SPIRE-ESA-MOM-002287

Herschel IA design review

Herschel-HSC-MoM-475

Place: KUL, Leuven, Belgium

Date: 16-17 November 2004

Time: 09:00– 17:00

Present:

Present:	Jorgo Bakker	JBa (HSC/ESTEC)
	Odile Coeur-Joly	OCJ (CESR/HIFI)
	Nicola de Candussio	NdC (HSC/ESTEC)
	Albrecht de Jonge	AdJ (SRON/HIFI)
	Wim de Meester	WdM (HSC/ESTEC)
	Lutfi Dwedari	LD (D-TEC/ESTEC)
	Steve Guest	SG (RAL/SPIRE)
	Rik Huygen	RH (KUL/PACS)
	Juliet Kemp	JK (IC/SPIRE)
	Do Kester	DK (SRON/HIFI)
	Andrea Lorenzani	AL (OAA/HIFI)
	Jean-Jacques Mathieu	JJM (HSC/ESTEC)
	Stephan Ott	SO (HSC/ESTEC)
	Jose Pizarro	JP (D-TEC/ESTEC)
	Johannes Riedinger	JRR (HSC/ESTEC, part time)
	Peter Roelfsema	PRR (HIFI/SRON)
	Hassan Siddiqui	HS (HSC/ESTEC)
	Bart Vandenbussche	BV (KUL/PACS, part time)
	Michael Wetzstein	MW (PACS/MPE)
	Peer Zaal	PZ (SRON/HIFI

1) Welcome

SO welcomed everybody, and thanked RH for organising the meeting.

2) Kick-off

SO introduced the rationale and objectives of the IA review (see appendix B). While during the HCSS review last year (CSDT #18) the majority of IA packages were considered not sufficiently mature to be reviewed[1], the IA WG felt that it is now worthwhile to perform a full review of the IA system and its packages. The conclusions of this meeting will become input to the Herschel Science Ground Segment Review

[1] JConsole, dataset, numeric and IO were reviewed during CSDT#18

3) First day of panel discussions

The individual review panels (task/dataflow, plot, classloader/sandbox, system/infrastructure, JConsole and numeric) commenced according to schedule (see appendix C)

4) Reflections of first day of panel discussions

SO welcomed everybody to the second day of the review meeting (see appendix D). The system panel asked for a second session, and the schedule was updated accordingly.

5) Second day of panel discussions

The individual review panels (image/dataset, UI/help and system/infrastructure) commenced according to the updated schedule (see appendix D)

6) Conclusions

SO welcomed everyone to the concluding session, outlined the organisation of the final session (see appendix E) and invited the panel chairs to present their conclusions.

6 (a) Presentation of panel chairs

6 b (i) Plot package

MW summarised the conclusions of the panel reviewing the plot package. The full report can be found in appendix F.

- the classes are very big (> 5000 lines)
- the packages needs to be redesigned in order to keep it maintainable and to separate the public API
- user documentation needs to be written
- the package is graded green

6 b (ii) Dataset package

RH summarised the conclusions of the panel reviewing the dataset package. The full report can be found in appendix G.

- the best way how to handle quantities remains an open issue
- metadata should be expanded to permit sets of array type data
- the package should be re-organised after system wide guidelines have been agreed
- the package is graded green

6 b (iii) Image package

RH summarised the conclusions of the panel reviewing the image package. The full report can be found in appendix H.

- the package is still under construction

- the package should be re-organised after system wide guidelines have been agreed
- the package is developed by PACS, and also mainly used by PACS. This might lead to a tunnel vision
- the package is graded green

6 b (iv) Numeric package

DK summarised the conclusions of the panel reviewing the numeric package. The full report can be found in appendix I.

- the effort to implement all types for up to five dimensions was necessary
- the generic toolbox framework is OK, but some of the individual toolboxes need to be refactored to make them work on all types and dimensions. At the same time they need to be redesigned to make them more OO (less a copy of IDL's versions).
- the package is graded green

6 b (v) Classloader package

JJM summarised the conclusions of the panel reviewing the classloader package. The full report can be found in appendix J.

- Classloader is nice to use and useful little tool
- link the documentation to JIDE so people using java can use it in their development
- integrate the classloader in JIDE so it does not need be to be initialized
- the package is graded green

6 b (vi) Sandbox package

JJM summarised the conclusions of the panel reviewing the sandbox package. The full report can be found in appendix K.

- the mechanism between properties and sandbox has to be sorted out
- the package is graded green

6 b (vii) Task package

PZ summarised the conclusions of the panel reviewing the task package. The full report can be found in appendix L.

- with the exception of history and a progress/completion indicator, all currently needed facilities are implemented
- a design/architecture document has to be written
- package is mainly used by HIFI
- the package is graded green

6 b (viii) Dataflow package

PZ summarised the conclusions of the panel reviewing the dataflow package. The full report can be found in appendix M.

- the package is in maintenance mode, but some documentation has top be updated
- package is mainly used by all ICCs
- the package is graded green

6 b (ix) UI package

NdC summarised the conclusions of the panel reviewing the UI package. The full report can be found in appendix N.

- package is a collection of utility classes
- package could be promoted to HCSS/SHARE
- the package is graded green

6 b (x) Help package

NdC summarised the conclusions of the panel reviewing the help package. The full report can be found in appendix O.

- the package is still under construction, but fulfills the *currently* formalised use-cases
- some restructuring is needed
- as the impact of not yet specified QLA related use-cases is unknown, the package is graded amber[2]

6 b (xi) JConsole package

SG summarised the conclusions of the panel reviewing the help package. The full report can be found in appendix P.

- the package receives many SCRs. They are mainly non-IA related or request for plug-ins
- the package should be restructured to improve cohesion, maintainability and documentation
- the package is graded green[3]

6 b (xii) System and infrastructure

AdJ summarised the conclusions of the panel reviewing the help package. The full report can be found in appendix Q.

- a coherent or compete set of use cases for IA is missing
- several areas have to be investigated as a matter of urgency:
  - the potential problems of the shopping basket concept, and of large downloads
  - evaluation of database implementation
  - implementation of history
  - stopping of tasks
- a 'system architect' should be appointed to coordinate the development efforts on a technical level
- the package is graded amber. While nothing came up that would prevent the IA system from being used by end users to do Interactive Analysis, but there are major issues identified that require a significant amount of work to address

[2] The missing use-cases were received and analysed and found to be not critical Consequently the status was re-set to green.

[3] In the final report the chair decided to downgrade the status to amber

6 (b) Round table discussions

6 b (i) Reflections on review

The participants considered the time for preparing and executing the review well spent, and encouraged to have a big IA review (with objectives adapted to the development progress) each year.

In addition, the following comments were made:

- more detailed review guidelines/objectives per panel would have been welcome
- more time for preparation would have been welcome
- the panel recommendations should be followed up
- the work of some panels was hampered by the number of observers who influenced the discussion
- it is not clear that the panel deputy should act as review chair, as there might be a conflict of interests (neutrality vs. development wishes)

6 b (ii) Priorities for IA#7

The participants considered the following items a priority for IA#7. It is expected that additions will come from the documentation review to be held in London in the near future

- improvements in formal documentation, notably for the user requirements and the design
- improvements in user documentation
  - navigation to find documentation
  - online help aspect
  - recommendations for writers
- interruption of tasks in an IA session
- completion indicator for tasks
- help aspect of documentation
- database aspects
- follow-up of review recommendations

6 b (iii) Message to CSDT

- we have to talk about the packages that cross the border (product, database issues, properties)
- installation of HCSS should be improved

6 b (iv) Message to the Herschel Science Ground Segment Review

- overall, our indicators are green
- no major technical issues are hindering the development
- Additional manpower is needed to bring the Herschel Common IA development to a successful conclusion

7) Closure

SO thanked all contributors to this review (in person or remote), and closed the meeting.

Appendix A

Actions from panels:

04-plot-01	3D plotting is not available now. Research on available 3D packages
04-plot-02	User groups should discuss if they want graphical annotations, and, if yes, if they want an example implementation first or if they don't need such a prototype
04-plot-03	Investigate the possibility to implement graphical annotations. To see how professional it can be done, one should implement a simple example and afterwards make a time estimate.
04-plot-04	Extend accepted datatypes to numeric Array1dData, solve the problem with quantities on the level of the numeric package, then remove Dataset plotting from PlotXY
04-plot-05	Refactor the package with the main goal of better package structure, smaller classes and a clear separation of the public API
04-plot-06	Update architecture and design documents after the refactoring of the package
04-plot-07	New methods should be documented. It's a good working style to write at least one line of description before implementing a method to clarify its purpose.
04-plot-08	Document the ideas to increase performance and usability
04-plot-09	Put the use of class private fields in the constructor into the HowTo (and user documentation).
04-dataset-01	use and documentation of Product needs updating
04-dataset-02	take care that the existing SxRs are solved e.g. • remove .data part • addressing columns • listeners for datasets
04-dataset-03	solve the dependencies for dataset.gui
04-dataset-04	IA System Architecture Group to clarify how and where does a dataset live in the system
04-dataset-05	IA System Architecture Group to clarify what package is responsible on quantities, and where and how are quantities handled in the IA
04-dataset-06	prioritise implementation of history

04-dataset-07	IA System Architecture Group to clarify the location of SpectrumDataset and its View
04-dataset-08	IA System Architecture Group to clarify the location of ImageDataset and its View
04-image-01	go over improvements section from input by Jorgo and make the necessary changes
04-image-02	IA System Architecture Group to see whether one can get statistics on the use of individual packages
04-image-03	IA System Architecture Group to ensure that Axis, Layer, Annotations, Histogram interfaces should at least be shared and decide whether there can be common implementations and, if so, where should they reside?
04-image-04	IA System Architecture Group to ensure that Axis, Layer, Annotations, Histogram interfaces should at least be shared and decide whether there can be common implementations and, if so, where should they reside
04-image-05	IA System Architecture Group to see whether GUIs used by image (like printing) can be shared
04-numeric-01	Investigate and implement selection for all dimensions
04-numeric-02	Refactor individual toolboxes to bring them in line with the latest version of numerics, make them run for all dimensions and types, and make them more OO
04-numeric-03	Propose mechanism how to contribute to numerics package
04-numeric-04	Investigate special mathematical functions contained in JSci.math library http://jsci.sourceforge.net/api/JSci/maths/AbstractMath.html to see whether we can use them
04-numeric-05	implement workaround for long integers in Jython
04-classloader-01	Ask to ICC if classloader is used and, depending on what feedback people provide, if there are any plans for the future that would change the way it works now.
04-classloader-02	Investigate whether there are new packages for java class loader that could reduce our maintenance efforts
04-classloader-03	Write a design document

04-classloader-04	IA System Architecture Group to consider whether subpackages for classloader should be split
04-classloader-05	Check the error handling within classloader
04-classloader-06	Link documentation to jide so people using java can use it in their development
04-classloader-07	Integrate the classloader in Jide so it does not need be to be initialized explicitly by the user
04-sandbox-01	IA System Architecture Group to ensure better integration of Sandbox + jide + properties + classloader (only one approach rather than the current two (sandbox and properties))
04-task-01	add new HowTo for the user
04-task-02	top-level documentation should be included under the JavaDoc-package description
04-task-03	current HowTo should become a developer's guide
04-task-04	add design documentation (dealing with task-design especially: JTask - Task)
04-task-05	add new HowTo for the user
04-task-06	IA System Architecture Group to sort out up to what basic hcss.ia level Task should be used as base component. Should the Task list be part of the ia.numeric.toolbox?
04-task-07	IA WG to promote the use of task components for all IA processing steps / functions.
04-task-08	IA DOC WG to ensure that system level documentation includes explanation / guide on intended use of IA components including Task
04-task-09	define clearly the concept of task and signature
04-task-10	prioritise implementation of completion notification
04-dataflow-01	update documentation
04-dataflow-02	architecture & design documentation should be moved from the User Guide into a special dedicated arch./design document
04-dataflow-03	Wim de Meester to give detailed feedback on current available documentation

04-dataflow-04	improve code for DataFlowManager and change implementation such that this Manager can be (1) used as a JComponent inside another application and (2) be added to the testharness
04-dataflow-05	IA WG to give input whether - and if, so on which time scale - within ia.dataflow end-user functionality should be included. This is needed as suggested by SG. An example might be stream type processing of a large set of data (database pointers) belonging to one observation. This would request the implementation of components which assure jython friendly usage within jython as well as an additional HowTo for the end-user.
04-dataflow-06	IA WG to promote that tasks should be the IA base component which can be re-used in processes (possibly via the TaskWrapper the use of task components for all IA processing steps / functions.
04-dataflow-07	IA DOC WG to ensure that system level documentation includes explanation / guide on intended use of IA components including dataflow components
04-UI-01	JK to add the Mouse use case use case to the package documentation [dd 15th December]
04-UI-02	NdC to open an SPR to refactor SystemPopup [dd 30th November]
04-UI-03	NdC and HS to document Window Manager class with Tim Lock [NdC 30th November]
04-UI-04	JK to deprecate help class as it is superseded by the help class in the help package. [dd 30th November]
04-UI-05	SG to open an SPR to address concerns on ScreenshotGenerator [dd 30th November]
04-UI-06	SG to open an SPR to address concerns on GuiUtils [dd 30th November]
04-help-01	SG and MW to provide all the relevant QLA use cases [due date November 26th]
04-help-02	Documentation WG / review to discuss the implementation of a strategy for retrieval the most appropriate documentation [dd 30th November]
04-help-03	JK to move herschel.ia.sessioninspector in herschel.inspector.session [30th November]
04-help-04	JK to investigate the XSL-FO technology [dd end on January 2005]

04-help-05	NdC to update documentation on HelpTask in order put more emphasis on the limitations [dd 30th November])
04-help-06	NdC to remove the use of System.out.println [NdC 30th November]
04-help-07	NdC to check if declaration within for loop is cause of bad performances [30th November]
04-help-08	NdC, HS, SG Fix the misuse of configuration in HelpLocator.defaults
04-help-09	DOC WG to check standards and exceptions in the formats of displayed documents
04-help-10	NdC to open spr for checking compliance with javadoc [dd 30th November]
04-help-11	JK to the documentation explaining limits on tests architecture [30th November]
04-jconsole-01	Restructure the package to make it easier to understand. The goal of moving things is to improve cohesion, maintainability and documentation
04-jconsole-02	Ensure that only classes and methods intended as public API appear in the JavaDoc
04-jconsole-03	IA system architecture group to specify package structure guidelines
04-jconsole-04	Look whether general utility classes can be moved outside the package.
04-jconsole-05	CSDT/IA system architecture group to define a general place to put utility classes
04-jconsole-06	Move any Herschel-specific code into a separate Herschel subpackage. Ensure that JIDE still runs when this package is not there
04-jconsole-07	Define a protocol such that plug-ins can be added to jide. Use this protocol to add the dataset inspector, session browser, and help functionality.
04-system-01	Review if the risks identified in the tiger-team report for large downloads and the shopping basket concept are still risks
04-system-02	The scope of History and its relation to IaProcess and Task need to be (re-) identified, and the priority to implement it has to be reestablished
04-system-03	Persistence implementation by versant database needs evaluation

04-system-04	It should be investigated how to prevent mistakes caused by Jython exposing Java implementation details
04-system-05	Performance requirements for IA should be drafted
04-system-06	Allocate resources to give design directions for extending the GUI-based capabilities of the system.
04-system-07	CSDT should investigate how to decouple the java package structure from the CVS module structure
04-system-08	The IA developers need an IA system architect (or architecture group), that can coordinate the technical design issues between the separate IA developers independent of management.
04-system-09	

Appendix B

Procedure for IA review 16/17th November 2004

Version 1.0, 27/10/04

1 Objectives

The purpose of the review is

- to verify that the architecture and design of IA provides a sound basis for further development
- to identify major potential development risks
- to check that the packages are maintainable even in the absence of owner and deputy
- to identify any other points we would like bring toe the attention of the Herschel Science Ground Segment Review

The output should be a list of items describing

- where we complied with the objectives
- identify the major shortcomings/problems, and propose solutions
- which objectives were not reviewed
- proposed HCSS use-case document updates
- proposed IA ADD updates
- identify minor shortcoming/problem requiring the attention of the package owner

The minutes (including the review reports of each panel) will become input to the Herschel Science Ground Segment Review.

2 Inputs

The following items are inputs of the review:

- IA#6/1 release candidate, including the metrics and test-coverage reports[1]

Applicable documents:

- HCSS User Requirements Document (FIRST/FSC/DOC/0115, Issue 2.0, Date 31 August 2001)
- HCSS Preparatory Calibration Database Requirements (Herschel-HSC-DOC-0350, Issue Draft, 0.4 Date 12 December 2003)

[1] In case this is not possible due to the changed delivery schedule, the metrics and test-coverage reports of IA#5/4 will be used instead

- Herschel Common Science System: Use Case Definitions (FIRST/FSC/DOC/0157, Issue 2.1, Date 11 April 2003)[2]
- End user requirements for HIFI Interactive Analysis (HIFI-ICC/2001-004, DRAFT 0.41, date 19/12/01)
- HIFI calibration uses cases (HIFI-ICC-2001-005, DRAFT 1.0, date 20/10/2003)
- PACS IA Software User Requirement Document (PACS-ME-RD-002, Issue: 0.6, Date: 25.04.03)
- PACS User Requirements for plotting and image displaying in IA (PICC-ME-RD-003, Issue: Draft, Date: 03-02-03)
- PACS - Standard Process Generation (SPG) PICC-ME-TN-001, Date: April 21, 2000, Issue: 1.0)
- SPIRE Instrument Control Center: Use-Case Definitions (Version 1.0, March 15, 2002)
- SPIRE IA Concept
- Java Coding Standard and Guidelines for the Herschel Common Science System Development, HSCDT/TN009, Issue 2.0, Date 30.10.02
- Herschel Common Science System Jython Coding Standards, HSCDT-TN040, Issue: 1.0, 7 April 2004

Reference documents:

- HCSS IA ADD (in preparation)
- IA vision documents
- HCSS Jython Developer's Recommendations, HSCDT-TN039, Issue: 1.0, 7 April 2004
- HERSCHEL IA COMMON FRAMEWORK ARCHITECTURE TECHNICAL REPORT, HERSCHEL-HSC-DOC-0241, issue 1, 12/04/02

[2] This document is currently being updated with additional IA use-cases. A later version might be made available shortly before the IA review

3 Review outline

3.1 Suggested points of attention

The items in appendix A (for the system review) and B (for each package) are a list of suggested points to be considered by the reviewers. They don't form a checklist that has to be completed during the review, as this wouldn't be possible in the available time. Instead, they are aimed to identify a list of points that might require attention. Each review panel will decide which are the most critical points to address, and list those points that were not addressed due to a lack of time. In case of need, each review panel is invited to augment this list.

Care has to be taken not to be bogged down in details, and thereby neglect the major design aspects.

We should

- concentrate on the IA system itself

and on subsystems that are

- recent
- were not reviewed one year ago
- old, but not frequently used, and understand the reasons why

3.2 Review panels and procedure

- each panel should have between 4 and 7 active members, e.g. IA contributors that did subscribe to this panel and did provide written input. Other persons who want to learn more about a package are welcome as observers (listening in only).
- at least the package owner or deputy has to be in the review panel
- each panel chairman (preferably the package deputy) is responsible
    - to select the main review points per package[3]
    - to organise his session and ensure that all points expected to be discussed will be discussed
    - to minute the conclusions[4]. For each point of the checklist it shall be stated whether it was reviewed, what was the outcome of the review, and what corrective are actions are proposed
- each panel member will provide written input to the panel covering the main review points by Friday, 12th of November, e.g. *before* the meeting

[3] A general checklist is not possible, as each package is in a different stage, and has different points needing attention

[4] The chairman might appoint a secretary to take minutes, if (s)he considers this necessary

- review starts with general session (1hour), followed by splinters. It ends with a general session (2 hours)
- IA coordinator is responsible for overall organisation and final minutes

A Suggestion list for System Review

A.1 Use cases/requirements

A.1.1 Can the system fulfill all foreseen operational and community needs?

A.1.2 Can the system meet the needs of both HIFI/PACS "IA as command line with GUIs" and SPIRE "IA as GUI with command line" view?

A.1.3 What are the use-cases/requirements that are expected to hold major development difficulties for the IA system? Are they clearly specified?

A.2 Architecture

A.2.1 Will the currently developed subsystems (calibration sources database, calibration data access), future subsystems (SPG, QCP, archive browser, pointing refinement system) and instrument specific parts (H/P/S..SS) fit into the overall architecture of astronomers IA (delivered to community) and operations[5] IA (used by ICCs/HSC)?

