This layer also provides cross-cutting aspects to the system like transaction boundaries and security checks.

• **Domain Layer**: responsible for the core business logic of the ROSS application.

• **Utility Layer**: responsible for providing non-domain specific utility functions.

Each of these logical layers is addressed in detail in a section below.

### 5.3 Presentation Layer

#### 5.3.1 Overview of Presentation Layer

The Presentation Layer of ROSS is the visible, interactive part of ROSS. It is a thick Java 6SE client, built using the Swing UI libraries and NetBeans. NetBeans is used to wire together the UI components and to create a well-structured, standardised GUI layer.

**5.3.1.1 The NetBeans Platform**

The NetBeans Platform [NETBEANS_PLATFORM], forms the basic foundation of the ROSS Presentation layer implementation. It is a leading Java/JEE application framework for constructing Java rich client graphical user interfaces in a configurable, standards-compliant and efficient manner. The current version of NetBeans Platform supports Swing GUI applications. In addition to a set of reusable libraries, the NetBeans Platform also provides a proven foundation of patterns and good practises for quickly developing Swing applications. The NetBeans Platform was selected for ROSS because:

- It provides a structured and standardised approach to building Swing applications which adheres to well-established principles of object-oriented software development.

- It promotes consistent, well-designed, configurable Swing applications which have the look-and-feel of native applications. This increases the familiarity of the GUI for end-users and reduces the learning curve for new users of the system.

- It provides a large set of useful GUI development libraries and patterns which increase the speed and efficiency of development.

- Its plug-in architecture will allow ROSS to be easily extended and provides the potential for its functionality to be reusable in other CTBTO projects.

- It is actively maintained and its community provides good developer support in the form of Wikis, books, tutorials, forums, examples, and screen casts.

#### 5.3.1.2 Jasper Reports

JasperReports [JASPER_REPORTS] is a comprehensive Java reporting solution developed by JasperSoft. It’ll be used in ROSS to generate the various required reports. It was selected for the reporting requirements in ROSS because:

- JasperReports produces pixel-perfect reports that can be viewed, printed
or exported in a variety of document formats including Microsoft Excel, PDF, RTF, Open Office, CSV, and HTML.

- **JasperReports** supports data coming from a variety of data sources including JDBC, XML, CSV, Hibernate, EJBs, Oracle PL/SQL stored procedures.
- JasperReports supports data coming from a variety of data sources including JDBC, XML, CSV, Hibernate, EJBs, Oracle PL/SQL stored procedures.
- JasperReports can contain charts and graphs such as Pie, Bar, Stacked Bar, Line, Area, Scatter Plot, Bubble, and Time series.
- Reports developed using JasperReports can be separated from the Java code. This allows reports to be modified without requiring a recompile of the Java source code.
- JasperReports can contain drill-down reports.
- JasperReports provides a component for viewing JasperReports inside a Java application.
- There is an open source WYSIWYG report designer specifically designed for JasperReports.

### 5.3.1.3 JFreeChart

JFreeChart [JFREECHART] will be used to plot the various graphs that are required in ROSS. It was selected for the graphing requirements in ROSS because:

- It is already used in the NG software application. Using JFreeChart will allow reuse of the existing NG graphs.
- It provides a number of out-of-the-box features that ROSS users have requested e.g. overlapping graphs, zoom in and out.
- It is the leading Java graphing technology.
- It is both free and open source.
- It is well supported with good documentation and examples.
- It is the graphing technology used in JasperReports.

### 5.3.2 Connections and Communication Channels

The Presentation Layer also communicates directly with the Business Layers (both the Application Service and Domain Layers). There also exist implicit dependencies on the ORM component of the Data Layer. The Presentation Layer may also communicate with any components in the Infrastructure Layer that it requires access to. These communications are by means of in-process method calls made against the assemblies that encapsulate these layers.

### 5.3.3 Summary of Key Features

Most ROSS end-user functionality is made available to the users in this layer.
5.4 Application Service Layer Architecture

5.4.1 Overview of Application Service Layer
The Application Service Layer is the interface between the Presentation Layer and the ROSS Domain Layer. It provides the Presentation Layer with simple and directly usable methods to access the logic encapsulated within the entities of the Domain Layer. This layer co-ordinates and simplifies access to the finely-grained Domain Layer, and provides non-Domain-specific aspects (e.g. security, transactions etc.) to ROSS. Application Services additionally provide a more “use-case literate” interface over the domain layer, considering, as an example a method `StationService.RetrieveParticulateStations()`. The Application Service classes used in ROSS are instances of the Façade pattern [GoF95] and do not contain any real domain logic.

5.4.2 Connections and Communication Channels
The Application Service Layer communicates directly with the ROSS Domain Layer. It also utilizes the ORM component of the Data Layer in order to specify transaction boundaries. It may also communicate with any components in the Infrastructure Layer that it requires access to. All of these communications are by means of in-process method calls made against the assemblies that encapsulate these layers.

5.4.3 Summary of Key Features
The Application Service Layer is responsible for delivering the following key features:
- Method entry points for the execution of key use cases.
- Methods for basic creation, retrieval, updating and deleting (CRUD) of domain objects. This includes methods for both simple and complex object querying (e.g. single objects by primary key value, lists of objects by search criteria).
- Sorting and filtering of lists of domain objects.
- Transaction boundaries, particularly for aggregate use case methods (use case methods which reuse several other use cases at the method invocation level).
- Security permission checks on method calls.
- Logging exceptions generated by lower layers.

5.5 Domain Layer

5.5.1 Overview of Domain Layer
The Domain Layer contains the actual ROSS business/domain logic and as such is the most important layer in the application. The classes contained in the domain layer fall into three broad categories, as documented in [Evans] and [Fowler]:
- **Domain Objects** or **Entities**: a type of business object that is defined by having a unique identity throughout its lifetime (e.g. a primary key). An example of such an object in ROSS is an object representing a specific “Station”. Instances of such types are usually persisted to the data store.
• **Value Objects**: immutable business objects without identity that are distinguished only by the value of their attributes e.g. an enumeration value or an amount of currency. An example of such an object in ROSS is an enumeration value representing a “Station Type”.

• **Domain Services**: domain concepts that tend to be "actions" rather than "things" e.g. "FundsTransferService.TransferFunds()". These services contain real business logic that does not clearly belong on any Entity class. The distinction between Application Services and Domain Services can often be subtle; the discriminator is the presence of actual domain logic.

5.5.2 **Connections and Communication Channels**

The Domain Layer communicates with the ORM component of the Data Layer and with any components in the Infrastructure Layer it requires access to. These communications are by means of in-process method calls made against the assemblies that encapsulate these layers.

5.5.3 **Summary of Key Features**

The Domain Layer is responsible for delivering the following key features:

- Encapsulating state and behavior of domain objects.
- Defining the associations between domain objects.
- Declaring the object-relational mapping of entities to database tables using Hibernate mapping files (see below).

5.5.4 **Object-oriented Data-Access**

The Domain Layer also contains the logic necessary to map persistent Domain Objects to the IDC operations database. This will be done using an Object-Relational Mapping (ORM) Framework [HIBERNATE_FRAMEWORK]. ORMs abstract relational database access away from an object-oriented domain model and GUI. No database dependencies or structures (e.g. SQL, Datasets etc.) need to be exposed to the higher layers when using an ORM. An ORM usually reduces the volume of data access code by over 50% and significantly increases the clarity, testability and maintainability of the code.

5.5.4.1 **Hibernate**

ROSS will use the Hibernate Object-Relational Mapping (ORM) Framework [HIBERNATE_FRAMEWORK] to access the IDC OPS relational database. Hibernate allows persistent classes to be developed following the object-oriented idiom, including association, inheritance, polymorphism, composition and collections. Hibernate is a Professional Open Source project run by JBoss, a division of Red Hat.

5.5.4.2 **NHibernate Licensing**

Hibernate is licensed under the open source GNU Lesser General Public License (LGPL). The LGPL license allows the use of the unmodified Hibernate libraries in commercial projects without any restrictions. If modifications are made to the
Hibernate source code then this source must also be shared with users of the project (not the wider community) under the same terms of the LGPL. As a result the LGPL is a risk-free open source license appropriate for CTBTO projects.

5.6 Utility Layer Architecture
The utility layer is a simple layer that contains shared classes used to carry out non-domain specific logic. Examples include:

- Writing to log files.
- Executing common functions like string manipulation, time manipulation, user role lookups etc.
- Sending emails

5.7 Data Layer Architecture

5.7.1 Overview of Data Layer
The data layer incorporates all persistent data used by ROSS. The low-level mechanisms for accessing the relational database and the file store are also incorporated within the scope of this layer.

5.7.2 File storage
ROSS will obtain the raw data messages (SPHD, QCPHD, DETBKPHD, CALIBPHD, BLANKPHAD, SOH, MET, and ALERT) from the IDC NFS file systems. These raw message files are stored in two locations on the IDC NFS file system: /ops/data/rn and /ops/products/rn. The file store is a RedHat Linux file server accessible through SSH.

