INFORMATION SOCIETY TECHNOLOGIES
(IST)
PROGRAMME

Information Society
Technologies

OpenMolGRID

PROJECT QUALITY PLAN

Contract reference	**IST-2001-37238**
Document identifier:	**OpenMolGRID-7-D7.1-0101-1-4-ProjectQualityPlan**
Date:	**17/12/2003**
Work package:	**WP 7: Project Management**
Partner:	**UT, UU, NEGRI, FZJ, CGX, OMC**
Lead Partner	**FZJ**
Document status:	**APPROVED**
Document classification	**PUBLIC**
Deliverable identifier:	**D7.1**

Abstract: The Project Quality Plan defines the organisation and the methodology that all the partners shall apply throughout the project

Delivery Slip

	Name	Partner	Date
From	GHF DIERCKSEN (TC)	UT/OMC	12/09/2003
Verified by	WPM's M.Romberg	WP1 – WP7 FZJ	30/08/2003 15/12/2003
Approved by	G.H.F.Diercksen (TC) R.Ferenczi (QE)	OMC CGX	17/12/2003 07/04/2004

Document Log

Issue	Date	Comment	Author
0-0	14/07/03	Submitted to WPM's	GHF DIERCKSEN (OMC)
1-1	23/08/03	Submitted to WPM's	GHF DIERCKSEN (OMC)
1-2	08/09/03	Submitted to WPM's	GHF DIERCKSEN (OMC)
1-3	09/12/03		GHF DIERCKSEN (OMC) R FERENZI (CGX)
1-4	17/12/03		GHF DIERCKSEN (OMC) R FERENZI (CGX)

Document Change Record

Issue	Item	Reason for Change
0-0	New document	Recommended by reviewers. This document clearly defines the duties and responsibilities of the Project Coordinator, the Technical Coordinator and the Quality Engineer. It defines the problem solving strategy and the risk identification and management.
1-3	Major revision	Recommended by reviewers. The project presentation has undergone a major revision: The description of the work has been adapted to the workplan. The project management structure has been streamlined by reducing the hierarchy layers and by putting the project coordinator on top of the hierarchy and the reporting/decision making structure. The roles and responsibilities of the members of the project management team have been precisely defined. Guidelines for resolving potential conflicts between the quality assurance plans of the consortium partners and the quality assurance plan of this project have been defined. The software quality assurance has been completely revised: Code review and testing guidelines, previously set up in two separate technical documents, have been defined to insure code quality.
1-4	Minor revision	Typographical errors and the layout have been corrected.

Files

Files in this section relate to actual storage locations on the BSCW server located at https://hermes.chem.ut.ee/bscw/bscw.cgi. The URL below describes the location on BSCW from the root OpenMolGRID directory

Software Products	User files / URL
Word 2000/XP	OpenMolGRID/Workpackage 7/Deliverables/ OpenMolGRID-7-D7.1-0101-1-4-ProjectQualityPlan

Project information

Project acronym:	OpenMolGRID
Project full title:	Open Computing GRID for Molecular Science and Engineering
Proposal/Contract no.:	IST-2001-37238
European Commission:	
Project Officer:	Annalisa BOGLIOLO
Address:	European Commission - DG Information Society F2 - Grids for Complex Problem Solving B-1049 Brussels Belgium
Office:	BU31 4/79
Phone:	+32 2 295 8131
Fax:	+32 2 299 1749
E-mail	annalisa.bogliolo@cec.eu.int
Project Coordinator:	Mathilde ROMBERG
Address:	Forschungszentrum Jülich GmbH ZAM D-52425 Jülich Germany
Phone:	+49 2461 61 3703
Fax:	+49 2461 61 6656
E-mail	m.romberg@fz-juelich.de

Content

1. INTRODUCTION

1.1. PURPOSE OF THE PROJECT QUALITY PLAN

The Project Quality Plan (PQP) defines the organisation and the methodology that all the partners shall apply throughout the project. It forms a common standard for the entire project lifecycle. The PQP is kept as pragmatic as possible because the project includes only 5 Partners without making any compromises concerning the quality assurance of the project.

The PQP is a project deliverable: D7.1. Its purpose is to:

- Identify processes that will be applied to assure quality: Chapter 2: Quality Objectives
- Defines roles and responsibilities to ensure a successful project with deliverables on time:
  - Chapter 3: Project presentation and organisation
  - Chapter 4: Project deliveries
- Provide the OpenMolGRID management with indicators to allow them to take appropriate decisions, and to track and report on project progress: Chapter 5: Project monitoring, reporting and risk management
- Describe software management practices: procedures, rules, and applicable methods for the OpenMolGRID project:
  - Chapter 6: Documentation management
  - Chapter 7: Software Quality Assurance
  - Chapter 8: Configuration management
  - Chapter 9: Anomalies management
  - Chapter 10: Standards and tools
  - Chapter 11: Archiving and storing
  - Chapter 12: Delivery Procedure

1.2. APPLICATION AREA

The Project Quality Plan shall be applied:

- by all partners,
- for all deliverable to European Commission,
- and for deliverables between partners.

Consortium Partners, supervise and check the work performed by their own staff in accordance with the OpenMolGRID PQP.

This PQP is designed to be consistent with ISO 9001 and should be interpreted with reference to:

- The Terms and Conditions for EC contracts.
- The OpenMolGRID proposal: IST proposal: Annex 1: Description of Work (in its present version D7.1a)

1.3. PROJECT QUALITY PLAN EVOLUTION PROCEDURE

Any project partner may request amendments but each amendment must be analysed by the Project Technical Coordinator and the Project Quality Engineer and then approved by the OpenMolGRID Project Coordinator.

1.4. LACK OF ADHERENCE TO THE PROJECT QUALITY PLAN

If there is a conflict between a Consortium Partner's Quality Assurance procedures and those imposed on this project, this should be brought to the attention of the OpenMolGRID Project Coordinator. Normally, precedence shall be given to the OpenMolGRID Project Quality Plan.

1.5. APPLICABLE DOCUMENTS AND REFERENCE DOCUMENTS

Applicable documents

[A1] OpenMolGRID-7-D7.1a-0100-1-1	Deliverable D7.1a: Work Plan
[A2] OpenMolGRID-7-D7.1b-0106-1-1	Deliverable D7.1b: Risk Management procedure

Reference documents

[R1] OpenMolGRID	IST proposal: Annex 1: Description of Work (in its present version D7.1a [A1])
[R2] The Java Language Reference,	http://java.sun.com/docs/books/jls/second_edition/html/j.title.doc.html
[R3] Java Web Start 1.0	http://java.sun.com/products/javawebstart/
[R4] Javadoc – The Java API Documentation Generator	http://java.sun.com/j2se/1.3/docs/tooldocs/win32/javadoc.html
[R5] Javadoc Doclets	http://java.sun.com/products/jdk/javadoc

1.6. PROJECT QUALITY PLAN EVOLUTION PROCEDURE

Different events may cause the content of this document to be modified. For instance:

- Evolution project characteristics
- Changes in techniques or tools.

Any project partner may request amendments but each amendment must be analysed and approved by the Technical Coordinator and the Quality Engineer as defined in section 1.3.

1.7. TERMINOLOGY

Glossary

[Ax]	Applicable Document
[Rx]	Reference Document
API	Application Programming Interface
CORBA	Common Object Request Broker Architecture.
EC	European Commission
HTML	Hypertext Markup Language
IST	Information Society Technologies
JVM	Java Virtual Machine.
PC	Project Coordinator
PDF	Portable Document Format
PM	Person Month
PO	Project Office
PPR	Project Progress Report

PQP	Project Quality Plan
QE	Quality Engineer
QuI	Quality Indicator
TC	Technical Coordinator
SDK	Software Development Kit.
WP	Work Package
WP1	Grid Data Warehousing of Molecular Structure-Property -(Activity) Information
WP2	Molecular Descriptor Generation and Quantitative Structure-Property Relation (QSPR) Model Bilding on the Grid
WP3	Computational Molecular Engineering of New Compounds and Materials
WP4	Grid Integration
WP5	Test Application for Chemical and Pharmaceutical Predictions
WP6	Dissemination of Results
WP7	Project Management
WPM	Work Package Manager

2. QUALITY OBJECTIVES

The OpenMolGRID quality objectives are to provide quality support to partners and monitor adherence to the Project Quality Plan throughout the project lifecycle.

The PQP is designed to provide for the assurance of quality, according to the main OpenMolGRID project characteristics.

2.1. PROJECT CHARACTERISTICS

The OpenMolGRID project is characterised by:

- Cooperation of partners from IT and computational chemistry
- Distributed development of software (4 partners)
- Software portability (Linux, Solaris, ...)

The OpenMolGRID Project is driven by the Applications requirements in terms of time-scale, performance and reliability.

From a usability point of view, the OpenMolGRID project must address the following quality factors:

- Conformity: conformance to requirements
- Efficiency: the software must use resources in the best way (memory, CPU, ...)
- Reliability: the software run without anomaly
- Portability: the software can be easily ported on different environments and computers
- Flexibility: aptitude of the software to evolve with appearance of new requirements
- Maintainability: easy to debug and correct
- Testability: complete test suite for non regression testing
- Documented: the deliverables must be well documented

To summarise: OpenMolGRID involves partners from IT and computational chemistry. It is a Research & Development project aiming at developing a Grid based complex problem solving environment for molecular engineering.

2.2. QUALITY SYSTEM

The quality system applied on the project is described in the present PQP and the following Project Plans:

Project Plans	**Delivered by**	**Maintained by**	**Described in:**
Project Deliverables Plan	QE	QE	Section 4.2.1
Project Documentation Repository	WP7	WP7	Website: https://hermes.chem.ut.ee/bscw/bscw.cgi/0/378
Project Folder	All	WP7	Section 5.6
Documents Templates	QE	QE	Section 3.4.5

2.3. QUALITY ORGANISATION

A Project Quality Engineer (QE) is appointed. The QE is in charge of providing quality support to partners and of checking for the application of the Project Quality Plan for each deliverable produced by the consortium.

The responsibilities of the QE are described in section 3.4.5.

3. PROJECT PRESENTATION

3.1. PROJECT OBJECTIVES AND GOALS

The overall objectives of the project are given in [R1]. The full description of the project may be viewed on the OpenMolGRID Website.

- The main objective of the OpenMolGRID project is to develop and deploy the application layer for a system for molecular engineering and molecular design, integrating heterogeneous, distributed data and computational resources. This application layer will be based on the UNICORE Grid infrastructure.