A.2.2 Do we have additional requirements on HCSS infrastructure?
(Note: Currently I noted: maintenance of packages having contributors for distinct sub-packages (dataset))

A.2.3 Is the package structure within IA OK? Do we have unnecessary dependencies?

A.2.4 Are the interfaces to other parts of the ground segment clear?

A.2.5 What are the areas of overlap with the HCSS? Are there "grey areas" where things might fall between the cracks?

A.3 Documentation[6]

A.3.1 Is there an IA user manual?

- Does it have a document number? If no, should it have one?[7]
- Does it describe the major functionality?
- To which IA version does it apply?

[5] Also sometimes called "expert" IA. However, this phrasing is deprecated in order to avoid the impression that the "astronomer's IA" is only a second-class IA

[6] Note that IA documentation will be subject to a detailed review early December

[7] According to QA, all documentation that is not automatically generated from version controlled modules, should have a document and version number

A.3.2 Do we have an IA architecture and design document, showing:

- Package decomposition
- Design notes, explaining design decisions
- UML diagrams, where appropriate, to illustrate design features
- Does it have a document number? If no, should it have one?

A.3.3 Is the place for the package documentation consistent?
Note: As input for this the review reports of the packages are needed

B Suggestion List for Package Review

B.1 Use cases/requirements

B.1.1 Does the subsystem satisfy the relevant use-cases and user requirements? (Note: Use-cases are mainly applicable to applications, rather than utility libraries.)

B.1.2 What are the use-cases/requirements that are expected to hold major development difficulties for this package? Are they clearly specified? (Note: Currently I noted: stopping of tasks, adding of debugging features, handling of big datasets, reported performance problems, IDLizing, saving of intermediate data)

B.1.3 Does the package have a high number of SCRs? Are the modification of use-cases and additional use-cases?

B.2 Architecture

B.2.1 What is the "raison d' être" for this package? To which currently existing use-cases is it linked? Do we need additional usecases to specify the most important functionality?

B.2.2 Does the package fit coherently into the overall architecture of IA?

B.2.3 Is there an architecture and design document, including:

- Subsystem decomposition
- Design notes, explaining design decisions
- UML diagrams, where appropriate, to illustrate design features

B.2.4 Is it a component architecture, with cleanly decoupled components?

B.2.5 Does it have understandable interfaces?

B.2.6 Does the API hide the underlying implementation (i.e. provide suitable abstraction)?

B.2.7 Are there public classes/methods which should not form part of the public API?

- Conversely, are there classes/methods that should be made public?
- Is it clear which packages form part of public API?

B.2.8 Does it have direct interfaces with other subsystems or other libraries? Is it incorrectly/overly coupled to other packages? Are there circular dependencies?

B.2.9 Is the subsystem decomposed into packages? Is this decomposition sensible, or a cause of user irritation and/or increase in complexity?

B.2.10 Do you have suggestions on the improvement of the architecture or design (E.g. would this package benefit from features introduced by JDK 1.5)?

B.2.11 Are there new external packages that are better suited to provide the underlying services?

B.2.12 Would it be worthwhile to make the package open source?

B.3 Documentation

B.3.1 Is there a user manual for the package? Does it have a document number? Should it have one?

B.3.2 Is there adequate developer's documentation? Does it have a document number? Should it have one?

B.3.3 Is there adequate design documentation? Does it have a document number? Should it have one?

B.3.4 JavaDoc

- Does the JavaDoc follow the Sun guidelines?
  - Does each method have an understandable JavaDoc description?
  - Is the general level of JavaDoc comments adequate (especially for public API)?
  - Does the JavaDoc include code examples, where appropriate?
- Does each subsystem have a top-level "package.html" file, including:
  - An initial summary sentence documentation, etc
  - A description of the contents and purpose of the package
  - A "Package Specification" section (may be blank)
  - A "Related Documentation" section with hyperlinks to design

B.3.5 Where is the documentation located?

B.3.6 Does the subsystem have a readable CHANGELOG file?

B.4 Design

B.4.1 Are there any indicators of design problems (code metrics, test coverage, high number of SPRs)?

B.4.2 Is the package easy to use? If not, why?

B.4.3 Are appropriate Design Patterns employed?

- Should further design patterns be introduced to improve abstraction, flexibility, modularity, etc?
- Conversely, are inappropriate design patterns employed?

B.4.4 Is there a proper separation of concerns between classes and between packages?

B.4.5 Is the design properly Object Oriented?

B.4.6 Is the principle of operation sound?

B.4.7 Might the design lead to performance problems?

B.4.8 Is reflection used unnecessarily?

B.4.9 Is inheritance used in an appropriate way? These checkpoints should be true wherever inheritance is used:

- It is a "kind of", not a "role played by", relationship.
- The object never needs to transmute into another class.
- It extends rather than overrides or nullifies the superclass.
- It does not subclass what is merely a utility class.

B.4.10 Would composition be better than inheritance?

B.4.11 Should any classes be immutable?

B.4.12 Is error handling adequate?

- Are there any 'exit' statements (except in "main" method)?
- Are the error and log messages understandable and appropriate?
- Is exception handling employed for all exception / error conditions?
- Are the exceptions thrown misleading or not the right ones?
- Should the exception be thrown at all?
- Should the exception be checked or non-checked?
- Can a situation be envisaged where the code will obviously fail?

B.4.13 Does the design allow the subsystem to be properly tested?

- Is a pluggable architecture needed to support test stubs?
- Is the GUI decoupled, so that the underlying logic can be tested on its own?
- Are there specific testing problems (e.g. Jython code, access to database server, GUIs)?

C Review panels

C.1 IA packages including lines of code

Currently each IA package has the following lines of code (simple wc). An R indicates that the package was reviewed one year ago. Bold face indicates that this package will be reviewed.

herschel.ia.cal	794	
herschel.ia.classloader	**1185**	
herschel.ia.dataflow	**12737**	
herschel.ia.dataset	**10438**	**R**
herschel.ia.demo	588	
herschel.ia.doc	6	
herschel.ia.help	**2434**	
herschel.ia.image	**10747**	
herschel.ia.io	6705	R
herschel.ia.jconsole	**9301**	**R**
herschel.ia.numeric	**70240**	**R**
herschel.ia.plot	**13091**	
herschel.ia.task	**3830**	
herschel.ia.testbed	387	
herschel.ia.ui	**2038**	

C.2 review panels

C.2.1 major panels (2 sessions)

- plot
- image and dataset
- task and dataflow
- ui and help

C.2.2 normal panels (1 session)

- classloader and sandbox

- numeric
- JConsole
- system and infrastructure

C.2.3 Organisation of the review

As the composition and schedule of the panels might need several updates, it will be subject to a separate note.

IA Review

Kick-off

Stephan Ott

Some words at the start …

- Welcome & thanks to Rik for hosting and organising!
- Our chance to review the complete IA design to see whether we are on the right track for a launch 2007
- Concentrate on major issues
  - Does the system do what the user community needs (will need in the future)?
  - What requirements drive the system? Can we reasonably hope to implement them?
  - Is the package maintainable?
  - What other items do we want to bring to the attention of the Herschel Ground Segment Review
- Don't invent additional requirements
- Don't get bogged down in details
- Be honest and list problem areas and proposed corrective actions

Don’t redesign on the fly

Individual panel sessions

- Panel chairs are responsible to produce review reports
  - Which objectives were reviewed?
    - Did we comply with the objective?
    - Where do we see major shortcomings/problems, and which corrective actions are proposed?
  - Which objectives were not reviewed? Why? (no time, not critical, not applicable)
  - Which additional objectives were reviewed?
  - Was the preparation of the participants adequate?
  - Draft reports: Friday, 19th of November
  - Final reports: Friday, 26th of November
- Observers should not impede the review process

Appendix C

Review panels and schedule for IA review 16/17th November 2004

Version 1.5, 11/11/04

A.1 Review panels --- list of owners and deputies

A.1.1 major panels (2 sessions)

	package deputy	package owner
• plot	**Wetzstein**	*Lorenzani*
• image	**Huygen**	*de Meester*
• dataset	**Brumfitt**	*Bakker*
• task	**de Candussio**	*Mathieu*
• dataflow	**Corrales**	*Zaal*
• ui	**Guest**	*Kemp*
• help	**de Candussio**	*Kemp*

A.1.2 normal panels (1 session)

	package deputy	package owner
• classloader	**de Candussio**	*Mathieu*
• sandbox	**Mathieu**	*Porret*
• numeric	**Brumfitt**	*Bakker*
• JConsole	**Guest**	*de Candussio*
• system	n/a	
• infrastructure	n/a	

A.2 Review panels --- proposed chairs and defenders

Principle: A person can chair only one session. Preferably, the session chair is the package deputy, and the defender the package owner.

The loading should be balanced between all participants.

A.2.1 major panels (2 sessions)

package	**chair**	*defender*	reviewer[1]
• plot	**Wetzstein**	*Lorenzani*	Coeur-Joly, Dwedari, Kemp
• image dataset	**Huygen**	*de Meester* *Bakker*	Dwedari, Kester, Mathieu, [Rector], Zaal
• task dataflow	**Zaal**	*Mathieu*	Pizarro, de Meester
• ui help	**de Candussio**	*Kemp* *Kemp*	[Borkowsky], Coeur-Joly, Guest, Pizarro, Siddiqui, Wetzstein

A.2.2 normal panels (1 session)

package	**chair**	*defender*	reviewer[1]
• classloader sandbox	**Mathieu**	de Candussio de Candussio	Kemp, Lorenzani, de Meester
• numeric	**Kester**	*Bakker*	Dwedari, Guest, Wetzstein
• JConsole	**Guest**	*de Candussio*	Huygen, de Jonge, [Borkowsky]
• system infrastructure	**de Jonge**	*Guest*	Bakker, Huygen, Siddiqui

[1] a reviewer in brackets [ ] denotes that he will not participate in person, but will review the package remotely and send his inputs to the review chair and his instrument representative before the meeting

A.3 Sequence of review panels (3 parallel sessions)

First day (16/11/2004)

09:00-10:00	**10:00 – 12:00**	**13:00 – 15:00**	**15:30 – 17:30**
kick-off (plenary session) St.-Barbara	**task/dataflow I** **Zaal** *Mathieu* Pizarro, de Meester	**task/dataflow II** **Zaal** *Mathieu* Pizarro, de Meester	
	plot I **Wetzstein** *Lorenzani* Coeur-Joly, Dwedari, Kemp	**plot II** **Wetzstein** *Lorenzani* Coeur-Joly, Dwedari, Kemp	**Classloader / Sandbox** **Mathieu** *De Candussio* Kemp, de Meester, Lorenzani
	system / infrastructure **de Jonge** *Guest* Bakker, Huygen, Siddiqui	**JConsole** **Guest** *de Candussio* Huygen, de Jonge	**numeric** **Kester** *Bakker* Dwedari, Guest, Wetzstein

Second day (17/11/2004)

09:00-09:30	**09:30 – 11:30**	**11:45 – 13:00**	**14:00-16:30**
reflections of first day (plenary session) St.-Barbara	**image/dataset I** **Huygen** *de Meester, Bakker* Dwedari, Kester, Mathieu, Zaal	**image/dataset II** **Huygen** *de Meester, Bakker* Dwedari, Kester, Mathieu, Zaal	**conclusions** (plenary session) St.-Barbara
	UI / help I **de Candussio** *Kemp* Coeur-Joly, Guest, Pizarro, Siddiqui, Wetzstein	**UI / help II** **de Candussio** *Kemp* Coeur-Joly, Guest, Pizarro, Siddiqui, Wetzstein	

Appendix D

Reflections of the first day

Stephan Ott

Good morning!

- Thanks to Rik for the wonderful dinner!
- System panel asked for a second session. Can we accommodate the following (morning panels and conclusions reduced by 20/30 minutes, meeting lasts until 17:00)?
- During conclusions, panel chairs are expected to present highlights of their panels (3+2 minutes for normal panels, 6+4 minutes for major panels)
  - Did to manage to cover what you intended?
  - Major positive conclusions
  - Major shortcomings/problems
  - Major corrective actions
  - Any other information of general importance

Previous schedule

09:00-09:30	09:30 – 11:30	11:45 – 13:00	14:00-16:30
reflections of first day (plenary session) St.-Barbara	**image/dataset I** **Huygen** *de Meester, Bakker* Dwedari, Kester, Mathieu, Zaal	**image/dataset II** **Huygen** *de Meester, Bakker* Dwedari, Kester, Mathieu, Zaal	**conclusions** (plenary session) St.-Barbara
	UI / help I **de Candussio** *Kemp* Coeur-Joly, Guest, Pizarro, Siddiqui, Wetzstein	**UI / help II** **de Candussio** *Kemp* Coeur-Joly, Guest, Pizarro, Siddiqui, Wetzstein	

Suggested schedule

09:00-09:20	09:20 – 11:00	11:15 – 12:30	13:30-14:45	15:00-17:00
reflections of first day (plenary session) St.-Barbara	**image/dataset I** **Huygen** *de Meester, Bakker* Dwedari, Kester, Mathieu, Zaal	**image/dataset II** **Huygen** *de Meester, Bakker* Dwedari, Kester, Mathieu, Zaal	**System / infrastructure II** **de Jonge** *Guest* Bakker, Huygen, Siddiqui + panel chairs St.-Barbara	**conclusions** (plenary session) St.-Barbara
	UI / help I **de Candussio** *Kemp* Coeur-Joly, Guest, Pizarro, Siddiqui, Wetzstein	**UI / help II** **de Candussio** *Kemp* Coeur-Joly, Guest, Pizarro, Siddiqui, Wetzstein	**TBC** Florin	

Appendix E

Concluding session

Stephan Ott

Organisation of the final session

- **Presentation of panel chairs**
  - Major positive conclusions
  - Major shortcomings/problems
  - Major corrective actions
  - Did to manage to cover what you intended?
  - Any other information of general importance
- **General discussion**
  - What do we suggest as priorities for IA#7?
  - What needs additional discussion in the IA WG?
  - What do we want to bring to the attention of the IA documentation review?
  - What do we want to bring to the attention of the CSDT?
  - What do we want to bring to the attention of the Herschel Science Ground Segment review?
- **Round table**
  - Was the review worth the efforts spent?
  - What can be improved for the next review?

Let’s go!

- **Presentation of panel chairs**
  - M. Wetzstein (plot)
  - R. Huygen (image/dataset)
  - D. Kester (numeric)
  - J.-J. Mathieu (classloader/sandbox)
  - P. Zaal (task/dataflow)
  - N. de Candussio (UI/help)
  - S. Guest (JConsole)
  - A. de Jonge (System)

Appendix F

B Conclusions of Package Review

Name of package reviewed: herschel.ia.plot

Panel composition:
chair: Wetzstein
defender: Lorenzani
reviewers: Coeur- Joly, Dwedary, Kemp
observer: Kester

B.1 Use cases/requirements

B.1.1 Does the subsystem satisfy the relevant use- cases and user requirements?

B.1.2 (Note: Use- cases are mainly applicable to applications, rather than utility libraries.)

Reviewed:
yes
conclusions and proposed actions:
Most user requirements are fulfilled by the PlotXY class. Exceptions and how they are treated are listed below:

- 3D plotting is not available now
  Reasearch is being done on available 3D packages.
- Histogram plotting
  jfreechart provides the required functionality. Time is allocated for the developement.
- Adding graphical annotations to a plot like arrows and user defined graphical decorations for certain points
  A way to add graphical annotations has been found. They could be defined within a graphical toolbox and be added as datasets. Already implemented Plot functionality could be used to draw the items. This has the advantage that the graphical annotations will automatically be scaled together with the plot.
  The Review board expressed doubts that the level of professionality of this solution can compete with the functionality of a pro graphical package like Photoshop that could be used to decorate a plot after it

has been saved as an image. Agreement has been achieved that developement of graphical annotations should not be done in this case.

More missing functionality at the level of SCRs:

- User interaction with the plot
  User want to define regions of data in the plot, export the data into ia, modifiy and re-import them into the plot.
  The export functionality has been done in a first draft. A group of datapoints can be selected with a mouse and exported to ia. Re-import and more comfortable selection methods have to be implemented.
- A Layer object still has two identifiers: The layername and an internal integer id. This leads to confusion, especially if the layername is changed in a session.
  The solution that has been found by the board is that only an internal string or integer id should be used. Several methods to display the internal id have been discussed
  - click on the legend toggles between the internal ids and layernames
  - define in the preferences if the id is always shown together with the layername in the legend but never printed
  - show a list of available ids by applying an easy method like „p.layers“
  - show a list of available ids by typing the appropriate methods like getLayer(+ TAB
- Have more than one symbol to plot on a layer, ideally a special symbol for every point of the layer
- The Legend should contain the symbol that is used on the layer

As can be seen from this list, the open requirements are very special and detailed. So most of the required functionality is implemented.

B.1.3 What are the use-cases/requirements that are expected to hold major development difficulties for this package? Are they clearly specified?

B.1.4 (Note: Currently I noted: stopping of tasks, adding of debugging features, handling of big datasets, reported performance problems, IDLizing, saving of intermediate data)

Reviewed:
yes
conclusions:

No problematic requirements have been identified. The most complicated developement would be graphical annotations that can compete with professional graphic packages.
proposed actions:
A solution how to implement graphical annotations has to be found. To see how professional we can do it, we should implement a simple example and make a time estimate. Users input is also required here.

B.1.5 Does the package have a high number of SCRs? Are the modification of use- cases and additional use- cases?

Reviewed:
yes
conclusions:
The package has had 20 SCRs overall where 7 have been open at the day of the review. They all contain fine tuning of the package and don’t need significant developing time.
proposed actions:
none

B.2 Architecture

B.2.1 What is the "raison d' être" for this package? To which currently existing use- cases is it linked? Do we need additional usecases to specify the most important functionality?

reviewed:
yes
conclusions:
Raison d' être is obvious.
Usecases for graphical annotations are required if they should be implemented.
proposed actions:
User groups should discuss if they want graphical annotations, if they want an example implementation first or if they don't need it. Then budget should speak.

B.2.2 Does the package fit coherently into the overall architecture of IA?

reviewed:
yes
conclusions:

Plot fits well into the architecture of ia. Its interfaces to ia are the jython console and ia datatypes.
Plot supports jython keyword service for the constructor and it accepts Double1d and TableDatasets as ia datatypes.
Further extension of the accepted datatypes can and will be done by using services of the numeric library. Acceptance of TableDatasets on the other hand will be moved to an outside plotting service for datasets.
Further discussion with the maintainer of the numeric package is necessary because Datasets are the carriers of quantities. Removal of plot service for TableDatasets would thus also mean removal of quantities if not another interface can be found. This is an open issue since the beginning of plot developement.
proposed actions:
Extend accepted datatypes to numeric Array1dData, solve the problem with quantities on the level of the numeric package, then remove Dataset plotting from PlotXY.

B.2.3 Is there an architecture and design document, including:

- Subsystem decomposition
- Design notes, explaining design decisions
- UML diagrams, where appropriate, to illustrate design features

reviewed:
yes
conclusions and proposed actions:
Architecure and design documents are available and have to be updated after the refactoring of the package.

B.2.4 Is it a component architecture, with cleanly decoupled components?

Reviewed:
yes
conclusions:
It is a component architecture, with cleanly decoupled components .
However, the decoupled design has become less clear after user requirements to access Axis properties through the Layer object have been implemented.
proposed actions:
This has been tackled by the package redesign discussions (see design section). An abstract class has been proposed that contains the axis accessor methods an that can be extended by multiple other classes.

B.2.5 Does it have understandable interfaces?

reviewed:
yes
conclusions:
It has understandable interfaces.
proposed actions:
none

B.2.6 Does the API hide the underlying implementation (i.e. provide suitable abstraction)?

reviewed:
yes
conclusions:
Besides public methods there are also a lot of protected methods especially in the PlotXY interface. This has puzzled users who consulted the javadoc as a reference documentation for the public api of plot because also protected methods are shown in the javadocs.
One of the refactoring goals is thus to develop a structure where user accessible classes will only have public or private methods to keep the api documentation clean.
But, in terms of hiding the underlying implementation protected methods are a suitable way to go. Thus the answer to the question is yes.
proposed actions:
package redesign (see design section)

B.2.7 Are there public classes/methods which should not form part of the public API?