5.7.3 Relational Database
The majority of the data (messages & processing results) used by ROSS will only be read from the IDC-OPS RDBMS and the IMS RDBMS, other systems (e.g. RMS Input) at the CTBTO are responsible for inserting, deleting, updating of this persistence data. However ROSS will be responsible for the Creation, Retrieval, Updating and Deleting (CRUD) of ROSS configuration entities (e.g. message timeliness configuration) that will also be persisted to the IDC-OPS-RDBMS. New database tables will be added to the IDC-OPS-RDBMS for these ROSS configuration entities. The numbers of tables that will be added is expected to be low.

5.7.4 Connections and Communication Channels
Communication with the IDC-OPS-RDBMS shall be via Oracle OCI library. This will open an Oracle database connection over TCP/IP between the client application (ROSS) and the database server.

Communication with the IDC-OPS-RDBMS can be initiated from both the ROSS Application Service Layer and the Domain Layer. Database communication in most
cases will be brokered by the Hibernate ORM (see the Domain Layer section above); although in some cases JDBC may be used where performance is required.

Communication with the CTBTO-File-Store will be via the standard Java IO libraries. The CTBTO-File-Store is accessible from the client through SSH.

5.8 Packages and Package Dependencies

The main Java packages identified for ROSS are:

<table>
<thead>
<tr>
<th>Java Package Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.ctbto.ross</td>
<td>Root Java package name for all ROSS classes.</td>
</tr>
<tr>
<td>org.ctbto.ross.common</td>
<td>Root Java package for ROSS common classes.</td>
</tr>
<tr>
<td>org.ctbto.ross.common.utility</td>
<td>Root Java package for ROSS common utility classes</td>
</tr>
<tr>
<td>org.ctbto.ross.domain</td>
<td>Root Java package for ROSS domain classes.</td>
</tr>
<tr>
<td>org.ctbto.ross.appservices</td>
<td>Root Java package for ROSS application service classes</td>
</tr>
<tr>
<td>org.ctbto.ross.ui</td>
<td>Root Java package for ROSS GUI/Presentation Layer</td>
</tr>
<tr>
<td>org.ctbto.ross.ui.common</td>
<td>Root Java package for ROSS common GUI modules</td>
</tr>
<tr>
<td>org.ctbto.ross.ui.workflow</td>
<td>Root Java package for ROSS Workflow module</td>
</tr>
<tr>
<td>org.ctbto.ross.ui.station</td>
<td>Root Java package for ROSS Station module</td>
</tr>
<tr>
<td>org.ctbto.ross.ui.reports</td>
<td>Root Java package for ROSS Reports module</td>
</tr>
<tr>
<td>org.ctbto.ross.ui.commands</td>
<td>Root Java package for ROSS Command module</td>
</tr>
<tr>
<td>org.ctbto.ross.ui.dataviewer</td>
<td>Root Java package for ROSS Raw Data Viewer module</td>
</tr>
<tr>
<td>org.ctbto.ross.ui.admin</td>
<td>Root Java package for ROSS Administration Layer</td>
</tr>
<tr>
<td>com.awst</td>
<td>Root Java package for generic libraries provided by AWST</td>
</tr>
<tr>
<td>com.awst.common</td>
<td>Root Java package for common classes</td>
</tr>
<tr>
<td>com.awst.common.utility</td>
<td>Root Java package for common utility classes</td>
</tr>
</tbody>
</table>

Table 3: Main ROSS Java Packages

5.9 ROSS Domain Model

A domain model is an attempt to model real world entities (existing or conceptual) from within the business domain of a software system. The basic entities of a domain model are conceptual classes of real-world objects. Attributes representing state are assigned to the conceptual classes and relations are defined between them. At a later stage in the project many of these conceptual classes will be expressed as software classes rather than simply conceptual classes.

TODO Define ROSS Domain Model
6 Process View

Under the 4+1 View Model the “Process View” shows the concurrent execution aspects of a system (the processes and threads) and the collaborations needed to support them. Performance and throughput are partially addressed in this view. Note that “process” in this context refers to execution processes, not business or user processes.

In the case of ROSS there are three application processes to be considered in this context:

1. The ROSS Client Application.
2. The IDC OPS Database Server.
3. The ROSS Publication Web Server.

6.1 ROSS Client Application

6.1.1 System Startup and Shutdown

The ROSS client application will be started by a user navigating to the .jnlp URL of the required ROSS distribution on the ROSS Publication Web Server. This will launch the ROSS client in Java Web Start, which will download all necessary binaries, configuration files and other dependencies to the user workstation.

Shutdown of the ROSS client will be at user request. The host process will be terminated upon shutdown.

6.1.2 Processes and Threads

The ROSS client will run within a single OS process. Internal to the client process multiple threads may be used in addition to the main event-dispatching thread used to update the display of user controls.

ROSS may defer some data retrieval operations to separate threads to enable a user to continue using ROSS while long running data retrieval operations run in the background. This approach has already been used in the Noble Gas application when loading the workflow data.

6.1.3 Performance

Performance of the ROSS client is most likely to be constrained by the speed of the database connection and the commands executed on this connection. Should it be necessary, improved performance could be achieved here by reducing the number of database round trips and also by reducing the volume and frequency of data transfers. Client-side caching of infrequently-updated data is another possibility.

The process performance of the Java application itself is affected by available computational resources (see the Deployment View section below) and the efficiency of the client code itself. In a long running Java client process, early disposal of expensive resources and careful management of object references in order to maximise the efficiency of garbage collection is important.

Perceived performance can also be affected by the responsiveness of the GUI application during long running operations, and the speed at which the ROSS application starts up. Startup will also be affected by the time taken to download the ROSS components from the Java Web Start URL hosting the deployment. By using
Java web start ROSS will download and cache the ROSS components on the client machine the first time ROSS is accessed. After that only ROSS components that have changed since the last time ROSS was used would be downloaded. Therefore any downloading performance issues are only really experienced when ROSS is accessed for the first time.

6.2 ROSS Relational Database Cluster

6.2.1 System Startup and Shutdown

The IDC runs two physical Oracle DBMS Servers in each cluster. This provides high availability, so that the logical database is, from the point of view of the ROSS application, always on. Start-up and Shutdown of physical databases are being taken care of by the IDC's database administrators. They are out of scope of the ROSS design.

6.2.2 Processes and Threads

Oracle runs as multiple processes depending on configuration and enabled services. The internal execution model of Oracle is out of scope of this document. Threads are dispatched to service client connections on demand.

In addition to interactive sessions with clients, Oracle will also automatically run daily batch jobs.

6.2.3 Performance

Performance of the ROSS Oracle database server is dependent on a large number of factors, including:

- Number of concurrent connections from ROSS users.
- Number of batch jobs running.
- SQL query performance.
- Index tuning and optimisation.
- Query plan optimisation.
- Frequency of database queries.
- Size of result sets transferred to clients.
- Locking of records, pages and tables during edits.

During operations, performance tuning and maintenance may be required.

6.3 Web Server

6.3.1 System Startup and Shutdown

The Apache Web Server is started and stopped automatically with the host operating system, or on-demand by a server administrator.

6.3.2 Processes and Threads

The internal execution model of Apache Web Server is not relevant to this document.
6.3.3 Performance

Performance of the Apache Web Server is not a concern. Apache is a well-established web server that is used in extremely high-volume websites. The ROSS website simply acts as a publication endpoint for static files. No dynamic content is hosted (e.g. JSP or PHP) and the main constraint, if any, when downloading the ROSS components over Java Web Start will be the available bandwidth between client and server. The low user levels of ROSS are unlikely to cause any performance issues in this regard. Furthermore, the total size of the ROSS application is expected to be no more than 15MB. This small component size combined with the high available bandwidth on the CTBTO intranet indicates that download speed will not be a performance constraint.
7 Component View

Under the 4+1 View Model the “Component View” or “Development View” depicts the software elements of the system as static components, such as modules, libraries, files and sub-systems. The main aim here is to show the mapping of the design elements into concrete “deliverables” or software configuration management elements.

ROSS is decomposed into several distinct layers and packages (shown in the Logical View above), which in turn are implemented, grouped and deployed as a number of physical components. These are described below, and diagrammed in the subsequent Deployment View section.

7.1 Client Tier Components

7.1.1 ROSS Client Application

The ROSS User Interface is an executable Java application composed of a number of Java packages deployed as binary .jar files. Each ROSS plug-in (see Chapter 5: Logical View) will be packaged as separate jar files, e.g. the ROSS Workflow View, ROSS Station, ROSS Domain etc.

7.1.2 ROSS Client Configuration Files

User preferences for ROSS will be configured in ROSS using the NetBeans preferences API. These preferences are saved to .properties files that are stored in the application’s user directory.

Global configuration settings (e.g. message timeliness configuration) that apply to all ROSS users will be configured through the IDC OPS database.

Any other configuration files for the ROSS client are deployed as .properties or XML files, in the CLASSPATH of the executing client .jars.

