

# High-Level Requirements for a Future Astronomical Software System

OPTICON Network 3.6\*  
US National Virtual Observatory<sup>†</sup>

Version 1.00 - 2008-02-29

---

\*OPTICON is funded by the European Commission under Contract no. RII3-CT-2004-001566

<sup>†</sup>The US National Virtual Observatory (NVO) is funded by the National Science Foundation under cooperative agreement AST0122449

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Purpose . . . . .	3
1.2	Reference documents . . . . .	4
1.3	Definition of terms . . . . .	4
1.4	Acronyms . . . . .	5
<b>2</b>	<b>High-Level Requirements</b>	<b>6</b>
2.1	Installation and running . . . . .	7
2.1.1	Installation . . . . .	7
2.1.2	Patches and updates . . . . .	9
2.1.3	Documentation . . . . .	11
2.1.4	License issues . . . . .	12
2.1.5	Deployment . . . . .	13
2.1.6	Support of coding languages . . . . .	14
2.1.7	Revision control . . . . .	15
2.1.8	Logging and errors . . . . .	17
2.1.9	Test and validation . . . . .	19
2.1.10	Internationalization . . . . .	20
2.1.11	Support of standards . . . . .	21
2.1.12	Access to external packages . . . . .	22
2.2	Definition of scripting and execution . . . . .	24
2.2.1	Execution . . . . .	24
2.2.2	Task parameter . . . . .	25
2.2.3	Scripting . . . . .	28
2.2.4	Scalability . . . . .	29
2.2.5	Graphical User Interfaces . . . . .	31
2.3	Access and handling of data . . . . .	33
2.3.1	Meta-data . . . . .	33
2.3.2	Astronomical coordinates . . . . .	34
2.3.3	Grouping of data . . . . .	35
2.3.4	Manipulation of data . . . . .	36
2.3.5	Data items . . . . .	37
2.3.6	Uncertainties . . . . .	37
2.3.7	Data management services . . . . .	38
2.3.8	Reference data . . . . .	39
2.3.9	Shared access to data . . . . .	40
2.4	Applications . . . . .	42
2.4.1	Access to code . . . . .	42
2.4.2	New applications . . . . .	42
<b>A</b>	<b>OPTICON Network 3.6</b>	<b>44</b>

# 1 Introduction

Most of the general purpose, data processing systems currently used in astronomy (e.g. IRAF, AIPS, MIDAS) were designed and developed in the 1980's. Although still very useful, they do not fully satisfy modern demands in areas such as modular design, open interfaces, and scalability. Limited support for many of these legacy systems also suggests that one must consider a future software environment for astronomical data processing and analysis. This task was taken up by the OPTICON Network 3.6, which started in 2004, with the aim to define high-level requirements, architectural concepts and system interfaces for such an environment. Shortly thereafter the US National Virtual Observatory joined the effort, with the goal of coordinating OPTICON planning and development with related efforts in the US, while ensuring that any such future astronomical environment be fully integrated with the virtual observatory.

## 1.1 Purpose

This document lists high-level requirements for a future software system for processing and analyzing astronomical data. System is here used to cover both the environment which provides an infrastructure and applications to be executed within the environment. Although the majority of requirements are associated to the environment also generic requirements related to application have been included.

The initial version was the result of discussions in the OPTICON Network 3.6 (see Appendix A) during the period 2004-2006. It is presented with the hope that it will help projects which intend to implement such a software system by listing important features to be considered. Although any specific project will have its own explicit requirements, many high-level requirements are common to systems for processing and analyzing astronomical data. For this reason, the current set of requirements is generic in nature and needs to be detailed by an actual project. It is expected that they will form the basis for an architectural concept and interface specifications.

By presenting these generic high-level requirements, we hope that different projects will share more objectives and thereby produce more compatible environments. Inter-operability between such environments starts by having similar objectives but is ultimately depending on adoption of compatible architectural concepts and interfaces. These aspects are also being considered by the OPTICON Network 3.6.

The underlying vision for these requirements is an open, modular system which can evolve in time and be adapted to future needs. To avoid that the current document becomes a pure academic exercise, a real and useful implementation must be based on it. As a minimum, a reference implementation must be established with specific choices for architecture and interfaces. It is essential that such an implementation solves a real-world, astronomical use case (e.g. reduction or analysis of a specific type of data) and thereby demonstrates its usefulness for the astronomical community.

## 1.2 Reference documents

This section lists documents referred to in these requirements.

**AIPS:** see <http://www.aoc.nrao.edu/aips/>

**Doxygen:** D. van Heesch, see <http://www.stack.nl/~dimitri/doxygen/>

**FITS:** see <http://fits.gsfc.nasa.gov/>

**IRAF:** see <http://iraf.noao.edu/>

**IEEE-754:** “American National Standard - IEEE Standard for Binary Floating Point Arithmetic” ANSI/IEEE 754-1985 (New York: American National Standards Institute, Inc.)

**MIDAS:** see <http://www.eso.org/sci/data-processing/software/esomidas>

**OASIS:** see <http://www.oasis-open.org/>

**PARM:** Tody, D. 2005: “Notes on a Parameter Model for Data Analysis” see <http://archive.eso.org/opticon/twiki/bin/view/Main/ParameterModel>

**WCS-I:** Greisen & Calabretta, 2002, *Astronomy & Astrophysics*, 395, 1061

**WCS-II:** Calabretta & Greisen, 2002, *Astronomy & Astrophysics*, 395, 1077

**WCS-III:** Greisen et al., 2006, *Astronomy & Astrophysics*, 446, 747

**XHTML:** see <http://www.w3.org/TR/xhtml1/>

## 1.3 Definition of terms

The following terms have been used in the text with the meaning listed below:

**client:** A module which requests services from another element.

**component:** A software module which provides a specific service. Normally, it implements a set of methods which performs the required functions such as a scientific computation. It interacts with a container through which it exchanges information with the environment (e.g. input parameters and results).

**container:** A software module which acts as a connection between a framework and a component. A container can accommodate one or more different components.

**environment:** The software infrastructure which enables the user to perform data processing tasks.

**framework:** Set of software routines or/and classes which provide a consistent set of utilities. Thus, an execution framework will offer an infrastructure which makes it easy to execute tasks.

**regression test:** A repeatable test which can verify the nominal performance of a software module or system.

**software bus:** A software layer which enables different executables to exchange information in a generic way. It may be used by an execution framework for communication between different modules possibly on different computers.

**standard system:** A configuration of the system (i.e. environment plus components, tasks, and clients applications) which will satisfy the demands of a typical astronomical desktop user.

**task:** A special type of components which implements a single service e.g. astronomical computation. A task takes a set of parameters, performs the calculations and returns results. It is stateless i.e. does not retain information from previous executions but starts always in the same state.

## 1.4 Acronyms

The acronyms listed below have been referenced in the text:

**ALMA:** Atacama Large Millimeter/submillimeter Array

**ANSI** American National Standards Institute

**ASCII** American Standard Code for Information Interchange (ref. ANSI X3.4-1986)

**API:** Application Program Interface

**CL:** Command Language such as the Unix shell language

**CORBA:** Common Object Request Broker Architecture

**FITS:** Flexible Image Transport System

**GNU:** GNU's Not UNIX - a Free Software Foundation project

**GPL:** GNU General Public License

**GUI:** Graphic User Interface

**HTML:** HyperText Markup Language

**IAU:** International Astronomical Union

**IEC:** International Electrotechnical Commission

**IEEE:** Institute of Electrical and Electronics Engineers

**ISO:** International Organization for Standardization

**IT:** Information Technology

**PDF:** Portable Document Format

**POSIX:** Portable Operating System interface for uniIX (ref. IEEE standard 1003.1-1990 or ISO/IEC 9945-1:1990)

**SI:** Système International d'Unités or International Standard Units

**TAI:** Temps Atomique International or International Atomic Time

**VO:** Virtual Observatory

**WCS:** World Coordinate System

**XML:** eXtensible Markup Language

## 2 High-Level Requirements

The detailed requirements are listed below arranged in 4 subsections. Each requirement is preceded by a reference number in the form **Rx.x.x** followed by a single letter denoting its priority. A scheme of 3 priority levels is used where **A)** indicates essential requirements which must be fulfilled to provide a fully functioning environment, **B)** is used for important requirements associated to a highly desirable feature, and **C)** for useful additions to the system which will enhance its usability. After each requirement, a short description or narrative is given (in *italic*). Definition of terms and abbreviations used in this document can be found in sections 1.3 and 1.4.

## 2.1 Installation and running

The requirement on installation and maintenance of the systems is given in this section where also general points like documentation and license issues are discussed.

### 2.1.1 Installation

This section deals with the installation and configuration of the astronomical software system consisting of an environment for data processing and application packages which offer explicit reduction and analysis tasks.