- Conversely, are there classes/methods that should be made public?
- Is it clear which packages form part of public API?

reviewed:
yes
conclusions:
The answer is given in B.2.6. The public api should be better localized after refactoring.
proposed actions:
none

B.2.8 Does it have direct interfaces with other subsystems or other libraries? Is it incorrectly/overly coupled to other packages? Are there circular dependencies?

reviewed:
yes
conclusions:
no problems here.
proposed actions:
none

B.2.9 Is the subsystem decomposed into packages? Is this decomposition sensible, or a cause of user irritation and/or increase in complexity?

reviewed:
yes
conclusions:
The package has a very flat design now. With increasing complexity of the software this has to be changed.
proposed actions:
package redesign (see design section)

B.2.10Do you have suggestions on the improvement of the architecture or design (E.g. would this package benefit from features introduced by JDK 1.5)?

reviewed:
yes
conclusions and proposed actions:
No benefits by introducing jdk 1.5 could be found. Design improvements see design section

B.2.11 Are there new external packages that are better suited to provide the underlying services?

reviewed:
yes
conclusions:
JFreechart is still decided as the best known choice.
proposed actions:
none

B.2.12 Would it be worthwhile to make the package open source?

reviewed:
yes
conclusions:
Other groups might benefit from Plotxy. But PlotXY can only be made open source if numeric and jide are made open source because of the close integration of plot into the ia environment.
proposed actions:
none

B.3 Documentation

B.3.1 Is there a user manual for the package? Does it have a document number? Should it have one?

reviewed:
yes
conclusions:
Plot has a HowTo but no complete user documentation. During the CodeReview a complete user documentation was requested.
proposed actions:
The results of the documentation review will show which way the whole system should go.

B.3.2 Is there adequate developer's documentation? Does it have a document number? Should it have one?

reviewed:
yes

conclusions:
There is a design documentation in the package.html which serves together with the javadoc as a developer documentation
proposed actions:
none

B.3.3 Is there adequate design documentation? Does it have a document number? Should it have one?

reviewed:
yes
conclusions:
As mentioned before there is a design documentation in the package.html which serves together with the javadoc as a developer documentation
proposed actions:
The design documentation will be updated after the redesign of the package.

B.3.4 JavaDoc

- Does the JavaDoc follow the Sun guidelines?
  - Does each method have an understandable JavaDoc description?
  - Is the general level of JavaDoc comments adequate (especially for public API)?
  - Does the JavaDoc include code examples, where appropriate?
- Does each subsystem have a top- level "package.html" file, including:
  - An initial summary sentence documentation, etc
  - A description of the contents and purpose of the package
  - A "Package Specification" section (may be blank)
  - A "Related Documentation" section with hyperlinks to design

reviewed:
yes
conclusions:
The javadoc is complete and follows the Sun guidelines. Some new methods are still undocumented.
proposed actions:
Ongoing developement should document also new methods. Its a good working style to write at least one line of description before implementing a method to clarify its purpose.

B.3.5 Where is the documentation located?

reviewed:
yes
conclusions:
In the HowTo section, javadoc and package.html.
proposed actions:
none before the documentation review.

B.3.6 Does the subsystem have a readable CHANGELOG file?

reviewed:
yes
conclusions:
no problems here
proposed actions:
none

B.4 Design

B.4.1 Are there any indicators of design problems (code metrics, test coverage, high number of SPRs)?

reviewed:
yes
conclusions and proposed actions:
The most obvious design problem of the PlotXY class is the number of lines (around 5000). Thus the main goal of the code review was to reduce the size of the PlotXY class.
This has been done by discussing a couple of designs and package structures. The goal is still to be able to use multiple plot packages together with the plot framework (like jfreechart, maybe a new 3d package or jsky for image).
Andrea Lorenzani has taken the action to discuss the pros and cons of the discussed designs and make a proposal for the best redesign of the package.
The redesign will also solve problems associated with the protected classes in the javadoc. That has been puzzling for users who use the javadoc as a reference documentation.

A couple of longer methods need a few lines of comments. Refactoring of these methods was not decided to be useful.
Refactoring of methods has been recommended on very few methods like public Layer addLayer(PyObject[] args, String[] kws) or draw(...) in the PlotXYCompositeRenderer. These two methods have been considered too long to maintain even with additional comments.

B.4.2 Is the package easy to use? If not, why?

reviewed:
yes
conclusions:
no problems have been mentioned
proposed actions:
none

B.4.3 Are appropriate Design Patterns employed?

- Should further design patterns be introduced to improve abstraction, flexibility, modularity, etc?
- Conversely, are inappropriate design patterns employed?

reviewed:
yes
conclusions:
no problems here
proposed actions:
none

B.4.4 Is there a proper separation of concerns between classes and between packages?

reviewed:
yes
conclusions:
with the exception of the coupling of Layer and Axis no problems have been found.
proposed actions:
see B.2.4

B.4.5 Is the design properly Object Oriented?

reviewed:
yes
conclusions and proposed actions:
The plot package has well defined classes with well defined responsibilities.
However, the object oriented design has become less clear after user requirements to access Axis properties through the Layer object have been implemented. It is questionable (and maybe confusing for the users) if axes should belong to layers.
The package redesign should also correct this problem and make axes belong to the overall plot without any ambiguity how to adress an axis.
This has also been tackled by the package redesign. (as mentioned in B.2.4)

B.4.6 Is the principle of operation sound?

reviewed:
yes
conclusions:
no problems here
proposed actions:
none

B.4.7 Might the design lead to performance problems?

reviewed:
yes
conclusions:
Plot has no real performance problems if hardware acceleration for 2d graphics is supported by the system. This can be a problem of linux systems due to the lack of appropriate drivers, especially for notebooks. On Windows and Macintosh systems drivers are usually available.
Nevertheless it has been discussed that plot could perform faster if the shutdown of the plot engine would not be coupled to the closing of the plot window. Instead the engine could remain in memory and support rendering of new windows on request.

proposed actions:
The ideas to increase performance and usabiliy should be documented.
The driver problem is related to the system in general (every qla will have this problem). So a central point for documenting this should be found.

B.4.8 Is reflection used unnecessarily?

reviewed:
yes
conclusions:
refection isn't used at all.
proposed actions:
none

B.4.9 Is inheritance used in an appropriate way? These checkpoints should be true wherever inheritance is used:

- It is a "kind of", not a "role played by", relationship.
- The object never needs to transmute into another class.
- It extends rather than overrides or nullifies the superclass.
- It does not subclass what is merely a utility class.

reviewed:
yes
conclusions:
no problems here
proposed actions:
none

B.4.10 Would composition be better than inheritance?

reviewed:
yes
conclusions:
jfreechart is accessed with composition. This has been decided as correct.
proposed actions:
none

B.4.11 Should any classes be immutable?

reviewed:
yes
conclusions:
no problems have been found here
proposed actions:
none

B.4.12 Is error handling adequate?

- Are there any 'exit' statements (except in "main" method)?
- Are the error and log messages understandable and appropriate?
- Is exception handling employed for all exception / error conditions?
- Are the exceptions thrown misleading or not the right ones?
- Should the exception be thrown at all?
- Should the exception be checked or non- checked?
- Can a situation be envisaged where the code will obviously fail?

reviewed:
yes
conclusions:
It has been mentioned that the javadoc shows that PlotXY throws no exceptions. This could be useful for error propagation if other packages use PlotXY.
More user friendly error messages have been required. This is a problem of ia/hcss in general. Discussions on how to handle errors and make them usable for users and developers are right now going on in the user groups.
proposed actions:
Throwing of exceptions should be implemented. User friendliness of exceptions requires at least two levels of error detail (for users and developer). This is achieved by the logging system now. Anyway, users are discussing this problem and will come up with a proposal for the whole ia system.

B.4.13 Does the design allow the subsystem to be properly tested?

- Is a pluggable architecture needed to support test stubs?
- Is the GUI decoupled, so that the underlying logic can be tested on its own?
- Are there specific testing problems (e.g. Jython code, access to database server, GUIs)?

reviewed:
yes
conclusions:
no problems here. But the graphical output cannot be tested so far.
proposed actions:
none

B.5 Summary/other comments

B.5.1 What are the main conclusions of the review panel? How do you rate the state (green/amber/red)

After the necessary redesign there are no problems for the plot package. Since it was the first plot review, its a good sign that so few problems could be found.
So very green light is the overall result.

B.5.2 What are the main recommendations of the review panel?

Redesign of the package with the main goal of better package structure, smaller classes and a clear separation of the public api.

B.5.3 Were additional points reviewed? If so, please list them, including conclusions and proposed actions

The use of class private fields in the constructors is not well documented
conclusions:
There is a clear rule: All fields that are used in the complete constructor can be used from the jython commandline in the user required way (PlotXY(y=myYData)) .
proposed actions:
Put this rule into the HowTo (and user documentation).

Appendix G

```
IA Code Review 16-17-Nov-2004 @ K.U.Leuven
------------------------------------------
Session: ia.dataset
Chair: Rik Huygen
Package Owner: Jorgo Bakker
Reviewers: Lutfi Dwedari, Do Kester, Jean-Jacques Mathieu, John Rector,
           Peer Zaal

General comments by reviewers at the start of the meeting

JJM - What exactly is the link between dataset and the rest of the system? How
      and where does a dataset live in the system? E.g. SpectrumDataset, where
      will this reside? ia.spectrum? like ImageDataset is in ia.image?. Other
      dataset instances, where will they reside?

DK  - use of quantities in TableDataset? Columns have quantities and we have to
      do something with it. Maybe we need a framework?

    - metadata? We should be able to have small sets of array type data. For
      example SpectrumDataset with some calibration data which can go into
      something like an ArrayParameter, on the other hand you could make a
      CompositeDataset where the calibration data goes into a ArrayDataset.

PZ  - Dataset is providing the basic building block for Product but has no
      documentation on how to make this Product persistent in your database or
      on its relation with other objects in the CCM environment.

Some answers to the suggested list of questions for package review

Use cases and user requirements

1.1 Does the subsystem satisfy the relevant use-cases and user requirements?

    We do not have use cases specific for dataset. Also the user requirements
    are not written out as such. How is ia.dataset then designed? Against which
    use cases or user requirements? The package was prototyped against the need
    for these data containers. There might be a need to retrofit to write the
    use cases.

1.2 What are the use-cases/requirements that are expected to hold major
    development difficulties for this package? Are they clearly specified?

    There are a number of additional user requirements:
      - access to a column
      - same syntax as in ia.numeric i.e. remove the .data
      - merging/sorting TableDataset
      - insertion of rows
      - ArrayParameters as metadata

    Performance issues? JBa: in principle there are no performance problems with
    dataset. Some specific methods might cause problems like appendRow() in
    TableDataset because these methods need reflection.

    Clarification by JBa: some methods (like appending) were required for
    convenience only. The design does not stop you to do it in a more optimized
    fashion.

1.3 Does the package have a high number of SCRs?

    Here we discussed the contents of a number of SxRs rather then answering the
    question.

    - the calling syntax for TableDataset and arrays in numeric is different
      which might confuse the user.
```

tds[1][3] versus array[1,3] where the former is a TableDataset and the latter is ArrayData. The tds call returns the third row in the first column while the array returns the third element in the first row.

It was also mentioned that a TableDataset **is not an** array and therefore it was questioned if we should even try to mimic the calling syntax of an array.

Drop the column number option and only address columns by name? This is against a user requirements where it should be possible for two columns to have the same name. It might also give a slight performance overhead when all columns need to be addressed.

Do we need an iterator over the columns then? Could this replace the index for columns?

- User friendlyness, getting rid of the .data part when addressing Arraydata in Columns. This might be solved by implementing the __getitem__ method on Column.

Architecture

2.1 What is the "raison d' être" for this package?

Obvious.

2.2 Does the package fit coherently into the overall architecture of IA?

What about History?
From the Javadoc: An empty slot for History. In the end we should remove this one, and use whatever becomes available in the herschel.ia tree.

Where is History now? Is this still a requirement? Where will it be picked up? According to JBa History is not a dataset specific thing. The system group should decide who is reponsible for history and in what package it will reside, do we need a specific history package etc.

2.3 Is there an architecture and design document?

There is currently no ADD as such. There is design documentation though in the overall package documentation but no specific document that contains the architectural design. Some reviewer had therefore problem finding this documentation...for clarification the URL to the design document is:

ftp://ftp.rssd.esa.int/pub/HERSCHEL/csdt/releases/doc/ia/dataset/doc/design.html

2.4 Is it a component architecture, with cleanly decoupled components?
2.5 Does it have understandable interfaces?
2.6 Does the API hide the underlying implementation?

There is a dependencies of ia.dataset.gui to intepreter and circular to plot/image!

There is also a dependency of ia.io to ia.dataset wrt inheritance, might be just a documentation issue.

Bart brings up the open SPR on Listeners on datasets. This is in the pipeline to be implemented, planned for iteration #7.

2.7 no

2.8 Does it have direct interfaces with other subsystems or other libraries?

ia.dataset.gui.DatasetInspector has a number of dependencies to PlotXY, Display.

2.9 Is the subsystem decomposed into packages?

no, except for the gui package

After Meeting remark by JBa: I rendered this question useless unless you put it into the context: "Is the package decomposition suitable".
Currently we have (no documentation is considered):

```
    dataset/demo
    dataset/gui
```

Whether gui should be part of dataset or somewhere else, was moved to system panel.

All tests are currently within the package itself. This should have been split into testing package private stuff and a separate test sub-package.

2.10 Do you have suggestions on the improvement of the architecture or design?

Unit tests are in the package itself. Why isn't there a separate test sub-package? The current location of the tests is to be able to test package private issues. Does this mean it doesn't test the public interface?

2.11 Are there new external packages that are better suited to provide the underlying services?

no

2.12 Would it be worthwhile to make the package open source?

In principle yes, but we think the package will have limited audience.

Other questions on the architecture

- is the column design a performance problem for row access?

  no performance issue really if you index in the Column Data

- responsibility on quantities - where and how are quantities handled in the IA?

  see above, this is passed on to the system architecture group.

- How to handle Datasets that are larger than available memory?

  Do we need to handle that at the level of Dataset, provide slicing etc.?

Documentation

Overall documentation is fine and very elaborate. The structure however could fit better into the overall IA documentation. This can only be done after the IA documentation review and when clear guidelines are available on IA documentation. Users and devlopers should send their comments on documentation to Jorgo.

There is a need for a consistent view on where the architectural design document etc are located, restructuring is needed.

Conclusions

* rate: GREEN, provided that the recommendations of this panel are implemented and/or discussed and documented.

* Investigations needed
  - location of derived Datasets
  - use of Quantites
  - extentions to metadata

* Actions

```
    - use and documentation of Product needs updating
    - take care that the existing SxRs are solved e.g.
      --> remove .data part
      --> addressing columns
      --> listeners for datasets
    - dataset.gui: solve dependencies

System items identified which should be picked up by a IA System Architecture
Group:

  - How and where does a dataset live in the system?
  - responsibility on quantities, where and how are quantities handled in the IA?
  - History
  - SpectrumDataset and its View
    ImageDataset and its View (Display)

--- appendices ---

Appendix A:

Review of IA dataset
JJ Mathieu
11 November 2004

Following list for package review (appendix B)
==============================================
1. Use cases/requirements
-------------------------
1.1 Not relevant. Library subordinated to numeric and needs to store
    data.
1.2 Requirements not considered (yet) which may have an impact:
    - 'Easy' retrieval of specific data off a database
    - Integration of products in the system.
      - Should they be outside of dataset or within?
      - Who then assures maitenance, garantee cohesion, uniformity...
      - If products are meant for DB and IA should they be here?
      - Other?
    - Large dataset handling? Is this really a dataset problem or a
      data design problem (or a DB schema problem)?
    - Performance of access (adding layers to 'true' data). Is this
      really a requirement that is problematic?
1.3 - Statistics say that we 17 S?R and 11 are still open. The system
      is fairly stable since about three months. What is remarkable
      is that the S?R are mostly very recent. Either the users are
      beginning to use the dataset or start to use it without having
      had a needed understanding of what the package is about.

Overall assessment GREEN

2. Architecture
---------------
2.1 Clearly package is subordinated to numeric in its design. What
    is less obvious are:
    - The needs it fulfills for science products as such (versus
      defining these with standard java classes) since the package does
      not seem to offer much functionality.
    - The apparent contradiction or hazy aspects linked with 'quantity'.
      It is stated that the package provides quantifying of data. But in
      SCR 999, for example, it is stated that numerics will not handle
      quantities. This issue has a long history because of this ambiguity.
2.2 Not quite sure. It could be under numerics, on the other hand it is
    functionally at the same level. It would be good to know also how and
    where to integrate specialised data sets (like spectrum or image). In
    the absence of a written and approved IA overall architecture we have
    to trust that the current architetcure is adequate.
2.3 There is no architectural document but design documents which are
    not explaining much in terms of how the design came to existence.
    This results in the remarks made in 2.1 and 2.2, also this would
```

    mean that to transfer knowledge to another maintainer would need
    human coaching and not just pointing at documentation.
2.4
2.5 The interface definitions are mixed in with the implementation which
    makes it difficult to spot 'easily' who is what. Even though there is
    a documentation for developer explaining how to extend the dataset
    package it is actually more a recipe for building classes by
    composition. It does not seem that the design is made for inheritance
    and therefore the issue of interface might not be relevant.
2.6 The interfaces indeed hide implementation which is delegated to
    Abstract... classes.
2.7 Probably OK, java language here plays a more important role than one's
    architectural or design wishes.
2.8 Using dataset are demo, image, io and plot. This means we do have
    circular dependancy with respect to image, io and plot packages.
2.9 Package could have followed the subpackage standard split with api
    implementation and test for instance. This may only cause a bit
    of confusion to developers.
2.10 Split properly into subpackages (api and test at least). Probably
     some of the classes (who are helper classes like the Abstract...)
     would then be hidden (in an implementation package).
2.11 No
2.12 No

Overall assessment ORANGE
justification: seems architecture is generally missing or has been thought
of as you develop rather than beforehand.

3. Documentation

3.1 A user guide also looking like a how to guide. Mainly geared towards
    jython usage, which is ideal since the java usage would appear (does so)
    in a developer's guide.
3.2 Developer documentation is missing a big point: where do you put your
    derived dataset. For instance image is a derived dataset (I think) and
    ends up in its own package while (may be) spectrum would end up in
    dataset. Which one is the law? Do we follow a casuistic approach?
3.3 Design documentation is inadequate to ensure an 'easy' takeover by
    another maintainer. Who does have the time to improve this?
3.4 No comments.
3.5 Within package in doc.
3.6 No comments.

Overall assessment GREEN

4. Design

4.1 Nope. I guess the unit test is not completed yet (for instance missing
    serialisation test).
4.2 Yes.
4.3 Probably. Apart from visitor none is really explicit though.
4.4 Yes
4.5 Yes
4.6 Yes
4.7 Possibly on queries in the database.
4.8 No
4.9 Yes
4.10
4.11
4.12 Yes
4.13 Yes

Overall assessment GREEN

Overall assessment against IA review procedure appendix B GREEN

Points noted by Rik

```
Architecture
------------
- I think it can be used as a generic dataset package. But may be
  the problems of dependancies (circular) have to be resolved or
  at least documented well enough for all to be aware of a potential
  problem.
- May be the issue is 'When I make my own specialised dataset, should
  I put it in its own package (image for example) or inside dataset?
  The ia.io issue is addressed by the point above.
- I am not sure this is an 'interesting' question. If the choice of
  implementation had been on 'row' mode, I am sure someone would have
  asked about adding a column.
- See above
- See remarks above. I am very uneasy with the quantity question. Also
  I think if any the processing of quantity (I'd rather say units)
  ought to be in numeric not in dataset. Dataset should only provide
  methods to alter the current unit, numeric should be the user. I
  believe this is what we have now, and in this sense this is OK.

Overall assessment ORANGE
Justification: Seems tha the package has been (at least partially)
developed on practical track and not on fully organised and structured
development. It may have suffered also from the lack of architecture at
the top level of IA (ia.io, ia.image, ia.dataset dependancies)

Documentation
-------------
- I think most of the needed documentation is adequate even for
  java development. May be looking into demo code would help
  developers; this could go into a How to for developers (there is
  an how to for users which elaborates on dataset building in
  jython).

Assessment GREEN

Usability
---------
- What we are after is a generic mechanism. Like any generic mechanism
  the visitor pattern may be heavy in use, but it offers a great way
  to crawl over structures (by definition their composition is unknown
  or rather varying). I think good examples of specialised dataset with
  their visitors leading to lightweight use in jython would be useful.
- I think the design covers the different types: homogenous arrays,
  heterogeneous tables and scalars. Only ragged arrays ans sparse
  matrices are missing.
- The request came often, may be a serious candidate for updates.

Assessment GREEN

Extensions
----------
Extensions of dataset should be very clearly defined. For instance
sorting could be left to numeric (or not). How to decide.

Dataset inspector introduces a dependancy wrt plot (image?) which
may not be justified (The question being of the nature of inspection
may not be the nature of displaying).

Miscellaneous
-------------
Together with numeric this package could be subject to many kinds of
conflicting requirements which all boil down to the following request:
'Make your system look like my favourite XXX but also make sure it
is more generic and adds YYY and ZZZ facility'. Clearly we have to
avoid this (within reason). What triggers this is the periodic
resurgence of: the unit questions and why dataset does solve it on its
own, the access to the inner side of data structure (and its
```

corollary the non-casting issue). Finally a point that also comes often is that the developed system is too complicated; that is probably the case, on the other hand if people expect only to do arithmetic at the toddler level, then yes it is too complicated, if they expect to do science, then I think the system is probably as complicated as science but no more.