Database client configuration (server name, default user name etc.) is stored outside of the CLASSPATH. Oracle client configuration (TNS Names) is stored in the Oracle client directories.

7.1.3 Other Components Utilized by ROSS Client

ROSS is deployed with a number of 3rd party components. These are mainly supporting Java libraries such as log4j, NetBeans Platform, JasperReports, JFreeChart, JGraph and Hibernate, deployed as .jar files. ROSS also depends on the Java Runtime Environment (Java 6 SE) being available on the client.

The Oracle Client libraries must also be deployed on the client.

7.2 Database Server Tier Components

The Database Server Tier contains many ROSS components. Some of these can be considered as logical components of ROSS. Other components are physical components of the database server such as data and binary files and do not need to be considered here. The main logical components deployed on the server are:
- Database objects installed as part of the ROSS schema (tables, views, indexes, triggers, users, permissions etc.). This is the logical Database Layer described in Chapter 5.7: Data Layer Architecture

7.3 Web Server Tier Components

The ROSS client components (.jar files and configuration files) will be deployed on the web server for download via Java Web Start to client workstations. A .jnlp URL will act as the publication endpoint for clients wishing to download and install the ROSS files. Informational static HTML pages may also be provided with links to different release versions of the ROSS client.
8 Deployment View

In the 4+1 View Model, the Deployment View describes where the software elements of the system execute in a physical network of hardware elements, and how these software elements communicate when in a distributed environment. The focus here is on distribution, communication and provisioning.

8.1 Processing Nodes

ROSS is deployed onto the following processing nodes. The components and processes in these nodes are described in the appropriate sections above.

- **ROSS Client Workstations**: a number of client PCs. The operating system on these PCs is 90% Microsoft Windows XP and 10% Linux Redhat v5.3/6.0.
  
The prerequisite software that should be installed on the client workstations is:
  
  1. Java Runtime version 6.0 or above
  2. Adobe PDF Reader 9.0 or above.
  3. Microsoft Excel 2003 or above (Windows Only)

- **ROSS Publication Web Server**: a single Linux server running the Apache Web Server, hosting the **ROSS Client Application** located at a Java Web Start URL. The same platform that is currently being used for the NG software will be reused for ROSS.

- **IDC Ops Database Cluster**: a database cluster consisting of two Oracle RDBMS including any new tables that are needed for ROSS. The operating system is Redhat Linux and the target RDMS software is Oracle 11g. Test and development databases are also available.

In addition to the above, ROSS will interact with the following external processing nodes:

- **External RN Web Applications**
  
  A number of independently hosted web applications that can be hosted on heterogeneous hardware anywhere within the CTBTO network.

- **IDC Email Server**:
  
  The email server used for sending and receiving command emails. The hardware and software platform of this Server is TBD.

- **CTBTO-File-Store Tier**
  
  A file server that stores the raw sample messages in plain text format. Access to the CTBTO-File-Store tier will be via SSH. The operating system is Linux RedHat.

- **LDAP/Active Directory Server**
  
  If a single sign on solution is to be implemented in ROSS, ROSS will have to connect to an LDAP or Active Directory Server to verify that a user is authorized to use ROSS. The hardware and software platform for these Servers is TBD.
8.2 Cross-Process Connectors

The following logical connectors allow communication between components running in distributed processes:

- **Oracle**: database connection over TCP/IP between the ROSS client and the Oracle database server.
- **HTTP**: to request and download the ROSS client from the Apache web server.
- **HTTP**: to link to external RN web applications.
- **SMTP**: for sending command emails over TCP/IP.
- **IMAP**: for retrieving command emails from a remote server over TCP/IP.
- **SSH**: for retrieving the raw sample messages from the CTBTO-FileStore.

8.3 UML Deployment Diagram

The nodes and connectors described above are shown below in the form of a UML Deployment Diagram. Each node contains one or more ROSS system components. These components are described in more detail in Chapter 7 Component View. This diagram depicts the hosting of the ROSS components, and the means of communication between these nodes and the external nodes.

![UML Deployment Diagram](image)

**Figure 3: ROSS Deployment Diagram**
9 Technology and Vendor Selections:

Most key technology selections are documented in the Design Considerations, Component View and Deployment View chapters of this document. Additional technology selections not mentioned above are discussed in this chapter.

9.1 Development Languages

The development languages to be used in Java:

- Java: used for development of the Java client.
- XML: used for configuration of the Java client.
- HTML: used for the web page on the Web Server containing the Java Web Start hyperlinks that will launch the ROSS client.
- SQL: used for database operations.
- JNLP: used for defining how to launch Java Web Start applications.
- JRXML: used for defining JasperReports.

9.2 Development Tools

- NetBeans Platform 7.0 is used to develop all Java software for ROSS.
- iReports is used to develop the JasperReports.

9.3 Code Quality Tools

- Source Monitor is used to provide code metrics and to identify the code that is most likely to contain defects.
- PMD & CheckStyle is used to check that ROSS conforms to recommended Java patterns and practices.
- FindBugs is used to find bugs in the Java code using static analysis (e.g. code that could never be executed because of logical irregularities)

9.4 Test, Build and Source Control Tools

- NB JUnit and Jemmy: unit testing.
- Maven: build management.
- TeamCity: build server.
- Subversion: source control system.
9.5 Software Licensing

9.5.1 Open Source Licenses

Various open source software components will be used in the development and deployment of ROSS. In all cases the licenses for these components are non-copyleft and non-viral, and impose no obligations or constraints on the ROSS software. Relevant open source components included in the ROSS application are:

- NetBeans Platform is available under the terms of a dual license consisting of the Common Development and Distribution License (CDDL) v1.0 and the GNU General Public License (GPL) v2 [NETBEANS-LIC]
- The log4j Logging Framework is licensed under the terms of the Apache License, Version 2.0 [APACHE-LIC].
- The Hibernate framework is licensed under the open source GNU Lesser General Public License (LGPL), Version 2.1, June 1991 [LPGL-LIC]
- JFreeChart is licensed under the terms of the GNU Lesser General Public Licence (LGPL), Version 2.1, June 1991 [LPGL-LIC]
- JasperReports library is available under the terms of a dual license: consisting of the JasperReports License, Version 1.0 (based upon "Apache Software License, Version 1.1") [JASPER-LIC] and GNU Lesser Public License (LGPL) [LPGL-LIC]

The Apache 2.0 licence only requires that a notice is kept informing recipients that Apache licensed code has been used. Such a notice should be included in all distributions of the ROSS software. The Apache license also applies to the Web Server software used to publish the ROSS client.

The LGPL/MPL licensing of Hibernate, JFreeChart and JasperReports allows the inclusion of this component in closed source or commercial applications.

9.5.2 Proprietary Licenses

ROSS depends on several proprietary software components. These are:

- Oracle 11g: various license options available. At the time of writing The IDC database server is deployed with an Oracle Enterprise Edition license.
- Windows client and server licenses.
- RedHat client and server licenses.
10 Architectural Factors and Decisions

This section provides, in the form of technical memos, the requirements, constraints and other considerations that have had a bearing on the architectural decisions described in this document.

The template for the technical memos is as follows:

<table>
<thead>
<tr>
<th>Technical Memo</th>
<th>Issue: &lt;Name&gt;</th>
<th>Solution Summary</th>
<th>ID</th>
<th>Factors:</th>
<th>Solution:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>&lt;Summary of solution&gt;</td>
<td>TM-000</td>
<td>&lt;Relevant factors influencing the decision&gt;</td>
<td>&lt;Details of chosen solution&gt;</td>
</tr>
<tr>
<td>Motivation:</td>
<td></td>
<td>&lt;List of motivations and constraints related to the solution&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unresolved Issues:</td>
<td></td>
<td>&lt;Any outstanding issues at time of writing&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alternatives Considered:</td>
<td></td>
<td>&lt;Other technical solutions that were considered and rejected&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Discussion:</td>
<td></td>
<td>&lt;Any other points&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Technical Memo**

**Issue: Choice of Operating System and Environment**

<table>
<thead>
<tr>
<th>Solution Summary:</th>
<th>ROSS will be deployed on the standard IDC infrastructure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>TM-001</td>
</tr>
<tr>
<td>Factors:</td>
<td>Existing operating systems and environment at the IDC</td>
</tr>
<tr>
<td>Solution:</td>
<td>• Java provides platform-independence</td>
</tr>
<tr>
<td></td>
<td>• Client OS is 90% Windows XP and 10% Red Hat Linux v5.3/6.0.</td>
</tr>
<tr>
<td></td>
<td>• Oracle 11g database</td>
</tr>
<tr>
<td></td>
<td>• Apache Web Server</td>
</tr>
<tr>
<td>Motivation:</td>
<td>Complies with IDC architecture.</td>
</tr>
<tr>
<td>Unresolved Issues:</td>
<td>The IDC have plans to upgrade Oracle to 11g. Will the planned upgrade of Oracle take place before deployment of ROSS into OPS?</td>
</tr>
<tr>
<td>Alternatives Considered:</td>
<td>None</td>
</tr>
<tr>
<td>Discussion:</td>
<td></td>
</tr>
</tbody>
</table>