3.2. DESCRIPTION OF THE WORK

The structure of the programme of work is as follows:

- WP1 Grid Data Warehousing of Molecular Structure-Property (-Activity) Information:

  The objective of this workpackage is to locate, extract, transform (e.g. descriptor calculation), and integrate globally distributed data sets describing environmental, toxicological, chemical, biological, and pharmacological information. This workpackage will place emphasis on the quality of the warehouse's content with an effort to provide added value to the greater chemical community. Fundamental to this is the need to provide a substructure search facility. To do this the warehouse will act as a UNICORE client. As such it can access other resources available on the Grid (i.e. within the virtual organisation). In addition, the warehouse will provide a subset of commonly used molecular descriptors that can be viewed as a complex transformation process that requires the use of specialised descriptor calculation programs, a task which is computationally intensive.

- WP2 Molecular Descriptor Generation and Quantitative Structure-Property Relation Model Building on the Grid:

  The objective of this WP is the adaptation of existing software modules for QSPR/QSAR model building. The following modules will be implemented: 2D to 3D conversion, semi-empirical quantum-chemical calculations, molecular descriptor calculation and QSPR/QSAR model development. This WP specifies application neutral input and output formats for relevant software modules and develops UNICORE plugins in close cooperation with WP4.

- WP3 Computational Molecular Engineering of New Compounds and Materials:

  The objective of this WP is development of software modules and respective UNICORE plugins for the reliable prediction of chemical structures with predefined chemic al properties and/or activity values. The system will provide the following functionality: structure generation, fragment library, and theoretical validation of generated structures with the help of descriptors and/or QSPR/QSAR models.

- WP4 Grid Integration:

  The objective of this workpackage is to model the workflow for molecular design and engineering using the EUROGRID Grid technology. Its task is to enhance the EUROGRID seamless interface for database access, new applications like descriptor calculation or predictive model building, and workflow by exploiting the EUROGRID plugin mechanism and the server's application interface. Thereby the OpenMolGRID user will be guided through the complex process of molecular engineering. In addition, it establishes a Grid testbed at the partner sites.

- WP5 Test Application of the System for Chemical and Pharmaceutical Predictions:

The main objective of this work package is to test OpenMolGRID performance in different aspects. It includes the functional testing and the testing of the model building and molecular engineering parts of the system. Beside the *in silico* testing, the systems will be tested in real life chemical and pharmaceutical applications using data of *in vitro* human fibroblast toxicity. To prove the efficiency and predictive power of the system towards industrial users, a considerable number of compounds (30,000) are synthesized and measured. Based on the experimental results, a model will be built by the QSPR tools of the OpenMolGRID system. The accuracy of the model will be validated, and the predictive capability will be tested. Ultimately, the molecular engineering capability (generation of structures with favorable properties; the OSPR tool) will be evaluated whether novel potential anticancer molecules can be identified using the system.

- WP6 Dissemination of Results:

  Establish Web presence for OpenMolGRID. Communicate research results to the international community through conferences and articles. Demonstrate the OpenMolGRID results by presentations at trade shows. Exploit the OpenMolGRID technology, results and expertise by the commercial partner

- WP7 Project Management:
  Goal-oriented scientific, technical administrative and financial management of OpenMolGRID. Quality assurance for project deliverables and other output. Timely submission of project deliverables. Organisation of regular steering committee and technical meetings.

3.3. CONSORTIUM ORGANISAT ION

3.3.1. Participants List

Partic. Role	Partic. no.	Participant name	Participant short name	Country	Date enter project	Date exit project
P (1)	1	University of Tartu	UT	Estonia	Start of project	End of project
P	2	University of Ulster	UU	Northern Ireland	Start of project	End of project
P	3	Mario Negri Institute	NEGRI	Italy	Start of project	End of project
C (2)	4	Forschungszentrum Jülich GmbH	FZJ	Germany	Start of project	End of project
P	5	ComGenex, Inc.	CGX	Hungary	Start of project	End of project
A	6	OpenMolCONSULTING	OMC	Germany	Start of project	End of project

Legend:

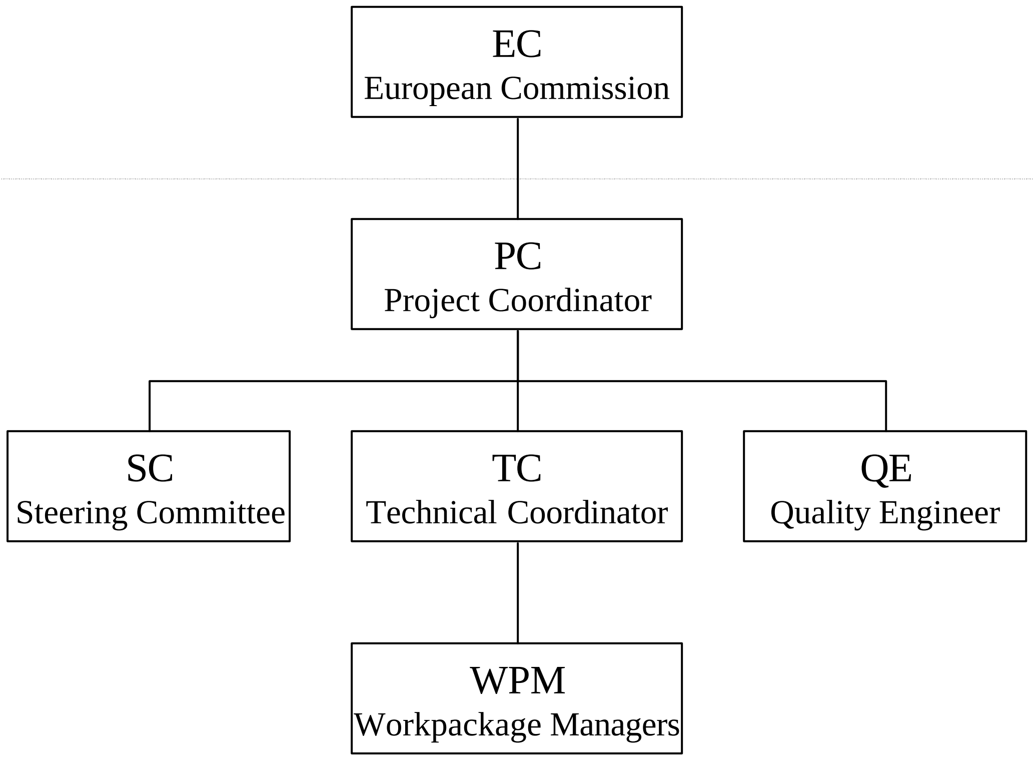
C = Coordinator (subject to approval by European Commission); P = Principal contractor; A = Assistant Contractor;

(1) Coordinating site: 01-15 month; (2) Coordinating site: 16-27 month

3.4. PROJECT MANAGEMENT STRUCTURE

The project management structure has been designed to fit the size and requirements of the project. It runs under the control of the Project Coordinator, the Project Technical Coordinator, and the Project Steering Committee. They are supported by a Quality Engineer.

The project management is presently organised with the follow ing structure:



3.4.1. Project Coordinator

The project coordinator (PC) is determined by the contract with EC. The PC has the following responsibilities:

- Administrative management;
- Final conflict resolution authority for managerial and technical matters;
- Relationship and correspondence with the Commission and third Parties and information thereoff to the Steering Committee;
- Collection of the Partners' documents and cost and other statements and forwarding thereof to the Commission;
- Provision of the Chairperson of the Project Steering Committee. Preparation and distribution of its minutes. Follow-up of its decisions on administrative issues;
- Scheduling and coordinating of Steering Committee Meetings;
- Supervision of the progress rela tive to the time schedules in workplan or otherwise set up by the common agreement of the Partners;
- Receiving requests, and deliberating on the same, from the other Partners to transfer funds between budget heads.

3.4.2. Steering Committee

The SC is composed of one representative from each Partner. It will be chaired by the PC or its representative. Each representative shall have one vote. The TC is ex-officio member and has one vote. If not specified otherwise, decisions are taken by simple majority of the Partners present. In case of a draw the vote of the Chairperson is counted twice. On the request of the PC the SC may take decisions by e-mail. Decisions are taken by simple majority of the votes received by e-mail within 5 working days. No reply is counted as agreement. The SC shall meet at least once in 6 months at the request of the PC or at any other time at the request of one of the Partners if deemed appropriate by the Chairperson.

The SC shall:

- Review the progress of the Project;
- Make the necessary decisions on administrative and technical issues;
- Lay down and review or amend procedures for publication and press release with regard to the Project.

3.4.3. Technical Coordinator

The Technical Coordinator (TC) supports and reports to the PC on technical issues among all WPs. The TC is appointed by the coordinating partner in agreement with the SC. The responsibilities of the TC are:

- Coordination of the planning and technical execution of workpackages 1-5;
- Monitoring the progress and quality of the technical work;
- Software bug tracking;
- Approving of the technical contents of deliverables;
- Identifying potential technical and scheduling problems;
- Proposing corrective actions and any necessary changes in strategy;
- Implementing and following up of the decisions by the SC on technical issues;
- Agreeing on the technical choices and standards as required;
- Resolving technical conflicts between work packages;
- Scheduling and co-ordinating technical meetings;
- Reporting to the PC of cases where partners fail to meet their commitments and of unresolved conflicts between work packages.

3.4.4. Work Package Manager

Each WP is under the responsibility of a single partner who designates a WP Manager (WPM). They organise appropriate contacts between the concerned partners and are in charge of producing the deliverables. The responsibilities of the WPM are:

- Co-ordinate and supervise all WP technical and professional activities;
- Formal reporting of progress, anticipated problems and deviations from work plan;
- Initiate work, discussions, visits and papers;
- Ensure the quality of the deliverable products;
- Resolving technical conflicts within the individual work packages;
- Drawing the attention of the TC to cases where there are unresolved conflicts within the individual work packages.

3.4.5. Quality Engineer

The responsibilities of the QE are:

- Preparing and proposing the Project Quality Plan;
- Providing software coding and testing guidelines;
- Providing deliverable templates;
- Monitoring the quality and timely production of deliverables;

- Formally approving all deliverables (documents and software);
- Reporting on deviations from the Project Quality Plan.

4. PROJECT DELIVERIES

4.1. PROJECT PHASES

The project starts with a user requirement gathering phase for each WP, iterating with a global architecture specification phase, followed by the development phase.