May be these last comments also reflect what we could say at system level.

Appendix B:

Rik Huygen

Hi All,

Apart from the general review guidelines below some review objectives you might want to concentrate on while you dig into the ia.dataset package. As the package has been reviewed before for design and architecture please focus more on usability and extendability.

* Architecture

  - are all components there to make this a generic dataset package?

  - can the package easily be extended for specific (package external) dataset types? and (how) does this possibly break export with ia.io?

  - does the column oriented design put major performance restrictions on the row accessibility required for some applications?

  - isn't there too strong coupling of this package with ia.numeric and ia.io?

  - does the package have any responsibility in the use and processing of Quantities?

* Documentation

  - documentation is very Jython tuned. Developers using Datasets from Java are left a bit in the dark. How to improve this?

* Usability

  - again Jython versus Java usage. Especially the calling mechanism for generic element access is cumbersome in Java through the use of the visitor pattern.

  - metadata and parameters: are they sufficient for the intended purpose? Do we need other parameter types e.g. ArrayTypes?

  - Can we get rid of the .data?

* Extensions

  - there are plenty of thing users want to do with Dataset, should such new functionality be implemented inside the package or as external utilities (Tasks?)
    - sorting tables by a particular column
    - joining/merging tables
    - ...

* miscellaneous

  - anything on the DatasetInspector?
  - anything else you find important to address...

```
* noticable SxRs

  - SCR-0786: Individual element access for Datasets
  - SCR-1013: User friendliness of Dataset
  - SPR-1143: A TableDataset is a collection of columns ALL OF
              THE SAME LENGTH.
  - SCR-1145: a 'type' field for clearly identifying Products
```

Appendix H

```
IA Code Review 16-17-Nov-2004 @ K.U.Leuven
------------------------------------------
Session: ia.image
Chair: Rik Huygen
Package Owner: Wim De Meester
Reviewers: Jorgo Bakker, Lutfi Dwedari, Do Kester, Jean-Jacques Mathieu,
           John Rector, Peer Zaal

General comment

Extensive input was received by Jorgo for this package review. This input is in
HTML format and is attached as a tar-file. The notes presented here are largely
based on the input from Jorgo.

Use-cases / requirements

1.1 Does the subsystem satisfy the relevant use-cases and user requirements?

    No use-cases. The developer has collated the user-requirements into a
    document which is available through the API doc (package level).

    There is no implementation or documentation of the following functionality.
      - Contour plots
      - Cursor feedback, how does the User gets access to it?
      - Animation, is this supported (and how)?
      - Addition of 2d-plots (PlotXY)

    Note on Animation: Showing different images after each other is already
    implemented. It is possible to show one image after another by clicking in
    the gui or using a java method.

1.2 What are the use-cases/requirements that are expected to hold major
    development difficulties for this package?

    no.

1.3 Does the package have a high number of SCRs? Is there a need for
    modification of use-cases and additional use-cases?

    The functionality provided in the User Requirements seems to fulfil the
    user's needs. Package is under construction. We could only identify two
    additional/complementary requirements:
      - coordinate axes
      - True color display

Architecture

2.1 What is the "raison d' être" for this package? To which currently existing
    usecases is it linked? Do we need additional usecases to specify the most
    important functionality?

    There is no user-friendly alternative around. It would be best if an effort
    is made to convert current functionality into a use-case document, as that
    will make it visible whether certain functionality is completely
    implemented.

2.2 Does the package fit coherently into the overall architecture of IA?

    The package seems to be a mix of display rendering, display manipulation and
    dataset definition. One could argue for a separation of these concerns.

2.3 Is there an architecture and design document?

    An Architectural Document is found via the package API documentation.
```

2.4 Is it a component architecture, with cleanly decoupled components?

OK!

The location of some of the components is maybe not optimal, i.e. Tasks, Datasets and Display are all located in the same package. The ImageDataset definition and the specific Tasks might better go into specific packages.

2.5 Does it have understandable interfaces?

Yes. Some methods are cluttering the interfaces, but they can be phased out (see improvements section from input by Jorgo)

2.6 Does the API hide the underlying implementation?

ImageDataset is encapsulating properly. The WCS class more resembles structure.

2.7 Are there public classes/methods which should not form part of the public API?

The improvements section in the input document from Jorgo contains a detailed analysis of public versus private classes/method.

2.8 Does it have direct interfaces with other subsystems or other libraries? Is it incorrectly/overly coupled to other packages? Are there circular dependencies?

The package is coupled to various sub-systems of IA:

```
  dataset : definition of image dataset
  numeric : contents of image dataset
  task    : for display manipulation
  plot    : for internal purposes
  ui      : for gui components
  jconsole: some gui utilities
```

Note on jconsole coupling: The only thing that is used from jconsole is JIDEUtilities. Other components are not used.

Many couplings are unavoidable, though the jconsole coupling might be something that can be phased out. That is, jconsole apparently has components that are useful beyond jconsole package itself. Hopefully these components become part of a utility package like 'ui'.

2.9 Is the subsystem decomposed into packages? Is this decomposition sensible, or a cause of user irritation and/or increase in complexity?

For the implementation, a single sub-package exists. This package (image.gui), contains gui components required by the Display class.

2.10 Do you have suggestions on the improvement of the architecture or design?

See section on improvements in Jorgo's input.

2.11 Are there new external packages that are better suited to provide the underlying services?

No knowledge of new packages that would do a better job.

2.12 Would it be worthwhile to make the package open source?

We see currently no reason to make this package open source.

Documentation

We did not have time to go over the documentation review and left this work for the IA Documentation review itself. There are however some comments in the input by Jorgo to keep in mind during this review.

From the architectural design documnet it is not always clear what the acronyms mean e.g. Wcs.

Other comments made during the session

- Can we expect unnecessary performance problems when handling huge datasets in the image package?

  We see no direct problem with this apart from the fact that huge datasets indeed need more processing time. There are no special optimisation strategies forseen to handle huge datasets.

- How much of the package is actually used? Are all instruments using this package?

  The package is used extensively by PACS, the use by HIFI and SPIRE is rather isolated.

- On the question if the package fits coherently into IA, we think that the data, model and the view should be better seperated, e.g. Display (view) - ImageDataset (data) - Tasks (operations on data).

- The definition of what exactly the ImageDataset is is maybe not really clear. Is it a generic image or more specific as a SkyImage? Should it be redesigned as such? SkyImageDataset inherit from ImageDataset?

- Is ia.image fully bound to JSky? Can another external library be plugged in?

- Wcs attributes seem to be a direct copy of FITS keywords. This might need a different approach as the keywords limit you to a linear transform.

- Interfaces of Axis, Layer, Annotations need to be shared with the plot package. --> system architectural item

- JBa found ImageDataset exactly reflecting the intension of sub-classing datasets. It might be worthwhile to put it as an example.

Conclusions

  * rate: GREEN, provided the recommendations of the panel are implemented
          and discussed/documented.

  * Investigations needed
    - how to make the WCS class less FITS/structure dependent

  * Actions
    - go over improvements section from input by Jorgo and make the necessary changes.

System items identified which should be picked up by a IA System Architecture Group:

  - How much of the system individual package is actually used by the users? Is there any means to measure that? Do the (number of) SCRs reflect this properly?

  - Axis, Layer, Annotations, Histogram interfaces should at least be shared, can there be common implementations and where should they reside?

  - GUIs used by Image? Like printing...

Appendices:

Appendix A:

Jose A.Pizarro De La Iglesia

General comment methods in this package are way too long and it is due a major refactoring exercise.

General Comment No effort is made to distinguish which exceptions are to be caught instead all exceptions are caught and then blocked or ignored . Some are logged but none are thrown up the chain.

Image .AxisGraphicsHandler.setWcs catches all exceptions (not just axis specific exceptions)
with out passing JVM etc exceptions up the chain.

+++

image.AxisGraphicsHandler.drawImageGraphics line 398 to line 1595

Please refactor this method as it is over 1000 lines long and contains a LOT of repetitive code.
It is unmaintainable as is.

+++
line 781 image.AxisGraphicsHandler.drawImageGraphics

```
//  Temporarily solution
          g.drawString(this._axesLabel[Axis.LEFT_AXIS],
                       x + 10 - this._spaceForAxes[Axis.LEFT_AXIS]
                       - this._extraSpace[Axis.LEFT_AXIS] +
                       this._labelFont[Axis.LEFT_AXIS].getSize() / 2,
ycoor);
```

Please provide none temporary solution
+++

image.Display.setImage please correct the Fixme comment

```
// FIXME : We cannot calculate the minimum and the maximum on the
      // imageDataset because the pixels that are masked out are taken into
      // account then. But we also cannot calculate the minimum and
maximum on
      // the image where the masked out pixels are NaN. This gives NaN as
      // minimum and maximum.
```

Please clarify how the pixels are masked out and what are you trying to fine the min max of ?

+++

image.Crop.execute() catches ALL exceptions that are thrown and does not restrict it self to just crop exceptions

+++

image.Translate.execute() catches ALL exceptions that are thrown

+++

image.Rotate.execute() catches ALL exceptions that are thrown
+++

image.Clamp.execute() catches ALL exceptions that are thrown
+++

image.Scale.execute() catches ALL exceptions that are thrown
+++

image.Transpose.execute() catches ALL exceptions that are thrown

+++

image.Histogram.execute() catches ALL exceptions that are thrown
+++

image.gui.ImageDisplayStatusPanel.update() catches ALL exceptions that are thrown

+++
image.gui.AnnotationToolbox(Display) catches ALL exceptions that are thrown and then ignores them
+++

image.gui.AnnotationToolbox.getJythonCode()

line 548 what does this method do if the thing that is to be annotated is not a line ?
+++

image.gui.AnnotationToolbox.Texthandler.mouseClicked()

line 772 inner class Texthandler creates multiple objects in the for loop

Appendix B:

Hi All,

Apart from the general review guidelines below some review objectives you might want to concentrate on while you dig into the ia.image package. This is a new package and should therefore be reviewed in detail.

* use cases/user requirements

  - these are accessible from the javadoc of the package in the section related documentation. What is the current status of the requirements coverage? Are there any major requirements still to be covered?

* Architecture and design

  - please check the general guidelines given in the IA Review document.

  - how generic is this package with respect to 'images'. The ImageDataset e.g. provides methods like getSkyCoordinates() and getWavelenght() which indicates a specific use/type for this Dataset.
    Is this the intention or do we need more generalization?
  - Are there classes functionality that could be shared with the ia.plot package? I'm thinking about things like Axis, Layer etc. or are these image specific.
  - can there be any performance/memory problem expected for huge Images?
  - how is the coupling of this package wrt packages like ia.numeric,

```
    ia.dataset, ia.plot?
  - Is the class Wcs specific for this package or has this a more general use?
  - this is mainly a GUI driven package, are all errors and exceptions
    properly thrown and clear and understandable by 'normal' users? Where do
    the messages appear?

* Documentation

  - Is the documentation clear and up-to-date?
  - does the documentation address all points in the user requirements/use
    cases?

* Useability

  - do you forsee problems with higher resolution screens (too
    small images ;-)
  - is all functionality from the commandline reflected in GUI actions?

* Extensions

  - nothing pops into my mind currently but I'm sure you have something
    here...

* noticeable SxRs

  - SCR-0902: ImageDataset should support non-celestial coordinate systems
  - SCR-1170: There should be a way to change the default color lookup
              table used by Display

Have fun!?
Rik


Appendix C:

Jorgo Bakker

 Display / import export (FITS) Separation of concerns?
- separate gui package, why?
- Display should go into image.gui
- tasks should go into image.manip ?
- Cursor feedback, what feedback?
- WCS is FITS

- public vs private/protected
  updateImage(...)



- Code:
  ImageDataset: void setErrors(Double2d errors, String description)
  **BAD
     if ( ( ( (Double2d) ( ( (ArrayDataset)
(this.get("image"))).getData())).getDimensions()[0]
          != errors.getDimensions()[0]) ||
        ( ( (Double2d) ( ( (ArrayDataset) (this.get("image"))).getData())).
         getDimensions()[1]
         != errors.getDimensions()[1])) {
  **BETTER
     if (!Arrays.equals(getImage().getDimensions(),errors().getDimensions()))
     {

  **BEST
     if (!isSameShape(getImage(),errors())
      ...

     private void isSameShape(ArrayData lhs,ArrayData rhs) {
       return Arrays.equals(lhs,rhs);
     }



  Rotate (Task)
  **BAD: large execute block
```

Use-cases / requirements
Use case compliance
Does the subsystem satisfy the relevant use-cases and user requirements? (Note: Use-cases are mainly applicable to applications, rather than utility libraries.)

No use-cases. The developer has collated the user-requirements into a document which is available through the API doc (package level).
I failed to find the implementation or documentation of the following. The developer may correct these findings:
  Contour plots
  Cursor feedback, how does the User gets access to it?
  Animation, is this supported (and how)?
  Addition of 2d-plots (PlotXY)
Killer use-cases
What are the use-cases/requirements that are expected to hold major development difficulties for this package? Are they clearly specified? (Note: Currently I noted: stopping of tasks, adding of debugging features, handling of big datasets, reported performance problems, IDL-izing, saving of intermediate data)

Change requests
Does the package have a high number of SCRs? Is there a need for modification of use-cases and additional use-cases?
Functionality provided in the User Requirements (see above) seems to fulfil the user's needs.
Architecture
Raison d' être
What is the "raison d' être" for this package? To which currently existing use-cases is it linked? Do we need additional use-cases to specify the most important functionality?
There is no user-friendly alternative around. It would be best if an effort is made to convert current functionality into a use-case document, as that will make it visible whether certain functionality is completely implemented.
Relation to architecture
Does the package fit coherently into the overall architecture of IA?
The package seems to be a mix of display rendering, display manipulation and dataset definition. One could argue that a separation of these concerns.
Design & Architecture documentation
Is there an architecture and design document, including: " Subsystem decomposition " Design notes, explaining design decisions " UML diagrams, where appropriate, to illustrate design features"
An Architectural Document is found in via the package documentation
Internal Coupling
Is it a component architecture, with cleanly decoupled components?
yes
Comprehensibility
Does it have understandable interfaces?
Yes. Some methods are cluttering the interfaces, but they can be phased out (see improvements section)
Encapsulation
Does the API hide the underlying implementation (i.e. provide suitable abstraction)?
ImageDataset is encapsulating properly. The WCS class more resembles structure.
Are there public classes/methods which should not form part of the public API? " Conversely, are there classes/methods that should be made public? " Is it clear which packages form part of public API?
Yes. (see improvements section)
External Coupling
Does it have direct interfaces with other subsystems or other libraries? Is it incorrectly/overly coupled to other packages? Are there circular dependencies?
It is coupled to various sub-systems of IA:

  dataset: definition of image dataset
  numeric: contents of image dataset
  task - for display manipulation
  plot - for internal purposes
  ui - for gui components
  jconsole - some gui utilities

Many couplings are unavoidable, though the jconsole coupling might be something that can be phased out.

Package Decomposition

Is the subsystem decomposed into packages? Is this decomposition sensible, or a cause of user irritation and/or increase in complexity?

For the implementation, a single sub-package exists. This package (image.gui), contains gui components required by the Display class.

Improvements

Do you have suggestions on the improvement of the architecture or design (E.g. would this package benefit from features introduced by JDK 1.5)?

Package Private 1

The image.gui package seems to contain gui components that are useful to the Display class only. One should consider whether these components should be package private and co-exist in the same package as the Display class is residing.

Package Private 2

Is the AxisGraphicsHandler required to be part of the public API?

Class Private methods

The Display class is revealing more than it should. The essence is to create a display of (layered) images; annotate and decorate (fonts, labels,...), and manipulation.

Example: updateImage(...).

Hiding implementation details

Display is implementing the ArrayDataVisitor. It seems that this should be part of the private implementation rather than becoming part of the public API. One could consider to hide this implementation detail with:

```
class Display {
   ArrayDataVisitor _visitor=new AbstractArrayDataVisitor() {
      void (Bool2d) {...}
      :
      void (Double2d) {...}
   }
}
```

Readability of Code

If a particular cast happens frequently, one might consider to do so in a (private) method. An example is the way the ImageDataset is checking errors:

```
class ImageDataset
   void setErrors(Double2d errors, String description) {
     // current
     if ( ( ( (Double2d) ( ( (ArrayDataset)
(this.get("image"))).getData())).getDimensions()[0]
          != errors.getDimensions()[0]) ||
        ( ( (Double2d) ( ( (ArrayDataset) (this.get("image"))).getData())).
         getDimensions()[1]
         != errors.getDimensions()[1]))
     { ... }

     alternative 1
     if (!Arrays.equals(getImage().getDimensions(),errors.getDimensions()))
     { ... }

     alternative 2
     if (!isSameShape(getImage(),errors)
     { ... }

     private void isSameShape(ArrayData lhs,ArrayData rhs) {
       return Arrays.equals(lhs.getDimensions(),rhs.getDimensions());
     }
   }
}
```

Considering that this kind of checking is happing all over the place, the latter is probably improving readability and testability.

Component vs stand-alone
Display is dealing with the JComponent concept by virtue of a special constructor. Not quite sure whether this constructor is appropriate. Perhaps a separation of the DisplayComponent and the Display as a stand-alone window utility may be an option. This could be done either via composition or sub-classing.
Avoiding unnecessary creation of objects
In the implementation there are a few areas where objects are created unnecessary. One could consider to use the mutability of an object, rather than to replace it by a new instance.
Duplication of Class Names
Both ia.plot and ia.image have the notion of Layer and Axis. This may raise confusion.
Heavy methods
Some methods are very lengthy (20-150 lines). For maintenance reasons, it may help to break up these methods into private implementation methods. General rule of thumb: stick to 10-15 lines at most.
Documentation directory
Should be renamed to doc.
Split of functionality
Tasks like Crop, Transpose etc... manipulate image datasets, Display,Axis,Layer,... and the image.gui package deal with rendering. It seems that if one has a gui sub-package, the manipulation tasks may be part of a sub-package as well
WCS and FITS?
WCS is highly coupled to a particular implementation of WCS systems (Is it an exact copy of the FITS WCS annotation?). As such the methods naming convention is rather cryptic. It seems to stem from the dates that variable and function names were limited to a length of six characters.
External Updates
Are there new external packages that are better suited to provide the underlying services?
No knowledge of new packages that would do a better job.
Open Source
Would it be worthwhile to make the package open source?
Hardly, too many couplings to other packages within IA
Documentation
User
Is there a user manual for the package? Does it have a document number? Should it have one?
A HowTo for ia.image exists within ia.doc package. The HowTo is too big to be a HowTo and should be converted into a User document.
Developer
Is there adequate developer's documentation? Does it have a document number? Should it have one?
The Java API partially documented. For example sometimes it is unclear whether the returned value of a method is shared or it is a copy. Due to inherent performance issues, documentation becomes rather important.
Design
Is there adequate design documentation? Does it have a document number? Should it have one?
yes
Standards
JavaDoc " Does the JavaDoc follow the Sun guidelines? "
Does each method have an understandable JavaDoc description? Is the general level of JavaDoc comments adequate (especially for public API)?
Does the JavaDoc include code examples, where appropriate?

