SDP Processing Component
Module View

Document number..... SKA-TEL-SDP-0000013
Document Type..... REP
Revision..... 05
Author..... R. Nijboer, D. Mitchell, P. Wortmann
Release Date..... 2018-04-18
Document Classification..... Unrestricted
Status..... Released

Lead Author	**Designation**	**Affiliation**
Ronald Nijboer	SDP Pipelines Lead	ASTRON
Signature & Date:	RJNijboer RJNijboer (Apr 19, 2018)	

Released by	**Designation**	**Affiliation**
Paul Alexander	SDP Project Lead	University of Cambridge
Signature & Date:	Paul Alexander Paul Alexander (Apr 19, 2018)	

Table of Contents

List of Abbreviations

CPU	Central Processing Unit
CSP	Central Signal Processor
FPGA	Field Programmable Gate Array
GPU	Graphics Processing Unit
SDP	Science Data Processor
SKA	Square Kilometre Array

1. Primary Representation

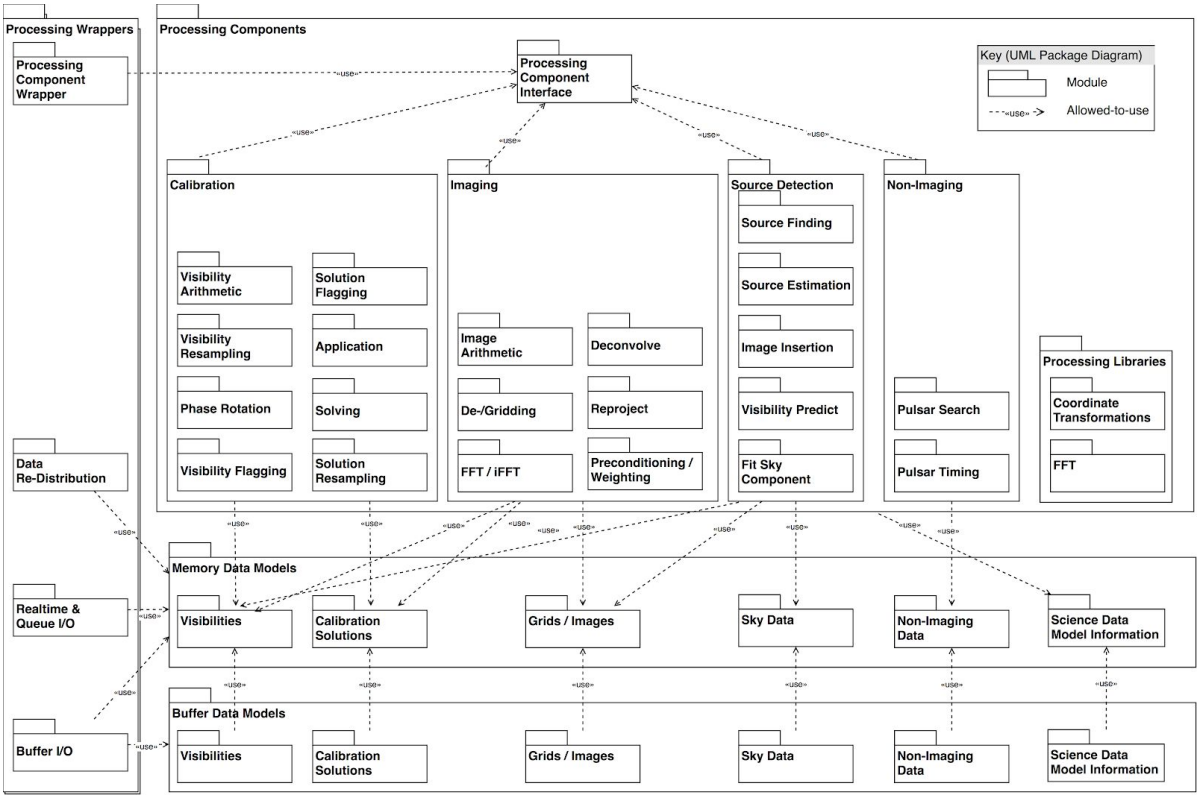

Figure 1 Module decomposition showing the relation between Workflow Components, the underlying Data Models, and the Execution Framework through Component Wrappers. Each Module may make use of Libraries for Processing or Data Access; this is not explicitly shown.