Project Milestones	***Target Month***
Procedures for project management and internal communication operational	01
Web site for OpenMolGRID implemented D6.1)	02
Specification of UNICORE-compliant software components	05
Report on target properties for pharmaceutical and phytopharmaceutical compounds	06
Setup of initial testbed which includes providing Certification Authority and Certification Authority-Policy (D4.5b))	07
Design of data base access integration (D4.1a, D4.1c)	09
Peliminary structural fragment library for algorithm development implemented	10
First Project Review (D7.4b)	10
Initial version of UNICORE-compliant software components	15
Design of Meta Plugin for automated workflow support (D4.2a)	15
First version of software for data base access (D4.1b, D4.1d)	15
Design of workflows for DC and model mevelopment (D4.2c, D4.3)	15
Initial test report on syntactic testing. (D5.1)	15
Design of integrated interface (D4.4)	17
Second project review (D7.7b)	18
First version of the Meta Plugin for automated workflow support (D4.2b)	19
First OpenMolGRID results presented in conferences (D6.2)	19
Software for descriptor estimation implemented	20
Working prototype of data warehouse available for testing	20
Test reports on *in silico* testing for SAR. (D5.2)	21
Initial version of UNICORE-compliant software components	21
Test of molecular engeneering tools	22
Improved version of Meta Plugin for automated workflow support	23
Software for generating molecular structures implemented	23
Test report on model building capability for large data set. (D5.3)	24
OpenMolGRID results presented in conferences and through articles in Journals; demonstration reports of the Grid system and customers feedback	24
Final version of UNICORE compliant software components.	25
Final version of the integrated interface and services	26
Validation of the Grid system with real life cases. (D5.5)	27
Second test report on syntactic testing. (D5.1)	27
Test report on predictive capability for large data set (D5.4)	27
OpenmolGRID results presented in conferences and through articles in Journals	27
Demonstration reports of the OpenMolGRID and customer feedback (D6.3, D6.6)	27
Final project review (D7.10a)	30

4.2. EC PROJECT DELIVERABLES

4.2.1. Deliverables

Deliv. ID	*Deliverable name*	*Lead Partner*	*Deliv. Type*[1]	*Plan*
	First Year Deliverables			
D7.1	Project Quality Plan	OMC	Report	1
D7.1a	Project Plan	UT	Report	1
D7.1b	Risk Management Procedure	UT	Report	1
D6.1	Web presence for OpenMolGRID	UT	Other	2
D7.2a	Quarterly Project Report	UT	Report	3
D7.3b	Consortium Agreement	UT	Other	3
D1.1a	Data warehouse software specification	UU	Report	5
D2.1	Specification of software modules for descriptor calculation and model development and their Grid interface components	UT	Report	5
D3.1	Specification of software modules for molecular structure generation and for descriptor value estimation and their Grid interface components	UT	Report	5
D1.3	Properties and priorities of the data for pharmaceutical and phytopharmaceutical compounds	Negri	Report	6
D7.3a	Quarterly Project Report	UT	Report	6
D3.2a	Specification of the structural fragment library	UT	Report	7
D4.5b	The OpenMolGRID Certification Authority Policy	FZJ	Report	8
D4.1a	Specification of the generic user interface for database access	FZJ	Report	9
D4.1c	Specification of the database access interface	FZJ	Report	9
D7.4a	Quarterly Project Report	UT	Report	9
D7.4b	First Project Review	UT	Report	10
D4.5a	Description of the OpenMolGRID Grid architecture, security architecture, and infrastructure and the deployment of the project testbed	FZJ	Report	12
D7.5a	Annual Project Report	UT	Report	12
	Second Year Deliverables			
D1.1b	ECOTOX – Terratox data specification	UU	Report	13
D1.1c	ECOTOX – Aquire data specification	UU	Report	13
D1.1d	NTP data specification	UU	Report	13
D1.1f	Specification of the database access tool for the OpenMolGRID data warehouse	UU	Report	13
D1.4a	Description of Ecotox	UU	Report	13
D1.4b	Description of NTP	UU	Report	13
D1.4e	Description of data warehousing	UU	Report	13
D2.4a	Description of the quantitative structure property / activity relation model: model building and application	UT	Report	13
D2.4b	Description of Codessa Pro modules (and Mopac)	UT	Report	13

1 Software = Code plus Documentation

Deliv. ID	*Deliverable name*	*Lead Partner*	*Deliv. Type*[1]	*Plan*
D3.6	Description of the molecular engineering procedure	UT	Report	13
D1.1e	Experimental data specification	CGX	Report	15
D1.2	A set of software components (and their documentation) which implement various parts of the OpenMolGRID warehousing processes.	UU	Software	15
D1.4c	Description of database with experimental data	CGX	Report	15
D4.1b	Software and documentation: Generic user interface plugin for database access	FZJ	Software	15
D4.1d	Software and documentation: Database access tool for selected databases	FZJ	Software	15
D4.2a	Specification of the Grid interface for classes of applications to support automated workflows	FZJ	Report	15
D4.2c	Specification of the workflow for descriptor calculation (DC)	FZJ	Report	15
D4.3	Specification of the workflow for QSPR/QSAR model development	FZJ	Report	15
D6.4	First report on exploitation results	CGX	Report	15
D7.6a	Quarterly Project Report	UT	Report	16
D4.4	Specification of the workflow for the overall molecular engineering process	FZJ	Report	17
D3.4	Software and documentation: Modules for rapid descriptor value estimation and their Grid interface components	UT	Software	18
D7.7b	Second Project Review	FZJ	Report	18
D4.2b	Software and documentation: Meta plugin for workflow support	FZJ	Software	19
D4.6a	Specification of the command line client for UNICORE	FZJ	Report	19
D6.2	Presentation of first year OpenMolGRID results at conferences	UT	Report	19
D7.7a	Quarterly Project Report	FZJ	Report	19
D3.5	Software and documentation: Modules for descriptor value calculation and their Grid interface components	UT	Software	20
D4.7	Integration test report	FZJ	Report	20
D1.5	Specification, design, and implementation of the Custom Data Repository	CGX	Software	21
D2.2	Software and documentation: Modules for descriptor calculation and their Grid interface components	UT	Software	21
D2.3	Software and documentation: Modules for QSPR/QSAR model development and their Grid interface components	UT	Software	21
D5.2	QSPR/QPSR models for Multi-Drug Resistance (MDR) and G-Protein Coupled Receptor (GPCR) activity as well as test results	CGX	Report	21
D6.5	Second report on exploitation results	CGX	Report	21
D7.8a	Quarterly Project Report	UT	Report	22
D3.2b	Software and documentation: Structural fragment library	UT	Software	23
D3.2c	Software and documentation: Database access tool for the structural fragment library	UT	Software	23
D3.3	Software and documentation: Modules for molecular structure generation and their Grid interface components	UT	Software	23

Deliv. ID	***Deliverable name***	***Lead Partner***	***Deliv. Type*[1]**	***Plan***
D5.3	QSPR model for large number of newly generated structures; evaluation results	CGX	Report	24
	Third Year Deliverables			
D4.6b	Software and documentation: Command line client for UNICORE	FZJ	Software	25
D7.9a	Annual Project Report	UT	Report	26
D5.1	Test reports on functional testing for algorithms, modules and software frames, suggestion for further development and fine tuning of the system	CGX	Report	27
D5.4	Test result for prediction power and structure design capabilities of the Grid system	CGX	Report	27
D5.5	Validation of the Grid system for chemicals, including pharmaceutical and phytopharmaceutical compounds	NEGRI	Report	27
D6.3	Presentation of second year OpenMolGRID results at conferences	UT	Report	27
D6.6	Third report on exploitation results	CGX	Report	27
D7.10a	FinalProject Report	FZJ	Report	30

5. PROJECT MONITORING AND REPORTING

5.1. INTRODUCTION

The following reporting documents are described hereafter:

- Quarterly Project Reports;
- Project Plan;
- Project Deliverable Plan;
- Risk Status Reports.

Project monitoring and reporting documents are reviewed by the TC and QE.

5.2. COMMUNICATION

5.2.1. E-mail

The preferred communication among OpenMolGRID Partners is e-mail. All official communication must be sent to one or both of the following official OpenMolGRID mailers depending on the subject:

omg-tech@chem.ut.ee : all mail concerning technical matters
omg-steering@chem.ut.ee : all mail concerning administrative matters (PC & SC)

All mail must be addressed to an individual/individuals, either by function or by name. The addressee must respond to the mail preferably on the same day but in any case within three working days. The WPM's are responsible that all mail concerning their Workpackage is answered within the deadline. All mail will be archived and will be monitored by the TC and the PC, respectively.

5.2.2. BSCW documentation server

A BSCW groupware system (see http://www.orbiteam.com) has been established at the UT partner site. The system provides crucial communication features such as notes, discussions, calendar, and daily activity reports.

This system is the central repository for all deliverables, presentations, reports, technical documents, internal reviews and other documents of interest.

5.3. CONFLICT RESOLUTION

Conflict resolution follows a three step procedure and depends on the subject. For each conflict the resolution procedure must be documented.

5.3.1. Technical matter

- **Step 1:** The concerned WPM's try to work out a solution for the case that is accepted by all WPM's. This process must be completed within 5 working days. If no solution is agreed within this time the case is declared a conflict and the next conflict resolution step is started by notifying the TC.
- **Step 2**: The TC decides the conflict by proposing a solution after studying the case and hearing the arguments of the WPM's. The TC may take outside advice. This process must be completed within 10 working days. If the solution is not accepted by all concerned WPM's the next conflict resolution step is started by notifying the PC.
- **Step 3**: The PC decides the conflict by proposing a solution after studying the case and hearing the arguments of the WPM's, the TC and the QE, if appropriate. The PC may take

outside advice. This process must be completed within 5 working days. If no solution is agreed upon within this time the PC must bring the conflict to the attention of the EC Project Officer.

5.3.2. Managerial matters

- **Step 1:** The concerned Consortium Partners try to work out a solution for the case that is accepted by all Partners. This process must be completed within 10 working days. If no solution is agreed upon within this time the case is declared a conflict and the next conflict resolution step is started by notifying the PC.
- **Step 2:** The PC decides the conflict by proposing a solution after studying the case and hearing the arguments of the Partners. The PC may involve the SC, TC, QE and the EC Project Officer. This process must be completed within 10 working days. If the solution is not accepted by all concerned Partners the PC must bring the conflict to the attention of the EC Project Officer.

5.4. RISK MANAGEMENT

The risk management procedure defines the rules to identify, estimate, treat and monitor risks. It is described in Deliverable D7.1b, Risk Management Procedure [A1].

5.4.1. Initial potential risks

5.4.1.1. Product definition risks

From the product definition point of view, the following initial risks can be listed:

- Mismatch between the OpenMolGRID requirements and the capabilities of UNICORE.
- Conflicting requests coming from the Application areas for the functionalities of the Data Warehouse or for the priorities in its development.