The JavaDoc could have more meat. No examples are within the JavaDoc (they are within the HowTo though!) See Developers section mentioned above.
Does each subsystem have a top-level 'package.html' file, including:
"An initial summary sentence documentation, etc "
A description of the contents and purpose of the package
A "Package Specification" section (may be blank)
A "Related Documentation" section with hyperlinks to design

Yes.
Location

Where is the documentation located?
Apart from the HowTo, all documentation is within the package, and accessible through the JavaDoc API.
Change Log
Does the subsystem have a readable CHANGELOG file?
Yes.
Design
General
Are there any indicators of design problems (code metrics, test coverage, high number of SPRs)?
For details, see improvements section above.
The package is testing some of the functionality of WCS and ImageDataset classes. It is rather difficult to test issues dealing with GUI components, as they may rely on a display to be available.
Usage
Is the package easy to use? If not, why?
The HowTo is quite helpful to start using this package
Are appropriate Design Patterns employed?
"Should further design patterns be introduced to improve abstraction, flexibility, modularity, etc?"
"Conversely, are inappropriate design patterns employed?"
Separation of Concerns
Is there a proper separation of concerns between classes and between packages?
See comments in the improvements section
OO Design
Is the design properly Object Oriented?
See comments in the improvements section. Additional note: Display is acting like a wrapper class. Due to the nature of it, it has quite a few methods.
Sound Principle of Operation
Is the principle of operation sound?
Yes
Performance Issues
Might the design lead to performance problems?
See comments in the improvements section. Unnecessary copying of data may become an issue
Reflection
Is reflection used unnecessarily?
No. Casting should be localized, preferably within a private method.
Inheritance
Is inheritance used in an appropriate way? These checkpoints should be true wherever inheritance is used:
It is a "kind of", not a "role played by", relationship.
The object never needs to transmute into another class.
It extends rather than overrides or nullifies the superclass.
It does not subclass what is merely a utility class.

Display should not implement the array visitor.
Is a JComponent a different view of Display?
De-composition
Would composition be better than inheritance?
See previous answer
Mutability
Should any classes be immutable?
Most are already mutable. Mutability of these objects as well as the objects within the implementation could be used more in a mutable fashion.
Error Handling
Is error handling adequate?
Are there any 'exit' statements (except in "main" method)?
Are the error and log messages understandable and appropriate?
Is exception handling employed for all exception / error conditions?
Are the exceptions thrown misleading or not the right ones?
Should the exception be thrown at all?
Should the exception be checked or non-checked?
Can a situation be envisaged where the code will obviously fail?
All tasks catch a general exception (a catch all), converting it into a warning message. This seems to be intentional (for the Display), though the display should catch an specific TaskException generated by the tasks, rather than the tasks silently eating up exceptions.

Testability
Does the design allow the subsystem to be properly tested?
Is a pluggable architecture needed to support test stubs?
Is the GUI decoupled, so that the underlying logic can be tested on its own?
Are there specific testing problems (e.g. Jython code, access to database server, GUIs)?

Provided that the method implementations are split up, the system can be tested properly (for those parts that do not require a window manager).
This goes hand in hand with the decoupling of GUI and logic.

Appendix I

```
        Minutes of the Numeric session 16/11/2004
                30/11/2004 Do Kester.


Present: Bakker, Wetzstein, Dwedari, Guest, Kester
         Huygen, Vandenbussche, Siddiqui

These minutes consist of 3 sections. A number of issues brought up
during an open discussion, answers to part of the questionaire and a
closing  remark.

A. Open discussion.
===================

On efficiency:
Can we build benchmarks into automatic testing?

On types and dimensions:
All types seem to be needed, at least most of them were requested some
way or another.

More Dimensions could be generated at the price of less performance.
Higher than 3 could fall into nD-classes like this. It should be
investigated.  Where the boundary should be is a subject of the
investigation and of the usage made of the higher dimensions. In due
course the boundary could shift to higher dimensions.

On selection:
Selection is needed in all dimensions. Investigate.

On toolboxes:
1. The toolbox framework is OK. It does not need refactoring.

2. Some of the individual toolboxes need to be refactored to bring them
in line with the latest version of numerics. At the same time they need
to be redesigned to make them more OO (less a copy of IDL's versions).
The convolution class (toolbox.filter), the interpolation class
(toolbox.interp) and the fitter class ( toolbox.fit) have some kind of
base classes. Transforms (toolbox.xform) has no base class. I am not
sure if it can have a base class. But both Hamming and Hanning inherit
from AbstractRealXdProcedure while FFT does not.

3. The toolboxes need to work on all possible types and dimensions. From
the documentation it seems that filter, interp and xform work for
Double1d (sometimes Float1d) only. Nothing forbids these things in more
dimensions. And some thoughts should be spend on addressing the integral
types (Int, Long etc.). It could quite well be that we decide that "we
won’t do it; it is not worth the effort", but at least it should be
thought about and documented so.

4. How can we easily (without too much formality) contibute to the
numerics package. There are still quite some areas where we could add
things. Eg. Special math functions (Bessel, Gamma, Errf etc), inner and
outer products for matrices (only done for Double2d), wavelets,
integrals etc. A quick search brought me to the JSci.math lib
http://jsci.sourceforge.net/api/JSci/maths/AbstractMath.html
There is quite a lot to be found there. For a complete overview go to
```

http://jsci.sourceforge.net. This library has not been investigated when we started IA as it is quite new. We should have an official look at it.

Recommendation: implement workaround for long integers in Jython.

B. Replies on the questionaire.

B.1.1
There are hardly any use cases on numeric. And the one that is there is not specific enough.
There was a list of requirements (contents of IDL) which should be prioritized. See also item 4 of toolboxes.

B.1.2
There are no show stoppers. We only need manpower to implement things.

B.1.3
No. In total 21 of which 7 are open.

B.2.1
Obvious.
Use cases are not the proper steering mechanism. Most of the requirements came in as implicit remarks at meetings, coffee talk etc.

B.2.2
Yes.

B.2.3
By now yes, not yet complete.

B.2.4
Yes, there is a strong coupling on the Jython command line, because all things need to know about each other, but no problem

B.2.5
For the casual user it is quite OK. The developer (of new functionality) needs proper explanations, a workshop and examples.

B.2.6
Yes and no, deliberately

B.2.7
There are some "internal" methods. They could be phased out.

B.2.8
There are interfaces to share.log, which should be removed.
As far as we know there are no circular dependencies.

B.2.9
Yes (2x) and no problems.

B.2.10
JDK 1.5 will help the developer, not the user. The template mechanism however, cannot be used to solve the type proliferation problem.
There were no suggestions for improvement on the numerics package proper.
The suggestions were about the individual toolboxes as described above.

B.2.11
Not that we know.

B.2.12
Yes, making it public would entail the same for other packages like Jide, Plot, Dataset (and maybe others). Otherwise it is not very usefull. Benefits: bugfinding, most importantly.

B.3.1
Yes. No, but should

B.3.2
Limited, should be expanded. No docnr, yet

B.2.3
Yes. No, but should.

The remaining questions were not tackled, mainly due to time pressure.

B.5 Summary/other comments

B.5.1 What are the main conclusions of the review panel? How do you rate the state (green/amber /red)

Green, in general for the design. Still quite some work to do in the specific toolboxes.

B.5.2 What are the main recommendations of the review panel?

There are some recomendations in the part A of the minutes.

B.5.3 Were additional points reviewed? If so, please list them,including conclusions and proposed actions

Again see part A of the minutes.

C. Closing remark.

Happiness is no complaints.

Appendix J

B Conclusions of Package Review

Name of package reviewed: Class loader

Panel composition:
JJ Mathieu (JJM)
J. Kemp (JK)
W de Meester (WdM)
A Lorenzani (AL)
N de Candussio (NdC)
J Pizarro (JP)

B.1 Use cases/requirements

B.1.1 Does the subsystem satisfy the relevant use-cases and user requirements? (Note: Use-cases are mainly applicable to applications, rather than utility libraries.)

reviewed:[1]

conclusions: Satisfy the needs and is simple to use.
proposed actions:

B.1.2 What are the use-cases/requirements that are expected to hold major development difficulties for this package? Are they clearly specified? (Note: Currently I noted: stopping of tasks, adding of debugging features, handling of big datasets, reported performance problems, IDLizing, saving of intermediate data)

reviewed:
conclusions: hifi) using sandbox/classloader to look at different data (comparing two data sets)
proposed actions:

B.1.3 Does the package have a high number of SCRs? Are the modification of use-cases and additional use-cases?

reviewed:
conclusions: few SCR
proposed actions:
ACTION JJM:
- Ask to ICC if classloader is used and what feedback people provide.
- Any plans for future that would change the way it works now.

[1] If it was not reviewed, please specify why (not applicable, not critical, lack of time)

B.2 Architecture

B.2.1 What is the "raison d' être" for this package? To which currently existing use-cases is it linked? Do we need additional usecases to specify the most important functionality?

reviewed:
conclusions: `Place of classloader in IA is justified by its use in interactivity.`
proposed actions:

B.2.2 Does the package fit coherently into the overall architecture of IA?

reviewed:
conclusions: `Provide some documentation of the design /implementation`
proposed actions:

B.2.3 Is there an architecture and design document, including:

- Subsystem decomposition
- Design notes, explaining design decisions
- UML diagrams, where appropriate, to illustrate design features

reviewed:
conclusions: `API okay for the purpose and simple enough.`
proposed actions:

B.2.4 Is it a component architecture, with cleanly decoupled components?

reviewed:
conclusions: Not relevant
proposed actions:

B.2.5 Does it have understandable interfaces?

reviewed:
conclusions: Not relevant
proposed actions:

B.2.6 Does the API hide the underlying implementation (i.e. provide suitable abstraction)?

reviewed:
conclusions: Not relevant
proposed actions:

B.2.7 Are there public classes/methods which should not form part of the public API?

- Conversely, are there classes/methods that should be made public?
- Is it clear which packages form part of public API?

reviewed:
conclusions: Not relevant
proposed actions:

B.2.8 Does it have direct interfaces with other subsystems or other libraries? Is it incorrectly/overly coupled to other packages? Are there circular dependencies?

reviewed:
conclusions: Not relevant
proposed actions:

B.2.9 Is the subsystem decomposed into packages? Is this decomposition sensible, or a cause of user irritation and/or increase in complexity?

reviewed:
conclusions: Not relevant
proposed actions:

B.2.10 Do you have suggestions on the improvement of the architecture or design (E.g. would this package benefit from features introduced by JDK 1.5)?

reviewed:
conclusions: `Any changes from switching to 1.5?`
proposed actions:

B.2.11 Are there new external packages that are better suited to provide the underlying services?

reviewed:
conclusions: `May be new packages for java class loader (cheaper maintenance) Look at it`
proposed actions: `Look at them`

B.2.12 Would it be worthwhile to make the package open source?

reviewed:

conclusions: Not relevant
proposed actions:

B.3 Documentation

B.3.1 Is there a user manual for the package? Does it have a document number? Should it have one?

reviewed:
conclusions: No user manual
proposed actions:

B.3.2 Is there adequate developer's documentation? Does it have a document number? Should it have one?

reviewed:
conclusions: `How to document is enough to get started and understanding the classloader`
proposed actions:

B.3.3 Is there adequate design documentation? Does it have a document number? Should it have one?

reviewed:
conclusions: `Design document should be written (see B2.2)`
proposed actions:

B.3.4 JavaDoc

- Does the JavaDoc follow the Sun guidelines?
  - Does each method have an understandable JavaDoc description?
  - Is the general level of JavaDoc comments adequate (especially for public API)?
  - Does the JavaDoc include code examples, where appropriate?
- Does each subsystem have a top-level "package.html" file, including:
  - An initial summary sentence documentation, etc
  - A description of the contents and purpose of the package
  - A "Package Specification" section (may be blank)
  - A "Related Documentation" section with hyperlinks to design

reviewed:
conclusions: `Javadoc adequate`
proposed actions:

B.3.5 Where is the documentation located?

reviewed:
conclusions: OK, in package
proposed actions:

B.3.6 Does the subsystem have a readable CHANGELOG file?

reviewed:
conclusions: Yes
proposed actions:

B.4 Design

B.4.1 Are there any indicators of design problems (code metrics, test coverage, high number of SPRs)?

reviewed:
conclusions: No
proposed actions:

B.4.2 Is the package easy to use? If not, why?

reviewed:
conclusions: Yes
proposed actions:

B.4.3 Are appropriate Design Patterns employed?

- Should further design patterns be introduced to improve abstraction, flexibility, modularity, etc?
- Conversely, are inappropriate design patterns employed?

reviewed:
conclusions: Irrelevant
proposed actions:

B.4.4 Is there a proper separation of concerns between classes and between packages?
reviewed:
conclusions: `Reconsider split of subpackage.`
proposed actions: `Raise the point with system group.`

B.4.5 Is the design properly Object Oriented?
reviewed:
conclusions: Irrelevant
proposed actions:

B.4.6 Is the principle of operation sound?
reviewed:
conclusions: Irrelevant
proposed actions:

B.4.7 Might the design lead to performance problems?
reviewed:
conclusions: Irrelevant
proposed actions:

B.4.8 Is reflection used unnecessarily?
reviewed:
conclusions: Irrelevant
proposed actions:

B.4.9 Is inheritance used in an appropriate way? These checkpoints should be true wherever inheritance is used:

- It is a "kind of", not a "role played by", relationship.
- The object never needs to transmute into another class.
- It extends rather than overrides or nullifies the superclass.
- It does not subclass what is merely a utility class.

reviewed:
conclusions: Irrelevant
proposed actions:

B.4.10 Would composition be better than inheritance?

reviewed:
conclusions: Irrelevant
proposed actions:

B.4.11 Should any classes be immutable?

reviewed:
conclusions: Irrelevant
proposed actions:

B.4.12 Is error handling adequate?

- Are there any 'exit' statements (except in "main" method)?
- Are the error and log messages understandable and appropriate?
- Is exception handling employed for all exception / error conditions?
- Are the exceptions thrown misleading or not the right ones?
- Should the exception be thrown at all?
- Should the exception be checked or non-checked?
- Can a situation be envisaged where the code will obviously fail?

reviewed:
conclusions: `Error handling to be checked`
proposed actions:

B.4.13 Does the design allow the subsystem to be properly tested?

- Is a pluggable architecture needed to support test stubs?
- Is the GUI decoupled, so that the underlying logic can be tested on its own?
- Are there specific testing problems (e.g. Jython code, access to database server, GUIs)?

reviewed:
conclusions: Irrelevant
proposed actions:

B.5 Summary/other comments

B.5.1 What are the main conclusions of the review panel? How do you rate the state (green/amber/red)[2]

green

B.5.2 What are the main recommendations of the review panel?

```
- ClassLoader is nice to use and useful little tool.
- Link documentation to jide so people using java can use it in their
development
- Integrate the classloader in Jide so it does not need be to be
initialized
  explicitly by the user --use of properties-- (NdC look at it)
ACTION JJM:
  - Send request to Nicola for Jide to initialise the classloader.
```

B.5.3 Were additional points reviewed? If so, please list them, including conclusions and proposed actions

[2] If the state is amber or red, please state the reason

Appendix K

B Conclusions of Package Review

Name of package reviewed: Sandbox

Panel composition:

```
JJ Mathieu (JJM)
J Kemp (JK)
W de Meester (WdM)
A Lorenzani (AL)
N de Candussio (NdC)
J Pizarro (JP)
```

B.1 Use cases/requirements

B.1.1 Does the subsystem satisfy the relevant use-cases and user requirements? (Note: Use-cases are mainly applicable to applications, rather than utility libraries.)

reviewed:[1]

conclusions: `How would the datapath (defined or described in TT) be implemented?`

proposed actions:

B.1.2 What are the use-cases/requirements that are expected to hold major development difficulties for this package? Are they clearly specified? (Note: Currently I noted: stopping of tasks, adding of debugging features, handling of big datasets, reported performance problems, IDLizing, saving of intermediate data)

reviewed:

conclusions:

```
      Datapath and how to use it?
      Some aspects are not clear yet.
```

proposed actions: `Sandbox + jide + properties + classloader need to be better integrated (promote to the system group).`

B.1.3 Does the package have a high number of SCRs? Are the modification of use-cases and additional use-cases?

reviewed:

conclusions: `Integration within a coherent approach is important`

[1] If it was not reviewed, please specify why (not applicable, not critical, lack of time)

proposed actions: `Ask the system group which way the system is going to develop and stress the need of users to use only one approach rather than 2 (sandbox and properties).`

B.2 Architecture

B.2.1 What is the "raison d' être" for this package? To which currently existing use-cases is it linked? Do we need additional usecases to specify the most important functionality?

reviewed:
conclusions: `provide safe environment for development. See TT`
proposed actions:

B.2.2 Does the package fit coherently into the overall architecture of IA?

reviewed:
conclusions:OK
proposed actions:

B.2.3 Is there an architecture and design document, including:

- Subsystem decomposition
- Design notes, explaining design decisions
- UML diagrams, where appropriate, to illustrate design features

reviewed:
conclusions: No
proposed actions:

B.2.4 Is it a component architecture, with cleanly decoupled components?

reviewed:
conclusions: No
proposed actions:

B.2.5 Does it have understandable interfaces?

reviewed:
conclusions: Irrelevant. Note that a small API is available and used by the class loader package. A similar interface would be necessary in any refactored properties/sandbox system.
proposed actions:

B.2.6 Does the API hide the underlying implementation (i.e. provide suitable abstraction)?

reviewed:
conclusions: Irrelevant
proposed actions:

B.2.7 Are there public classes/methods which should not form part of the public API?

- Conversely, are there classes/methods that should be made public?
- Is it clear which packages form part of public API?

reviewed:
conclusions: Irrelevant
proposed actions:

B.2.8 Does it have direct interfaces with other subsystems or other libraries? Is it incorrectly/overly coupled to other packages? Are there circular dependencies?

reviewed:
conclusions: Irrelevant
proposed actions:

B.2.9 Is the subsystem decomposed into packages? Is this decomposition sensible, or a cause of user irritation and/or increase in complexity?

reviewed:
conclusions: Irrelevant
proposed actions:

B.2.10 Do you have suggestions on the improvement of the architecture or design (E.g. would this package benefit from features introduced by JDK 1.5)?

reviewed:
conclusions: Irrelevant
proposed actions:

B.2.11 Are there new external packages that are better suited to provide the underlying services?

reviewed:
conclusions: Irrelevant

proposed actions:

B.2.12 Would it be worthwhile to make the package open source?

reviewed:
conclusions: Irrelevant
proposed actions:

B.3 Documentation

B.3.1 Is there a user manual for the package? Does it have a document number? Should it have one?

reviewed:
conclusions:OK
proposed actions:

B.3.2 Is there adequate developer's documentation? Does it have a document number? Should it have one?

reviewed:
conclusions:OK
proposed actions:

B.3.3 Is there adequate design documentation? Does it have a document number? Should it have one?

reviewed:
conclusions:OK
proposed actions:

B.3.4 JavaDoc

- Does the JavaDoc follow the Sun guidelines?
    - Does each method have an understandable JavaDoc description?
    - Is the general level of JavaDoc comments adequate (especially for public API)?
    - Does the JavaDoc include code examples, where appropriate?
- Does each subsystem have a top-level "package.html" file, including:
    - An initial summary sentence documentation, etc
    - A description of the contents and purpose of the package
    - A "Package Specification" section (may be blank)
    - A "Related Documentation" section with hyperlinks to design

reviewed:
conclusions:OK
proposed actions:

B.3.5 Where is the documentation located?

reviewed:
conclusions:OK
proposed actions:

B.3.6 Does the subsystem have a readable CHANGELOG file?

reviewed:
conclusions:OK
proposed actions:

B.4 Design

B.4.1 Are there any indicators of design problems (code metrics, test coverage, high number of SPRs)?

reviewed:
conclusions:OK
proposed actions:

B.4.2 Is the package easy to use? If not, why?

reviewed:
conclusions:OK

proposed actions:

B.4.3 Are appropriate Design Patterns employed?

- Should further design patterns be introduced to improve abstraction, flexibility, modularity, etc?
- Conversely, are inappropriate design patterns employed?

reviewed:
conclusions:OK
proposed actions:

B.4.4 Is there a proper separation of concerns between classes and between packages?
reviewed:
conclusions:OK
proposed actions:

B.4.5 Is the design properly Object Oriented?
reviewed:
conclusions:OK
proposed actions:

B.4.6 Is the principle of operation sound?
reviewed:
conclusions:OK
proposed actions:

B.4.7 Might the design lead to performance problems?
reviewed:
conclusions:OK
proposed actions:

B.4.8 Is reflection used unnecessarily?
reviewed:
conclusions:OK
proposed actions:

B.4.9 Is inheritance used in an appropriate way? These checkpoints should be true wherever inheritance is used:

- It is a "kind of", not a "role played by", relationship.
- The object never needs to transmute into another class.
- It extends rather than overrides or nullifies the superclass.
- It does not subclass what is merely a utility class.

reviewed:
conclusions:OK
proposed actions:

B.4.10 Would composition be better than inheritance?

reviewed:
conclusions:OK
proposed actions:

B.4.11 Should any classes be immutable?

reviewed:
conclusions:OK
proposed actions:

B.4.12 Is error handling adequate?