**Technical Memo**

**Issue: Choice of Programming Language and Environment**
### Software Architecture Document for ROSS

#### Solution Summary: ROSS will be written in Java.

<table>
<thead>
<tr>
<th>ID</th>
<th>TM-002</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Factors:</strong></td>
<td>Problem domain, current usage at IDC, existing code, open-source considerations, platform-independence</td>
</tr>
<tr>
<td><strong>Solution:</strong></td>
<td>The programming environment will be NetBeans IDE 7.0. The programming language will be Java 6.</td>
</tr>
</tbody>
</table>
| **Motivation:** | • Java is platform independent  
• Java will allow maximum reuse of existing code at the CTBTO.  
• Java is mature, well supported, robust, feature-rich, and object oriented  
• Java is commonly used at the CTBTO.  
• AWST development team are experienced in developing Java applications  
• NetBeans IDE is free and is fully integrated with the NetBeans Platform |
| **Unresolved Issues:** | None |
| **Alternatives Considered:** | Eclipse & Oracle JDeveloper Dismissed because the integration with the NetBeans Platform would be more difficult in either of these IDEs. |
| **Discussion:** | |

---

#### Technical Memo

**Issue:** Method for database access

**Solution Summary:** ROSS will use the Hibernate ORM.

<table>
<thead>
<tr>
<th>ID</th>
<th>TM-003</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Factors:</strong></td>
<td>Programming environment, deployment, data access, maintenance</td>
</tr>
<tr>
<td><strong>Solution:</strong></td>
<td>The Hibernate Object-Relational Mapping Framework will be used as the mapping layer between the CTBTO-RDMS and the ROSS Domain Layer.</td>
</tr>
</tbody>
</table>
| **Motivation:** | • Domain-driven OO design of ROSS suggests use of ORM.  
• ORM significantly reduces amount of data-access code.  
• Hibernate is the de-facto Java ORM, and is both free and open-source. |
| **Unresolved Issues:** | None |
| **Alternatives Considered:** | Oracle TopLink: AWST Development team has no experience using this technology. |
| **Discussion:** | |
## Technical Memo
### Issue: Choice of Rich Client Platform

<table>
<thead>
<tr>
<th>Solution Summary:</th>
<th>ROSS will use the NetBeans Platform.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ID</strong></td>
<td>TM-004</td>
</tr>
<tr>
<td><strong>Factors:</strong></td>
<td>Deployment, GUI widget set, extensibility and maintainability</td>
</tr>
<tr>
<td><strong>Solution:</strong></td>
<td>The ROSS modules will be developed to plug into the NetBeans Platform</td>
</tr>
</tbody>
</table>
| **Motivation:**   | - The NetBeans platform provides various required and/or desirable features straight out of the box.  
                     - The platform’s plug-and-play mechanism allows applications to be developed independently and “plugged” together to create an integrated solution.  
                     - The platform is Swing compatible and will allow the existing Java GUI code to be migrated without requiring any major code changes.  
                     - The NetBeans community actively maintains the platform and provides developer support in the form of Wikis, books, tutorials, forums, examples, and screen casts. |
| **Unresolved Issues:** | - The NetBeans community are about to deliver (due for release in April 2011) a new major version of its NetBeans platform. ROSS will target the new major version, 7.0 but we may need to revert if there are stability issues with new version. |
| **Alternatives Considered:** | - The Eclipse RCP. The Eclipse RCP platform was rejected because it does not support the standard Swing widget-set  
                     - Spring RCP: The Spring RCP was rejected because it’s not as feature-rich as the other platforms and its future is uncertain |
| **Discussion:**   |                                      |

## Technical Memo
### Issue: Choice of Reporting Tool

<table>
<thead>
<tr>
<th>Solution Summary:</th>
<th>ROSS reports will be generated using Jasper Reports</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ID</strong></td>
<td>TM-005</td>
</tr>
<tr>
<td><strong>Factors:</strong></td>
<td>Programming environment, reporting requirements, open-source considerations, developer experience</td>
</tr>
<tr>
<td><strong>Solution:</strong></td>
<td>The ROSS reports will be developed using the Jasper Reports technology</td>
</tr>
</tbody>
</table>

Software Architecture Document for ROSS
**Motivation:**
- JasperReports is a comprehensive Java reporting solution. It's the world's most complete, most popular Java reporting solution.
- JasperReports produces pixel-perfect reports that can be viewed, printed or exported in a variety of document formats including Microsoft Excel, PDF, RTF, Open Office, CSV, and HTML.
- JasperReports supports data coming from a variety of data sources including JDBC, XML, CSV, Hibernate, EJBs, Oracle PL/SQL stored procedures.
- JasperReports can contain charts and graphs such as Pie, Bar, Stacked Bar, Line, Area, Scatter Plot, Bubble, and Time series.
- Reports developed using JasperReports can be separated from the Java code. This allows reports to be modified without requiring a recompile of the Java source code.
- JasperReports can contain drill-down reports.
- JasperReports provides a component for viewing JasperReports inside a Java application.
- There is an open source WYSIWYG report designer specifically designed for JasperReports.

**Unresolved Issues:** None

**Alternatives Considered:**
- Building reports from scratch would require far more effort.
- Crystal reports require a license and are less flexible.

**Discussion:**

---

**Technical Memo**

**Issue: Application Type**

**Solution Summary:** ROSS will be developed as a thick client/server application.

**ID** TM-006

**Factors:** GUI centric requirements, existing code, distribution model

**Solution:** ROSS will be developed as a client server application with a rich Java client executing on the user's workstation.

**Motivation:**
- A thick client application tier was selected for ROSS because:
  - Distribution outside the LAN is not important.
  - Responsiveness and performance of the client tier is important.
  - Will maximum reuse of existing code at the CTBTO.

**Unresolved Issues:** None.

**Alternatives Considered:**
- A web-based application (e.g. JSP, PHP) was considered technically feasible but rejected due to a Client-Server application providing the maximum opportunity to reuse code.

**Discussion:** None.
### Technical Memo

#### Issue: Choice of graphing technology

<table>
<thead>
<tr>
<th>Solution Summary:</th>
<th>ROSS will use JFreeChart for plotting graphs.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ID</strong></td>
<td>TM-007</td>
</tr>
<tr>
<td><strong>Factors:</strong></td>
<td>GUI requirements, open-source considerations, existing code</td>
</tr>
<tr>
<td><strong>Solution:</strong></td>
<td>JFreeChart will be used to plot graphs in ROSS</td>
</tr>
</tbody>
</table>
| **Motivation:**   | - JFreeChart is already used in the NG software application. Using JFreeChart will allow reuse of the existing NG graphs.  
- JFreeChart is the leading Java graphing technology.  
- JFreeChart is both free and open source.  
- JFreeChart is well supported with good documentation and examples.  
- JFreeChart is the graphing technology used in JasperReports |

#### Unresolved Issues:

**Alternatives Considered:**
- TeeChart for Java, The TeeChart technology was used in the legacy SOH Monitor application but for ROSS this was dismissed because of the license costs (single developer license costs $469) involved.

### Technical Memo

#### Issue: Target Database Instance

<table>
<thead>
<tr>
<th>Solution Summary:</th>
<th>ROSS will target the IDC OPS database</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ID</strong></td>
<td>TM-008</td>
</tr>
<tr>
<td><strong>Factors:</strong></td>
<td>Centralisation of data storage, existing databases</td>
</tr>
<tr>
<td><strong>Solution:</strong></td>
<td>ROSS will target the IDC OPS database.</td>
</tr>
<tr>
<td><strong>Motivation:</strong></td>
<td>Some of the legacy systems are not working with the IDC OPS database but with the IMS database (e.g. the SOH Application developed by Robert Werzi). These databases should have the same contents, but don't always. This sometimes leads to confusion with the recipients of the resulting reports because of results differences. Therefore it is desired to target the same database to remove this confusion.</td>
</tr>
</tbody>
</table>

#### Unresolved Issues:

If a station is not available in the IDC OPS database should ROSS look for data in the IDC test database? The first prototype versions of ROSS will target a single database platform. This issue will be addressed when we address the database config requirements.
Alternatives Considered: None

Discussion: ROSS should be developed against the IDC DevLan/Testbed database schema and not the IDC OPS database schema because this database contains the latest database schema that will be installed into OPS by the time ROSS is ready to go into production.

It should be possible to configure ROSS to work against the IDC TestBed and DevLan databases.

---

**Technical Memo**

**Issue: Connection to RAC Database**

**Solution Summary:** ROSS will connect to the Oracle RAC

**ID**

TM-009

**Factors:**

Existing RDBMS practices at IDC

**Solution:** ROSS will connect to the Oracle RAC rather than to an individual database in the cluster

**Motivation:** The IDC are using database clustering technology to make the IDC OPS and IDC Testbed databases highly available. For the clustering technology to work properly an application should not connect directly to a single database instance but connect to the RAC.