**R1.1.1 A** The environment must be easy to install on typical systems used by astronomers

*Computer knowledgeable astronomers must be able to get the environment up and running without special knowledge of the system configuration or help from IT system managers.*

**R1.1.2 A** It must be possible to customize the installation for local needs

*There will be a large variety of systems with different hardware and software configurations or/and operational requirements. Local customization is vital both to avoid that all users must install everything and to allow for special local features (e.g. availability of Beowulf clusters).*

**R1.1.3 A** A simple configuration tool must be provided

*It is important that installation and configuration of the standard system is as simple as possible in order to make it easily available. Reasonable defaults must be provided by the tools so that a standard installation can be done with a minimum of interaction.*

**R1.1.4 A** The installation of the single-user, standard system shall not require system/root privileges

*Many users may only have access to systems which are managed by IT people. Getting access to system privileges or submitting change requests to an IT department creates very significant overheads or may be impossible for security reasons.*

**R1.1.5 A** All warnings and errors occurring during the installation must be provided in a comprehensive log

*It is not practical for a user to sit and wait during an installation as it may take some time. It should be possible to determine all installation problems from the logs so that one needs a minimum of trails to get it all to work. At a minimum the installation procedure should inform the user whether it completed successfully or not.*

- R1.1.6 A** Both single-user and shared installations must be possible  
*Whereas most private usage (e.g. on lap- or desktops) would be done in a single-user mode, larger groups or institutes may find it more efficient to provide a common installation to all their members.*
- R1.1.7 A** Installation shall rely only on general, commonly available tools  
*It is very annoying to hunt around on Internet for special tools with specific versions in order to install the environment. Thus, care must be taken to avoid this by mainly relying on major tools with stable versions for the installation of the environment.*
- R1.1.8 A** Installation from source code must be possible  
*To ensure that the environment can be installed on a wide variety of platforms (including different compiler and library versions), it is important to be able to install from source code. This also allows people to try installation on systems which have not been tried before.*
- R1.1.9 A** It must be possible to verify an installation with a simple user test  
*Although the system will be tested and certified on standard platforms, users may want to install it on slightly different configurations. To ensure that it works as specified, a simple, relative short test procedure must be available to verify this.*
- R1.1.10 B** Both configuration and installation tools should be offered as CL and GUI versions  
*Although most users are likely to use a GUI for configuration and installation, it is very useful to offer a CL based version which can be used in batch for larger installations.*
- R1.1.11 B** Simple binary installation should be supported for main platforms  
*It is often faster to download and install a binary version. For people with no special configuration requirements, this may be the preferred way. Several package management systems (e.g. yum, deb, and rpm) exist which may serve as examples for highly automated installation systems.*
- R1.1.12 B** The environment should have a well defined package structure  
*Eventually the environment will grow and provide a large amount of features. Most people would only need a part of them. Thus, a good package structure will make it easier to overview the environment and select just what is required for a specific usage.*
- R1.1.13 B** It should be possible to view package dependencies in the system  
*As not everybody will want all features of the system it is important to know the dependencies so that one can get only required packages installed and avoid failures due to low level dependencies.*

**R1.1.14 B** All packages should have checksum, signature and size information

*As we live in a world with computer viruses and other malware, it is of high importance to be able to verify that the software one downloads is from an authorized site and in its original form.*

**R1.1.15 B** It should be possible to install packages with static libraries

*Some applications may require special, possibly unstable, libraries to work properly. To allow this, it should be possible to specify that a given application is linked with static, non-standard libraries.*

## 2.1.2 Patches and updates

Although a system would be expected to have official releases at regular intervals, the access to updates and patches is essential in order to provide users with the latest improvements.

**R1.2.1 A** Standard procedures for easy distribution of patches and updates must be provided

*Users wishing to install updates or patches to the environment should be able to do so easily by following simple instructions. Such procedures should be based on the same tools used for installation, and should be kept stable throughout the environment life (i.e. they should be always the same). Several open revision control systems (e.g. cvs and subversion) are available and could form a basis for handling patches and updates.*

**R1.2.2 A** It shall be possible to view available patches/updates and compare their version with the ones installed

*Each patch and update should be identified by a version number. Differences between locally installed version and latest version must be viewable prior to download and installation, both in the form of version/patch number, and in form of detailed ChangeLog/bugs fixed list.*

**R1.2.3 A** Installation of updates must be supported through Internet

*Similar to what can be done with most Linux Operating Systems, the installation of new versions can be performed without explicitly downloading them first. In such cases, the environment shall automatically take care of downloading and launching the installation procedure.*

**R1.2.4 A** All patches/updates must have associated documentation

*Whenever an update (i.e. a new version) is produced, all the associated documentation (user's manual, cookbook, developer's manual, etc.) must also be updated and distributed with the new version. In the case of patches, documentation should be patched as well, if applicable (these patches are probably not required in the case of trivial bug fixing).*

- R1.2.5 A** It must be possible to view patch/update documentation without downloading the entire package  
*All documentation must be distributed within the package itself, but manuals must also be down-loadable separately, so that potential users can get an idea of the environment without necessarily going through the whole installation process.*
- R1.2.6 A** Patches/updates must be linked to bug reports  
*Bug reports should have unique, clear identifier (e.g. bug report number), and each patch/update should explicitly mention which of these bug-fixes/new features have been included in the current version. For the environment it is essential to have a bug reporting system while for applications it would be desirable. Several open problem reporting systems exist (e.g. gnat and Bugzilla) and could provide a well defined work-flow for handling bug reports.*
- R1.2.7 B** It should be possible to select patches/updates and have them install automatically  
*Once the user has decided to install a given patch/update, not necessarily the latest available one, the environment should be able to automatically perform the update, without further intervention by the user.*
- R1.2.8 B** Repositories of packages and patches/updates should be available through Internet  
*Patches and updates should be kept in a repository, accessible through Internet from where to download them. Version number and ChangeLogs must be promptly inspectable without prior downloading.*
- R1.2.9 B** New patches/updates should be announced  
*Whenever a patch or an update of the environment is available, 'registered users', i.e. users wishing to be notified, should be automatically warned, e.g. by e-mail.*
- R1.2.10 B** It should be easy for individual users to install their own patches/updates  
*Problems in specific tasks may always occur. Users should be able to find and correct such issues themselves if they want. Thus, a simple way to patch a routine and re-install it should be provided. This will encourage users to help themselves and, hopefully, provide their corrections to the community.*
- R1.2.11 C** Updates may be installed from a standard file system  
*Not all users may have high-bandwidth Internet connections. In such cases, it is convenient to obtain the system or updates of it on a physical medium (e.g. CDROM or DVD) which supports a standard file system. Installations can then be performed in two steps: first a download or mounting the file system, followed by the installation, which at that point will not require Internet connection.*

### 2.1.3 Documentation

As part of the quality requirements, a common, homogeneous level of documentation is foreseen. The documents should be addressed to different actors: the astronomers as end users, the developers community and system integrators. A minimum set of documents should be produced as part of the software development process, based on open standards, including on-line help and end user manuals, design documents for developers and release notes and installation guides for system integrators.

The selected standard for documentation should not impose or assume a given software development model; on the contrary it should be open enough to allow for different development methodologies.

**R1.3.1 A** Standards for documentation and help information must be defined

*It is expected that many people contribute with packages to the environment. In order to ensure a proper information sharing and to facilitate a consistent design, it is required to define a minimal set of standards for documentation both for developers and for the end user. Subsets of this documentation will also be available in the form of on-line help.*

**R1.3.2 A** Editable documentation must be available in an open standard format

*Within the open structure of the environment, no proprietary formats should be used for documentation. Documents must be available in open standard format, environment neutral form (e.g. OASIS).*

**R1.3.3 A** Documentation must be searchable and have indices

*Access to the information in documents must be based on a proper indexing mechanism of structured topics, allowing searching. An open source document management system could be foreseen as document repository.*

**R1.3.4 A** All commands/methods must have on-line help available including version information

*The foreseen on-line help documentation should be detailed enough to cover individual commands at the end-user level or methods at the level of developer documentation. As part of the configuration control, identification of the software version must be clearly associated to the on-line help, and also associated to all documents.*

**R1.3.5 A** Help text and documentation must be part of any package

*As part of the common documentation structure in **R1.3.4**, the following minimum set of documentation is foreseen for any package within the environment: on-line help text, and package documentation according to the predefined standard in **R1.3.1**.*

**R1.3.6 A** End-user, development, and integration documentation must be part of any package

*Documentation must be addressed to three different communities: (1) end users, at different levels - novice, standard and expert -, containing: on-line help and tutorials, (2) developers, containing: design documents and technical notes as typical documents and (3) system integrators, containing: installation guides, release notes and configuration control documents.*

**R1.3.7 B** Tutorials should be part of a package

*Tutorial information to introduce new users into the usage of the package is very important. Documentation which details how to write new applications for a package or improve existing ones should be considered. Tutorials may not be relevant in some case and are therefore optional.*

**R1.3.8 B** All packages should have a common documentation structure

*To ensure a homogeneous design and to allow the sharing of information in a systematic form, the structure of the documentation should be the same for all packages. Complex packages may require extended documents and technical notes, above this common level.*

**R1.3.9 B** In-line code documentation should follow an open standard format

*Design documents and detailed technical information on packages and interfaces should rely as much as possible on automatic document generators, based on in-line source code documentation. This approach minimizes the documentation effort and ensures an updated information, in parallel with the underlying code. A good example is [Doxygen], a documentation system under GNU General Public License, that support source files for C++, C, java, Python, etc. The resulting documentation, directly extracted from the sources, can be generated in HTML format for on-line browsers, and also in Latex for off-line reference manuals. It should be possible to cross-reference commonly used routines such as mathematical functions and standards fitting procedures.*

#### **2.1.4 License issues**

One must consider license issues to ensure that the environment and associated science packages are widely available and can be shared by the community. To ensure this, the user should be granted the following essential permissions: free access to source code, right to free re-distribution of the system, and right to make code modifications when proper acknowledgment of the original authors is made. Another issue which may limit the free distribution and usage of software is *patents*. Although it is widely accepted that software

algorithms are not patentable there are attempts to make it possible.