- Are there any 'exit' statements (except in "main" method)?
- Are the error and log messages understandable and appropriate?
- Is exception handling employed for all exception / error conditions?
- Are the exceptions thrown misleading or not the right ones?
- Should the exception be thrown at all?
- Should the exception be checked or non-checked?
- Can a situation be envisaged where the code will obviously fail?

reviewed:
conclusions:OK
proposed actions:

B.4.13 Does the design allow the subsystem to be properly tested?

- Is a pluggable architecture needed to support test stubs?

- Is the GUI decoupled, so that the underlying logic can be tested on its own?
- Are there specific testing problems (e.g. Jython code, access to database server, GUIs)?

reviewed:
conclusions:OK
proposed actions:

B.5 Summary/other comments

B.5.1 What are the main conclusions of the review panel? How do you rate the state (green/amber/red)[2]

green

B.5.2 What are the main recommendations of the review panel?

Ask system panel to sort out mechanisms between properties and sandbox.
There is a new work package to look into and implement a sub system which will be tackling that particular issue and has the title “Unification of configuration/Property generator/Sandbox/Installer”.

B.5.3 Were additional points reviewed? If so, please list them, including conclusions and proposed actions

[2] If the state is amber or red, please state the reason

Appendix L

Output Task review

Overal Summary

- package got status color green; functionality and design good; documentation needs to be extented
- no USE CASE document available

- action items:
  - **package owner**: add new HowTo for the user and
  - **package owner**: top level documentation should be included under the javadoc-package description
  - **package owner**: current howto should become a developers guide
  - **package owner**: add design documentation (dealing with task-design especially: JTask - Task)
  - **package owner / Jorgo / System / IA WG**: should sort out up to what basic hcss.ia level Task should be used as base component. Should the Task list be part of the ia.numeric.toolbox ?
  - **System / IA WG**: System level doc. should:
    - include explaination / guide on intended use of IA components including Task.
    - promote the use of task components for all IA processing steps / functions.
  - **package owner / Hassan**: to sort out open issue Hassan raised as input for this review.
  - **System / IA WG**: to decide whether - and if, so on which time scale - "history" and/or "completion notification" should be implemented

- B1: no USE CASE document available, however there is a requirement documents available under livelink
- B2:
  - support for (standard way of) data processing within IA, especially for processing datasets
  - no architure document available, is part of AI following from this review
  - has clear interfaces
  - api hides underlying implementation
- B3: see AI above and for more detailed info below (section documentation)
- B4: design is not discussed as owner had no design document available

Architecture and design as a sound basis?

- task's requirement document is the base for current implementation
- JJ will add design documentation (dealing with design of JTask - Task)
- **positive** feedback from users on curent functionality and behaviour

Development risks:

- history is not implemented:
  - => is needed for SPG (estimate first time needed: next year)
  - => JJ: effort is 3 to 6 man-month without replay
  - => up to management to determine whether this is really feasible

- status of a task. Completion/progress event mechanism.
  - => not yet impl.
  - => up to management and prioritasation when this should be done

Maintainability:

- design and api is well structured
- code follows Java Standards closely
- logging mechanism and exception handling in task is good
- check test coverage and metrics report: no real problems show up: OK!

Other points of interest:

- **documentation:**

  Lack of documentation on the IA level which should introduce users on the different IA components, including Task. Current user guide is not including yet an overview of IA components and a guideline how the task component is intended to be used.

  - current status on Task-usage is that is not known /used as one should expect.
  - usage of ia.task is not privileged yet at this time of the IA life cycle, but will be needed in the near future
  - users have no idea to combine / develop tasks
  - system's guideline should promote the use of task components for all IA processing steps / functions.
  - top level documentation should be included under the javadoc-package description
  - current howto should become a developers guide
  - JJ should provide a howto for the user (including comments made by Russ)
  - System level doc. should:
    - include explaination / guide on intended use of IA components including Task.
    - promote the use of task components for all IA processing steps / functions.

  - **User Guide**

  Numeric toolbox should support a catalog of tasks, besides the more primitive functions like sin() and cos() types.

- **Developer jython/java use:**

  Can the task easily be used by the developer?

  => yes.

- the **task-help** is implemented though not useable for the end-user

=> help should be defined on one place, in order to make it maintainable
=> integrated with the common IA help, no use to have a task specific help
=> help should support help/feedback for a specific task, for example: help(fft), Also on the command line

=> check above with ia.help

- **interactive task:**

  is implemented: within an IA session one can first set parameters ; execute ; check results; adjust task; check results.......

  Stephans UC: the functionality is already available under dataflow.

  - where task is on the lower algorithm level calculating the outcome with a fixed set of parameters
  - the dataflow can adjust the parameters for the next task call.
  - only the dataflow is not aware of the last entry and therefor is not stopped

  => ? should this UC be implemented our was this a theoretical question?

- **SCR:**

  SCR1123: is jconsole problem

  SCR0796:

* possibility to give feedback end-status for a task (as can for example be passed to logging object). Useful
for SPG/IA: i.e. should be able to interrupt a task and ask its status and eventually kill it
=> input for jconsole
* investigate whether dataflow-indicator like functionality can be impl. for a task

- **Hassan's input:**

What is meant by a 'Task'? And the related concept I am aware of :'Signature'. Both need to be clearly defined somewhere.

=> Hassan and JJ need to check what Hassan is missing in the documentation (if that's is the case)

- **open issues / requested SCR**:

  Task should provide interface in order to allow to check whether Task contains gui (instanceof(GuiTask)) and such a Gui Task should provide method to
  hand over its GUI (input by Odile)

Appendix M

Output Dataflow Review

Overal Summary

- package is in maintaince mode; status color: green
- there is no dedicated USE CASE document present; the tiger team document serves as a base for the dataflow package
- Resulting Action Items:

- **package owner**: documentation should be updated
- **package owner**: architecture & design should be moved from the User Guide into a special dedicated arch./design document
- **wim de meester**: give detailed feedback on current available documentation
- **package owner**: improve code for DataFlowManager and change impl such that this Manager can be (1) used as a JComponent inside another application and (2) be added to the testharness
- **IA WG / System**: give input whether - and if, so on which time scale - within ia.dataflow end-user functionality should be included. This is needed as suggested by Steve. An example might be stream type processing of a large set of data (database pointers) belonging to one observation. This would requests the implementation of components which asure jython friendly usage within jython as well as an additional HowTo for the end-user.
- **IA WG / System**: like stated for task, the top level IA documentation should give overview of IA components including:
  - dataflow components
  - guideline that tasks should be the IA base component which can be re-used in processes (possibly via the TaskWrapper)

- B1: The packages fullfill the use-cases / requirements as pictured in the Tiger Team Report
- B2:
  - povides functionality / components for stream type processing within IA
  - Architure design document currently is part of the User guide for the dataflow package
  - after restructuring a year ago package has clear api
  - api hides undelying implementation

- B3: see AI above and for more detailed info: "documentation" section below
- B4: design is not discussed (was not regarded to be part of this review) as this was already enlighted & restructured in the 2003

Architecture and design as a sound basis?

- is described in current Dataflow User Guide

- after redesign of dataflow package last year all three ICCs use the dataflow package for there QLA

    = success for common IA development

- Design / requirements ; what is implemented
    - dataflow is in maintaince mode
    - no longer responsibility for history = moved to task
    - has a pure java api which confirms the dataflow package is for developers/specialists
    - an process can be a:
        - pure java
        - wrapper for a task; java or **jython based**
        - where the task allows pluggable algorithms; java or **jython based**

= including most of the requirements setup by the Tiger Team , i.e. support **dynamic** behaviour

Development risks:

- dataflow currently support QLA and test / trend analysis.
- It does **not** offer a good user api / jython facade for the use within jide.

Questions for the IA WG / system:

- should dataflow also be an end-user tool?
- should dataflow be used within SPG? (handling huge sets of data in a stream avoiding out of memory errors)

=> dataflow should also provide a good jython user-api

Maintainability:

- apart from DataFlowManager/Viewer classes - here proposed to be cleaned up - maintainability is ok.

Others:

- **Documentation:**
    - like stated for task, the top level IA documentation should give overview of IA components including:
        - dataflow components
        - guideline that tasks should be the IA base component which can be re-used in processes (possibly via the TaskWrapper)
- **developers howto's:**
    - WdeM: wording / english could be improved. WdeM will give feedback her**e**
- **user guides:**

    - WdeM: howto is a subset of userguide. Owner should be aware to keep them in line.
    - JJ: user guide is a mix between design & Architecture and user guide. should be split up.
    - dataflow user guide is out of date. It deals with obsolete information (contains howto-move-over issues from refactoring in the past...)
- **ia.help**
    - how can the common ia.help (component based -) be used to extend current more OO type of help facilities under the DataFlowManager

- **Code review:**
    - code review Jose all dealing with DataFlowViewer:

      => creating objects within a loop
      => too many if statements, should be if... else
      => some methods should be cleaned up.
      => some comments should be removed.

    - DataFlowManager should be a JPanel instead of a JFrame

      => allow it to be used as JComponent
      => allow it to be tested in batch mode

- **check metrics report:**

* DataFlowViewer needs to be cleaned up.

- **check test coverage report:**

* DataFlowManager extends JFrame is excluded from the test as it cannot be tested in the background without
an (X) window in batch mode. Testharness however exists

SCRs:

SCR-1200: check contents
SPR-1160: to be answered by Javier
SPR-0938: dealing with DataFlowManager's testharness lacking X window

Appendix on code review issue (Jose's more detailed feedback):

```
Minor issues
=============
datafolw.DataFlowViewer.createConnectorBoxes , create
processboxes,createFeedthroughBoxes,

  Objects declared within a for loop
+++++++++++++++++++++++++++++++++++++++++++++++
class datafolw.DataFlowViewer.GuiMouseListener extends MouseAdapter{
            public void mousePressed declares a new integer within a while
loop
++++++++++++++++++++++++++++++++++++++++++++++++

Major Issues
==============
 class dataflow.DataFlowViewer.MenuActionListener implements
ActionListener{

        public void actionPerformed(ActionEvent e)

 Use else if (cmd.equals("sometext"))  instead of a lot of if (cmd.equals
("sometext"))statements

It seems that you want to use a case/switch statement here but cannot due
to
comparison of strings


refactor      the following    DataFlowViewer.this.validate();
                               DataFlowViewer.this.repaint();

so that you are not entering the same repetitive commands. or do it outside
the if block as it is done in nearly all if conditions.

refactor the if  (cmd.equals("sometext")) and   if (cmd.startsWith("get")
&& cmd.endsWith("-proc") )
blocks  to call methods contacting these blocks to make it clear what each
if
block is doing and to improve maintainability

Does this class really need to be an inner class?

++++++++++++++++++++++++++++++++++++++++++++

class datafolw.DataFlow.createProcess     please clarify comment
      //In case there are security problems, make
            //constructor explicitly accessible
            // XXX why is this needed, AdJ
```

Appendix N

B Conclusions of Package Review

Name of package reviewed: UI

Panel composition:

Chair	Candussio
Defender	*Kemp*
Reviewers	Guest
	Pizarro
	Wetzstein
	Coeur-Joly
Remote Reviewer	[Borkowsky]

Introduction

The package is more a collection of utility classes then a tool or a comprehensive framework for user interface development. Therefore not every point of the list was considered relevant for this review. This is mentioned in the pertinent points.

B.1 Use cases/requirements

Much of the package is not easily covered by Use Cases (see Introduction), with the only exception of the use case describing the behavior of the mouse (Annex A)

B.1.1 Does the subsystem satisfy the relevant use-cases and user requirements? (Note: Use-cases are mainly applicable to applications, rather than utility libraries.)

reviewed:[1] Yes

conclusions:

The subsystem satisfies the only applicable use case (Mouse use case).
Problems: the use case is not available in the documentation

proposed actions:

[1] If it was not reviewed, please specify why (not applicable, not critical, lack of time)

Action to JK to add the use case to the package documentation [dd 15th December]

B.1.2 What are the use-cases/requirements that are expected to hold major development difficulties for this package? Are they clearly specified? (Note: Currently I noted: stopping of tasks, adding of debugging features, handling of big datasets, reported performance problems, IDLizing, saving of intermediate data)

reviewed: *[Not Reviewed as not Applicable (see Introduction)]*

conclusions:

proposed actions:

B.1.3 Does the package have a high number of SCRs? Are the modification of use-cases and additional use-cases?

reviewed: *[Not Reviewed as not Applicable (see Introduction)]*

conclusions:

proposed actions:

B.2 Architecture

B.2.1 What is the "raison d' être" for this package? To which currently existing use-cases is it linked? Do we need additional usecases to specify the most important functionality?

reviewed: Yes

conclusions:

No additional use cases required.

proposed actions:

B.2.2 Does the package fit coherently into the overall architecture of IA?

reviewed: Yes

conclusions:

The package fits coherently into the IA architecture.
However the point of promoting it as as hcss package was raised

proposed actions:

SG and HS to check if herschel.ia.ui can be promoted to herschel [dd 15th December]

B.2.3 Is there an architecture and design document, including:

- Subsystem decomposition
- Design notes, explaining design decisions
- UML diagrams, where appropriate, to illustrate design features

reviewed: Yes

conclusions:
The package does not require at this moment the mentioned documentation (see Introduction)

proposed actions:

B.2.4 Is it component architecture, with cleanly decoupled components?

reviewed: Yes

conclusions:
The package is compliant to this point

proposed actions:

B.2.5 Does it have understandable interfaces?

reviewed: Yes

conclusions:
The package is compliant to this point
proposed actions:

B.2.6 Does the API hide the underlying implementation (i.e. provide suitable abstraction)?

reviewed: Yes

conclusions:
The package is compliant to this point
proposed actions:

B.2.7 Are there public classes/methods which should not form part of the public API?

- Conversely, are there classes/methods that should be made public?
- Is it clear which packages form part of public API?

reviewed: Yes

conclusions:
The package is compliant to this point

proposed actions:

B.2.8 Does it have direct interfaces with other subsystems or other libraries? Is it incorrectly/overly coupled to other packages? Are there circular dependencies?

reviewed: Yes

conclusions:
The package is compliant to this point

proposed actions:

B.2.9 Is the subsystem decomposed into packages? Is this decomposition sensible, or a cause of user irritation and/or increase in complexity?

reviewed: Yes

conclusions:
This point doesn’t apply to this package (see introduction)

proposed actions:

B.2.10 Do you have suggestions on the improvement of the architecture or design (E.g. would this package benefit from features introduced by JDK 1.5)?

reviewed: Yes

conclusions:
No suggestions or improvements

proposed actions:

B.2.11 Are there new external packages that are better suited to provide the underlying services?

reviewed:Yes

conclusions:
No external package can provide at the moment a better service
proposed actions:

B.2.12 Would it be worthwhile to make the package open source?

reviewed:

conclusions:

proposed actions:

B.3 Documentation

B.3.1 Is there a user manual for the package? Does it have a document number? Should it have one?

reviewed: Yes

conclusions:
The package does not contain a user manual but it is not needed.

proposed actions:

B.3.2 Is there adequate developer's documentation? Does it have a document number? Should it have one?

reviewed: Yes

conclusions:
The package is compliant to this point

proposed actions:

B.3.3 Is there adequate design documentation? Does it have a document number? Should it have one?

reviewed: Yes

conclusions:

A design document is not need at this point (see introduction)

proposed actions:

B.3.4 JavaDoc

- Does the JavaDoc follow the Sun guidelines?
  - Does each method have an understandable JavaDoc description?
  - Is the general level of JavaDoc comments adequate (especially for public API)?
  - Does the JavaDoc include code examples, where appropriate?
- Does each subsystem have a top-level "package.html" file, including:
  - An initial summary sentence documentation, etc
  - A description of the contents and purpose of the package
  - A "Package Specification" section (may be blank)
  - A "Related Documentation" section with hyperlinks to design

reviewed: Yes

conclusions:

The package is compliant to this point

proposed actions:

B.3.5 Where is the documentation located?

reviewed: Yes

conclusions:

Documentation is located in the package javadoc document

proposed actions:

B.3.6 Does the subsystem have a readable CHANGELOG file?

reviewed: Yes

conclusions:

The subsystem has a readable CHANGELOG file.

proposed actions:

B.4 Design

B.4.1 Are there any indicators of design problems (code metrics, test coverage, high number of SPRs)?

reviewed: Yes

conclusions:

A fine grain metric inspection has highlighted that two classes have values not comparable with the rest of the package: SystemPopup WindowManager.

SystemPopup:
The different metric values were indeed pointing to some design problems.
The review pointed out the following ones:
a) The class violates inheritance principles
b) The implementation is partially exposed in the Javadoc description
c) There are a number of constructors that appear to be wholly unnecessary
The panel agreed to ask for a refactoring of the class
1] **Action** on NdC to open an SPR to refactor SystemPopup [dd 30th November]

WindowManager:
The metrics did not reflect a design problem.
2] **Action** on NdC and HS to document this class with Tim Lock [NdC 30th]

Some classes have evaluated in fine detail and the following has emerged as deserving attentions:

Help
This class should be removed as it is superseded by the help class in the help package.
3] **Action** on Juliet to deprecate the class [dd 30th November]

ScreenshotGenerator
This class has the following points of concerns:
1. Creation of an instance of this class is maybe not required. Could it go in GuiUtils as a single method?
2. Why is createImage factored out as a separate method? Should at least be declared static.
3. jpegFile and pngFile are declared as class instance variables yet only used locally.
4. The JFileChooser should be declared static (so current directory is remembered).
5. The main if/else block should be refactored, mostly duplicate code in both bits.
6. A general exception is both caught and thrown, should be specific, and documented in the Javadoc.
4] **Action** on SG to open an SPR to address this points [dd 30th November]

GuiUtils:
The method names are inconsistent. One has "all" the end, the other doesn't, even though they're both recursive.
5] **Action** on SG to open an SPR to address this points [dd 30th November]

proposed actions:

Actions 1] 2] 3] 4] 5] as mentioned along the conclusions.

B.4.2 Is the package easy to use? If not, why?
reviewed: *[Not Reviewed as not Applicable (see Introduction)]*

conclusions:

proposed actions:

B.4.3 Are appropriate Design Patterns employed?

- Should further design patterns be introduced to improve abstraction, flexibility, modularity, etc?
- Conversely, are inappropriate design patterns employed?

reviewed: *[Not Reviewed as not Applicable (see Introduction)]*

conclusions:

proposed actions:

B.4.4 Is there a proper separation of concerns between classes and between packages?
reviewed: *[Not Reviewed as not Applicable (see Introduction)]*

conclusions:

proposed actions:

B.4.5 Is the design properly Object Oriented?
reviewed: *[Not Reviewed as not Applicable (see Introduction)]*

conclusions:

proposed actions:

B.4.6 Is the principle of operation sound?

reviewed: *[Not Reviewed as not Applicable (see Introduction)]*

conclusions:

proposed actions:

B.4.7 Might the design lead to performance problems?

reviewed: *[Not Reviewed as not Applicable (see Introduction)]*

conclusions:

proposed actions:

B.4.8 Is reflection used unnecessarily?

reviewed: *[Not Reviewed as not Applicable (see Introduction)]*

conclusions:

proposed actions:

B.4.9 Is inheritance used in an appropriate way? These checkpoints should be true wherever inheritance is used:

- It is a "kind of", not a "role played by", relationship.
- The object never needs to transmute into another class.
- It extends rather than overrides or nullifies the superclass.
- It does not subclass what is merely a utility class.

reviewed: *[Not Reviewed as not Applicable (see Introduction)]*

conclusions:

proposed actions:

B.4.10 Would composition be better than inheritance?
reviewed: *[Not Reviewed as not Applicable (see Introduction)]*

conclusions:

proposed actions:

B.4.11 Should any classes be immutable?
reviewed: *[Not Reviewed as not Applicable (see Introduction)]*

conclusions:

proposed actions:

B.4.12 Is error handling adequate?

- Are there any 'exit' statements (except in "main" method)?
- Are the error and log messages understandable and appropriate?
- Is exception handling employed for all exception / error conditions?
- Are the exceptions thrown misleading or not the right ones?
- Should the exception be thrown at all?
- Should the exception be checked or non-checked?
- Can a situation be envisaged where the code will obviously fail?

reviewed: *[Not Reviewed as not Applicable (see Introduction)]*

conclusions:

proposed actions:

B.4.13 Does the design allow the subsystem to be properly tested?