**Unresolved Issues:** Not all Java Oracle database drivers support this type of connection. A suitable OCI database driver needs to be identified for use in ROSS. The next step to resolving this issue is to speak with the IDC database admins and find out which database driver is being used in other Java applications.

**Alternatives Considered:** None

**Discussion:** Java thin database drivers are unsuitable for this type of connection

---

**Technical Memo**

**Issue: Storage of Configuration Data**

**Solution Summary:** Splitting of personal and global configuration information

**ID**

TM-010

**Factors:**

Maintainability, personalization

**Solution:** ROSS will store personal user preferences in the application’s user directory in a .property file and will store global configuration data in the IDC-OPS database.

**Motivation:**

- NetBeans Platform provides a user preference API straight out of the box.
- Storing global configuration in the IDC-OPS database ensures all ROSS users use the same configuration.
### Unresolved Issues:

#### Alternatives Considered:

Storing all configuration data in the IDC-OPS database was considered but dismissed for the following reasons:

1. Would require database admin action when a user leaves the CTBTO.
2. If configuration becomes corrupted and the configuration is stored in the database, assistance from the IDC support staff would be required to resolve the corruption issues. If it’s stored in the application’s user directory a ROSS user could restore the default ROSS configuration themselves by deleting the .property file in the application’s user directory.

An advantage of storing personal configuration data in the IDC database was to allow ROSS users to switch machines and not require them to reconfigure their preferences. Switching machines is a rare occasion and the NetBeans Platform allows you to import/export the user preferences file. On Linux workstations, the same home directory is available from any machine. On windows, a shared drive can be used to store the user profile.

---

### Technical Memo

**Issue: Code Reuse**

**Solution Summary:** ROSS will reuse some of the source code from the NG Software

**ID:** TM-011

**Factors:** Existing code, coding standards/best practices

**Solution:**

ROSS will reuse the source code from the NG Software for the following features:

1. Date Filter
2. Station Hierarchy
3. NG Graphs Selector & NG Graphs
4. NG Flowcharts
5. NG Table Data
6. Raw Data Viewer
7. Statistics View
8. Workflow (TBD, See TM-012)
9. NG Processing, Alerts Messages & Data Gap List

**Motivation:** Reusing the existing source code will save effort and allow time to be spent elsewhere providing other functionality.

**Unresolved Issues:**

**Alternatives Considered:** Reuse of other parts of legacy code would not be cost-effective.
Discussion: The NG source code will require some major code refactoring when it is migrated to ROSS and the NetBeans platform. Appendix 2 lists our code review findings and recommendations. These review findings will be addressed as the code is migrated to ROSS.

<table>
<thead>
<tr>
<th>Technical Memo</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Issue: Reusing the NG Workflow Tool</strong></td>
</tr>
</tbody>
</table>

**Solution Summary:** TBD

**ID**
TM-012

**Factors:**
Existing code, coding standards/best practices

**Solution:** TBD

**Motivation:** Building the ROSS workflow on top of JFreeChart would give us a lot of infrastructure for free e.g. scaling of the Workflow, painting of the messages and labels on the x and y axes.

**Unresolved Issues:**
The workflow tool in the NG software is built on top of JFreeChart. We wish to reuse this component to provide the workflow tool for ROSS with some modifications. However before we can continue with this approach we need to identify if any APIs are provided by JFreeChart that allow us to:

- Convert the mouse pointer coordinates into model units. This is needed to provide the clicking and tooltip features.
- Override the default tooltips. This is needed to provide message tooltips.
- Number the blocks on a graph. This is needed to number the pre-PHD messages.
- Use several colours in a single row. This is needed to show the message statuses.
- Show the date and time on the x axis. This is needed to clearly indicate the date range being shown in the Workflow.

**Alternatives Considered:**
Develop a new component from scratch using Swing components and the Java Graphic libraries. This approach would require a lot more development effort than building on top of the JFreeChart because of the need to build the basic infrastructure.

**Discussion:**

<table>
<thead>
<tr>
<th>Technical Memo</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Issue: Noble Gas MySQL Database</strong></td>
</tr>
</tbody>
</table>

**Solution Summary:** The NG MySQL Database will be migrated to the Oracle IDC database

**ID**
TM-013

**Factors:**
Maintainability, Existing Databases

**Solution:** The NG MySQL Database will be migrated to the Oracle IDC database

Software Architecture Document for ROSS
| **Motivation:** | Remove the need to administer a MySQL database.  
Bring ROSS into line with current IDC database practices.  
Improve ROSS performance  
Remove an cross database query issues between the NG and IDC databases. |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unresolved Issues:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Alternatives Considered:</strong></td>
<td>Retain the MySQL database. This solution was dismissed because we will now have the necessary permissions to install the tables into the IDC database. The NG developers did not have the necessary permissions and created this database to work around the permission issues.</td>
</tr>
<tr>
<td><strong>Discussion:</strong></td>
<td></td>
</tr>
</tbody>
</table>
## Appendix 1: Acronyms

<table>
<thead>
<tr>
<th>Acronym, Technical Term</th>
<th>Longhand Form, Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CGI</td>
<td>Common Gateway Interface</td>
</tr>
<tr>
<td>COTS</td>
<td>Commercial Off-the-Shelf</td>
</tr>
<tr>
<td>CRUD</td>
<td>Create, Retrieve, Update, Delete</td>
</tr>
<tr>
<td>CTBTO</td>
<td>Comprehensive Nuclear-Test Ban Treaty Organization</td>
</tr>
<tr>
<td>DAO</td>
<td>Data Access Object</td>
</tr>
<tr>
<td>DBA</td>
<td>Database Administrator</td>
</tr>
<tr>
<td>DTO</td>
<td>Data Transfer Object see [JAVA-DTO]</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>LGPL</td>
<td>Lesser General Public Licence</td>
</tr>
<tr>
<td>IDC</td>
<td>International Data Centre</td>
</tr>
<tr>
<td>IMS</td>
<td>International Monitoring System</td>
</tr>
<tr>
<td>OPS</td>
<td>Operations</td>
</tr>
<tr>
<td>ORM</td>
<td>Object Relational Mapping/ Mapper</td>
</tr>
<tr>
<td>POC</td>
<td>Point of Contact</td>
</tr>
<tr>
<td>RAC</td>
<td>Real Application Cluster, Oracle’s term for clustered databases</td>
</tr>
<tr>
<td>RISM</td>
<td>Responsible IDC Staff Members</td>
</tr>
<tr>
<td>RN</td>
<td>Radionuclide</td>
</tr>
<tr>
<td>SOH</td>
<td>State-of-health</td>
</tr>
<tr>
<td>TER</td>
<td>Test and Evaluation Report</td>
</tr>
</tbody>
</table>
Appendix 2: Noble Gas Application Code Review Findings

Introduction
On 17th March 2008 Mark Harrison of AWST carried out a brief, high-level review of the Noble Gas Application’s Java source code as part of the design and planning of the development of ROSS. This code review was used as input into the ROSS SAD and planning of ROSS-2. The code under review was an export of the Noble Gas repository, provided by Dennis Giordanino on 26th February.

Summary of Static Analysis of ROSS Java Code
A simple static analysis of the ROSS Java code was carried out using the SourceMonitor tool (http://www.campwoodsw.com/sourcemonitor.html). Key statistics are:

- Code Lines: 11,466
- Classes: 91
- Methods Per Class: 3.02
- Avg Depth: 2.95
- Avg Cyclomatic Complexity: 3.54

General Observations on ROSS Java Code
1. There is no separation or layering between the GUI logic and business logic.
2. 40% of the code is contained in a single file (NGXAJframe)
3. 8 out of 14 files contain one or more functions that exceed the recommended Max Cyclomatic Complexity. Three files contains a function that is off the scale.
4. Functions are generally too complex.
5. Java best practices have not been followed when naming the project’s namespace.
6. There are several common string literals that are used throughout the code e.g. Station types, these should be converted into constants or Enums.
7. Arguments are commonly copied directly from the GUI into database queries. This exposes the database to SQL injection attacks. Database parameter lists should be used to avoid exposing the database to SQL injection attacks.
8. Code is poorly laid out and hard to read.
9. There are a number of constructors and functions that accept more than three variables as arguments e.g.

   public NGXAJframe(String username, String nameUser, String nameGrade, String section, int idGrade, int idSection, String supervisor, String abbrUser){
private JFreeChart jCharDraw(int connectionID, String stationID, JTabbedPane jTable, String nameSensor, Boolean tableview, String typeSystem, String nameSensorSampling, String titleStation, int connectionIDSampling)

It is recommended best practice to pass in a single class that contains the arguments when more than three arguments are to be passed into a function.