**R1.4.1 A** The standard system must have a license which allows free access to source code, re-distribution, and code modification.

*To ensure that the system can be utilized by all people in the astronomical community, it is essential that no restrictions (including financial ones) exist for obtaining it. The GPL provides a sample for such a license.*

**R1.4.2 A** All external software packages needed for running the environment must be available without license fees

*Although the environment (ref. **R1.4.1**) will be freely available, it may well depend on other external packages (e.g. XML decoding, message system etc.). For the entire system to be useful there must exist a combination of external packages which can be used to run the environment without license fees.*

**R1.4.3 A** Any licensed software must be clearly identified

*For more specialized purposes, it may be more efficient to rely on non-free software (e.g. NAG library). General users must be aware of this when they install the system so that they can easily avoid such components if not desired because of license or financial reasons.*

## 2.1.5 Deployment

It is foreseen that the environment will be deployed in open system architectures, with minimal dependencies on proprietary products and on commercial licenses. In addition, the target installation should encompass a wide variety of equipments, from the basic desktop to powerful distributed facilities.

**R1.5.1 A** It must be possible to install and run a version of the standard system on a single-user, desktop workstation or laptop

*This requirement specifies the minimum target installation, as a low cost single-user desktop workstation or laptop, so as to provide support to a wide community of scientists. This base version could implement the essential system functionality. Specific applications in the environment may require hardware with higher specifications, as indicated in the release notes.*

**R1.5.2 A** Internet connection must not be required for basic usage

*In order to simplify the usage of the system, it must be self-contained, so that Internet connectivity is not required to run the system in its basic form. Internet may provide additional functionality to the basic system, if available.*

- R1.5.3 A** The system must be available for Linux and POSIX compliant systems  
*This requirement defines the target system, limiting the scope to POSIX compliant systems. The requirement comprises a wide potential installation base. Incarnations in other architectures may be considered at a later stage, depending on the evolution of the technology. Several products offer a POSIX compliant environment on Microsoft Windows systems.*
- R1.5.4 A** Minimum resources (hardware and software) required to install and run the system must be defined  
*Release notes are foreseen as part of the standard documentation of a new version of the environment. The release notes should specify minimal requirements to run the system covering hardware, software and versions.*
- R1.5.5 B** It should be possible to use remote resources (e.g. clusters, grid nodes) dynamically when they become available  
*The basic system should run in stand alone mode on the minimal desktop environment, as indicated in **R1.5.1** and **R1.5.2**. However, the system should allow for the usage of remote resources via Internet in a dynamic mode, when available. Foreseen remote resources include storage (data grids), CPU (computer grids), services (semantic grid). In addition, it will be possible to implement the system in clusters*
- R1.5.6 B** Client access to the environment should be provided from a wide range of systems  
*The underlying architecture is based on the client - server paradigm. Servers may be large, powerful computers, with foreseen evolution in line with grid computing and distributed services in the Internet. Clients may have a different evolution, including thin clients, notebooks, palm computers or ... mobile phones. The foreseen architecture should be open enough so as to support protocols on the client side that could be implemented in these and future devices.*

## 2.1.6 Support of coding languages

In astronomy, a large variety of programming language is used: from the old fashioned, but still widely used, Fortran, to the new, but already much common, java. The success of an environment as described here, resides also in the possibility for the community to add new applications and tools to it, without an a priori knowledge of the environment itself. For these reasons, the environment must be able to support applications written in many different languages, i.e. applications written in one of the supported languages, can make

use of the “environment wide” facilities, like logging and messaging system, device support.

**R1.6.1 A** POSIX C and C++ must be supported for development of components

*The framework must be able to run applications written in C and C++, which are the most common languages used in existing legacy software. Such applications will therefore be able to use the environment embedded facilities of VO connection, scalability, logging and messaging system, etc.*

**R1.6.2 A** FORTRAN must be supported for development of components

*Fortran is still one of the most popular languages among astronomers. Astronomers are not only the users of the environment, but also the major contributors of new algorithms and applications. A large number of highly sophisticated astronomical software packages written in Fortran exists and is still maintained and used by the community. An easy way to use these applications from within the environment must be provided.*

**R1.6.3 A** Java must be supported

*Java is a most suitable language especially for web based services, which are one of the applications the environment should be able to handle. For this reason, support of java applications is recommended.*

**R1.6.4 A** Python must be supported

*Python is currently undergoing tremendous growth in usage, for it is easy to learn and use, and has good graphics capability. Development of applications in python is becoming more and more common, especially when graphical interfaces and interactions are involved. This is why python applications should be easily embeddable in the environment.*

**R1.6.5 A** Documentation of the API for supported languages must be provided

*All interface libraries to supported languages must be extensively documented, not only for users to be able to use such API (**R1.6.6**), but also for more skilled users to extend existing APIs or build new ones for additional programming languages.*

**R1.6.6 B** Style guides, tutorials and examples should be available for all supported languages

*Normal users must be capable of using their own applications from within the environment, as well as of developing new applications using all the facilities provided by the environment. For this purpose, clear examples, HOWTOs, tutorials and extensive documentation should be provided to make this integration as easy and smooth as possible.*

### 2.1.7 Revision control

Traceability of changes in the system is important in order to provide the user with a well tested and documented end product with a unique version. The final version may need to be

a combination of the system/application version, the system build version, and the version and patch level of the operating system.

**R1.7.1 A** All parts of the system (i.e. source code, documentation configuration files etc.) must be under revision control

*It is essential for any major system to be under strict configuration control so that all modifications are recorded and traceable.*

**R1.7.2 B** Tools for generating version information in a standard format should be available

*It cannot be assumed that all parts of the system will use the same configuration control tools. However, it should be possible to present the user with a consistent picture of the exact version being employed. To provide the end-user with consistent version information, it should be possible to transform the internal used information on version to a generic format.*

**R1.7.3 B** All releases of the system or parts of it should be done with explicit version and listing of versions of all its parts

*Since results may depend on versions of several software components in the system, the user should be able to verify the actual version of all relevant components used.*

**R1.7.4 B** Links between bug reports and versions should be provided

*Not all users may be willing to implement all new versions e.g. due to overheads or operational requirements. Thus, it should be possible for users to check which version of a particular component solves a specific problem.*

**R1.7.5 B** It should be possible to get, dynamically, the explicit version of any science task being executed

*If a user detects that results have changed compared to previous versions, it should be possible to check if a new version of the software has been implemented. A user who cannot perform upgrades of the system himself needs a easy, direct way to determine the exact version he is using.*

**R1.7.6 B** It should be possible to get version numbers added to the execution logs and results being generated

*Especially for batch execution of procedures it is convenient to have the version numbers recorded to ease comparison with previous executions. The version of the environment may be given when it is started. For individual application programs or tasks either the environment or the actual program could report version information.*

**R1.7.7 B** Read access to the system source code should be possible

*If a particular bug is severe for a user, it may not be acceptable to wait for a next release of the entire system to have it resolved. Thus, it should be possible to download new versions directly from the source code repository so that patches can be obtained as soon as they are released. The access could either be directly to the source code repository for registered users or through a mirror/ftp site for generic access.*

## 2.1.8 Logging and errors

It is always important to track and record all actions (e.g. execution of tasks) performed by a system especially when it is distributed. The execution log serves this purpose and provide the user with a historical record of what the system has done. It is important that such services are language neutral and are handled in a generic manner for all parts of the system.