- Is a pluggable architecture needed to support test stubs?
- Is the GUI decoupled, so that the underlying logic can be tested on its own?
- Are there specific testing problems (e.g. Jython code, access to database server, GUIs)?

reviewed: *[Not Reviewed as not Applicable (see Introduction)]*

conclusions:

proposed actions:

B.5 Summary/other comments

B.5.1 What are the main conclusions of the review panel? How do you rate the state (green/amber/red)[2]

The package is in a green state. Some classes have been carefully scrutinized and specific actions have been taken to address the few minor highlighted points.

B.5.2 What are the main recommendations of the review panel?

There are no main recommendations.

B.5.3 Were additional points reviewed? If so, please list them, including conclusions and proposed actions

[2] If the state is amber or red, please state the reason

Appendix O

B Conclusions of Package Review

Name of package reviewed: Help

Panel composition:

Chair	Candussio
Defender	*Kemp*
Reviewers	Guest
	Pizarro
	Wetzstein
	Coeur-Joly

Introduction

The package has completed its first complete iteration just in #6/1 by releasing the tools to generate the documentation files from the extended set of Tags. As a result it has been possible to complete the entire help generation cycle: Code->Generation->Display.

B.1 Use cases/requirements

B.1.1 Does the subsystem satisfy the relevant use-cases and user requirements? (Note: Use-cases are mainly applicable to applications, rather than utility libraries.)

reviewed:[1] Yes

conclusions:

There are no deviations from the use cases documented in the package.
However
1) SG objected the set of Qla use cases were not migrated into the Help package. As this set of use cases is not formalized within the help package, the panel agreed to consolidate the current use cases by asking SG and MW (who have new relevant ones) to produce them in time to be further discussed by the user group.
1] Action on SG and MW to provide all the relevant use cases [due date November 26th].

[1] If it was not reviewed, please specify why (not applicable, not critical, lack of time)

2) The use case defining the strategy for retrieval the most appropriate documentation is missing but, as it belongs to the realm of documentation, the following action was approved.
2] Action: Documentation review to discuss the implementation of a strategy for retrieval the most appropriate documentation [dd 30th November]

proposed actions:

1] SG and MW to provide all the relevant use cases [due date November 6th]
2] Documentation review to discuss the implementation of a strategy for retrieval the most appropriate documentation [dd 30th November]

B.1.2 What are the use-cases/requirements that are expected to hold major development difficulties for this package? Are they clearly specified? (Note: Currently I noted: stopping of tasks, adding of debugging features, handling of big datasets, reported performance problems, IDLizing, saving of intermediate data)

Reviewed: Yes

conclusions:

1) The Support for Jython script as already specified is clear.
2) Implementation of a strategy for retrieval the most appropriate documentation is under definition.

proposed actions:

B.1.3 Does the package have a high number of SCRs? Are the modification of use-cases and additional use-cases?

reviewed: *[Not Reviewed as not Applicable, the package has just been published]*

conclusions:

proposed actions:

B.2 Architecture

B.2.1 What is the "raison d' être" for this package? To which currently existing use-cases is it linked? Do we need additional usecases to specify the most important functionality?

reviewed: Yes

conclusions:

The point has been addresses in B.1.1

proposed actions:

B.2.2 Does the package fit coherently into the overall architecture of IA?

reviewed: Yes

conclusions:

The package fits coherently into the overall architecture of IA, with the exception of the sessioninspector which will have to be moved to its own package.

proposed actions:

Addressed in B.2.9

B.2.3 Is there an architecture and design document, including:

- Subsystem decomposition
- Design notes, explaining design decisions
- UML diagrams, where appropriate, to illustrate design features

reviewed: Yes

conclusions:

The current document is sufficient for the time being.

proposed actions:

B.2.4 Is it a component architecture, with cleanly decoupled components?

reviewed: Yes

conclusions:

The package is compliant to this point.

proposed actions:

B.2.5 Does it have understandable interfaces?

reviewed: Yes

conclusions:

The current structure has inherited the hcss standards for sub packages and the readability is partially compromised. As it is a general question it was redirected to the system panel.

proposed actions:

System panel to provide standards for package structure.

B.2.6 Does the API hide the underlying implementation (i.e. provide suitable abstraction)?

reviewed: No, see B.2.5

conclusions:

proposed actions:

B.2.7 Are there public classes/methods which should not form part of the public API?

- Conversely, are there classes/methods that should be made public?
- Is it clear which packages form part of public API?

reviewed: No, see B.2.5

conclusions:

proposed actions:

B.2.8 Does it have direct interfaces with other subsystems or other libraries? Is it incorrectly/overly coupled to other packages? Are there circular dependencies?

reviewed: Yes

conclusions:

proposed actions:

B.2.9 Is the subsystem decomposed into packages? Is this decomposition sensible, or a cause of user irritation and/or increase in complexity?

reviewed: Yes

conclusions:
The panel decided that herschel.ia.sessioninspector is not the right place for this package. The reason is that even though the session inspector and the help view are in the same logical domain they do not belong in the same package, as the sessioninspector is a graphical tool it should be located under its own package or under a generic tool package.

proposed actions:
JK to move herschel.ia.sessioninspector in herschel.inspector.session [30th November]

NdC to check if DatasetInsepctor could be moved in the same package [20th November]

B.2.10 Do you have suggestions on the improvement of the architecture or design (E.g. would this package benefit from features introduced by JDK 1.5)?

reviewed: Yes

conclusions:
The XSL-FO technology was suggested as a possible candidate

proposed actions:
JK to investigate the XSL-FO [dd end on January 2005]

B.2.11 Are there new external packages that are better suited to provide the underlying services?

reviewed: Yes

conclusions:
Aspect oriented technology was suggested as a possible candidate and it could explored in the next future.

proposed actions:

B.2.12 Would it be worthwhile to make the package open source?

reviewed: Yes

conclusions:
Not at the moment but the potentiality of the package suggests a later investigation.

proposed actions:

B.3 Documentation

The package still lacks a complete developers documentation.

B.3.1 Is there a user manual for the package? Does it have a document number? Should it have one?

reviewed: Yes

conclusions:

The users' documentation on help is fine however SG requested to put more emphasis on the limitations part of the help task

proposed actions:

NdC to update documentation on HelpTask in order put more emphasis on the limitations [dd 30th November])

B.3.2 Is there adequate developer's documentation? Does it have a document number? Should it have one?

reviewed: *[Not Reviewed (see introduction)]*

conclusions:

proposed actions:

B.3.3 Is there adequate design documentation? Does it have a document number? Should it have one?

reviewed: Yes

conclusions:

A design document is available and it is sufficient for the time being

proposed actions:

B.3.4 JavaDoc

- Does the JavaDoc follow the Sun guidelines?
  - Does each method have an understandable JavaDoc description?
  - Is the general level of JavaDoc comments adequate (especially for public API)?
  - Does the JavaDoc include code examples, where appropriate?
- Does each subsystem have a top-level "package.html" file, including:
  - An initial summary sentence documentation, etc
  - A description of the contents and purpose of the package
  - A "Package Specification" section (may be blank)
  - A "Related Documentation" section with hyperlinks to design

reviewed: *[Not Reviewed (see introduction)]*

conclusions:

proposed actions:

B.3.5 Where is the documentation located?

reviewed: *[Not Reviewed (see introduction)]*

conclusions:

proposed actions:

B.3.6 Does the subsystem have a readable CHANGELOG file?

reviewed: Yes

conclusions:
The package has a readable CHANGELOG

proposed actions:

B.4 Design

Limitations:
The session inspector package which is still under active development (it needs to be integrated with the Dataset Inspector) is not object of this review.

B.4.1 Are there any indicators of design problems (code metrics, test coverage, high number of SPRs)?

reviewed: Yes

conclusions:
No problems were highlighted by the metrics or by reviewers.
However the following suggestions were formulated:
1)Remove the use of System.out.println
2)Fixe the misuse of configuration

proposed actions:
1] NdC to remove the use of System.out.println [NdC 30th November]
2] NdC Check if declaration within for loop is cause of bad performances [30th]
3] NdC, HS, SG Misuse of configuration in HelpLocator.defaults [TDB]

B.4.2 Is the package easy to use? If not, why?
reviewed:Yes

conclusions:
The use of javascript in the displayed document should be avoided.
proposed actions:
Documentation group to check standards and exceptions in the formats

B.4.3 Are appropriate Design Patterns employed?
- Should further design patterns be introduced to improve abstraction, flexibility, modularity, etc?
- Conversely, are inappropriate design patterns employed?

reviewed:Yes

conclusions:
The package is compliant to these points.

proposed actions:

B.4.4 Is there a proper separation of concerns between classes and between packages?
reviewed: *[Not Relevant]*

conclusions:

proposed actions:

B.4.5 Is the design properly Object Oriented?
reviewed: *[Not Relevant]*

conclusions:

proposed actions:

B.4.6 Is the principle of operation sound?
reviewed: *[Not Relevant]*

conclusions:

proposed actions:

B.4.7 Might the design lead to performance problems?
reviewed: *[Not Relevant]*

conclusions:

proposed actions:

B.4.8 Is reflection used unnecessarily?
reviewed: *[Not Relevant]*

conclusions:

proposed actions:

B.4.9 Is inheritance used in an appropriate way? These checkpoints should be true wherever inheritance is used:

- It is a "kind of", not a "role played by", relationship.
- The object never needs to transmute into another class.
- It extends rather than overrides or nullifies the superclass.
- It does not subclass what is merely a utility class.

reviewed: *[Not Relevant]*

conclusions:

proposed actions:

B.4.10 Would composition be better than inheritance?
reviewed: *[Not Relevant]*

conclusions:

proposed actions:

B.4.11 Should any classes be immutable?
reviewed: *[Not Relevant]*

conclusions:

proposed actions:

B.4.12 Is error handling adequate?

- Are there any 'exit' statements (except in "main" method)?
- Are the error and log messages understandable and appropriate?
- Is exception handling employed for all exception / error conditions?
- Are the exceptions thrown misleading or not the right ones?
- Should the exception be thrown at all?
- Should the exception be checked or non-checked?
- Can a situation be envisaged where the code will obviously fail?

reviewed: Yes

conclusions:
Odile asked to check if the warnings when compiling documentation are compliant to the javadoc standards

proposed actions:
NdC to open spr for checking compliance with javadoc [dd 30th November]

B.4.13 Does the design allow the subsystem to be properly tested?

- Is a pluggable architecture needed to support test stubs?
- Is the GUI decoupled, so that the underlying logic can be tested on its own?

- Are there specific testing problems (e.g. Jython code, access to database server, GUIs)?

reviewed:Yes

conclusions:

The subsystem can be properly tested.
However there are limitations due the javadoc architecture that limits the test to the only black box testing.

proposed actions:

JK to update the documentation explaining limits on tests architecture [30th November]

B.5 Summary/other comments

B.5.1 What are the main conclusions of the review panel? How do you rate the state (green/amber/red)[2]

The package needs to be checked against the set of user requirements. As they are not completely available the package is formally in amber state.

B.5.2 What are the main recommendations of the review panel?

B.5.3 Were additional points reviewed? If so, please list them, including conclusions and proposed actions

[2] If the state is amber or red, please state the reason

Appendix P

JConsole Review 16/11/04 Minutes SG 30/11/04

N de Candussio (HSC/ESTEC, "defender")
S Guest (SPIRE/RAL, chair)
R Huygen (PACS/KUL, reviewer)
A de Jonge (HIFI/SRON, reviewer)
P Roelfsema (HIFI/SRON, observer)
B Vandenbussche (PACS/KUL, observer, p/t)

Purpose

SG introduced the purpose of the review as to concentrate on future maintainability, as it was felt that the large number of SCRs on the package appears to be putting a strain on some aspects of the design. The package was not reviewed against the checklist for the review as it had previously been reviewed in that context in 2003.

The panel agreed to stress:

1. Maintainability
2. Robustness
3. Extensibility

Reusability, while desirable, is not an end in itself. It must be clear what is gained by it, and it is possible to go too far. However, it is expected to be a by-product of a design-for-maintenance approach with high cohesion and loose coupling. Only classes that are intended to be reusable outside a standalone JIDE environment should appear in the Javadoc.

A reusable and extensible JConsole is a good candidate for open source.

General

There was some dicussion on whether we actually need to maintain this package at all. NdC asked if there are now other options that could be used as an alternative. It was agreed that we should keep our eyes open. SG asked if maintenance and further development of the package could be given to an external body ie outsourced, as there would likely be some interested parties. RH argued that we should be careful of giving it to someone else as it is too critical to the system as a whole. The panel agreed that control of this package should be kept in-house, though there is a possibility that some parts, if sufficiently modular, could be worked on by other groups. [SG: there is a contradiction between this and looking for alternatives]

NdC was uncomfortable that restructuring work recommended by the panel would not be covered by written requirements and that such work is uncosted. He would like to work against Use Cases. It was agreed though that there is an implicit requirement to make the system maintainable. RH remarked that an SCR-driven system precludes future planning. The panel suggested that the analysis of an SCR should reference an existing Use Case (or other written requirement) if there is one, and include a new Use Case if there is not.

NdC agreed to answer every point brought up by the package reviewers. He mentioned that the debugging and Control-C issues might have implications on the overall package architecture.

Package Name

It is not clear what the difference is between jconsole and jide, and this is confusing.
The panel felt that:
1. JIDE could refer to the standalone application. The package contains all the elements of the environment and could be called jide.
2. JConsole could refer to the (reusable) command line interface.

This was intended purely as a suggestion for further discussion - no action is appropriate.

Package Structure

All the reviewers were confused by the package structure. It is unclear which classes are a part of a public API, and what the reasons for the decomposition are. In particular the main API appears to be in the "gui" subpackage and not the "api" subpackage. Moreover, the api/impl split implies the use of a pluggable implementation pattern which is not actually followed, and which was also felt to be confusing. NdC stated that "api" is intended to be a "user API" and "gui" the "developer API". As API = application *programmer* interface, and GUI = graphical *user* interface, this is particularly confusing.
The package structure was based on one found in some other HCSS packages. It was generally felt that a "util" subpackage was expected to be part of the public API.

NdC indicated that to consider the package as having an API was a new view of it, as it was originally conceived as purely an application.

The panel recommended to:
1. Restructure the package to make it easier to understand. The goal of moving things is to improve cohesion, maintainability and documentation.
2. Ensure that only classes and methods intended as public API appear in the Javadoc.
3. Ask the system review to make package structure guidelines clear.
4. Look into whether general utility classes can be moved outside the package. There should be a general place to put utility classes.

Dependencies and Pluggability

The inclusion of the dataset inspector and help buttons (both from SCRs) has resulted in the core jconsole (gui) package depending on not only the entire IA system, but also external packages such as JSky. Some of the dependencies are circular.

The panel recommended to restructure the package to support "plug-ins" that can be used to add extra items and change the configuration. Dependencies on other packages should be placed outside the core JConsole package and loaded with the plug-in mechanism. NdC pointed out that the dependencies still exist even with a plug-in. However, they are separated out and localised. The plug-ins should probably be loaded from a start-up script. This mechanism will make jconsole fully extensible and easier to maintain.

There is some debate about which package dependencies are acceptable and which aren't. AdJ raised the question of whether the help feature should be a plug-in. It was agreed that it should. Dependencies on utility libraries such as share and ia.ui might be ok. It was felt that the locations of utility classes in the HCSS/IA system should be rationalised and that there should be a general place or places to put them. This was referred to the system panel. Many of the classes in jconsole.util are utilities and could be moved.

It was asked whether it was possible to make the editing environment pluggable, so users could for example use an integrated JEdit. AdJ pointed out that people are very attached to their own editing environment. Input from the user group might be useful here. NdC agreed to investigate the possibility. It was agreed that this feature is less important than generic plug-ins.

RH specifically brought up the point of being able to add keyboard shortcuts, and to be able to "hook in" user-defined functions. For example, it should be possible to write a plug-in function that shows all methods of a class if the user presses a TAB after a dot is typed on the commandline. This would be supported by the plug-in mechanism.

Properties

Most properties are defined twice with the same defaults ie in xml and defaults file. It was agreed to rationalise this.

There was some discussion on the best way for JConsole to handle its properties. SG had provided a list of five alternative ways as input. The panel felt that JConsole components using properties directly (option 1) was acceptable, but hiding it behind a pluggable API would be better (option 4).

NdC felt that guidelines were lacking in how to use properties properly. BV remarked that a distinction should be made between properties and configuration. AdJ suggested that there should be lightweight property implementation with just system properties for use outside an HCSS environment.

Appendix A: Review kickoff (SG e-mail 04/11/04)

Hi,

I have already mentioned to a few people that I would like a JConsole review to focus on a design for maintainability. I was also the one who asked for it to be reviewed, which I suppose is how I ended up the chair. At the moment we seem to have Nicola and Rik signed up.

JConsole was carefully designed and was successfully reviewed in 2003, but the large number of SCRs on the package appears to be putting a strain on some aspects of the design, and introducing some dependencies in possibly unexpected places. My opinion is that the design needs some rework with particular emphasis on future maintainability.

These are some general aims:

1. A standalone "core" package with no or minimal dependencies on other packages. Purely utility packages (eg ia.ui) should be ok, Herschel-specific ones not so.
2. Avoid direct coupling to Herschel-specifics anyway. This will allow the package to be reused, and possibly offered to the Jython community, hopefully in return for additional external effort. There should be a way to include things like session and dataset inspector without direct coupling.
3. Improved modularity which could enable more people to work on it independently.
4. We can't anticipate all future change requests, so keep it as flexible as possible.
5. Identify and avoid any arbitrary restrictions.
6. Identify what the public API is, document it properly, and make sure nothing else appears in the Javadoc.

I know that not everyone agrees with me that we should be producing reusable software. Still, it's an aim of mine.

A few more specific points:

1. I would like to look at the way properties are used, with the aims of cohesion and loose coupling between components. How can this be best achieved? (I also note that most properties are defined twice with the same value).
2. Which, if any, components are useful in their own right? Are they reusable elsewhere?
3. Does the package structure make sense? Could it be better organised? (I note that an "api" package exists, yet is not the main api, "gui" is).

Steve.

Appendix B: SG review input (e-mail 12/11/04)

Introduction

JConsole was carefully designed and was successfully reviewed in 2003, but the large number of SCRs on the package appears to be putting a strain on some aspects of the design, and introducing some dependencies in possibly unexpected places. My opinion is that the design needs some rework with particular emphasis on future maintainability.

These are some general aims:

1. A standalone "core" package with no or minimal dependencies on other packages. Purely utility packages (eg ia.ui) should be ok, Herschel-specific ones not so.
2. Avoid direct coupling to Herschel-specifics anyway. This will allow the package to be reused, and possibly offered to the Jython community, hopefully in return for additional external effort.
3. Improved modularity which could enable more people to work on it independently.
4. We can't anticipate all future change requests, so keep it as flexible as possible.
5. Identify and avoid any arbitrary restrictions.
6. Identify what the public API is, document it properly, and make sure nothing else appears in the Javadoc.

Package Structure

The purpose of each package and the reasons for it being split are not always clear. The api package is *not* the main api, gui is - very confusing. Consider locating the main API within a single package, though this might be at odds with modularity - the jython and util packages also appear to contain classes that are a part of the API.

impl: It seems like overkill to have a package for a single class. Why is it not a package-private class in api?

Can util and tools be combined? Why not? (properties?).

Is the package breakdown sufficiently modular that a developer at a different site can take over a subpackage? If not, can it be made so?

Dependency tree (incomplete)

```
jconsole    - jython
            - share.util
            - share.property
            - share.log
            - ia.ui
            - ia.dataset.gui (JIDEComponent)
            - ia.numeric (JythonUtil)
            - ia.task (JythonUtil, ClearTask)
            - ia.help (JIDEComponent)

help        - jconsole (many classes in sessioninspector use JythonUtil, **mutual
dependency**)
            - share.util
            - share.log
            - ia.ui
            - ia.task
            - ia.numeric
            - ia.image          -  jsky    -   jai
            - com.sun.javadoc
            - jython

dataset.gui - jconsole (DatasetInspector uses JIDEUtilities, **mutual dependency**)
            - ia.dataset
            - ia.numeric
            - ia.io.ascii
            - ia.image
            - ia.plot
            - ia.ui
            - jython
```

JConsole actually depends on JSky + JAI!!! SessionInspector won't start without them.
In fact the main JIDEComponent depends on virtually the entire IA system.
Also note that JythonUtil claims to be in a "private" package, yet is used extensively in sessioninspector!