5.4.1.2. Management risks

From the management point of view, the following initial risks can be listed:

- Effectiveness of the overall co-ordination and management structure.
- Schedule slippage, late deliveries and slow progress in general.
- Under estimation of the required effort.
- Turn over of key-personnel.
- Late resource availability. The late availability of software, hardware and human resources can be an obstacle to the project's progress.

5.4.1.3. Technical Risks

From the technical point of view, the following initial risk can be listed:

- Unacceptable performance of the final system in terms of computers and network facilities.

A more detailed decription of the individual risks and procedures for their control are contained in the Annex of the Risk Management Procedure [A1]

5.5. QUARTERLY PROJECT REPORTS

Every quarter, each WP Manager submits its WP Quarterly Report to the PC, TC and QE. The PC generates the quarterly project report from the WPs Quarterly Reports.

Quarterly Project Report document templates are provided (see §14).

5.5.1. European Commission Quarterly Project Report

The European Commission Quarterly Project Report is composed of:

- Technical achievement synthesis
- Current situation of the project
- Current situation for each WP
- Status of milestones
- Resources
- Deviation
- Risks
- Other
- Plans of next reporting period
- Meetings/conferences/Papers/Dissemination
- Efforts for the reporting period
- Partner/WP: Consumed/Estimated Funded (and qualitative appreciation about Unfunded effort)
- Project Plan update

5.6. PROJECT FOLDER

The Project Folder is managed by WP7. It contains one folder for each workpackage. The Workpackage Folders are managed by the WPM's. Each Workpackage Folder contains three folders: Deliverables, Technical Documents and Unclassified. In addition, the Workpackage 7 Folder contains the EC File.

EC file:

- Project Proposal;
- Contract;
- Mails exchanged with the EC;
- Delivery Notes;
- Project Reviews;
- Work Plan;
- Quality Plan.

6. DOCUMENTATION MANAGEMENT

6.1. INTRODUCTION

The aim of this chapter is to describe the documentation management procedure for the OpenMolGRID project. It defines standard rules and procedures related to documentation production that all the partners should apply throughout the project.

The documentation management procedure is applicable

- by all partners
- for all deliverable documents to the European Commission
- for documents exchanged between partners.

It is recommended that documents internal to the consortium follow these guidelines.

6.2. DEFINITIONS

Deliverable:	A deliverable consists of one or more types of products (documents, software components, etc.). The list of deliverables is defined in section 4.2.
Deliverable identifier:	A deliverable identifier uniquely identifies each deliverable. The list of deliverables and associated deliverable identifier are given in section 4.2. For example, the deliverable Project Quality Plan is identified by the deliverable identifier: D7.1.

6.3. DOCUMENTS PUBLICATION RULES

6.3.1. Document presentation

All partners will use standard document templates in order to produce standardised documentation. These templates are provided in Annex 2.

Each document contains:

- a title page,
- an executive summary,
- a delivery slip table (for deliverables only),
- a document log and document change log,
- the file name and location on the project documentation server,
- a glossary if necessary,
- a list of referenced documents,
- annexes if necessary.

All documents will be written in English and produced using word processing software from the list of tools described in section 10.1.

The OpenMolGRID project name will be written in the final official format: OpenMolGRID.

English date format will be used for all documents, e.g. 01/02/2003 for 01 February 2003.

The following table gives an overview of the main attributes of a document.

Attribute	Description	Title page	Other pages
Logo	IST Logo (colour)	X	
Project Name	OpenMolGRID (blue)	X	
Document title		X	X
Document identifier	Section 6.3.2, e.g.: OpenMolGRID-7-D7.1-0101-0-0-ShortDocumentTitle	X	X
Date	Last update	X	X
Partner(s)	Involved Partner acronyms, e.g. FZJ, UU, etc.	X	
Lead Partner	WPM Partner acronym	X	
Document Status	See section 6.5, e.g.: DRAFT	X	
Work Package producing the document	e.g.: WP7: Project Management	X	
Classification	See section 6.4, e.g.: PUBLIC, CONFIDENTIAL, etc	X	X
Deliverable identifier: only for deliverable documents	See section 4.2, e.g.: D7.1	X	
Abstract	Executive summary	X	
Contract reference	IST-2001-37238	X	X
Page number	30/40		X
Sort title	Short document title		X

6.3.2. Document identifier

A unique document identifier to ensure effective version control must reference each document.

The document identifier is defined as:

For deliverable documents:

<Project name>–<WP number>–<Deliverable identifier>–<Document number>–<Version>–<Revision>–<Short document title>

e.g.: OpenMolGRID-7-D7.1-0101-0-0-ProjectQualityPlan

For non-deliverable documents:

<Project name>–<WP number>–<Document type>–<Document number>–<Version>–<Revision>–<Short document title>

e.g.: OpenMolGRID-7-TED-0111-1-0-WorkpackageQuarterlyReportQ4

6.3.2.1. Project Name

The Project Name = **OpenMolGRID**

6.3.2.2. WP Number

The WP Number represents the number of the work package producing the document:

OpenMolGRID-7-D7.1-0101-0-0-ProjectQualityPlan for WP7

6.3.2.3. Deliverable identifier

This field is required only for deliverable documents (see definition in §6.2).

For example:

OpenMolGRID-7-D7.1-0101-0-0-ProjectQualityPlan

6.3.2.4. Document Type

This field is required only for non-deliverable documents.

Document types are listed in Annex 1.

For example:

OpenMolGRID-7-TED--0111-1-0-WorkpackageQuarterlyReportQ4 (Internal technical document)

6.3.2.5. Document Number

For each WP the range 0001 to 9999 is allocated. Within each WP, a number (0001 to 9999) is assigned to each new document of any type. In order to register the previous documents, the document number will start at 0101 for new documents. Document numbers within each WP shall be assigned by the WPM:

OpenMolGRID-7-D7.1-0101-0-0-ProjectQualityPlan (1st new document of WP7)

6.3.2.6. Version-Revision

For a document in a draft version, the version and the revision starts at 0-0.

When a document is distributed internally or delivered, the Version-Revision number must always be updated. When the delivery concerns just a part of the document only the revision number is incremented (it is assumed that when the modification content is about 30 % of the document, it is necessary to deliver a new version).

For delivery of a revision, the log and the change log of the document must be updated, and the modifications marked on each page.

For a new version, if the log and change log become too large, only the history of Version number remains.

For example,

OpenMolGRID-7-D7.1-0101-0-0-ProjectQualityPlan (first new document of the WP7)

OpenMolGRID-7-D7.1-0101-1-0-ProjectQualityPlan (draft version in a consistent form)

OpenMolGRID-7-D7.1-0101-2-0-ProjectQualityPlan (Deliverable approved by TC and QE)

6.4. CLASSIFICATION ATTRIBUTE

This attribute defines the confidentiality level:

- CONFIDENTIAL: Restricted circulation list (specify in footnote)
- INTERNAL: Internal circulation within project
- PUBLIC: Public document.

6.5. DOCUMENT STATUS

The following statuses of a document appear on the document presentation page:

- DRAFT
- APPROVED (Approved by the TC and QE. See section 6.6 for more details.)

After delivery to European Commission, the status of the document becomes:

- ACCEPTED with no changes or minor changes
- ACCEPTED subject to changes proposed
- REJECTED

6.6. DOCUMENT REVIEW

All deliverables are reviewed by the TC (see Delivery procedure section 12). He may ask WPM's for assistance and comments. The final version of a document is checked by the QE for quality aspects (coherence):

- the format of the document is correct, the presentation, the identification, the title pages, the summary, the glossary, the annexes, the plan, the production rules are respected
- the content of the document is coherent itself and contains all the information necessary for its comprehension
- the content of the document is coherent and compliant with other documents
- if the document has to be incorporated into another document, the resulting document's coherence is also checked.

The review process is documented on a Dialog Form. The Dialog Form has the same name as the reviewed Deliverable with the word 'REVIEW' added at the end and is stored in the same folder as the reviewed document. The possible outcomes of a review are that the document is:

- Approved with no changes or minor changes (e.g. typos)
- Approved subject to changes proposed by the TC and/or QE
- Not approved

The follow-up procedure of each of the above case is:

- The author(s) simply correct the errors and issue the document as approved.
- The author(s) make the necessary changes, which are proposed to the reviewers (this may be iterative but must be achieved within a period decided by the QE). Once all the changes are made and accepted the document is re-issued as approved.
- The author(s) re-work the document. It is then re-issued as a further draft and reviewed again.

The document review process is monitored by the QE. He supplies the responsible WPM with the appropriate template and the WPM and TC with the time schedule of the reviewing process (calendar dates). The following time schedule applies to the reviewing process:

Action	***Time due (working days)***
Template for the deliverable send to the author(s) Time schedule for the deliverable send to the authors & TC	-25
Deliverable submitted to the TC & QE	-15
Deliverables Approved/Not approved by the TC & QE	-10
Deliverables submitted to the PC if APPROVED	-10
Deliverables returned to author(s) for corrections if NOT APPROVED Notification of the PC	-10
Deliverable resubmitted to the TC & QE	-8
Deliverables Approved/Not approved by the TC & QE	-7

Action	***Time due (working days)***
Deliverables submitted to the PC if APPROVED	-7
Emergency action initiated by TC & QE if NOT APPROVED	-7
Deliverables submitted to the EC	0

6.7. DOCUMENT MODIFICATION

As a document can be revised during the project lifecycle, it is necessary to use a version revision mechanism based upon the identification number in order to:

- Track all the modifications that affect a document after its delivery;
- Inform each partner on the last released version of a document by uploading it on the project document repository;
- Provide each partner, at any given time, with a consistent vision of the documentation state.

7. SOFTWARE QUALITY ASSURANCE

This section serves as a guideline to good practices involved with software quality assurance. Goals are intended to better quality in terms of robustness, maintainability and extensibility.

7.1. CODING GUIDELINES

7.1.1. Introduction

The purpose of this document is to provide a standard to enable developers to write Java code that is consistent and well defined, making the code easier to maintain and read. These guidelines refer to the language of the Java 2 Platform (Standard Edition Version 1. and above) as specified by Sun Microsystems [R1].

7.1.2. Language Usage

7.1.2.1. Packages

All source files should belong to a package. This base of this package should be "org.openmolgrid". Projects should be grouped together in logical units of classes. Packages should not be interdependent. See the examples in the following table.

Package	Description
org.openmolgrid.util	Generic utility classes
org.openmolgrid.corba	Generic CORBA classes
org.openmolgrid.<projectname>.*	Specific Java-based project source

Table 1 Package and Class Name Examples

7.1.2.2. Classes

Classes should be tagged with the *final* modifier if the class will definitely not be sub-classed. This instructs the compiler that the methods in this class will not be used polymorphically (Unlike C++ all methods in Java are virtual methods by default).