**R1.8.1 A** A system wide, standard error/logging interface must be available for all supported languages

*At system level, a standard way of producing messages and logging must be provided, to be used by any application irrespective of the programming language the application uses. Such standard should have a similar syntax in all languages.*

**R1.8.2 A** A detailed log of tasks executed must be available

*It is essential to have a record of what the system has done. This serves both to document what tasks have been executed and to trace problems if something went wrong.*

**R1.8.3 A** It must be easy to repeat execution of tasks listed in command logs

*It is often convenient to re-execute a set of tasks, possibly with slightly modified parameters. This is greatly facilitated if one can extract the commands from the log, which anyway would record them, and with simple tools (e.g. cut and paste) get them repeated.*

**R1.8.4 A** Errors occurring during the execution must be checked, reported and logged

*If tasks are executed remotely or in batch mode, the only way to verify that all was done is to check the logs. It is of critical importance that all errors are detected (even if the operating system aborts a task e.g. due to memory errors) and reported correctly.*

**R1.8.5 B** Error levels should be supported

*Errors should be clearly subdivided in different levels, e.g.: **fatal**, which prevents going on in the task, **warning**, for which a standard recovery action is taken and the task can continue, **information level**, which reports useful messages on the on-going task (like grade of completion, or input parameters), and **debug**, useful only to know exactly step by step, what is going on. The environment must be able to handle the different error levels in a uniform way.*

**R1.8.6 B** Logging level should be supported

*As in the case of error messaging, the level of logging should be configurable (following the example above, error, warning, info debug). The logging level can be different from the messaging level and as such should be configurable separately. For example, one may want to have just serious errors displayed on the console, but increase the information level being logged, for following inspection.*

**R1.8.7 B** It should be possible to configure the system so that only certain error/log levels are shown

*According to user's needs and wishes, it should be possible to configure the system so that only errors and messages of priority equal or higher than the chosen one are displayed. For example, in some cases only serious errors are needed, and only these should be shown. This option should be configurable at system level and be used at all levels. Both environment and applications filtering of log/error levels should be provided that is the environment should be able to filter the log levels it prints and applications should be able to choose an appropriate level.*

**R1.8.8 B** All error/log messages should contain time stamps and information on host and task versions

*Each error message, as well as all logged information should carry along date and time of when the message has been produced, as well as the name of the task which has produced it. This is of great importance when results of batch procedures are to be analyzed, as well as for debugging. In a distributed system where nodes may have slightly different configurations it is preferable to have an exact record of when, where and what was executed. In addition, input parameters and overview of results may be useful.*

- R1.8.9 B** It should be possible to add comments and notes to the command log (i.e. to use them as worksheets with an intermix of descriptions, notes, commands and results)

*It may take significant time to reduce and analyze a complex data set. For this reason the full reduction and analysis is often broken up into many shorter sessions which may be performed with long breaks in between. This makes it more difficult to remember the exact stage of the analysis from one session to the next. A convenient way of solving this problem is to insert explanatory notes, considerations and conclusions into the log which otherwise only would contain the sequence of commands and results but not the thoughts and reasons behind. Such extended logs or notes are also very useful when the work is written up for publication.*

## 2.1.9 Test and validation

Test and validation procedures are important to ensure the reliability of the system. System integrators need them to guarantee that new releases are performing to specifications while end users want to verify that their specific installation is correct.

- R1.9.1 A** Standards for test and validation of the system as well as for individual scientific tasks must be established

*It is foreseen that different hardware environments, especially in terms of parallel systems, will be employed. In many cases, source code compilation will be necessary. To ensure consistent system behavior, procedures for validation and test are essential.*

- R1.9.2 B** Automatic regression tests should be available for all parts of the system

*As the system will evolve it is necessary always to be able to verify if it is working according to specifications. Automatic regression tests which can be executed when ever needed (e.g. over night) are essential to ensure the stability and reliability of the system. Such test can also be used for quantifying performance on different platforms.*

- R1.9.3 B** Tests and associated data should be associated to all packages

*In order to ensure consistency and correct performance, tests with well-defined input and known output are needed.*

- R1.9.4 B** All tests should generate a comprehensive report which clearly indicates if problems were identified and the total time required

*Since tests may well be run by non-experts, reports must be clear and comprehensive enough to allow identification and correction of problems occurring. The timing data will also be useful for benchmark purposes.*

### 2.1.10 Internationalization

*Internationalization* is a general term, often associated to *localization*. In this context, we consider internationalization as the capability of a system to be adapted to a different country or culture, while localization is the process of adding or configuring the specific features for using the system in a concrete country or culture.

A good paradigm for the underlying model in this section is the support of internationalization provided by wikipedia, as open contribution from the community worldwide.

For the purpose of this document, the scope of internationalization is limited to the language. Other adaptations are not considered for this environment. Strict standards are adopted for calendars and physical units, like weight and length, so that there is no need to adapt them to a given locale. In this context, internationalization addresses: a) documents and manuals, b) on-line help facility, c) interactive dialogs, and d) messages and errors in system logs.

**R1.10.1 A** It must be possible to add internationalization for user interaction

*Although the reference language is English, it is expected that the system would be designed such that it will not prevent its usage or its development by communities with very limited knowledge of this language. To this aim, the architecture of the system and, in concrete, the user interaction, should allow the implementation of other languages and the configuration of the system for a concrete locale.*

**R1.10.2 B** It should be easy to provide internationalization of the general documentation (i.e. manuals, on-line help, API specifications)

*It is foreseen that documents are available in the reference language, i.e., English; however, the document management system for the environment, should allow the addition of versions in other languages (internationalization) and the configuration of a given installation to a concrete locale.*

**R1.10.3 A** An English version of all documentation (i.e. manuals, on-line help, API specifications) must exist

*As indicated above, English is the reference language used by the environment. Therefore is foreseen that master versions of documents are available in this reference language. In concrete, implementation of in-line code documentation **R1.3.9** is a clear case for the English version. Translation of documents are foreseen, understood as contributions by developers from the worldwide community.*

**R1.10.4 A** It must be possible to generate an English log

*Internationalization is good, but it should not prevent communication. System logs are often used as interprocess communication mechanisms or used by text processors for system diagnostics, etc. Therefore, the system must support the production of logs in the reference language, i.e., English.*

### 2.1.11 Support of standards

One should always try to benefit from work already done. It is important to adopt open standards, whenever possible, to ensure the maximum of compatibility with other systems. Further, one should always carefully check if existing standards can be used as new code is expensive both in terms of development and maintenance.

**R1.11.1 A** All interprocess communication must use open interfaces and protocols

*The environment will be able to use distributed resources. Thus, interprocess communication is essential and must rely on open standards to interoperate with the widest range of other tools. Protocols for messaging and events must be open and shared to ensure full interaction between components. This does not rule out the use of special protocols (e.g. Dbus or CORBA) for internal usage as long as they are not visible to a user.*

**R1.11.2 A** All communication through Internet browsers must be browser independent

*Internet browsing will be used frequently to exchange information with the system. It is important that such communications do not depend on a specific browser so that one can select one's preferred browser for such tasks.*

**R1.11.3 A** Printable documents must be available in PDF

*The PDF format is widely accepted as an open standard for printable documents offering both reasonable compression and search options. It will ease the access to printable documents to adopt a single standard which can be displayed by a large variety of tools.*

**R1.11.4 A** Standard languages for access to Internet information must be supported

*HTML is used extensively for exchange and presentation of information on Internet. Since many users will work through Internet this format must be supported. For the future, XHTML should be supported as HTML becomes obsolete. Other formats like XML may also be considered.*

**R1.11.5 A** Access to Web services must be supported

*With the increasing usage of Internet, it is vital to offer full access to Web services e.g. through SOAP. Many new services of interest will become available. Thus, seamless exchange of data between such services and the environment will be important.*

**R1.11.6 A** Access to VO services must be supported

*An increasing amount of data and computer services will become available through VO services. It is essential to get full access to them.*

**R1.11.7 A** It must be possible to read and write data in standard formats

*The environment and associated tasks must be able to read and write data in the formats most widely used in astronomy, like FITS, XML and simple ASCII tables (with a choice of most common delimiters, like tabs, spaces, commas).*

**R1.11.8 A** FITS data must be fully supported

*In particular, FITS format is the most widely used transportation format in astronomy, and it must be supported in all its features: FITS images, binary tables, FITS with extensions, hierarchical keywords, etc.*

**R1.11.9 A** Interaction with VO data must be provided

*Given the larger and larger diffusion VO will have in the future, it is fundamental that the environment can read data in standard VO format. Support for VO Table is essential. As VO format is not totally defined yet in all its subtleties (i.e. it is not as well defined as FITS is), it must be possible to update such VO I/O interfaces as soon as new standards become available.*

**R1.11.10 B** It should be possible to transform documentation into an open, standard format

*Although documentation may be saved internally in a number of formats, the transformation to an open, standard format (e.g. OASIS OpenDocument format) is important to ensure that it is not lost in a proprietary format but can be accessed and upgraded in times to come.*

**R1.11.11 B** Whenever appropriate, existing open standards should be used

*One should avoid to develop internal standards in areas where major open source community standards already exist. It is better to build on widely accepted, open standards as it increases the chance of utilizing common tools and reduce the overall development time.*

**R1.11.12 B** It should be possible to add support for other formats

*Handling of further formats not foreseen at the beginning should be possible, and easy to implement, e.g. by adding dedicated I/O libraries and keeping I/O dependencies within such libraries.*

## **2.1.12 Access to external packages**

Although the environment is based on open source software and non commercial packages, there are cases in which interaction with external licensed software can be useful. In view of these cases, the following requirements should be followed.