10. There is limited commenting throughout the code.
11. There is lots of code commented out. There is no indication if this is dead code or functionality that has not been completed.
12. There is different logging strategies used throughout the code base e.g.,

```
System.out.println("Error connection...");
```

```
Logger.getLogger(NGXAJframe.class.getName()).log(Level.SEVERE, null, ex);
```

It is recommended that a single strategy is used throughout for logging.
13. Exceptions are often swallowed by the application with no indication provided to the user that something may have gone wrong. E.g.

```
} catch (SQLException ex) {
    Logger.getLogger(NGXAJframe.class.getName()).log(Level.SEVERE, null, ex);
} catch (IOException ex) {
    Logger.getLogger(NGXAJframe.class.getName()).log(Level.SEVERE, null, ex);
}
```

14. The following code should be extracted into a Factory class that creates database connections instead of being repeated throughout the code base.

```
if (connectionID==0){
    con = openConnection("oracle",defaultProps.getProperty("dataBase_1"),
    defaultProps.getProperty("user_1"), defaultProps.getProperty("password_1"));
} else if (connectionID==1){
    con = openConnection("oracle",defaultProps.getProperty("dataBase_2"),
    defaultProps.getProperty("user_2"), defaultProps.getProperty("password_2"));
    pathRawData=defaultProps.getProperty("pathProcessedMessagesDataBase_2");
} else if (connectionID==2){
    con = openConnection("oracle",defaultProps.getProperty("dataBase_3"),
    defaultProps.getProperty("user_3"), defaultProps.getProperty("password_3"));
    pathRawData=defaultProps.getProperty("pathProcessedMessagesDataBase_3");
} else if (connectionID==3){
15. GUI elements are often poorly named e.g.: jTextField9.

16. Using the StringBuilder instead of the String class is recommended best practice when constructing large strings e.g.

```java
.executeQuery("SELECT queryGraph,queryTable FROM NOBLEGASSAMPLERESULTS WHERE ((typeStation="" + typeSystem + "") AND (aliasParameter="" + sensorName + ")");
```

17. The code contains hard coded configuration data that should be externally configurable e.g.

```java
conn = DriverManager.getConnection("jdbc:mysql://marie.ims.ctbto.org:3306/ngmonitoring?" + "user=dennis&password=dennis");
```

There is often different behavior for different station types as illustrated below. A better approach than using if … else would be to use the strategy design pattern.