7.1.2.3. Interfaces

- Interfaces or abstract class definitions should be created for all classes that may conceivably be extended in the future, or where data is shared across more than one class.
- Client-code should program to interfaces or abstract classes, not concrete classes. This aids re-use by reducing the dependency on class implementations.
- Constant data members such as application defaults should be defined in the interface or abstract class that will be available to a class,

```
public interface ISession
{
     public final static String APPLICATION_NAME = “Test Application”;
     public final static String DEFAULT_CONFIG_FILE = “test.conf”;
     public final static int MAX_NUMBER_OF_CONNECTIONS = 50;

     // any other method prototypes
}
```

7.1.2.4. Exceptions

See section 7.4.1, Error Handling.

7.1.2.5. Data Members

Encapsulation should be followed as closely as possible. That is, the class interface should be separated from the implementation to the extent that there is no dependency on the class implementation outside of the class. Therefore,

- Data members of classes that should not be used in subclasses should be given **private** access.
- Data members of classes that may be used in subclasses should be given **protected** access.
- Publicly accessible "accessor" methods should be provided for all private variables. Accessor methods for protected variables should have **protected** access. Clients must therefore use the accessor methods. See section 7.1.2.6 for required format of accessor methods.
- All data members that are read-only (constant) should be denoted with the **final** and **static** modifier. This will improve memory usage and performance. Constant values may be defined in common interface definitions (see section 7.1.2.3).
- Fields should be defined where declared if possible. This only applies if the field is not defined elsewhere, e.g. in a constructor.

7.1.2.6. Methods

- Methods should have a single, clearly defined purpose.
- The method body should be as short as possible given the purpose of the method.
- Similar method operations should be at the same level of abstraction.

`Access Level`

Method access control should be as strict as possible. For example,

- Methods should be marked **public** if they are to be used by any other class.
- Methods should be marked **private** if they are not to be used by any other class.
- Methods should only be marked with **default** access if they must only be available to classes in the same package.
- Methods should only be marked **protected** if they must be available to all sub-classes.

7.1.2.7. Constructors

- Many different constructors should be provided to allow various configurations to be set by the user. A default constructor with zero arguments though should always be provided. This is seen in the example for the next point.
- Constructors should be chained to increase code re-use. This means that a single constructor, the most configurable constructor will be used by all the other constructors. For example,

```
public class Diag implements IDiag
{
     private String sFilename_i = null;

     public Diag()
     {
       this(DEFAULT_FILENAME);
     }

     public Diag(String sFilename_p)
     {
       this.sFilename_i = sFilename_p;
     }
}
```

7.1.2.8. Imports

- Use the "global package import" (formally called the "type-import-on-demand") style of import instead of the "class import" style. This is easier-to-read and involves less maintenance. For example, to import the `java.util.HashMap`, `java.util.Calendar` and `java.util.Properties` classes,

```
    import java.util.*;
```

  instead of

```
    import java.util.Properties;
    import java.util.HashMap;
    import java.util.Calendar;
```

7.1.2.9. Core API

- The use of the `java.lang.String` class is restricted to immutable strings, that is, strings that will not have their value changed. Care should be taken when attempting to modify strings using `java.lang.String` of the method parameter (see below) and possible performance implications (see "Coding Issues" in section 7.1.8.1).
- The following idiom for String definition is preferred since it allows re-use of the same object for other strings with the same value. In the following example, the references `sTitle` and `sGreeting` both refer to the same object from the string pool, whereas `sAnotherGreeting` refers to a different object with the same value.

```
String sTitle = "Hello";                      // preferred
String sGreeting = "Hello";                   // preferred
String sAnotherGreeting = new String("Hello"); // not preferred
```

- The `java.lang.StringBuffer` class should be used for any mutable strings. It is more efficient than the `java.lang.String` class for manipulating strings values. Adequate StringBuffer sizes should be specified to reduce object re-creation.
- StringBuffer objects should be used for string-based output method parameters instead of String objects since all Java methods are passed-by-value,

```
public void appendName(StringBuffer sbName_p)
{
     String sName = this.getName();
     sbName_p.append(sName);
     return;
}

public static void main (String[] args)
{
     StringBuffer sbName = new StringBuffer("Name: ");
     this.appendName(sbName);
     System.out.println(sbName);
}
```

- Consider the Collection type that best suits your purpose. The following table indicates some attributes of common Collection structures.

Attributes	Suited To	Collection
Fixed ordering, integer-based indices.	Direct integer-based look-ups.	ArrayList - or the basic array.
No duplicates	Managing unique items.	Set, e.g. TreeSet, HashSet.
Balanced ordering, no indices.	Frequent searches over large data sets.	Tree, e.g. TreeMap

Fixed ordering, string-based indices.	Direct string-based look-ups.	Map, e.g. HashMap

Table 2 Collection Type Advantages

7.1.2.10. Packaging and Distribution

- JAR files should always be used to package collections of class files.
- Applications should be packaged into individual JAR files.
- Executable JAR files should be used (if possible) to package components. This reduces the need for start-up scripts.
- Configuration files used by any applications may be stored within the application JAR file if they are not to be edited manually. This will help suit remote retrieval of Java applications, for example using Sun's Java Web Start Framework [R3].

7.1.3. Naming Conventions

The following naming conventions are specified to help with naming classes, methods and variables.

7.1.3.1. Scope

There are four possible scopes:

- Class-scope. These are identified by the **static** modifier. Only one class member exists for all the object classes regardless of how many instances exist. These data members are can optionally be suffixed with "_c" or pre-fixed with "c_".
- Instance-scope. This is the default scope for class members. A data member exists for each class instance (i.e. object). These can optionally be suffixed with "_i" or pre-fixed with "i _".
- Parameter-scope. This is the scope for method parameters. Parameter-scope is valid for the lifetime of the method. These can optionally be suffixed with "_p" or pre-fixed with "_p".
- Local-scope. This is the scope for method members. These should not have any prefix or suffix to indicate scope.

7.1.3.2. Classes

Classes names should have first letter capitalised, use mixed case and be meaningful (nouns).

- Abstract classes should be prefixed with "Abstract", e.g. AbstractList, AbstractMap.

7.1.3.3. Interfaces

Interface names should also have first letter capitalised, use mixed case and be meaningful (nouns). Differently to class names, they should be prefixed with an "I". Normally they will be identical to the associated class name – except for the "I" prefix.

- Interfaces with only one operation may be appended with "able", e.g. Runnable, Cloneable.

7.1.3.4. Data Members

Java allows only two basic types of data member: primitive types and objects.

`Primitives`

The different kinds of primitive types can be distinguished with the following prefixes. Developers can use these at their discretion. Suitable naming of variables is considered more important.

Primitive Type	Prefix	Example
short	h	hMyShort

int	i	iMyInteger
long	l	lMyLong
float	f	fMyFloat
double	d	dMyDouble
byte	y	yMyByte
boolean	b	bMyBoolean

Table 3 Naming Conventions for Primitive Types

`Objects`

The Java language provides a large collection of standard abstract data structures. It is no longer necessary to write classes that handle re-sizeable arrays, lists, sets or trees as they are provided in the core language in thread-safe and unsynchronised forms. This framework creates considerable complexity with naming conventions. In fact, it is not possible or desirable to list short naming conventions for all the Java types.

The following table lists the Hungarian-style prefixes relevant for popular data structures classes. The conventions are limited to wrapper types for Java's primitives. All other reference types can be named using the general object convention.

Data Type	Prefix	Example
java.lang.Short	h	hMyShort
java.lang.Integer	i	iMyInteger
java.lang.Long	l	lMyLong
java.lang.Float	f	fMyFloat
java.lang.Double	d	dMyDouble
java.lang.Byte	y	yMyByte
java.lang.Boolean	b	bMyBoolean
java.lang.String	sz or s	szMyString, sMyString
java.lang.StringBuffer	sb	sbMyStringBuffer
java.lang.Object	o	oMyObject

Table 4 Naming Conventions for Reference Types

`Constants`

Constant values should be written in uppercase and delimited with underscores. As all constants will be class-scoped variables there is no need to explicitly scope a constant variable.

- Constant values should be defined in the interface or abstract class definition that is associated with the Interface.
- Default value constants should be prefixed with "DEFAULT_", e.g. DEFAULT_DIRECTORY.
- Range defaults should be prefixed with "MIN_" and "MAX_", e.g. MIN_SIZE and MAX_SIZE.

7.1.3.5. Methods

Methods should be given clear meaningful names (verbs). The following specific types of methods are given special names following the Sun Bean format:

- Non-boolean "getter" accessor methods should have the following prototype:
  ```
  public <return type> getXXX()
  ```
- Boolean "getter" accessor methods should have the following prototype:
  ```
  public boolean isXXX()
  ```
- All "setter" accessor methods have the following prototype:
  ```
  public void setXXX(<value>)
  ```
- Methods that return a value should be named "getXXX".
- Methods that return a list of values should be named "getAllXXX".
- Methods that return a boolean value should be named "isXXX".
- Methods that create objects should be named "createXXX".
- Methods that update should be named "updateXXX".
- Methods that add objects should be named "addXXX".
- Methods that delete should be named "removeXXX".
- Methods that convert should be named "toXXX", e.g. toString().
- Methods that do validation checks should be named "checkXXX".

7.1.3.6. Exceptions

- All exceptions must be named "<Name>Exception". The name must be meaningful even if it is fairly lengthy. For example *DatabaseUnavailableException*. Abbreviations should not be necessary.
- All exception handler variables within the same try/catch block should be named "ex". Nested handlers should increment the index of this variable name,

```
try
{
    try
    {
          Integer iAge = Integer.valueOf("21 and a bit");
    }
    catch (NumberFormatException ex2)
    {
          System.err.println("You didn't enter your age, did you?");
          ex2.printStackTrace();
          throw ex2;
    }
}
catch (Exception ex)
{
    ex.printStackTrace();
    throw ex;
}
```

7.1.4. Error Handling

There are two types of errors in Java: fatal and recoverable. These are termed "Errors" and "Exceptions" respectively. The JVM performs many roles for a Java class, one of which is its handling of recoverable errors. The JVM detects recoverable errors and generates the appropriate exception. The user’s class can then handle this if appropriate.

Problems that the JVM cannot handle ("Errors") are typically hardware related problems, for example, lack of memory. These are unforeseen and are not appropriate to catch in user code – as they are not catchable.