**R1.12.1 B** Standards for interfaces to commercial packages should be established

*There is a number of commercial packages which are widely used in astronomy. The system should have embedded the possibility to interface with commercial packages in a standard way. Differences driven by the commercial packages themselves are to be taken into account at system integration.*

**R1.12.2 B** Interface to external packages should provide the option to execute tasks in the external system and retrieve results

*Whenever an interface to a specific external package is provided, it must be able not only to “send commands” to the external package, but also to retrieve the results and “incorporate” them in the system.*

**R1.12.3 C** Access to external packages may be provided using vendor specific, dynamic libraries

*Whenever the vendor provides its own libraries for interfacing, such libraries may be used if it is more convenient with respect to a home developed interface.*

**R1.12.4 C** Tutorials on writing a client/server component to an external tool may be available

*It would be extremely useful to provide examples and tutorials on how to interface the environment to an external package, for example using sockets or pipes or other client/server methods.*

## 2.2 Definition of scripting and execution

This section deals with requirements for the usage of script procedures. The execution of task and procedures is also considered.

### 2.2.1 Execution

Data processing consists of the execution of a set of individual tasks which each perform a well defined processing step. The execution of such tasks is considered in this section.

**R2.1.1 A** It must be possible to execute individual tasks directly from the operating system shell

*All components of the application software must be packaged as individual tasks which can be called from a package specific execution framework as well as directly from the operating system level, e.g. a Unix shell. Both execution methods should use the same parameter mechanism.*

**R2.1.2 A** Both sequential order and key defined parameters must be possible

*Parameters supplied for task execution must be recognized by specifying them in a pre-defined order or by passing a keyword table to the task, listing the parameters as simple keyword value pairs. For the second method the full parameter set would not be needed. A mixed mode should be allowed where the key defined parameters may be given after the last ordered parameters. It would also be possible to pass parameters through a parameter set file (see **R2.1.3**)*

**R2.1.3 A** It must be possible to specify parameters in a parameter set file

*It must be supported that parameter keyword tables can be stored in a file from which they are passed to a task prior to its execution. This is particularly useful for pipeline and batch data processing. Tasks could update or create new parameter files.*

**R2.1.4 A** Logging of the execution of tasks, their parameters and results must be supported

*Logging of the execution of tasks, their parameters and results is an important part of scientific methodology and must therefore be supported. It must be possible to specify different levels of logging. Full logging should include time-stamps, task names, version numbers, supplied parameters, a summary of results or references to created files containing the results. Logging may be included in the processed data files in order to provide a history of data processing.*

**R2.1.5 A** Standard termination codes must be provided by tasks

*Upon normal and abnormal completion, all tasks should provide a signal from a pre-defined list of termination codes which specifies the success and completeness of task execution and the type of possible error conditions encountered during execution.*

**R2.1.6 B** It should be possible to save the state of an interactive session and later restore it for continued execution

*The analysis of a data set may take a significant period of time and extend over many individual data processing sessions. In order not to lose local variables and settings between such sessions, it must be possible to save and restore them explicitly.*

**R2.1.7 B** It should be possible to specify a specific version of a task to be executed

*For backward compatibility and to make data processing fully reproducible, different versions of the same task might be concurrently available. This implies that it should be possible to execute a specific version of a task rather than the last version installed. For practical reason, only a limited set of such older versions may be provided. The main usage would be for applications for which significant changes have been made.*

**R2.1.8 B** Failure recovery should be possible for batch mode processing

*When one executes a large number of tasks in batch it is unavoidable that something will fail sometimes. Thus, it is important to be able to restart the processing in a clever way (not just redoing everything).*

## 2.2.2 Task parameter

Handling of parameters is a critical aspect of the design that has to be coordinated with other projects (VO, ALMA, etc) to allow for a proper inter-operation. Therefore, parameter model, namespace and parameter passing mechanism/protocol should be based on common designs agreed with relevant projects.

In the basic form parameters respond to the simple scheme **keyword = value**. The actual implementation is however based on a more elaborated parameter model. A brief summary of the attributes foreseen in such a model is given below (for full description see [PARM]).

**Name** Required. Parameter name uniquely identifying the parameter within some namespace (<pset>.<pname>).

**Type** Required. Parameter data type. Parameter types can be 'string', 'int', 'float', 'boolean'.

**Mode** Required. The parameter mode defines how to handle the parameter, either as 'hidden', i.e., it can be defaulted, or as 'mandatory', i.e., input reviewed by the user.

**Value** Optional. The current parameter value. It could be an actual value, or undefined value or indefinite value.

**Default** Optional. Default parameter value.

**Unit** Optional. Physical unit of the parameter.

**Error** Optional. Estimation of the numerical error associated to the parameter value

**Min** Optional. Minimum numerical value for the parameter.

**Max** Optional. Maximum numerical value for the parameter.

**Prompt** Required. Short prompt string to be used in interactive mode.

**Help** Required. Paragraph of text or URI describing the parameter.

The following requirements are associated to the parameter model described above.

- R2.2.1 A** A standard for passing parameters between components must exist  
*It is expected that astronomical applications are packaged as components which can be called either from the host level, e.g., a shell, or from some execution framework, e.g., using SOAP or CORBA as low level communication protocol. Therefore, a mechanism to handle parameters for a component is required. Parameter sets should be defined in technology neutral form, e.g., XML.*
- R2.2.2 A** Sequences of parameters must be supported  
*Parameter sets are envisaged as collections of parameter objects which may be ordered. Parameter sets are usually associated to input / output data sets of a given task, but could also stand as data entities on their own within the environment.*
- R2.2.3 A** It must be easy to use output parameters of one task as input parameters for another  
*Parameter passing mechanisms should allow the use of output parameters from a task as input for the next one. This is an essential requirement to implement procedures in a simple form. Parameter sets could therefore appear in three forms: Input, output and independent data entity; the system should handle parameters in one form or another in a fully transparent way.*
- R2.2.4 A** Units must be specified for relevant parameters  
*As indicated in the parameter model above, parameters have associated the optional attribute **Unit**, to specify the physical units of the parameter value when appropriate. Default is 'Unitless'. A scheme based on normalized SI units is envisaged.*
- R2.2.5 A** Parameters with undefined values (i.e. NULL) must be supported  
*As explained in the model above, there should be some means to specify an undefined value (no value or uninitialized). This is called NULL value. NULL values should be handled in a consistent form by applications and system tasks. Care should be taken, so that NULL values are not mixed with indefinite values for numerical operations e.g., divide by zero (see [IEEE-754]).*

- R2.2.6 A** It must be possible to specify uncertainties for relevant parameters  
*As explained in the model above, there should be some means to specify uncertainties in numerical parameter values. The attribute **Error** is optionally associated to 'real' and 'int' data types. Estimation of the error and correct handling of error propagation is application dependent and relies on the underlying error model. This is therefore a complex issue, requiring very deep analysis before actual implementation. The system should allow the implementation of this mechanism in different phases.*
- R2.2.7 B** Both user and system default values for task parameters should be supported  
*The parameter model should support a mechanism, probably implemented by the parameter **Mode**, so that default values could be handled at system and user levels. System default values should be available in first instance, to be modified by the user in the local environment.*
- R2.2.8 B** It should be possible to use values previously given as default for parameters of a given task  
*The mechanism proposed in **R2.2.3** should allow to revert parameter values to previous defaults.*
- R2.2.9 B** Range and type checking of input parameters should be supported  
*As indicated in the parameter model above, parameters have specific attributes to identify the **Type** and, optionally, to define the allowed range of values (**Min**, **Max**). The parameter handling mechanism should use these attributes to validate parameters, both in input and output modes. The following data types are foreseen: 'string', 'int', 'float', 'boolean'. Additional types could also be considered in the final version. It should be possible to bypass such type and range checking in cases where the actual value is represented by a string e.g. an expression including mathematical functions. In such cases, checking would first be done in the application.*
- R2.2.10 B** Naming convention for parameters should be defined  
*In the basic form parameters respond to the simple scheme **keyword** = **value**. The **keyword** is the parameter name, i.e., attribute **Name** in the model above. The name is a unique identifier within a well defined namespace. Naming convention for the name space is therefore required. It is foreseen to have a structured name (...<pset>.<pname>), so that parameters could be identified in complex distributed environments. It should be possible to declare or default the name space.*
- R2.2.11 C** It is recommended to establish a directory for parameter names  
*Due to the foreseen flexibility in handling parameters, allowing parameter identification in distributed environments, the formal definition of parameter namespace is an essential aspect of the system. General purpose parameters could then be registered in a directory to allow for easy inter-operation.*

### 2.2.3 Scripting

It is often convenient to use a flexible scripting language to control the flow of execution. The typical attributes for such languages are weak typing, dynamic allocation of variables, and direct execution.