2. Element Catalogue

This section is organized as a dictionary where each entry is an element of the Primary Presentation. This section builds on the SDP Pipelines Design document [RD01] and its supporting documents [RD02], [RD03], [RD04], and [RD05].

2.1. Elements and Their Properties

The following properties are defined to explain the elements in this section:

- **Variants:** How many function variants should be implemented? For our purposes variants mean sub-modules providing similar Processing Component interfaces such that they could be used interchangeably by Workflows. This should be used to wrap different algorithmic approaches, usage of certain optimised libraries and especially optimisations for certain accelerator hardware (CPU, GPU, FPGA…).
- **Performance:** How compute intensive is the function? Is performance critical to pipelines? When appropriate we give a rough estimate of how much computational cost this module will likely contribute to the SDP compute budget according to the parametric model [AD04]. The value given will be percentage of projected compute averaged over pipelines required for high-priority science objectives.

- **Implementation:** Software re-use. Identify which components from existing software could be adapted to implement this function. How much wrapping is required? Note that there are a number of data processing software suites from precursors that implement SDP functionality (CASA [RD10], LOFARSoft [RD11] [RD12], ASKAPSoft [RD13]). This does not necessarily mean that we would always re-use those software parts.

Processing Components are organised in sub-modules according to the data that they work on, see also section 2.4. Details on the Data Model related to Workflows are out-of-scope for this document, but will be described in a Workflows Data Model View document which is to be written for CDR. For now we refer to the System-level Data Model View [AD03]. Furthermore, modules are grouped by function, not according to 'pipeline' or workflow they are used by.

2.1.1. Processing Components

2.1.1.1. Processing Component Interface

The Processing Component Interface is an interface common to all Processing Components. It is used by Processing Wrappers to instantiate, invoke and pass data between Processing Components as required for the Science Pipeline Workflow under execution.

This means that Processing Components are implemented independently from Execution Frameworks and Science Pipeline Workflows, which means that it is de-coupled both from the work distribution as well as the purpose of the workflow.

Variants: Only one, but might have to be implemented / get wrapped for a number of programming environments depending on the Processing Wrappers
Performance: Needs to able to pass in-memory data efficiently, ideally using raw pointers to in-process data for bulk memory data models
Implementation: No re-use from existing astronomy software

2.1.1.2. Calibration

The measured data from the telescope is corrupted by various effects. As a result an image produced from the measured data is limited in its quality. To improve the quality of the image, a model of the instrument and its environment is fitted to the measured data and used to correct the measured data for all corrupting effects. The resulting corrected image will have an improved quality.

Calibration sub-modules operate on Visibility data items and Calibration Solutions data items. Visibilities are handled by the Visibility Arithmetic, Visibility Resampling and Visibility Flagging as well as the Phase Rotation modules. Calibration Solutions are worked on by the Solution Flagging, Application, Solving, Solution Operations, and the Solution Resampling modules. We will take a closer look in the following sub-sections.

2.1.1.2.1. Visibility Arithmetic

Simple arithmetic visibility operations with low computational complexity.

- uv-subtract: subtract one set of visibilities from another (in order to create residual visibilities)
- uv-add (e.g. Visibility Predict or Degridding output)
- uv-scale (e.g. change flux scale)
- uv-ratio (e.g. XX/I, Q/U, XX/YY)
- polarisation conversions
- DFTs (reverse of Visibility Predict)
- simple baseline calculations like length and azimuth angle (e.g. for weighting)
- weights arithmetic? (e.g. estimation, calculation after averaging, tapers, inverse tapers to down weight short baselines and diffuse emission, etc.)
- simple statistics?
- sorting?
- numerical derivatives in time or frequency? (but this could also be part of the Solving module)
- Calculation of Doppler-shifts

Variants: -
Performance: Estimated < 0.1% of total computation
Implementation: This is probably simple enough to be re-implemented by SDP. Maybe for some functions the Casacore library needs to be used [RD07].