7.1.4.1. General

Exceptions can be caught in user's code. Exceptions can simplify "error" handling by moving all post-operation checks into well-grouped exception handlers. The following are recommended:

- A single try/catch block should surround all methods that perform actions that may throw exceptions. This is the "main" try/catch block.
- The main try/catch block should define exception handlers for all the exceptions needing to be uniquely identified. A superclass exception handler if appropriate, may process subclassed exceptions.
- Any resources that always need freed should be placed in the finally block for the method's main try/catch. The following example shows some code retrieving result set from the database. In this case, the statement and connection objects are freed regardless of the success or failure of the processing,

```
Connection oConnection = null;
PreparedStatement oStmt = null;
try
{

     Class.forName(sJDBCDriverName_i);
     oConnection = DriverManager.getConnection(
          “sun.jdbc.odbc.JdbcOdbcDriver”,
          “jdbc:odbc:workflow”, “sa”, “”);
     oStmt.prepareStatement(“SELECT * FROM TEST_TABLE”);
     ResultSet oTeams = oStmt.executeQuery();
     // process the result set values
}
catch (ClassNotFoundException ex)
{
     oDiag_i.diag(“Failed to find class for JDBC Driver”);
     throw ex;
}
catch (SQLException ex)
{
     throw ex;
}
finally
{
     oStmt = null;
     oConnection = null;
}
```

1. Exceptions can be propagated back by adding a throws clause to the method prototype if it is not suitable to handle them.
2. Nested try/catch blocks should be avoided if possible since they clutter the code. They are useful when dealing specifically with a problem that can be ignored. The following example attempts to create a class dynamically. If this fails, it creates a default class object and continues processing.

```
    3.     try
{
    Class oDynamicClass = null;
    try
    {
          oDynamicClass = Class.forName(sNewClass);
    }
```

```
        catch (ClassNotFoundException ex)
        {
            oDynamicClass = Class.forName(DEFAULT_CLASS);
        }

        // some more work using the dynamically-created class
    }
    catch (Exception ex)
    {
        ex.printStackTrace();
        throw ex;
    }
```

- The highest-level client should always provide a generic exception handler.
- Any superclass exception handlers should always be listed after its subclass exception handlers.
- Exception handlers are preferred to code checks. However, it is appropriate to use simple checks *before* an operation in some cases to improve performance. This is often useful for checking array size before accessing it, objects are not null, etc.

7.1.4.2. Checked and Unchecked Exceptions

There are two types of exceptions: checked exceptions and unchecked exceptions. Checked exceptions are checked at compile-time by the compiler to ensure that any exceptions a method throws are handled by the calling code. The compiler allows unchecked exceptions (or "Runtime exceptions") however, to be ignored by the calling code. It is recommended that,

- Checked exceptions should be used in almost all circumstances.
- Unchecked exceptions should be used when a problem can be disregarded. Clients should disregard a problem if for example; it is not expected to occur except for a *programming* error. This is the case when calling methods on null values, exceeding array indices, etc. as it is expected the programmer has not made these mistakes.

7.1.5. Memory Management

Java's JVM memory management ensures there are no memory leaks in a pure Java program (it does not ensure there are none in native methods). However, it does not promise any more than this. For example, it does not guarantee that memory will be freed at any point except when the JVM terminates. Therefore, the JVM should not be relied on to free memory at *any* point in a program's execution.

Relying on `finalize()` for example is not recommended. It does not guarantee when an unreachable object is garbage-collected. It doesn't even guarantee it will ever be called. It merely guarantees there will be no memory leaks (when the JVM is terminated).

The following measures are recommended:

- Provide explicit "exit" methods to free all resources. This includes scarce resources, e.g. file handles, sockets, and non-scarce resources, i.e. ordinary objects.
- Free resources explicitly by calling the explicit exit methods.
- Free resources automatically by calling the exit methods from finally blocks.
- Set all variables to null if they are no longer being used.
- Chain finalizers within sub-classes to ensure all base classes have their `finalizer()` method called. Java does not chain superclass finalizers by default.
- Use the most suitable reference type to limit memory use. In addition to hard references there are now soft, weak and phantom references. These new references allow the Garbage Collector

to use different policies to determine whether they should be freed – other than the default "is this object reachable" policy.

7.1.6. Threading

Support for threading in Java is included in the core API. Java threading therefore benefits from the advantages of the Java language, i.e. portable, well defined. In addition to this it is mainstream Java: every Java application whether explicitly or implicitly uses the threading-model.

The following measures are recommended:

- Implement the `java.lang.Runnable` class instead of extending the Thread class if your class is already a subclass.
- The `java.util.Vector` and `java.util.Hashtable` container classes are thread-safe. They should therefore be considered carefully before use in a single-threaded environment since the performance will be unnecessarily degraded. Consider Collections instead.
- No Collections are thread-safe by default. Use thread-safe views of collections to get the initial thread-safe view of a Collection. Iteration through this synchronised collection will still need to be explicitly synchronised,

```
List oUserList = Collections.sychronizedList(new ArrayList());
```

- Use finest granularity of lock possible, i.e. do not use object lock (e.g. locking a method) when other methods may be called legitimately without the need for thread-safety. Use a variable-lock if possible.

  The first example demonstrates the preferred approach. The second example shows a coarse level of granularity (i.e. lock at the object instance level) for comparison. The first example has two main advantages over the second approach: the lock is only held for the stack operation, not the `diag()` calls that do not require synchronisation; and secondly, the threads trying to access the `incCount()` method are trying to gain access to a different monitor than the threads trying to access the `push()` method. Neither of these are the case in the second example.

```
Object oCounterLock_i = new Object();
int iCounter_i = 0;
ArrayList oStack_i = new ArrayList(100);
public void push(Integer iItem_p)
{
      oDiag_i.diag("About to add item (" + iItem_p + ")");
      synchronized (this.oStack_i)
      {
            oStack_i.add (iItem_p);
      }
      this.incCount();
      oDiag_i.diag("Successfully added item (" + iItem_p + ")");
}

private void incCount()
{
      synchronised (this.oCounterLock_i)
      {
            ++this.iCounter_i;
      }
}
```

instead of,

```
ArrayList oStack_i = new ArrayList(100);
public sychronized void push(Integer iItem_p)
{
      oDiag_i.diag("About to add item (" + iItem_p + ")");
```

```
                oStack_i.add (iItem_p);
                this.incCount();
                oDiag_i.diag("Successfully added item (" + iItem_p + ")");
        }

        private synchronized void incCount()
        {
                ++this.iCounter_i;
        }
```

7.1.7. Documentation

Java allows for two types of comments: plain comments and Javadoc comments. Plain comments are either `/* Multi Line Comment */` or `// Single line comment`. These are not affected by Javadoc comments (/** ...*/) which are designed for API specifications.

It follows then that all comments that the developer wishes to be included in the API specification should be Javadoc comments and the rest as plain comments. This means for instance, that all method body comments should be plain comments, while all class, method, field, package-heading comments should be Javadoc comments.

7.1.7.1. Plain Comments

Plain comments should follow the "Multi Line Comment" defined above

7.1.7.2. Javadoc Comments

Javadoc should be used for API specification. It will describe packages, classes, interfaces, fields, methods and exceptions. In effect it can be used as a replacement for a handcrafted Specification document. Public, private, protected and package methods should be included in this Javadoc-generated API specification. Design documents should be written to explain the workings of the class for these public, protected, private and package members. The Javadoc Specification can be created during the design to formally specify the interface to the classes by defining methods with an empty method body.

The Javadoc mechanism in Java 2 can be used with any pluggable documentation generator (or "doclet"). The standard Sun doclet is the HTML doclet. It produces HTML output and can therefore have HTML directly embedded into its comments. Other vendor's doclets are available for generation to other formats, e.g. MIF, PDF.

See Ref. [R4] for more information or Ref. [R5] for listings of alternative doclets.

`Sun HTML Doclet Usage`

- Comment all packages
- Comment all classes – explaining class functions
- Use @author for author names. Multiple @authors are allowed.
- Use @version with a Source Control version keyword or suitable release number.
- Comment all data members
- Comment class fields as succinct one-line comments.
- Comment all methods.
- Comment parameters, return values and exceptions.
- Package overview pages are helpful.

7.1.8. Performance Considerations

7.1.8.1. Coding Issues

- Do not re-allocate objects within loops. This is often seen in Java code, but is inefficient. The single object should be instantiated outside the loop and initialised if necessary on each iteration,

```
Iterator itPhoneNumbers = mpPhoneNumbers.iterator();
while (itPhoneNumbers.hasNext())
{
      // some work
}
```

- Synchronisation is a lot slower than unsynchronised code. It is said to be 7 times slower using the Classic JVM. Therefore, consider not using synchronised types like `java.util.Vector` and `java.util.Hashtable` if thread-safety isn't required.
- Use buffered I/O. Caching input and output to a buffer is considerably better for performance than writing each object to file,

```
String sData = "some data";
BufferWriter oBufOut = new BufferWriter(new FileWriter ("file.txt"));
PrintWriter oOut = new PrintWriter(oBufOut);
oOut.println(sData);
```

- Promote in-line methods/classes. Compilers often in-line "final" methods that only have two-three lines of code. If all methods in a class are final, declare the class as final. Note that methods by default are virtual methods, and that synchronised methods cannot be in-lined.
- Compilers often store the earliest declared variables in a method in registers. Therefore declare the most frequently used variables first in a method.
- Primitive based keys for hashes are considerably more efficient than string based keys. Create a Key class that maps string keys onto primitive values.
- Simple checks are much faster than catching an equivalent exception. These may be suitable for pre-operation checks, for example to check if an array has sufficient size before accessing an index.

7.1.8.2. Database Access Optimisation

- The `java.util.PreparedStatement` class uses pre-parsed SQL. This should be used instead of `java.util.Statement`.
- Replace simple (interpreted) SQL statements with precompiled alternative. Examples of this include using precompiled vendor-specific SQL as Java code, e.g. SQLJ for Oracle, or calling vendor-specific Stored Procedures using `java.lang.CallableStatement`.
- Cache the `java.util.CallableStatement` object within the object instead of re-creating for every method call.
- Turn AutoCommit off.
- JDBC resources such as ResultSets, Statements, etc. should be released as soon as they are no longer required.
- Investigate and consider alternative JDBC drivers that can offer improved performance and support for many more RDBMS that the default Sun driver.

7.1.8.3. JVM Optimisation

Once your code is compiled, the choice of Java Virtual Machine can greatly impact the performance of your application. There are two types of optimisers used in JVMs:

- Static Optimiser, e.g. Just-in-time compiler, which is standard in Sun's Java 1.2 Development Kit. This compiles the class immediately after the JVM loads the class at run-time. Native-code compilers are also becoming more popular, although some tests seem to show that the native-code compilers' performance does seem to be second to the existing (fairly mature) bytecode compilers.
- Dynamic Optimiser, e.g. Java HotSpot adaptive optimiser. This compiles only those sections of code that are used multiple times (the "hotspots"). It therefore saves time by not having to compile everything before it is executed. It also collates run-time information on the statistics of the code, allowing informed decisions to be made on how best to optimise it. In addition to this it also claims to virtually eradicate synchronisation overheads.