- R2.3.1 A** Both interactive and batch mode execution must be supported  
*Users of the system are expected to have various degrees of expertise. Interactive mode is essential for introduction to the system as well as for many standard applications, while batch mode must be available in order to execute heavy, repetitive task sequences, similar to pipeline applications. This is especially true in environments where parallel hardware is available.*
- R2.3.2 A** Standard flow control like looping, branching and conditional executing must be available  
*Any type of meaningful scripting must support these minimal requirements since they allow more complex and powerful data processing.*
- R2.3.3 A** It must be possible to use multiple scripting languages  
*Several modern scripting languages are available today. Recent developments show that scripting language technology evolve and that new languages may become popular rapidly. A flexible analysis system should benefit from such development.*
- R2.3.4 A** It must be possible for users to define procedures  
*Procedure definition is a standard tool in order to promote modularly designed software. It improves reliability and reuseability.*
- R2.3.5 B** Usage of both local and global variables should be possible  
*It is normally safer to use only local variables in procedures as this makes them more independent. There are cases where global variables may be easier to use for special purposes (e.g. flags for print levels) even if the risk for crosstalk is higher.*
- R2.3.6 B** It should be possible to make procedures thread safe  
*To provide scalability, it may be needed to execute the same procedure on different data sets in parallel. Although not all procedures would be required to be 'thread safe' (i.e. be able to execute in parallel), a way to ensure it should be available.*
- R2.3.7 B** It should be easy to turn an interactive session into a procedure which can be batch executed  
*To solve a new data analysis problem, one begins with an interactive test phase where various approaches are explored. The system should encourage this. After this initial phase, repetitive processing often is needed for further testing and for final reduced data production.*

**R2.3.8 B** It should be possible to execute procedures both in foreground and background (i.e. synchronous or asynchronous)

*Foreground execution is useful for testing purposes, where errors can be more easily spotted. Background execution is the preferred mode for production processing, especially in a parallel environment.*

**R2.3.9 B** The scripting environment should support common astronomical conventions such as projective geometries and magnitudes

*When dealing with astronomical data, it is convenient that parameters can be manipulated at the scripting level following common conventions.*

**R2.3.10 B** The scripting environment should allow support of application dependent conventions (e.g. time, wavelength specification conventions)

*Depending on the explicit astronomical application, slightly different conventions may be appropriate. When clearly indicated, such domain specific features should be possible.*

## 2.2.4 Scalability

The increasing amount of data to be processed in astronomy demands more powerful system to perform the processing. It cannot be expected that the speed of single computers can keep up with the growth of data in the future. Thus, scalability of the environment to larger data volumes becomes essential for satisfying future demands.

A general problem is often to join both compute power and data. This may involve strategies for migrating data to computer resources. Although no standard solution can be given, it is frequently better to let applications deal with algorithmic problems and have the environment to do data migration if needed. Special high performance, data intensive applications may have to consider the data location and structure themselves.

**R2.4.1 A** The system must be scalable

*The system is intended to be used on many different types of hardware, from laptops to massively parallel architectures. It should be reasonably easy to develop an application on a small system for subsequent processing on a parallel system.*

**R2.4.2 A** Parallel execution of tasks must be possible

*To take full advantage of modern hardware, the software structure of both environment and components (e.g. computational tasks) must support parallel execution.*

**R2.4.3 A** It must be possible to re-synchronize execution streams

*While some applications may run in parallel without any interprocess communication, there are many applications that demand resynchronization in certain stages of the processing.*

- R2.4.4 A** Monitoring of tasks must be supported  
*The system will allow execution of processes in a variety of environments, some of which may not be well-known in advance. To efficiently monitor tasks becomes essential in this situation.*
- R2.4.5 A** Location of execution of a task must be transparent to the users  
*A user may want to execute some task on a local desktop machine for testing but would select a larger system (e.g. a cluster) for reducing a large data set. This must be transparent for the user who should not need to care for where a task is actually executed.*
- R2.4.6 A** Support for multi-CPU and cluster systems must be provided  
*Current trends in hardware development favor parallel processing systems in various forms. Simultaneously, current and future data sets are expected to be large and growing. It is essential that the system supports this kind of hardware resources.*
- R2.4.7 A** Resource locking with a task's ability to suspend until a resource becomes available must be supported  
*In particular, locking of disk files is critical as a way of coordinating processing in a distributed system.*
- R2.4.8 A** It must be possible to suspend or abort processes externally  
*If all else fail, it must be possible to remove tasks and cleanup the system externally (e.g. using operating system commands).*
- R2.4.9 B** It should be possible to specify where tasks will be executed  
*In a practical parallel environment all nodes cannot be expected to look and perform exactly the same. This could be especially true in a development phase.*
- R2.4.10 B** Grid systems should be supported  
*Grid systems in the form of massively parallel systems are currently being developed and interconnected in many countries. The system should facilitate taking advantage of these new resources.*
- R2.4.11 B** For access to remote resources basic security and privilege authentication should be supported  
*Remote compute resources, such as grid nodes, are usually only accessible via enhanced security schemes.*
- R2.4.12 B** It should be possible to specify resources for a task (e.g. hardware)  
*There are cases when access to large memory is more important than access to many CPUs. It must be possible to specify such requirements to the system.*

**R2.4.13 B** It should be possible to pass parameters to executing process  
*It is useful to modify the parameters of long running processes. In particular, for iterative algorithms it is very useful to tell a process that its result is now acceptable and to shut down gracefully.*

**R2.4.14 B** It should be possible to execute several instances of the system in parallel  
*Not only should the environment be able to execute tasks in parallel during one session but several independent instances of the environment should also be allowed to be executed.*

## 2.2.5 Graphical User Interfaces

A Graphical User Interface provides an easy-to-learn, efficient tool for many types of pre-defined processing steps. Although it may limit some flexibility, it can show the user the standard route through a more complex reduction sequence.

**R2.5.1 A** It must be possible to support GUI's  
*GUI's can frequently improve the usability of a system. Thus it is essential that the environment allow the development and deployment of GUI's.*

**R2.5.2 B** GUI's may be available for execution of high level tasks  
*The availability of dedicated Graphical User's Interfaces to set parameters and execute tasks is very much appreciated by users, as GUIs are much easier to use and remember their usage. Providing GUIs for the most higher level tasks (although it is not strictly required) would surely increase the appreciation of the environment, and the Environment should be able to handle such GUIs if existing.*

**R2.5.3 B** It should be possible to record actions being done through a GUI (i.e. logging) and turn them into a procedure  
*Tasks, or other actions, executed through dedicated GUIs should be logged as all other tasks. If the output of such log can be easily converted into a script or batch procedure to be automatically executed, it would be an extra bonus for users not to have to learn "yet another scripting language".*

**R2.5.4 C** Definition of work-flows through a GUI may be considered  
*It is often desirable to have user's defined work flows (or scripts) to execute automatically a number of tasks in a predefined order. To define such work flows through a Graphical User's Interface would avoid to the user the burden of learning "yet another scripting language", and is therefore a useful extra worth considering.*

**R2.5.5 C** It should be possible to run GUI's on a system (e.g. disk-less workstations) different from that/those on which actual processing is done

*GUI's are extremely useful, but can become a burden if they have to be used remotely, especially in presence of firewalls or slow connections. For these reasons, it should be possible to have the GUI running on the local system, and the GUI itself would dispatch commands to a remote system where the task(s) is actually executed.*

**R2.5.6 C** Support for GUI's on a wide range of platforms is desirable

*Graphical User's Interfaces are often platform dependent, as they rely a lot on system graphical libraries. In designing GUIs particular care should be given to keep GUIs as platform independent as possible, keeping system dependencies to a minimum, if not totally avoidable.*

## 2.3 Access and handling of data

Requirements on type of data to be handled by the system is listed in this section where issues related to data access are considered. Further, required ways of manipulating data are given.

### 2.3.1 Meta-data

Data are described by additional meta-data (e.g. acquisition parameters, world coordinates, process history) but even simple values may require further specifications. Due to the amount of data, it is important that all quantities processed by the system are fully described.

**R3.1.1 A** Physical units in SI plus IAU recommended units must be supported (see WCS Paper I)

*Multi-wavelength data often will have different units. To combine such data the system must be able to associate the physical units to all relevant quantities so that the user always can check them e.g. being warned if different or incompatible units are used.*

**R3.1.2 B** It should be possible to associate meta-data to all numeric quantities of scientific interest

*Numeric values associated to physical quantities need to be described through meta-data. This may take many forms such as identifiers for individual values or definition of acquisition parameters for images.*

**R3.1.3 B** Processing of physical units should be supported

*When dealing with data from different sources and with a multitude of units, it is very convenient to have the system take care of conversions.*

**R3.1.4 B** It should be possible to use powers of 10 scaling factors

*Astronomy deals with quantities having value over a large range e.g. fluxes, densities, and temperatures. It is convenient to scale such values by factors of 10 (or using logarithms) when displaying them.*

**R3.1.5 B** Quality flags indicating explicit quality groups should be supported

*Often quality flags are required to better characterize data. This may give additional information on acquisition or reliability.*

**R3.1.6 C** Errors related to a specified statistical model may be supported

*It is common to use either Gaussian or Poisson models for errors but other error models are possible. Further, data with different error models may be combined or compared which may require a detailed knowledge of the errors. Thus, it may be necessary to explicitly specify the error model and its parameters for data.*

**R3.1.7 C** Pixel response functions may be supported (e.g. definition of sensitivity within a pixel)

*Several techniques may be used to extract spatial information from data close to the pixel size used for acquisition (e.g. combining frames with sub-pixel offsets). In such cases, it is of interest to use information of the pixel response function if available.*

## 2.3.2 Astronomical coordinates

The support for astronomical coordinate systems is essential when data sets from different sources are merged or compared. In an era when large, diverse data collections become available, through VO services, processing software must be able to deal with at least the most commonly used coordinate representations.