2.1.1.2.2. Visibility Resampling

Operations changing visibility data density. Low performance requirements, but critical for keeping the size of visibility data under control.

- Average: transform the visibilities to coarser time-frequency resolution
- Interpolate: transform the visibilities to denser time-frequency resolution
- Coalesce: apply baseline dependent averaging (BDA) to visibilities
- De-coalesce: invert from baseline-dependently averaged visibilities

Variants: -
Performance: Estimated < 0.1% of total computation
Implementation: SDP will implement this without re-use.

2.1.1.2.3. Phase Rotation

Rotate visibilities to a different phase center, possibly changing the projection plane in the process. Used for working with visibilities corresponding to different fields of view (such as facets) and reducing the complexity of de/gridding. Equivalent to Reprojection in the image domain.

Variants: -
Performance: Phase Rotation needs to be performed at the full visibility resolution and for every facet. This makes that, even though the cost per visibility - pixel pair is relatively minor,

the total computational performance contribution may become substantial. Estimated ~2% of total computation.
Implementation: SDP will implement this without re-use.

2.1.1.2.4. Visibility Flagging

Flagging of visibilities would be implemented as two sub-modules:

1. One based on ranges that are specified by the Observatory at the start of the processing (e.g. given stations / dishes, given baselines, given frequency channels, etc.).
2. The other based on the visibility data itself by detecting outliers from average and / or median values of the data

Variants: Module 2 (based on the visibility data itself) comes in two variants: one for Batch workflows and one for Real Time workflows. Since the time-frequency data ranges that the Flagger will work on will be different for Batch and Real-time workflows the optimal way of determining statistics will be different. Therefore, two optimized variants are foreseen.
Performance: Estimated ~1% of total computation
Implementation: AOFlagger [RD14], CASA [RD10], ASKAPSoft [RD13]

2.1.1.2.5. Solution Flagging

Flagging of visibilities based on calibration solutions. Outlier Calibration Solutions are flagged and those flags are propagated to the corresponding Visibility data samples.

Variants: -
Performance: Flagger is not performance critical. Estimated < 0.1% of total computation
Implementation: SDP may want to re-use existing Flaggers.

2.1.1.2.6. Application

Apply the calibration solutions to the visibilities

Variants: -
Performance: Estimated ~1% of total computation
Implementation: SDP will implement this without re-use.

2.1.1.2.7. Solving

Fit model parameters to the visibilities

There will be sub-modules for the implementation of different algorithms for fitting, like

- Levenberg-Marquardt,
- SAGECal (Consensus Optimization),
- StEFCal,
- Linear Fitting.

The use of different norms (L2, L1) can be dealt with through parameter settings.

Variants: There will be variants to allow for optimizations towards different accelerators (CPU, GPU, FPGA, …)
Performance: Parameter fitting can be a performance bottleneck, depending on the Solving Strategy. Various algorithms will have different computational complexity depending on number of stations, frequencies, directions, and granularity of the data. Some might only be appropriate in the context of Workflows. The Parametric Model [AD04] estimates 1% total compute, but the performance impact is likely higher especially for distributed algorithms.
Implementation: known implementations: CASA [RD10], LOFAR-BBS [RD12], SAGECal [RD12], MeqTrees [RD16], ASKAPSoft [RD13]. Whether these implementations are fit for re-use is TBD.

2.1.1.2.8. Solution Resampling

- Average: transform to coarser time-frequency resolution (but need to consider things like Jones matrix ambiguities).
- Interpolate:
    - transform to denser time-frequency resolution
    - interpolate over solutions that are missing due to flagging
- Merging of solutions from different parts of SDP (e.g. frequencies or times) into a single set of solutions.
- Combination of ionospheric phases from different frequency bands.
- Fitting of linear features to phases of neighbouring stations. (This may go elsewhere.)

Variants: -
Performance: Estimated ~1% of total computation
Implementation: SDP will implement this without re-use.