These optimisers are provided as alternative JVMs. So to use them requires no change to existing code, just an installation of the new JVM dlls.

There is no formula that will determine which optimiser should be used. It is best to speed-test both types of optimiser with various vendor implementations of JVM.

7.2. CODE REVIEWS

Code reviews have two primary goals:

- To identify issues regarding poor programming practice or language style (non adherence to naming rules for instance);
- To identify functionally issues or closeness of implementation to design.

These goals are intended to identify problems that will, when fixed, lead to better code quality in terms of robustness, maintainability and extensibility.

Code reviewing should be done in small steps, internally to each WP. Code reviews will be done both by the WP Manager and the QE; typically the QE will cover the language angle, while the WPM concentrates on functionality and design.

The procedure for code reviews is very like the document review procedure.

7.3. TESTING

Testing is the exposure of a system to trial input to see if it produces the expected output. It cannot guarantee correctness, but can increase confidence that the system will perform without failure. Testing can be considered as the process of detecting the presence of faults, whereas debugging is the process of locating these faults and correcting them as appropriate.

The aim of testing is two-fold:

- To detect errors in the system as early as possible - when they are least expensive to correct.
- To judge if the program is usable in practice. But testing can only demonstrate the presence of errors, not the absence of them. However, since exhaustive testing is not feasible, tests must be carefully designed to capture the maximum number of errors in the most cost effective way. Well thought out testing should lead to confidence in the product.

Testing, far from an add-on, encompasses a wide range of activities and is an integral part of the software process throughout its life cycle.

For each stage of the testing process, it will be convenient that the WP responsible of the test had internally performed the tests before delivery.

7.3.1. Testing Phases

There are a number of phases associated with testing. These are outlined below.

- Unit Testing – the testing of individual components
- Integration Testing – the testing of the interfaces between components
- System Testing – testing according to the expected usage of the system
- Acceptance Testing – testing by real system users

Each of these phases can be performed a number of times and may be halted due to the discovery of software defects. Once the defect is corrected, testing will resume at the lowest level of testing, i.e. unit testing

7.3.2. Unit Testing

A unit can be considered to be a method, class, or package. This phase performs testing at the lowest level. The person who wrote a particular code unit performs unit testing on that unit. Individual unit tests are carried out in isolation from other units. Unit tests are used to verify that the detailed design has been correctly implemented.

Many developers do not see the benefit of unit testing and argue that it is too time consuming and only proves that the code does what it is supposed to do. They have the attitude that integration tests will catch the software defects. However unit testing has many benefits. Defects are discovered early and do not propagate into subsequent phases where detection becomes more difficult. Unit testing simplifies the integration-testing phase and results in code that is easier to maintain.

Units are tested in isolation from the units that call them and in isolation from any units called. This means that units can be tested in any sequence, as no test is dependant on any other unit to be tested. Tests that must be carried out can be derived from the detailed design. Ideally tests should be designed before a unit is coded, however given the rapid prototyping approach being adopted during development, this may not always be the case.

Unit testing ensures that the individual modules or units operate correctly. This is done independently of other system components and usually performed by each coding team. Each unit is tested by the person responsible for its coding. The aim of the unitary test of the unit is to check that it works as designed.

The unitary tests of others units linked with the unitary tested units can be executed, where the behaviour of more and more complex sub-systems is verified.

Many of the errors that arise here do so because of problems with the interface between these modules.

Components are tested using a bottom-up testing strategy. This method allows integration of components during the unit tests phase.

A specific care should be devoted to the unitary tests.

Before the integration phase, the WP Manager will check their completeness in a special review meeting.

<u>Unit test specification:</u>

Each component is unitary tested.

<u>Unit Tests design</u>

Unit Tests design consists to define for each component:

- Tested operation(s)
- Input data
- Setting-up procedure
- Resources needed for tests setting up
- Expected result

Unit Tests preparation

This activity consists of:

- Setting-up the test environment
- Preparing the input data

Unit Tests setting-up

This activity consists of:

- Executing unit test of the component under test
- Comparing results with expected results
- Detecting anomalies
- Identifying errors corresponding to anomalies
- Correcting errors
- Replaying tests after correction
- Adding additional tests possibly to complete test coverage capacity
- Registering the tests results
- Setting-up configuration management for tests elements

Tests are registered in a unit test dairy within each WP. This dairy includes the following information:

- Component tested
- Operation(s) tested
- Input data
- Expected results
- Obtained results

Within OpenMolGRID the JUnit framework (http://junit.sourceforge.net/) will be used to test individual units. For small units, diagnostic checks are sufficient for unit testing purposes. Unit testing follows the bottom-up strategy.

7.3.3. Integration Testing

Integration tests are designed to test that the interfaces between components function as expected. Tests are performed on sub-components of several different units. Integration tests will be performed within each workpackage and between workpackages. On small projects the programmers who developed the software carry out integration tests.

If certain units are not yet available for testing, then stubs should be written to "simulate" their function, thus allowing testing to continue without the need to wait for other components. Inputs and outputs may be hard coded in this case, for example by reading and writing values from a file.

Test cases should be provided to test sub-systems or the system in his whole. These tests are performed by each WP team independently of other sub-systems.

The approach for integration tests is similar as unit test approach as described before.

Within OpenMolGRID formal integration procedures will be written before the scheduled integration tests, envisaged to be two weeks before prototypes are expected to be available.

7.3.4. System Testing

System testing is testing carried out on the system as a whole. It is normally carried out by the system testers or individuals not involved in the code development cycle. As system testers are not always end users, testing may not actually reflect actual system usage. Within OpenMolGRID system testers may also be end users.

System tests ensure that the software complies with the original requirements and tests hardware, software and interfaces. Therefore they are often referred to as validation tests. Tests often involve recovery tests, performance tests, stress tests and security tests.

A formal testing procedure for system testing will be developed two months before the final system is ready to be released.

7.3.5. Acceptance Tests

These are carried out in accordance with the actual use of the software by the end users of the system. These tests are considered ad-hoc and have no formal procedure.

7.3.6. Fixing Software Defects

As fixing a defect or bug can have knock-on effects to the remainder of the system regression tests must be carried out, meaning that tests must be performed again starting from the unit-testing phase.

7.3.7. Bug Lifecycle

Throughout the testing process, several incidents or bugs will arise. It is important that such incidents are tracked throughout the project. Bug-tracking will be done by using anomaly reporting forms mantained in the project document repository (BSCW), for details see Section 9.3.

7.4. QUALITY CONTROL

The aim of quality control is to enable and control the development and production of all OpenMolGRID software (code and documentation) compliant with the characteristics and the requirements defined for the project.

WPM's, the PC and the QE verify software before their delivery to the EC.

The TC verifies whether the organisation is compliant with the plan, and the reported progress status of the project.

The WPM's are responsible for the control of technical aspects and conformity to requirements while the QE is concerned with the quality aspects (coherence).

Different approaches can be employed by the persons involved in such as:

- participating into review,
- organising inspections: inspection is a software or documentation checking quality action which by examination observation and measurement evaluates conformity of a standard to pre-defined quality clauses,
- if necessary, organising audits: an audit is a methodical examination of a situation relating to a product, a process, an organisation, performed in collaboration with the parties involved in order to check the conformity of the situation to pre-defined standards and the extent of alignment of the standards with the desired goals.

Most of the actions executed by those involved in the quality control of the products are tracked. The results of the test or inspection are written on a specific dialog form, which is returned to the person responsible for the product. Then the author of the product has to respond to the dialog form as follows: either the remarks are not justified, or the remarks must be addressed and the product corrected. The dialog form is returned to the author of the inspection and a new review of the product is performed by the Quality Engineer, but only on those elements identified by the first inspection. The Quality Engineer controls the dialog form and ensures that all remarks have been processed. The Quality Engineer then stores the dialog form.

8. SOFTWARE CONFIGURATION MANAGEMENT

8.1. SOURCE CODE MANAGEMENT

Each work package will be responsible for managing the source code associated with each of its products. The source code from different work packages resides in different CVS repositories at the WPM's site.

8.2. CENTRAL REPOSITORY

All source code for the OpenMolGRID project will reside in the project document repository (BSCW). This repository will provide read-only access to the code. All code used for the tests, must be available from the central repository.

9. ANOMALY MANAGEMENT

This chapter describes the procedure to identify, manage and resolve anomalies that may occur. This procedure will be applied from the integration phase onward.

9.1. ANOMALY DEFINITION

An anomaly is generally defined as an apparent error in any item that does not conform to the specified requirements or specifications.

The anomalies can be classified in major or minor:

- Major Anomaly: is an anomaly that makes the system virtually unusable or compromises safety.
- Minor Anomaly: is an anomaly that affects functional use in a small number of cases or circumstances.

9.2. ANOMALY RESPONSIBILITIES

The person(s) discovering a bug is responsible for the bug reporting, control and dispatching to the TC using the bug report form. WPMs are responsible for resolving bugs. The TC informs the PC of all major bugs.

The escalation procedure that will be used is that the WPM will be assigned all bugs reported against a product produced by that work package.

The WPM may reevaluate the seriousness of the bug, and whether it is appropriately assigned. He/she may reassign the bug to another work package if that is appropriate.

If the WPM cannot immediately resolve the bug, then he/she will assign it to an appropriate person within that work package. The WP representative is responsible for ensuring that the bug is evaluated in a timely fashion.

The TC is responsible for ensuring that the WPM's process bug reports in a timely fashion and to act as an intermediary should a bug involve the products of more than one work package.