**R3.2.1 A** The general astronomical coordinate systems must be supported

*Astronomical coordinate systems are in general spherical coordinate systems defined by a reference plane and one reference point in space as the origin. Any location in space is then described by the radius vector providing the distance and the direction between the origin and the location. The direction is usually given by two angles which are conventionally specified in sexagesimal representation.*

**R3.2.2 B** Standard celestial coordinates should be supported

*Standard conventions as have been defined by the IAU FITS Working Group should be supported to specify the physical, or world (celestial), coordinates to be attached to each pixel of an N-dimensional image. The officially recognized conventions for defining celestial coordinates as they are projected onto a two-dimensional plane should be supported (world coordinate system; see [WCS-I] and [WCS-II])*

**R3.2.3 B** Frequency, wavelength and energy representation of spectra should be supported

*Spectral coordinates are commonly given in units of frequency, wavelength, energy, or Doppler velocity. Parameters and conventions needed to specify spectral information and a world coordinate function to describe ideal optical dispersers have been defined in the paper (see [WCS-III]).*

**R3.2.4 B** Standard time reference systems should be supported

*Coordinated Universal Time (UTC) has become effective in 1972 as the internationally recognized standard time reference system. UTC is derived from International Atomic Time (TAI) and runs basically as a linear function with occasional discontinuities when the difference between the atomic time and one based on the Earth rotation approaches one second and a one second adjustment is applied. Besides UTC, dates and times in astronomical data are frequently specified in Julian dates and fractions of a day. For space missions, Dynamical Barycentric time should be supported.*

**R3.2.5 B** Solar and planetary coordinate systems should be supported  
*The support for coordinate systems related to the surface of planets and stars is important for several areas of astronomy and should be given (see [WCS-I] and [WCS-II])*

**R3.2.6 C** Transformation of data to different coordinate, time and unit systems may be supported

*Support for transformations between standard coordinate systems and different unit systems is desirable, e.g. transformations of spherical coordinates between horizon, equatorial, and galactic systems, transformations between Gregorian and Julian dates, conversions between SI units and other unit systems frequently used in astrophysics.*

### 2.3.3 Grouping of data

Modern astronomy deals with ever growing data sets from which detailed information on very large collection of objects can be obtained. To keep track of such vast amount of information, it is essential to be able to group them.

**R3.3.1 A** It must be possible to associate data with different types together and refer to them as a single entity

*It is often useful to treat several, related data entities as one such as a spectrum with its wavelength calibration or an image with the transfer function of the filter used during acquisition.*

**R3.3.2 A** Collections of similar group entities must be supported

*Analyzing many similar objects, which each may be described by a set of data defined as a group entity, one needs to create collections of them (e.g. in tables or databases). Typical examples are collections of stars with their photometry, of galaxies with their morphological properties, and of quasars with their spectra.*

**R3.3.3 A** It must be possible to make group entities persistent either through files (e.g. FITS tables) or databases

*Since groups and collection of data will be used over a longer period of time and in many data processing sessions, it must be possible to save them in a persistent form.*

**R3.3.4 A** It must be possible to support a unique key for group entities

*As for databases, it is essential to have a unique reference to each entry in a collection. Such keys may either be a single entry in the group (e.g. an identifier) or a set of values (e.g. coordinates).*

- R3.3.5 A** It must be possible to select subsets of collections  
*In order to analyze special subsets of collections, it must be possible to define selective views e.g. only including those within a certain range.*
- R3.3.6 A** It must be possible to define data collections depending on their meta-data  
*Much of the information about data sets, such as maps, is given in their meta-data. Thus, it should be possible to include them in definition of groups and collections. The selection of subsets may involve complex logical relations between meta-data e.g. in order to identify relevant flat fields for a science exposure it is necessary to check at least instrument, detector, time, type and filter.*
- R3.3.7 B** It should be possible to associate an identifier to a group entity  
*For easy reference to groups of data, it is convenient to have an identifier.*
- R3.3.8 B** It should be possible to define relationships  
*Many astronomical objects are hierarchical by nature (e.g. galaxies consisting of stellar clusters with individual stars). To mimic such relations in the data, hierarchical relations should be supported.*
- R3.3.9 B** It should be easy to perform tasks for all members of a collection  
*When processing large collection of data, one normally tests the procedure on a few members to verify it works correctly. The same procedure should then be applied to the whole collection by simple means.*

### 2.3.4 Manipulation of data

The scripting environment is both used for simple manipulation of data and flow control of task execution.

- R3.4.1 A** Standard mathematical operations and functions must be provided for all relevant data  
*The user must have access to such standard tools in order to allow a meaningful analysis of data. In particular, such functions are useful to create new macros.*
- R3.4.2 B** Standard statistical functions, tests and robust estimators should be available  
*For some types of data the signal to noise ratio may be quite marginal. The application of statistical estimators on such and other data is helpful for their interpretation. Depending on the specific problem, other relevant functions may be provided.*
- R3.4.3 B** Iteration over arrays and collections should be efficient  
*One frequently tries out a procedure on a few sample data. For the production run, it should be easy and efficient to apply such a procedure on a large dataset in an array or collection.*

### 2.3.5 Data items

The basic data types important for astronomical data processing are listed. They apply to all data items, which should be manipulated by the system, including parameters, meta-data, and general variables. In addition to such basic types, string representation of numeric values should also be considered e.g. to express the number of significant digits for real values, or for expressions to be evaluated.

**R3.5.1 A** Numeric data items must support scalars and arrays

*This flexibility is needed in order avoid very cumbersome programming.*

**R3.5.2 A** Standard data types must be available such as integer, real, complex, boolean

*These are standard features in most programming languages and are needed for flexible and efficient programming.*

**R3.5.3 A** Values of at least 64-bit precision must be available

*While data from digital detectors hardly exhibits 64-bit precision, there are cases when such high inherent precision is needed e.g. for time stamps and coordinates. Also intermediate computations and theoretical computations may require 64-bit precision.*

**R3.5.4 A** Multi-dimensional arrays must be supported

*Efficient and clear organization of data is important. Many astronomical quantities are preferably collected together in multi-dimensional arrays.*

**R3.5.5 A** Strings must be supported

*Character strings must be included as they are used for meta-data such as identifiers, types, and classifications. Further, it is convenient to be able to manipulate string either to generate new strings or for comparison in logical expressions.*

### 2.3.6 Uncertainties

Data from different sources and with varying errors are often compared and/or merged. This requires a careful treatment of uncertainties.

**R3.6.1 A** Propagation of errors and quality flags must be supported

*The specification of measurement errors is essential for a proper interpretation of the results obtained from data reduction and analysis. All manipulation and processing tools must support proper propagation of error uncertainties according to a specified statistical model. Quality flags of data must be propagated through subsequent steps of data processing.*

**R3.6.2 A** Handling of undefined values must be consistent

*Undefined (NULL) values (no value or uninitialized) must be treated in a consistent way by all applications, system tasks, and numerical operations.*

**R3.6.3 C** It may be possible to specify the statistical model for error propagation

*Standard Gaussian error propagation is only valid if the function of a numerical operation is sufficiently linear within the standard deviations of the parameters. It would be desirable that more sophisticated and nonlinear statistical models for error propagation will be included with the system.*

### 2.3.7 Data management services

There are two driving factors related to the location of data: (1) the large data volume envisaged, which requires mechanisms for data storage in distributed environments and (2) complex nature of the data, with specific descriptors as meta-data.

The environment foresees a number of data management services for data access, data publication and data discovery, in such a way that information could be shared in a simple form, allowing transparent collaboration in distributed teams.

Critical services are **File location** services, to provide information on physical location of data and **File transfer** services to provide a secure, robust and efficient transfer of bulk data. **Data access control** and **user roles** will be used to implement concrete policies on data rights when required.

A **data replication** mechanism will produce copies of frequently used data sets. This would allow an efficient data access with minimal network load.

The model responds to the concepts in the data grid. Inter-operation with Virtual Observatory services is critical in this area.