The mutual JConsole dependencies can be solved by removing (or moving) the dependencies on ia.dataset.gui and ia.help. The problem class here is JIDEComponent.
The dependency of JythonUtil on ia.numeric and ia.task is less serious but should still be tackled. The references to these packages are localised in the static method isFunction, which can be moved elsewhere.
ClearTask is fully dependent on ia.task - this dependency can only be tackled by moving it (see recommendations).
JythonLogger also has a "herschel" string dependency. It is not clear whether this is necessary or can be avoided.

Recommendations:
1. Move any herschel-specific code into a separate herschel subpackage. Ensure that jide still runs when this package is not there.
2. Define a protocol such that plug-ins can be added to jide. Use this protocol to add the dataset inspector, session browser, and help functionality. The plug-ins should probably be added by a start-up script located by a property.

Documentation

Only the intended API should appear in the Javadoc.

There are many missing Javadoc comments. It's not always clear though whether the class is intended to be part of the API.

Are hooks/predefined variables (eg _interpreter) documented?

There are many classes with no CVS/copyright header.

What does this mean? (from JConsole Javadoc)
"JConsole offers the basic copy and paste features,
however they are not accessible by keywords."

Properties

Most properties are defined twice with the same defaults ie in xml and defaults file.
Should only be in defaults file if overriding package default.

Property handling *can* be delegated outside of the core classes but should not be
done at application level as this results in massive coupling.

What is the best way to handle this? There appear to be a number of options, eg:

1. Add the property handling code to each class that needs a property value, and just gets its own properties. This is simple and good for cohesion, but creates a dependency on property handling code in places where they should perhaps be none. Is this a problem?

2. Create subclasses of said classes to handle the properties and put them in a "herschel" directory. It might then need (TBC) some creational pattern (eg factory) to ensure you get the right one, which is a bit messy.

3. Variant of (2), where we accept we need the factory, but it handles all the property code itself, no subclasses needed. Note that this factory should be more elaborate than currently exists, as methods for each component that uses a property would be needed (for consistency everything should be created through it). Again, this seems a messy way of creating components.

4. Define an API over the top of the properties. JConsoleDefaults already does this in a sort of heavy-handed way (and it seems a little peverse to have that *and* the component creation factory). This would use an interface so that the implementation was pluggable. Any JConsole class could call something like
JConsoleProperties.getProperties().getFont(), which (IMHO) should return a Font object.

5. Stick with what we have and don't worry about it.
JConsoleDefaults now does little useful other than to initialize the properties.

Note that in each case it has to be possible to get the property names in order to make them available to the property editing popups. This seems a little messy, but doesn't pose a genuine problem (it is after all, just a name).

The gui package

I assume that all the public classes in here, other than test harnesses, are part of the jconsole API.

Are classes such as JIDE & JIDEFrame needed at all?

Why don't the test harnesses show up in the Javadoc? It's good that they don't, but how?

JIDEComponent:
- is directly coupled to dataset.gui ("Info" action)
- is directly coupled to help, which relies on Herschel configuration ie the non-reliance on properties is an illusion. Is this ok? (Problem goes away if dependency recommendations are followed).

- This code isn't right, *confusion* due to poor choice (by SG!!) of variable names.
```
      if (windows) GuiUtils.addMouseListenerToAll (cont, ml);  // = paste one
      GuiUtils.addPasteListener (cont, pl);
```

ScriptPane
- Creates a mouse listener. What does this do? Does it still work with JIDEComponent? Does it matter if it doesn't?

The tools package

The purpose of this package appears to be to run JIDE/JConsole within a Herschel-configured environment. As such, can it be combined with the proposed jconsole.herschel package?

JyLauncher:

Why does it use a non-documented property "filename" rather than a command-line argument?

The util package

This package is private according to the Javadoc (even though it's used by the help package). It contains a number of utility classes which might be generally useful and could logically go elsewhere. If these classes are not moved they should either be clearly labelled as being a valid part of the API, or not appear in the Javadoc. Here is a list of each, with a suggestion of where they might be able to go. Wherever I indicate ia.ui, share.swing could also be appropiate. Uncertainty is indicated by question marks.

ClearTask.java	jconsole.herschel
CopyStream.java	share.util
ExtensionFileFilter.java	ia.ui
JFilesSaveDialog.java	ia.ui (?)
JRefreshFileChooser.java	ia.ui (?)
JythonDocument.java	jconsole, but api or private?
JythonUndoManager.java	jconsole, but api or private?
JythonUtil.java	jconsole api (except isFunction -> jconsole.herschel)
LimitedLinesDocument.java	ia.ui
OptionParser.java	share.util (?)
TextAreaOutputStream.java	ia.ui
TextEditor.java	ia.ui (?)
UndoableTextEditor.java	ia.ui (?)
XSplitPane.java	ia.ui
JConsoleLogger.java	share.log (renamed)
JLogWindow.java	Unnecessary, should be a method in JConsoleLogger (cohesion)
JythonFilter.java	share.log (renamed)
JythonLogger.java	jconsole.herschel or share.log (rename) (?)
TextAreaHandler.java	share.log

Miscellaneous

The actions "Log", "Dataset" and "Info" should have better names as it's not clear what they do (they appear on the popup menu).

Can the script editor be made pluggable such that the user can plug-in a favourite?

Remark from Mahohai Huang:
"Do you think if it is an interesting idea to make the editor/console of a JIDE/jconsole detachable from the Jython interpreter (i.e. runnable from different JVMs)? This will allow existing IDEs (e.g. Eclipse) to be used as a platform for all the nice features to be borrowed or implemented with ease. The current JIDE/Jconsole can still remain as a "component'able" light weight Jython development environment."

Comments on specific classes

JIDEUtilities contains a catch of a generic exception, no rethrow and no comment.

Appendix C: RH Review input (e-mail 15/11/04)

Comment by Rik Huygen for IA Code Review 16 Nov 2004 @ K.U.Leuven

General Comments

- where is the architectural design document?

Package Structure

- general comments

  each (sub-)package should contain a package.html file explaining it's existence/purpose and the functionality it provides in its classes.

  The package structure itself is not really clear from its contents.

Please make clear what the difference is between Jide and JConsole and use them consistently throughout the package and its documentation.

- api

  should contain the public API i.e. JIDE (Jide), JConsole component, Launcher...factories if needed (see below)

- gui

  currently contains the main components of the public interface. Should this package exist at all? The whole jconsole package is centered around GUIs so what is so special about classes in the gui sub-package? This is different from packages like ia.dataset which is not GUI based, but contains a sub-package gui which contains the GUI for inspecting components of the main ia.dataset package.

- impl

  Only one class here. I wonder what the benifit is of this whole Factory construct in this particular case. If the purpose is flexibility and allowing experts to plugin different editor components or commandline components, I don't think the JConsoleFactory will be able to provide such flexibility.

- jython

  PACKAGE PRIVATE

  Nevertheless this package contains classes of general interest e.g. Interpreter, Log (?).

- tools

  aren't these classes kind of public API also?

- util

  PACKAGE PRIVATE ? Is this a good idea to have util sub-packages that are private? If at all there should be a gui sub-package to contain the non-public GUI components, most of the classes that are in this util sub-package belong there (e.g. JFilesSaveDialog, JythonDocument).

Documentation

- For a lot of classes the javadoc is either missing or incomplete. Even for essential interfaces no or incomplete javadoc is available e.g. Console.

- As mentioned above the package documentation is missing.

- javadoc is sometimes misleading e.g. a ConsoleEditor has the following in its javadoc:

  > A ConsoleEditor provides the support for loading (executing) and saving jython script file.

  Nevertheless the only methods it provides start with execXXX().

- It might be usefull to explain in the documentation (ADD?) which class is responisble for each Swing component e.g. ScriptPane contains several tabbed ScriptDebuggers, JConsolePane contains JConsoleEditor and a History component (where is this component?...gui/HistoryList which is for some reason package private?)

Individual class comments

- JConsoleDefaults

  is it possible to add/define your own properties here that you would like to use by default in an environment containing customised pluggins? Can pluggins add/register their default set of properties here? e.g. DatasetInspector.

  it is a little bit strange that this class defines properties that one would expect for the individual components. What does getFontSize() and getFontType() provide? Shouldn't e.g. ScriptDebugger (or one of its superclasses) define its own default Font which then can be overwritten by

```
   the setFont() method? What else is meant by reusable components?

 - JConsoleFactoryManager

   see previous comments about factory use (impl)

   if the only methods defined in this class are static, why do you need a
   getInstance() method?

 - JIDEUtilities

   incorrect name --> JideUtilities

 - PropertyHandler

   not sure what this does and why it returns a SystemPopup while its called
   addPropertyHandler.

 - Console

   what is the difference between JConsole, JConsoleEditor, and JConsolePane. It
   is not clear to me why all three of them should implement the Console
   interface.

 - JIDEFrame versus JIDE: why is the JIDEFrame class needed?

 - Editor (I)

   this is a very confusing inerface. It states in the documnetation that it
   defines a TextEditor and it only provide I/O kind of methods for loading and
   saving text.

 - Output

   ?

 - Log

   ?

 - History

   why doesn't this contain simple commands like add() and remove()/pop().


Pluggability

 - is the design such that it supports pluggability? And if yes, plugging in
   what? Are we talking about plugging in our personalised version of the command
   prompt or editor window? are we talking about plugging in our own version of a
   customised FileChooser? Can we replace the Interpreter by another version?
   Or is it more like adding icons to the toolbar to call isolated GUI tools like
   DatasetInspector?

 - Keyboard hook?


RH, 15-Nov-2004


Appendix D: Jerzy Borkowski input (e-mail 13/11/04)
---------------------------------------------------
---------------------------------------------------

Dear Steve, Stephan,

Here are my comments for the JConsole review.
I'm sending this email only to you two, so if
there are other members of  jconsole review
panel who should read it, may I ask
you to please forward this email to them.


1. General

1a.
```

The are two terms: JIDE and JConsole used to describe
similar functionality. There are even
2 identical hcss executables : jide and jconsole.
This is confusing. Only one name should be used, and
the second name should be dropped.

1b.
Ditto in the docs.
I think the problem here comes from the fact that (as
written in the doc design.html/par#3(Architecture)) :

```
  The jconsole package is split into sub-packages:
    * The gui package contains the gui components
    * The jython .....
    [...]
```

which is contrary to the par#5(Gui) which stipulates :

  "The basic component of the gui package is JConsole"

Thus, throughtout the docs and in the source code
jconsole means either the whole jconsole package
(consisting of gui, util, jython, etc...)
or the jconsole class (part of gui). I'd suggest
renaming one of the two (but I'm not sure how big
impact this will have on other parts of the hcss).

2. API vs. GUI

2a.
Since GUI files depend on API files (import herschel.ia.jconsole.api....), API source files
should not depend on GUI. This is mostly followed, with the exception of JConsoleFactory.java
and PropertyHandler.java files, which import herschel.ia.jconsole.gui.JIDE..... stuff. IMHO
the PropertyHandler.java and JIDEComponent.java are best placed in the same directory/class.

2b.
The api/gui directory (and package) names are misleading.
The real API interface (or at least its critical part) is in the gui directory, and even in
the doc it is written that "Clients of the jconsole should only include the gui package".
Therefore api directory/package could be renamed to, say, jconsole-core, and gui
directory/package (or part of it) renamed to api.

kind regards,

Jurek

Appendix Q

A Conclusions of System/infrastructure Review

Panel composition:

Chair: Albrecht de Jonge
Defender: Steve Guest
Members: Jorgo Bakker, Rik Huygen, Hassan Siddiqui.

A.1 Use cases/requirements

A.1.1 Can the system fulfill all foreseen operational and community needs?

Reviewed:

Yes

Conclusions:

- The shopping basket concept and large downloads were identified as risk areas in Tiger Team report. No feasibility demonstration has been given so far. In view of progressing technology we may doubt if network/database independency is still needed in four years.
- It is unclear what happened to the (product)history as defined in the Tiger Team definition. No clear support is available in the task/process classes. Lacking this support Spire software developers are currently making workarounds. The requirements for history are still ambiguous, several idea's about history are around
- Sandbox seems currently not used by end users - leave to sandbox panel
- The relation between CCM product and IA product is well-defined. Not completely clear from the CCM is how the following issues are to be dealt with:
    - SPG will have to put IA products in the database(s)
    - How are these are retrieved or navigated to.
    - Products for calibration have to be stored in a way compatible with their use by uplink/downlink

  We understand the current requirements to imply that calibration data have the form of IA products
- Persistence implementation by versant database needs discussion, these questions were raised:
    - Is what we have this sufficient?
    - Is navigation used where needed?
    - Is the support by HCSDT development sufficient?
    - There seems a lack of documentation/instruction?
    - Are the browsing facilities sufficient?

  The overall feeling was ''current IA use of database may not be as intended/designed''. Additional work/support is needed, and experience feedback from IA end users is needed

- There are no obviously missing area’s with respect to Tiger Team report
- It was briefly questioned if Jython is still the scripting language of choice. The panel agreed that it surely is, but that there are some questions and potential pitfalls:
    - Documentation is problematic. Any public class should be documented at end user level
    - Tailoring Jython and numeric to each other was a large effort. A dedicated scripting language instead might have been more efficient in implementing the numerical part but that does not outweigh the benefits of using Jython.
    - Jython allows people to uncover private methods/classes and expose implementation detail. The risk is that end user scripts might (inadvertently) become dependent on these.
- Performance was discussed. It is unclear what is expected in terms of display speed, computation speed, data access speed.

Proposed actions:

- Review if the risks identified in the tiger-team report for large downloads and the shopping basket concept are still risks.
- The scope of History and its relation to IaProcess and Task need to be (re-) identified, and the priority to implement it has to be reestablished.
- Persistence implementation by versant database needs evaluation
- It should be investigated how to prevent mistakes caused by Jython exposing Java implementation details.
- Performance requirements for IA should be drafted.

A.1.2 Can the system meet the needs of both HIFI/PACS "IA as command line with GUIs" and SPIRE "IA as GUI with command line" view?

Reviewed:

Yes

Conclusions:

- The SPIRE view of IA is not completely GUI-driven. For example it is not required that a fit *algorithm* will be *designed* in a GUI. Given that restricted interpretation of “GUI with command line” the panel concluded that current IA system meets the needs of all three instrument groups
- Some GUI-based functionality may be required in later developments. Examples are to drag a product into a viewer, or to select and run a task from a menu. Care is required to guarantee that current system development does not block this type of extensions. Some basic design work is needed in this area:
    - No problems are currently foreseen to extend the system in this direction
    - Framework development for this needs use cases/requirements
    - Designs will be need for, for example, how GUI’s interact with Task, in particular, how a GUI can control a task.

    - This needs manpower

Proposed actions:

- Allocate resources to give design directions for extending the GUI-based capabilities of the system.

A.1.3 What are the use-cases/requirements that are expected to hold major development difficulties for the IA system? Are they clearly specified?

Reviewed:

Yes

Conclusions:

- The lack of a unified use-case and requirement overview makes it difficult to answer this question. Two areas certain potential
- The "Control C" problem in Jython was considered a development risk. It exposes a fundamental problem in 'safe scripting languages'
    1. A cheap solution is to forbid aborts/suspensions, leading to a safe environment with uncorrupted objects
    2. Another cheap solution is to fully allow aborts/suspensions, leading to 'half-processed', i.e., corrupted objects, with no simple way to detect which objects are corrupt, and the only 'safe' way out at this point is to discard all objects. The current solution, with a forced 'discard all objects' resembles this – you just have to kill your whole session.
    3. The correct (expensive) solution is to code *all* processing in an 'interruptible way'. This allows the process to detect 'half-processed objects' so that they can be flagged/deleted/recovered
    4. It might be possible to identify a number of 'risk' tasks/processes that should be coded in an interruptible way so that the problem of solution 2) is avoided without having the expense of 3).
- Jython class definitions that extend Java classes are not 'backward compatible'. For example, if an end user defines a Jython class MyTable that extends the Java class herschel.ia.dataset.TableDataset, Java code will not recognize objects of class MyTable as being TableDataset objects. It is not clear if the ability to define and extend classes is an explicit requirement on the 'IA scripting language'.

Proposed actions:

None proposed by the panel.

A.2 Architecture

A.2.1 Will the currently developed subsystems (calibration sources database, calibration data access), future subsystems (SPG, QCP, archive browser, pointing refinement system) and instrument specific parts (H/P/S..SS) fit into the overall architecture of astronomers IA (delivered to community) and operations IA (used by ICCs/HSC)?

Reviewed:

Yes.

Conclusions:

- Support of future subsystems depends on their global design. This prevents a general answer to this question.
- Currently SPIRE are experimenting with SPG-like processing. Lack of navigation to calibration objects applicable to particular observation data is known to be a problem area .

Proposed actions:

None proposed by the panel

A.2.2 Do we have additional requirements on HCSS infrastructure? (Note: Currently I noted: maintenance of packages having contributors for distinct sub-packages (dataset))

Reviewed:

Yes

Conclusions:

- Currently, there is an almost one-to-one mapping of a Java package (group) to a single CVS module with one responsible developer. This makes collaboration on the package awkward.
-

Proposed actions:

- HCSS should investigate how to decouple the java package structure from the CVS module structure

A.2.3 Is the package structure within IA OK? Do we have unnecessary dependencies?

Reviewed:

Yes

Conclusions:

- It was felt that the systematic division of a package into xxx.api, xxx.impl, xxx.test etc, as conventional in HCSS was not very good for IA:
    - Package xxx should contain the API directly
    - A subpackage xxx.impl should only be present if the package is intended to actually use multiple independently developed implementations in parallel
    - Test should be in a separate xxx.test package if they are ‘black-box’ tests. Internal functional tests should be in the xxx package, and then only one top level ‘test’ method or class should be publicly visible as entry point for test runs.
- Without going into detail (left to the package panels) the system panel concluded that more attention should be paid to integrating the different packages. A lot of developers are apparently unaware of the technical details of development going on in other groups, and tend to re-invent things. A coordinator (or coordinating group) that is aware of ongoing development at a fair level of detail, could prevent duplication of work and/or design mismatches.
- It might be worth while to find tools that can identify dead code. Dead code at private or package level can be found be software metric tools, but how do we find out what end users and user/instrument contributed software uses?

Proposed actions:

- The IA developers need an IA system architect (or architecture group), that can coordinate the technical design issues between the separate IA developers independent of management.

A.2.4 Are the interfaces to other parts of the ground segment clear?

Reviewed:

Yes

Conclusions:

- (could be a A.2.3 review item) An area of concerns is that we need to fit user contributions, demo’s etc in a clear way into the system. We also have to make sure that as the IA system develops, demo’s and user contributions can be checked against new IA releases, to make sure that they are not outdated.

Proposed actions:

No actions proposed by the panel.

A.2.5 What are the areas of overlap with the HCSS? Are there "grey areas" where things might fall between the cracks?

Reviewed:

Yes.

Conclusions:

- The end user facilities to access the database are a “gap” area. The access and store package are part of HCSS but do not seamlessly integrate with IA, and do not typically provide for IA user retrieve and save scenarios. This applies especially to calibration and SPG items.
- The boundary between HCSS and IA is at the moment not clearly defined. In the first place, there is no clear management decision about separation/unity of IA/HCSS. Secondly, there is no person/group identified to coordinate technical issues between HCSS and IA – a role that might be fulfilled by the system architect (group) proposed under A.2.3. Proposing to move functionality across the HCSS/IA boundary would be his responsibility.
- Area’s where overlap between HCSS and IA components is identified are
    - Herschel.access and herschel.ia.io
    - Herschel.binstruct and herschel.mib (although they are by their naming both HCSS, many components, especially of binstruct. are of an IA nature)

Proposed actions:

None proposed by the panel

A.3 Documentation

Reviewed:

No. Due to limited time and the fact that a separate documentation review will be held it was decided not to review the documentation here but only to record any remarks or questions that arose in the context of the other discussion

Conclusions:

- The package documentation location is not consistent across packages
- The Javadoc API should only document the real developers interface and hide any internal methods. When the package decomposition forces internal methods to be public, they should be very clearly marked, preferably even be excluded from the published java API documentation
- Documentation should investigate how to separate end-user documentation and developer documentation

Proposed actions:

None

A.4 Summary/other comments

A.4.1 What are the main conclusions of the review panel? How do you rate the state? (green/amber/red)

The state was rated green – nothing came up that prevents the IA system from being used by end users to do Interactive Analysis, except that it is not complete in some areas

A.4.2 What are the main recommendations of the review panel?

- A coherent set of use case/requirements is needed to review IA again
- History should be implemented
- The JConsole “CTRL C’ problem should be investigated
- A ‘system architect’ should be appointed to coordinate the development efforts on a technical level.

A.4.3 Were additional points reviewed? If so, please list them, including conclusions and proposed actions

None