```java
if (typeSystem.compareTo("ARIX")==0) {
    auxToken = new StringTokenizer(arix[1], ",");
    auxTokenAlias = new StringTokenizer(arix[0], ",");
    auxTokenAliasComponent = new StringTokenizer(arix[5], ",");
} else if (typeSystem.compareTo("SAUNA")==0) {
    auxToken = new StringTokenizer(sauna[1], ",");
    auxTokenAlias = new StringTokenizer(sauna[0], ",");
    auxTokenAliasComponent = new StringTokenizer(sauna[5], ",");
} else if (typeSystem.compareTo("SPALAX")==0) {
    auxToken = new StringTokenizer(spalax[1], ",");
    auxTokenAlias = new StringTokenizer(spalax[0], ",");
    auxTokenAliasComponent = new StringTokenizer(spalax[5], ",");
}
```
else if (typeSystem.compareTo("Manual") == 0) {
    auxToken = new StringTokenizer(data[0][0], "@");
    auxTokenAlias = new StringTokenizer(data[0][0], "@");
}
Appendix 3: ROSS storyboard

The Storyboard is accessible on the ROSS project Wiki: https://www.awst.at/intranet/display/ROSS/Storyboard
## Appendix 4: ROSS Requirements Traceability Matrix

### Key

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Req. #</td>
<td>The unique identifier assigned to a feature/requirement</td>
</tr>
<tr>
<td>Requirement /Feature Description</td>
<td>A description of the feature/requirement to be implemented.</td>
</tr>
<tr>
<td>Priority</td>
<td>The priority of the requirement using MoSCoW prioritisation (<a href="https://www.awst.at/intranet/display/ROSS/Requirements">https://www.awst.at/intranet/display/ROSS/Requirements</a>). M = Must-Have&lt;br&gt;S = Should-Have&lt;br&gt;C = Could-Have&lt;br&gt;W = Wont-Have</td>
</tr>
<tr>
<td>Module</td>
<td>The module where this feature will be implemented.</td>
</tr>
<tr>
<td>Screen Ref#</td>
<td>The Storyboard reference e.g. 1.2. The first number refers to the screen number and the second the annotation number.</td>
</tr>
<tr>
<td>Implementation Plan</td>
<td>Our plan for implementing this feature. The following options are available: &lt;br&gt;Reuse (System Name): Reuse existing source code from a legacy system &lt;br&gt;Re-Implement (System Name): Re-implement a feature that was available in a legacy system &lt;br&gt;New Feature: A new feature that needs to be developed from scratch</td>
</tr>
<tr>
<td>Implementation version</td>
<td>The version when this feature will be implemented.</td>
</tr>
</tbody>
</table>
## Requirements

<table>
<thead>
<tr>
<th>Req. #</th>
<th>Requirement /Feature Description</th>
<th>Priority</th>
<th>Module</th>
<th>Screen Ref#</th>
<th>Implementation Plan</th>
<th>Implementation version</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROSS-35</td>
<td>10.1.1 The ROSS Workflow shall be presented in the same format and using the same colour codes that are used in the Robert Werzi’s Workflow</td>
<td>M</td>
<td>Workflow</td>
<td>2</td>
<td>Re-implement (RWWF)</td>
<td>ROSS v0.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reuse (NG)</td>
<td>ROSS v0.2</td>
</tr>
<tr>
<td>ROSS-36</td>
<td>ROSS shall allow a user to configure which items are displayed in the Workflow.</td>
<td>S</td>
<td>Workflow</td>
<td>2.2</td>
<td>New Feature</td>
<td>ROSS v0.2</td>
</tr>
<tr>
<td>ROSS-11</td>
<td>There shall be a link from each station displayed in the workflow to the Particulate or Noble Gas station view</td>
<td>M</td>
<td>Workflow</td>
<td>2.8</td>
<td>New Feature</td>
<td>ROSS v0.2</td>
</tr>
<tr>
<td>ROSS-10</td>
<td>There shall be a link from the workflow tool to all the relevant data messages (i.e. RN samples, SOH messages, alerts, met data). Clicking on a point in the workflow would display the relevant message.</td>
<td>M</td>
<td>Workflow</td>
<td>4.1</td>
<td>New Feature</td>
<td>ROSS v0.2</td>
</tr>
<tr>
<td>ROSS-31</td>
<td>The ROSS Workflow shall show timeliness information for full samples. Colour codes shall be used to represent the timeliness, i.e. two states for late (&gt;3 hours) and very late (&gt;5 hours) data</td>
<td>S</td>
<td>Workflow</td>
<td>2.16</td>
<td>New Feature</td>
<td>ROSS v0.2</td>
</tr>
<tr>
<td>ROSS-37</td>
<td>10.1.2 ROSS shall allow the timeliness statuses to be configured</td>
<td>C</td>
<td>Workflow</td>
<td>N/A</td>
<td>New Feature</td>
<td>ROSS v0.2</td>
</tr>
<tr>
<td>ROSS-38</td>
<td>ROSS shall use the processing statuses and colour codes from the IDC Workflow Tool.</td>
<td>S</td>
<td>Workflow</td>
<td>N/A</td>
<td>Re-implement (IDCWF)</td>
<td>ROSS v0.2</td>
</tr>
<tr>
<td>ROSS-39</td>
<td>ROSS shall allow the user to toggle between two workflow colour/state schemes display: 1. timeliness status, and 2. processing status.</td>
<td>S</td>
<td>Workflow</td>
<td>2.17</td>
<td>New Feature</td>
<td>ROSS v0.2</td>
</tr>
<tr>
<td>ROSS-40</td>
<td>10.1.3 ROSS could show the (Input Failure &amp; Input Process) lines on the Workflow that are shown in the IDC workflow tool.</td>
<td>W</td>
<td>Workflow</td>
<td>N/A</td>
<td>Re-implement (IDCWF)</td>
<td>N/A</td>
</tr>
<tr>
<td>ID</td>
<td>Description</td>
<td>Priority</td>
<td>Component</td>
<td>Version</td>
<td>Notes</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------</td>
<td>--------------------</td>
<td>----------</td>
<td>----------------------------</td>
<td></td>
</tr>
<tr>
<td>ROSS-18</td>
<td>10.1.4 The user shall be able to scale the Workflow tool.</td>
<td>S</td>
<td>Workflow</td>
<td>2</td>
<td>Reuse (NG)</td>
<td>ROSS v0.2</td>
</tr>
<tr>
<td>ROSS-41</td>
<td>10.1.5 ROSS shall provide a configurable auto-refresh feature.</td>
<td>S</td>
<td>Workflow</td>
<td>2.18</td>
<td>Re-implement (RWWF)</td>
<td>ROSS v0.2</td>
</tr>
<tr>
<td>ROSS-42</td>
<td>10.1.6 ROSS shall provide an option to turn off the auto refresh feature.</td>
<td>S</td>
<td>Workflow</td>
<td>2.18</td>
<td>New Feature</td>
<td>ROSS v0.2</td>
</tr>
<tr>
<td>ROSS-43</td>
<td>10.1.7 ROSS shall display in separate lists: messages that failed to be processed, alerts that were generated and data gaps between messages over a configurable limit.</td>
<td>S</td>
<td>Workflow</td>
<td>2.11, 2.15</td>
<td>Reuse (NG)</td>
<td>ROSS v0.1</td>
</tr>
<tr>
<td>ROSS-44</td>
<td>10.1.8 ROSS shall display a tooltip when a message is hovered over</td>
<td>S</td>
<td>Workflow</td>
<td>2.3</td>
<td>New Feature</td>
<td>ROSS v0.2</td>
</tr>
<tr>
<td>ROSS-45</td>
<td>10.1.9 ROSS shall display properties of a message when a message is clicked on in the Workflow</td>
<td>S</td>
<td>Workflow</td>
<td>2.10</td>
<td>New Feature</td>
<td>ROSS v0.2</td>
</tr>
<tr>
<td>ROSS-46</td>
<td>10.1.10 ROSS shall remember which stations were being displayed in the Workflow view when ROSS is closed and reload them when ROSS reopens</td>
<td>C</td>
<td>Workflow</td>
<td>2</td>
<td>New Feature</td>
<td>ROSS v0.2</td>
</tr>
<tr>
<td>ROSS-16</td>
<td>10.1.11 On clicking the link to navigate to the Station or Workflow View the data for selected date range shall be loaded by default.</td>
<td>S</td>
<td>Workflow &amp; Station</td>
<td>1.6</td>
<td>Reuse (NG)</td>
<td>ROSS v0.1</td>
</tr>
<tr>
<td>ROSS-17</td>
<td>10.1.12 There shall be navigation buttons to move the date range of the data on by +/- one day in the Workflow and Station View.</td>
<td>S</td>
<td>Workflow &amp; Station</td>
<td>2.1 5.2</td>
<td>New Feature</td>
<td>ROSS v0.2</td>
</tr>
<tr>
<td>ROSS-47</td>
<td>10.1.13 ROSS shall set the date range by default to 3 days.</td>
<td>M</td>
<td>Date Range Filter</td>
<td>1.6</td>
<td>Reuse (NG)</td>
<td>ROSS v0.1</td>
</tr>
<tr>
<td>ID</td>
<td>Description</td>
<td>Priority</td>
<td>Date</td>
<td>Requirement</td>
<td>Feature Type</td>
<td>Version</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
<td>------</td>
<td>---------------</td>
<td>--------------</td>
<td>---------</td>
</tr>
<tr>
<td>ROSS-19</td>
<td>The user shall be able to configure the date range for the data shown in the Workflow and Station Views</td>
<td>M</td>
<td>1.6</td>
<td>Date Range Filter</td>
<td>Reuse (NG)</td>
<td>ROSS v0.1</td>
</tr>
<tr>
<td>ROSS-48</td>
<td>ROSS shall provide an option to terminate a long running query.</td>
<td>S</td>
<td>1.16</td>
<td>Date Range Filter</td>
<td>Reuse (NG)</td>
<td>ROSS v0.1</td>
</tr>
<tr>
<td>ROSS-49</td>
<td>ROSS shall provide an option to reset to the default date range</td>
<td>C</td>
<td>1.17</td>
<td>Date Range Filter</td>
<td>New Feature</td>
<td>ROSS v0.2</td>
</tr>
<tr>
<td>ROSS-50</td>
<td>ROSS shall provide a station hierarchy that allows a user to add a station to the Workflow or drill down into the Station data.</td>
<td>M</td>
<td>1</td>
<td>Station Hierarchy</td>
<td>Reuse (NG)</td>
<td>ROSS v0.1</td>
</tr>
<tr>
<td>ROSS-32</td>
<td>ROSS shall allow a user to filter the station hierarchy to show stations that they are responsible for</td>
<td>S</td>
<td>1.8</td>
<td>Station Hierarchy</td>
<td>New Feature</td>
<td>ROSS v0.1</td>
</tr>
<tr>
<td>ROSS-15</td>
<td>10.1.15 ROSS shall allow a user to filter the station hierarchy to show stations that have generated an alarm in the last X (TBD) hours</td>
<td>S</td>
<td>1.9</td>
<td>Station Hierarchy</td>
<td>New Feature</td>
<td>ROSS v0.1</td>
</tr>
<tr>
<td>ROSS-51</td>
<td>ROSS shall provide additional filters e.g. Show Stations with missing data, or show station groups only (e.g. Show Particulate Stations, Show NG Stations)</td>
<td>W</td>
<td>N/A</td>
<td>Station Hierarchy</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ROSS-52</td>
<td>10.1.16 ROSS shall have an input field that allows a user to search for a station.</td>
<td>W</td>
<td>N/A</td>
<td>Station Hierarchy</td>
<td>New Feature</td>
<td>N/A</td>
</tr>
<tr>
<td>ROSS-33</td>
<td>10.1.17 ROSS shall allow a user to identify the responsible person for a station from IRS</td>
<td>C</td>
<td>1.14</td>
<td>Station Hierarchy</td>
<td>New Feature</td>
<td>ROSS v0.1</td>
</tr>
<tr>
<td>ROSS-53</td>
<td>10.1.18 ROSS shall allow a user to switch the Station Hierarchy between a flat list view and a hierarchy view</td>
<td>S</td>
<td>1.19</td>
<td>Station Hierarchy</td>
<td>New Feature</td>
<td>ROSS v0.1</td>
</tr>
<tr>
<td>ROSS-54</td>
<td>ROSS shall provide an option to add all of the stations shown in the station hierarchy. If more than X stations are to be added into the workflow view, by clicking on one of the station checkboxes,</td>