**R3.7.1 B** Shared file and data access services should be available

*Although direct file and data access by individual components cannot be prohibited, it is often preferable to use shared services as they provide a homogeneous mechanism and are easier to maintain.*

**R3.7.2 B** Exact location of data should be transparent to the user

*File location and file transfer services allow users and applications to access data without knowing the exact physical location of the files. A replication mechanism together with a logical namespace will allow this transparent access in an efficient mode.*

- R3.7.3 B** It should be possible to acquire information on the location of data  
*File location services provide a transparent access to data (R3.7.2). These services will also include methods to obtain information on the actual location of the data files. Due to the data replication mechanism several copies of the file could be available in the global file system. Information on the location of the different copies should also be provided*
- R3.7.4 B** It should be possible to request that processing of data is done at their location to ensure minimum usage of networks  
*The environment will optimize resources, i.e., available CPUs, network bandwidth and storage, so that actual processing of the data is done by default in the CPU(s) where data files are located, for data intensive tasks. In addition, it is assumed that relevant data will be replicated in several active locations to minimize data transfers.*
- R3.7.5 B** It should be possible to search specified locations for relevant data  
*The file location services will make it possible to search for data files stored in remote locations. In addition, related meta-data could be used for complex searches. Inter-operation with Virtual Observatory services is foreseen in this requirement.*
- R3.7.6 B** It should be possible to specify the location of data sets created  
*The file location services include the allocation of output data sets in specified locations of the distributed file system. Access control and user privileges would be used to determine if the actual creation of the file is permitted. This requirement also allows explicit migration of data.*

### 2.3.8 Reference data

*Reference data* identifies special data sets under configuration control in a project or collaboration. In principle, the following cases of reference data are foreseen: (1) basic science data, e.g., atomic or molecular data to be used for physical models, (2) calibration data, required to calibrate observations in photometric or spectroscopic analysis, (3) test data, used in system integration tests or benchmarking of applications. However, the concept could be extended to encompass any data set that is frequently used in a given project or collaboration.

- R3.8.1 A** It must be possible to define collections of reference data (e.g. standard photometry or spectra) which can be read concurrently by multiple users  
*It is expected that the file management services will allow the definition of data collections to be used as reference data for calibrations and data processing.*

- R3.8.2 A** Changes of such reference data must be logged and available to users  
*Reference data are under rigorous configuration control, so that modifications are logged in the history records of the data. Users and applications can specify different versions of the reference data to be used in a concrete environment.*
- R3.8.3 A** The origin of reference data must be available  
*It is very important that users can trace the origin of reference data. This could be a bibliographic reference or a link to a standard data repository.*
- R3.8.4 C** Multiple copies of reference data may exist  
*Data replication mechanism allows the availability of multiple copies of frequently used data. Reference data is just a particular case of this generic feature. Since individual copies of a data set may have different versions, it should also be possible to select the explicit version to be used.*
- R3.8.5 C** It may be possible to define preferences for reference data used  
*The foreseen namespace allows the specification of the reference data, including the actual data set, the version to be used (**R3.8.2**) and the location of the replica (**R3.8.4**).*

### 2.3.9 Shared access to data

An increasing number of astronomical projects are done in collaborations where many participants need access to data and analysis tools. Instead of shipping new version of data and results around to collaborators, it is much more efficient to shared access. Requirements related to such data sharing is addressed in this section.

- R3.9.1 A** Access to data must be easy and support sharing  
*No matter where they reside, data must be easily available to the system. It must be also possible to share data among different users (e.g. calibration data), without duplicating them.*
- R3.9.2 A** It must be possible for well defined sets of users to share data depending on roles and privileges  
*For some applications (like calibration pipelines or some sort of proprietary data), it can be important to keep access restricted to some particular users or groups of users. A mechanism of data protection in this sense must be provided, so that some data can be read or written only by authorized users.*
- R3.9.3 A** It must be possible to protect data against multiple concurrent write accesses  
*While concurrent read access should always be possible, multiple concurrent write access should be avoided to avoid data corruption or bad data history tracking. Some mechanism of data locking should be devised to prevent such occurrences.*

- R3.9.4 A** There must not be imposed limits on size of data sets when handling and analyzing them  
*Data files are becoming larger and larger everyday. It is important for the environment to survive in time that data size is NOT an issue, i.e. files of any size must be supported and handled and no a priori limit exist on data size.*
- R3.9.5 A** It must be possible to lock data sets or parts of them  
*When data are grouped in data sets, it must be possible to apply a locking mechanism to the whole data set or to part of it, if wished, avoiding to lock singularly each data file belonging to the data set.*
- R3.9.6 B** Access to read-only data sets should be possible  
*Some data, like calibration data, should be protected against accidental corruption by the users. In these cases, only read access should be granted to all users, possibly granting write access only to privileged users*
- R3.9.7 B** A standard for adding history records should be defined  
*Adding history records to a data file should be a “centralized” task handled at environment level, so that all tasks will use the same mechanism. This will have the double advantage of easing developers from handling this sort of “logging” and of having histories in the same format irrespective of the task used, making history parsing easy to handle (if needed).*
- R3.9.8 B** It should be possible to associate a version to a data set  
*The actual version of a data set is often determined from several meta-data items such as time of creation, origin, date of modification and name of application modifying it. Instead of comparing multiple meta-data items, it is simpler to associate an explicit version number to it.*
- R3.9.9 B** Access to data sources with fast/concurrent updates should be handled  
*In real-time scenarios, it may be of interest to monitor data sources for which data values change at a high rate and concurrent with the component being executed. This may also be important for some interactive data processing.*
- R3.9.10 C** History of all changes to data sets may be recorded and associated to them  
*Whenever an operation is performed on some data, it must be recorded as part of the data themselves (e.g. with a history mechanism in the header of the file) to allow exact reconstruction of the analysis performed.*

## 2.4 Applications

The current section deals with more application specific requirements in the sense of scientific packages and contributed code.

### 2.4.1 Access to code

For free exchange of ideas within the science community, access to the actual source code of astronomical application is vital. Even for well documented applications, fine but important details may not be fully explained. The source code gives the final answer to complex questions and should be freely available. Scientific progress often relies on building on others achievements which, in the software context, may be to improve existing code (with proper acknowledgment of previous authors). Thus, this should also be facilitated.

**R4.1.1 B** General science packages should allow free access to source code, re-distribution, and code modification.

*To foster science progress, it should be possible for users to improve science packages and, hopefully, feed back improvements to the community. This is greatly facilitated by such software being under the Gnu Public License.*

**R4.1.2 B** All scientific packages should be available as open source

*Even with elaborate documentation available, the exact working of a science package may not be easy to understand. Access to source code for such packages is highly important in order to enable users to verify if results are reliable or affected by special assumption.*

**R4.1.3 B** Contributed science packages should be made available with permission to access to source code, re-distribution, and modify code

*It is hoped that scientists will contribute their packages to the environment so that a large community can benefit. Thus, such user should be encouraged to provide their software under GPL in order to facilitate further usage easier(see **R4.1.1**)*

### 2.4.2 New applications

It is important to make it easy for people to add new applications to the system so that the full community can take advantage of new developments.

**R4.2.1 A** It must be easy for astronomers with no special computer science background to add new application code

*It is important to facilitate inclusion of astronomers expertise and experiences into a new software system. Too often specialized algorithms remain inaccessible to the astronomical community.*

- R4.2.2 A** A simple, well defined package structure must be defined for astronomical applications  
*A structured approach makes it considerably easier to integrate new modules into an existing system. It facilitates compilation and linking procedures.*
- R4.2.3 A** Addition of new applications must be scalable  
*It is expected that the number of applications available from the environment will grow continuously. Thus, it is essential that the performance of the system does not decrease significantly when more are added.*
- R4.2.4 B** Tutorials on writing simple applications should be provided  
*To encourage algorithm designers to include their software into a larger environment, it is important to have as low introductory threshold as possible. This minimizes the time investment needed.*
- R4.2.5 B** Contributed packages should only be accepted into the standard system if they contain full documentation and regression tests  
*Since the standard system will be available to and used by many, the required level of documentation must be high, in order to facilitate debugging and maintenance.*
- R4.2.6 B** It should be possible for a task to request display of a data set  
*Doing interactive data analysis, the user frequently may want to view the data. Although this often is done through special *display-tasks*, it may be convenient (or more efficient) for some tasks to show their data themselves.*
- R4.2.7 B** It should be possible for a task to get parameter values through the placement of a cursor on the displayed data  
*Detailed interaction during data analysis frequently requires the user to point on special features on a display of the data. To facilitate this, tasks should have the option to request cursor values in an efficient way.*
- R4.2.8 B** It should be possible for a task to have a dialogue with the user in a way that fits naturally into any GUI being used  
*In some cases, not all parameters can be fully defined before a task is started. They may depend, in a complex way, on the data themselves. Thus, a task should be able to communicate efficiently with the user during interactive sessions.*

## A OPTICON Network 3.6

The OPTICON activities are funded by the European Commission under Contract no. RII3-CT-2004-001566. The Network 3.6 on 'Future software environments for processing and analysis of astronomical data' was started in the spring of 2004. The people involved in writing the present high-level requirements document are: P. Grosbøl (ESO), B. Garilli (INAF), D. Ponz (ESA), P. Linde (Lund), K. Reinsch (Göttingen), H. Terlow (Kapteyn), M. Ullgren (CSC), N. Caon (IAC), J. M. van der Hulst (Kapteyn), K. Banse (ESO), W. Pence (GSFC/NASA), W. Cotton (NRAO), and D. Tody (NVO/NRAO).