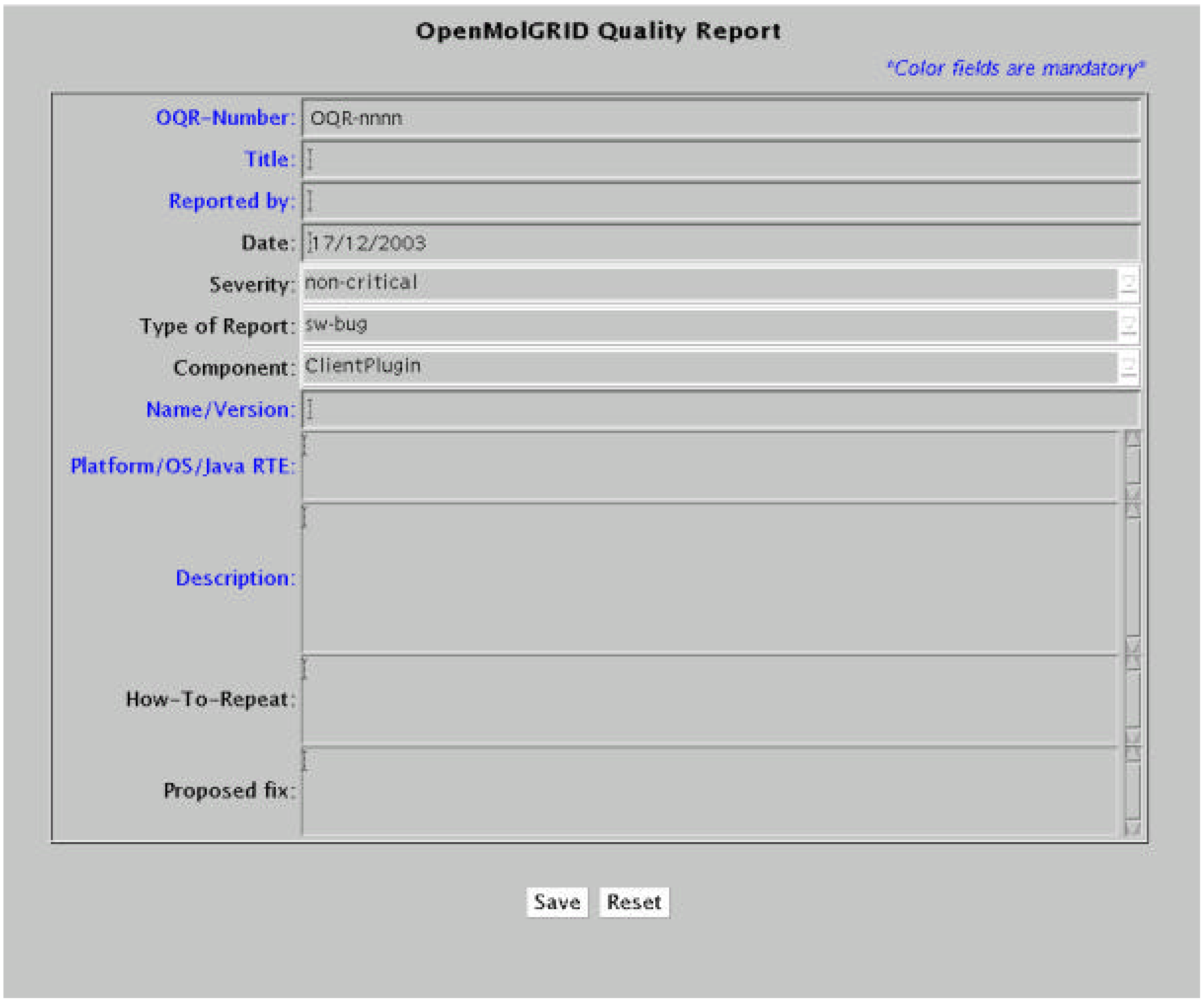
9.3. OVERVIEW OF THE BUGTRACKING PROCEDURE

Bugtracking is carried out using report forms on the project documentation server (BSCW). The ***QualityReports*** folder on the BSCW server contains a *QualityReport_template* file which includes a Java applet that gathers information from the user. The applet generates an ASCII file containing that information, assigns a unique report number to it and saves it to the ***open*** subfolder. Through the daily report generated by the BSCW system every OpenMolGRID participant is informed of the new bug report. A developer responsible for the bug checks the bug report, edits it and moves it to the ***suspended*** subfolder, until the problem has been fixed. When the issue is resolved, the report is moved to the ***closed*** subfolder. The report contains the original bug report by the user, comments by the developer(s) and information about bug fixing, in effect providing a full bug history.

9.4. DESCRIPTION OF THE REPORT FORM

The report form is utilised by users to create a new bug report. It is reached by viewing the document

QualityReport_template in the ***QualityReports*** folder on BSCW. The user needs to be logged on to the BSCW system.



OQR-Number:	OQR-nnnn
Title:	
Reported by:	
Date:	17/12/2003
Severity:	non-critical
Type of Report:	sw-bug
Component:	ClientPlugin
Name/Version:	
Platform/OS/Java RTE:	
Description:	
How-To-Repeat:	
Proposed fix:	

Several fields are set automatically by the system:

- **OQR number**, the unique OpenMolGRID quality report number, it is generated when the report is saved
- **Reported by**, gives name and organization of the reporting user, as determined by the BSCW system
- **Date**, the current date

The remaining fields have to be set by the user:

- **Title**, a one line description/title for the bug
- **Severity**, the severity of the problem, one of the following (with approximate descriptions)
  - *critical:* the software does not work
  - *serious:* the functionality of the software is severely impaired
  - *non-critical:* the software works, but not as documented or expected
- **Type of report**, specifies the report type, one of
  - *Sw-bug*: software bug
  - *Doc-bug*: Documentation bug, typing errors, missing filenames, etc.

  - *Setup-support*: Problem with installion of software
  - *Use-support*: Problem with using/handling the software
  - *Change-request*: Suggestion for modifying the software
  - *Extension-request*: Suggestion for extending the software
  - *Other*: other problem not covered by the previous categories
- **Component**, specifies the OpenMolGRID software component the report refers to. Currently, these are
  - *Client-Plugin*: Problem with client software
  - *Application*: Problem with server side software
  - *Admin*: Problems relating to system administration (tools, setup, user database, ...)
  - *Other*: Other problems
- **Name/Version**, the name and version of the component
- **Platform/OS/Java**, describes the local environment and operating system
- **Description**. gives a detailed problem descriptions
- **How-to-repeat**, should give instructions on how to reproduce the problem
- **Proposed-fix**, is the suggestion for fixing the problem

The **Save** button stores the report in the open subfolder on BSCW. The **Reset** button clears the user-filled fields. The form can be left without saving a report by using the *back* function of the browser.

9.5. DESCRIPTION OF BUG REPORT EDIT FORM

Developers can view and edit the bug reports using the *edit_OQR* applet in the ***QualityReports*** folder on BSCW.

The applet allows to select quality reports from the ***open***, ***suspended***, and ***closed*** subfolders, edit them and store them to one of the subfolders.

The generated report file contains fields for further processing of the report, which the Edit applet shows and allows to edit. These are

1. **Assigned to**, the name and organization of the developer responsible for taking care of the report
2. **Priority**, the priority the assigned person gives to the report, values are *low*, *medium*, *high*.
3. **Assigned date**, the date the respoonsible person is assigned
4. **Response/Solution(signed)**, contains all relevant information for solving the problem. In case the information comes from another person than the assigned, the text should be signed with the persons name and organization.
5. **Additional discussion**, gives the possibility to add comments relating to the problem.
6. The **Save** button stores the edited report to the selected subfolder.

10. STANDARDS AND TOOLS

This chapter will describe standards and tools used for the development of OpenMolGRID.

10.1. DOCUMENT MANAGEMENT TOOLS

In order to improve workflow activity, it is recommended to standardise tools. The following tools will be used:

- Word processing: MS Word.
- Spreadsheet: MS Excel.
- Slides presentation: MS PowerPoint.
- Project Planning: MS Project.
- Document Management tools: BSCW (see http://www.orbiteam.com)

The following formats will be used for exchanging documents:

- For documents type PPR (Project Progress Report): Word, Excel, PowerPoint
- PDF (need ACROBAT Writer to produce PDF format and ACROBAT Reader for reading)

All official documents must be available in PDF format.

10.2. SOURCE MANAGEMENT AND CENTRAL REPOSITORY

The raw source code management is done on a per Workpackage basis. A read-only repository is established on the project document repository (BSCW).

10.3. ANOMALY MANAGEMENT TOOLS

Bug-tracking will be done by using bug reporting forms mantained on the project document repository (BSCW).

11. ARCHIVING AND STORING

WP7 is in charge of archiving the Project Folder documents described in §5.6 and the deliverable documents. WP1 is in charge of archiving the project production (code, documentation, specification, etc.).

The following table summarises the archiving methodology:

Product type or document	**Responsible of product**	**Responsible of archiving**	**Archiving location**	**Archiving duration**
Project Folder documents described in §5.6. and the deliverables documents	WPM	WP7	UT	5 years
Production project (*Code, Documentation, Specifications,etc*)	WPM	WP7	UT	5 years

Each WPM is responsible for the backup of its own WP environment.

12. DELIVERY PROCEDURE

All deliverables will be provided in English.

12.1. DELIVERIES TO THE EUROPEAN COMMISSION

Before its delivery to the European Commission, each deliverable is reviewed by the TC and the QE as defined in §6.7 in order to assess that it is consistent with the project objectives (technical, quality and cost objectives).

The OpenMolGRID products will be delivered by the PC to the European Commission with one bound and one unbound paper copy and an electronic copy. A delivery note is sent with the deliverable.

13. ANNEX 1: DOCUMENT TYPES

The following table gives a first version of document types. The QE manages document types.

Document types	**Acronyms**
Deliverable Documents	Dxx.x (see §4.2)
For non deliverable documents:	
Technical Document	TED
Internal Document Review	IDR
Anomaly Report	ANR
Presentation	PRE
Minutes	MIN
Template	TEM

14. ANNEX 2: DOCUMENT TEMPLATES

The following table gives a first version of templates. The QE manages the templates.

Templates	File name	Comment
Deliverable Document	OpenMolGRID-7-TEM-0901-0-0-DeliverableTemplate.doc	Template for deliverable documents, such Quality Plan, Work Plan, etc..
Technical Document	OpenMolGRID-7-TEM-0902-0-0-SimpleDocumentTemplate.doc	Template for technical documents
Internal Document Review	OpenMolGRID-7-TEM-0903-0-0-InternalDocReviewTemplate.doc	Template for internal document reviews
Anomaly Report	OpenMolGRID-7-TEM-0904-0-0-AnomalyReportTemplate.doc	Template for Anomaly Reports
Presentation	OpenMolGRID-7-TEM-0907-0-0-Presentation.ppt	Template for point project presentations
Minutes	OpenMolGRID-7-TEM-0903-0-0-MeetingMinutesTemplate.doc	Template for meeting minutes.
Delivery Note	OpenMolGRID-7-TEM-0905-0-0-DeliveryNote.doc	Template for delivery notes to the EC.
Quarterly Project Report	OpenMolGRID-7-TEM-0908-0-0-EC-QuarterlyReport.doc	Template for quarterly project reports
WP Quarterly Project Report	OpenMolGRID-7-TEM-0909-0-0-WP-QuarterlyReport.doc	Template for WP quarterly project reports