<td>S</td>
<td>1.15</td>
<td>Station Hierarchy</td>
<td>Reimplement (RWWF)</td>
<td>ROSS v0.1</td>
</tr>
<tr>
<td>Requirement</td>
<td>Description</td>
<td>Priority</td>
<td>Station</td>
<td>Version</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>----------</td>
<td>---------</td>
<td>---------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>ROSS-12</td>
<td>There shall be a link from the graphs to the relevant SOH message. Double clicking on a point in the graphs would display the relevant SOH message.</td>
<td>C</td>
<td>Station</td>
<td>8.17</td>
<td>New Feature</td>
<td>ROSS v0.2</td>
</tr>
<tr>
<td>ROSS-55</td>
<td>ROSS shall allow a user to select graph types.</td>
<td>M</td>
<td>Station</td>
<td>8.1</td>
<td>Reuse (NG)</td>
<td>ROSS v0.1</td>
</tr>
<tr>
<td>ROSS-56</td>
<td>ROSS shall be able to generate the same graphs that are available in the NG Application and SOH Application.</td>
<td>M</td>
<td>Station</td>
<td>8</td>
<td>Reuse (NG)</td>
<td>ROSS v0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Re-implement (SOH)</td>
<td>ROSS v0.2</td>
</tr>
<tr>
<td>ROSS-57</td>
<td>ROSS shall allow multiple graphs to be displayed on the same page.</td>
<td>M</td>
<td>Station</td>
<td>9</td>
<td>Reuse (NG)</td>
<td>ROSS v0.1</td>
</tr>
<tr>
<td>ROSS-58</td>
<td>ROSS shall display a table that lists the sample data used in a graph.</td>
<td>M</td>
<td>Station</td>
<td>12</td>
<td>Reuse (NG)</td>
<td>ROSS v0.1</td>
</tr>
<tr>
<td>ROSS-59</td>
<td>ROSS shall display the Workflow with other types of station data.</td>
<td>M</td>
<td>Station</td>
<td>11</td>
<td>Reuse (NG)</td>
<td>ROSS v0.1</td>
</tr>
<tr>
<td>ROSS-60</td>
<td>ROSS shall provide a station flowchart for different types of Radionuclide particulate and Noble Gas stations, which shows colour-coded status of sensors at that station.</td>
<td>M</td>
<td>Station</td>
<td>13</td>
<td>Reuse (NG)</td>
<td>ROSS v0.1</td>
</tr>
<tr>
<td>ROSS-34</td>
<td>ROSS shall allow multiple SOH graphs to be overlayed (e.g. temperatures of different ovens) in a single graph.</td>
<td>C</td>
<td>Station</td>
<td>10</td>
<td>New Feature</td>
<td>ROSS v0.2</td>
</tr>
<tr>
<td>ID</td>
<td>Description</td>
<td>Required</td>
<td>Station</td>
<td>Re-implementation</td>
<td>Implementation Status</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------------------------------------------------</td>
<td>----------</td>
<td>---------</td>
<td>-------------------</td>
<td>-----------------------</td>
<td></td>
</tr>
<tr>
<td>ROSS-61</td>
<td>ROSS shall provide a replacement WindRose feature</td>
<td>W</td>
<td>Station</td>
<td>N/A</td>
<td>Re-implement (SOH)</td>
<td></td>
</tr>
<tr>
<td>ROSS-62</td>
<td>ROSS shall align the graphs based on relative times</td>
<td>S</td>
<td>Station</td>
<td>9.3</td>
<td>Re-implement (SOH)</td>
<td>ROSS v0.2</td>
</tr>
<tr>
<td>ROSS-63</td>
<td>ROSS shall display the reference sample info (sample ids of 1. Sample in air sampler, 2. Sample in decay, 3. Sample in acquisition) based on a point selected in a graph</td>
<td>S</td>
<td>Station</td>
<td>9.4</td>
<td>Re-implement (SOH)</td>
<td>ROSS v0.2</td>
</tr>
<tr>
<td>ROSS-64</td>
<td>ROSS shall display the currently selected x (i.e. time) and y axis values to the right of the graph when a point in the graph is selected</td>
<td>S</td>
<td>Station</td>
<td>9.5</td>
<td>Re-implement (SOH)</td>
<td>ROSS v0.2</td>
</tr>
<tr>
<td>ROSS-65</td>
<td>ROSS shall provide a tool for issuing commands to a station in an agreed format</td>
<td>S</td>
<td>Command</td>
<td>N/A</td>
<td>Re-implement (Command )</td>
<td>ROSS v0.3</td>
</tr>
<tr>
<td>ROSS-66</td>
<td>ROSS shall download the command messages from the IDC email server and display them in a suitable format to the ROSS user</td>
<td>C</td>
<td>Command</td>
<td>N/A</td>
<td>New Feature</td>
<td>ROSS v0.3</td>
</tr>
<tr>
<td>ROSS-25</td>
<td>The tool for displaying the command emails shall by default show only the command messages that have not received a reply and provide an option to show all command messages</td>
<td>W</td>
<td>Command</td>
<td>N/A</td>
<td>New Feature</td>
<td>N/A</td>
</tr>
<tr>
<td>ROSS-67</td>
<td>10.1.24 ROSS shall provide a feature to search and filter command messages</td>
<td>W</td>
<td>Command</td>
<td>N/A</td>
<td>New Feature</td>
<td>N/A</td>
</tr>
<tr>
<td>ROSS-68</td>
<td>10.1.25 ROSS shall provide a tool to review station sample data</td>
<td>M</td>
<td>Review</td>
<td>N/A</td>
<td>Reuse (NG)</td>
<td>ROSS v0.3</td>
</tr>
<tr>
<td>ROSS-69</td>
<td>ROSS shall provide a commenting tool that allows</td>
<td>W</td>
<td>Review</td>
<td>N/A</td>
<td>New</td>
<td>N/A</td>
</tr>
<tr>
<td>Feature</td>
<td>Requirements</td>
<td>Type</td>
<td>Section</td>
<td>Feature Status</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>--------------</td>
<td>------</td>
<td>---------</td>
<td>----------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ROSS-20</strong> 10.1.27</td>
<td>The Review tool shall be combined in the same screen with the Station data (e.g. graphs, table data, flowchart)</td>
<td>S</td>
<td>Review</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ROSS-21</strong> 10.1.28</td>
<td>By default the 'OK' options shall be selected in the Review Tool</td>
<td>S</td>
<td>Review</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ROSS-22</strong> 10.1.29</td>
<td>ROSS shall enforce the user to make a comment if the 'OK' option is not selected</td>
<td>S</td>
<td>Review</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ROSS-70</strong> 10.1.30</td>
<td>The raw data viewer shall be able to display the following raw message types: SPHD, QCPHD, DETBKPHD, CALIBPHD, BLANKPHAD, SOH, MET, and ALERT.</td>
<td>M</td>
<td>Raw Viewer</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ROSS-23</strong> 10.1.31</td>
<td>The raw data viewer shall allow the user to navigate to the previous/next message for the selected station and message type.</td>
<td>S</td>
<td>Raw Viewer</td>
<td>4.3 / 4.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ROSS-24</strong> 10.1.32</td>
<td>ROSS shall allow a user to view two messages side-by-side.</td>
<td>S</td>
<td>Raw Viewer</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ROSS-71</strong> 10.1.33</td>
<td>ROSS shall provide a text search feature</td>
<td>C</td>
<td>Raw Viewer</td>
<td>5.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ROSS-72</strong> 10.1.34</td>
<td>ROSS shall provide a text highlight feature</td>
<td>W</td>
<td>Raw Viewer</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ROSS-73</strong> 10.1.35</td>
<td>ROSS shall provide a message filters that allow a user to show only a specific section of a message.</td>
<td>W</td>
<td>Raw Viewer</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ROSS-74</strong> 10.1.36</td>
<td>ROSS shall provide dual scrolling when messages are being viewed side by side. As you scroll the other</td>
<td>W</td>
<td>Raw Viewer</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
viewer scrolls at the same rate

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Type</th>
<th>Report Location</th>
<th>Status</th>
<th>New Feature</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROSS-75</td>
<td>10.1.37 ROSS shall align the message data sections when messages are being viewed side by side</td>
<td>W</td>
<td>Raw Viewer</td>
<td>N/A</td>
<td>New Feature</td>
<td>N/A</td>
</tr>
<tr>
<td>ROSS-76</td>
<td>10.1.38 ROSS shall provide a statistics report</td>
<td>M</td>
<td>Report</td>
<td>N/A</td>
<td>Reuse (NG)</td>
<td>ROSS v0.3</td>
</tr>
<tr>
<td>ROSS-77</td>
<td>10.1.39 ROSS shall provide a Test &amp; Evaluation Report</td>
<td>M</td>
<td>Report</td>
<td>N/A</td>
<td>Re-impliment (TER)</td>
<td>ROSS v0.3</td>
</tr>
<tr>
<td>ROSS-13</td>
<td>10.1.40 There shall provide a link from ROSS to the IMS reporting tool (available everywhere).</td>
<td>S</td>
<td>Link</td>
<td>8.13 8.14</td>
<td>Reuse (NG)</td>
<td>ROSS v0.1</td>
</tr>
<tr>
<td>ROSS-26</td>
<td>ROSS shall use single sign on authentication</td>
<td>C</td>
<td>Security</td>
<td>N/A</td>
<td>New Feature</td>
<td>ROSS v0.3</td>
</tr>
<tr>
<td>ROSS-78</td>
<td>ROSS shall restrict access to administration features</td>
<td>S</td>
<td>Security</td>
<td>N/A</td>
<td>New Feature</td>
<td>ROSS v0.3</td>
</tr>
<tr>
<td>ROSS-79</td>
<td>ROSS shall allow user access levels to be configured</td>
<td>S</td>
<td>Admin</td>
<td>N/A</td>
<td>New Feature</td>
<td>ROSS v0.3</td>
</tr>
<tr>
<td>ROSS-80</td>
<td>ROSS shall provide a tool to add/remove stations</td>
<td>S</td>
<td>Admin</td>
<td>N/A</td>
<td>Reuse (NG)</td>
<td>ROSS v0.3</td>
</tr>
<tr>
<td>ROSS-81</td>
<td>ROSS shall allow a user to switch between databases</td>
<td>S</td>
<td>Admin</td>
<td>N/A</td>
<td>New Feature</td>
<td>ROSS v0.3</td>
</tr>
<tr>
<td>ROSS-82</td>
<td>ROSS shall allow a user to configure the database connection settings</td>
<td>S</td>
<td>Admin</td>
<td>N/A</td>
<td>New Feature</td>
<td>ROSS v0.3</td>
</tr>
<tr>
<td>ROSS-83</td>
<td>ROSS shall allow a user to configure the Flowchart database queries</td>
<td>S</td>
<td>Admin</td>
<td>N/A</td>
<td>Reuse (NG)</td>
<td>ROSS v0.3</td>
</tr>
<tr>
<td>ROSS-84</td>
<td>ROSS shall allow the following items to be configured in the personal preferences: 1) Auto-refresh on/off 2) Auto-refresh rate 3) Default Date Range 4) Station Filters</td>
<td>C</td>
<td>Admin</td>
<td>N/A</td>
<td>New Feature</td>
<td>ROSS v0.3</td>
</tr>
<tr>
<td>ID</td>
<td>Description</td>
<td>Role</td>
<td>Status</td>
<td>Type</td>
<td>Note</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
<td>--------</td>
<td>---------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>ROSS-85</td>
<td>ROSS shall provide a links administration tool to allow administrators to add external links into ROSS without requiring developer assistance</td>
<td>W</td>
<td>Admin</td>
<td>N/A</td>
<td>New Feature</td>
<td></td>
</tr>
<tr>
<td>ROSS-86</td>
<td>ROSS could provide a start page (See NetBeans IDE Start Page for an example)</td>
<td>W</td>
<td>Admin</td>
<td>N/A</td>
<td>New Feature</td>
<td></td>
</tr>
</tbody>
</table>