2.1.1.3. Imaging

The Imaging Components are aimed at creating images from visibilities and then deconvolving images to remove effects of imperfect sampling.

We have three groups of Imaging sub-modules centered around Images:

- Transformation between Visibilities and Images (and vice versa): Gridding/De-Gridding, Preconditioning/Weighting and FFT/iFFT
- (Minor Cycle) Deconvolution of Images, which creates Sky Components from Images: Deconvolve
- Manipulation of Images: Image Arithmetic and Reprojection

2.1.1.3.1. Gridding / De-Gridding

Transform between sampled visibilities and the regular uv-grid. This is a necessary prerequisite for allowing the reconstruction of images using FFT algorithms.

This requires compensation for the grid's regularity as well as correcting for instrumental effects such as baseline co-planarity, reception patterns and calibration. In practice this

means that visibilities will have to be convolved with a number of extra terms in this process: anti-aliasing, W-term, A-term, plus eventually one for ionospheric correction. Currently the standard algorithm is to use oversampled convolution kernels and then a nearest neighbour algorithm for putting visibilites on the grid. We may need a new Component that constructs Gridding Kernels from Calibration Solutions, e.g. for Ionospheric correction, or correction for refined Primary Beam patterns.

Variants: Performance optimized variants will be created depending on the particular Workflow (Buffer-Continuum Imaging, Buffer-Spectral Line Imaging, RT-Fast Imaging for Slow Transients). Optimizations will depend on type of hardware and data access patterns (and, hence, data distribution schemes). A high level of co-design is expected. Various variants of the W- and A-Projection algorithm exist, each having different performance characteristics (see below). Image Domain Gridding may be available for further optimization, depending on the particular data access pattern.
Performance: Gridding and FFT dominate the computational performance (see [AD04]). Optimizing Gridding needs co-design of Data Access Pattern and hardware architecture. Various optimizations exist for minimizing the convolution kernels that need to be applied: W-Stacking (smaller kernels at the expense of higher memory usage), W-Snapshots (smaller kernels at the expense of reprojection of snapshot images). Convolution Kernels will be computed on-the-fly (since they need updating on short timescales). Image Domain Gridding optimizes the computation of the Convolution Kernels.

This is estimated to contribute about ~13% to the total SDP computation. Unclear whether high algorithmic efficiency can be achieved.
Implementation: Known implementations: CASA [RD10], AW-Imager [RD12], WS-Clean [RD15], ASKAPSoft [RD13]. Whether these implementations are fit for re-use is TBD.

2.1.1.3.2. Preconditioning / Weighting

Different weighting schemes for the uv-samples trading sensitivity (i.e. image thermal noise level) against image resolution

- Uniform
- Briggs
- Natural

Furthermore, in order to minimize differences in the Point Spread Function (PSF) over frequency channels, the calculation of visibility weights based on frequency-integrated density is supported.

In addition, the PSF can be shaped by Gaussian and / or Tukey tapering of the visibility weights. There may be the need for other types of tapering as well, e.g. elliptical tapering. Also, with many short baselines it may be useful to have an inverse taper that weighs down short baselines (but perhaps only for calibration).

Variants: Traditional Weighting schemes (except for Natural Weighting) require a full pass through all UVW and FLAG to determine the imaging weights. Only then can the Gridding step start. ASKAPSoft implemented a-posteriori weighting using a Wiener filter, for which the extra pass through UVW and FLAG data is not needed.
Performance: Estimated <0.1% of total computation
Implementation: Known implementations: CASA [RD10], AW-Imager [RD12], WS-Clean [RD15], ASKAPSoft [RD13]. Whether these implementations are fit for re-use is TBD.

2.1.1.3.3. FFT / iFFT

Transform between uv-grid and image grid (and vice versa). Various optimized implementations for the Fast Fourier Transform exist, which may give rise to different sub-modules. If > 95% of the pixels are zero a Sparse Fourier Transform (SFT) may be useful, but this will only be relevant for the Fast Imaging workflow (as part of the Real-time workflows).

Variants: -
Performance: Performance critical for the large image sizes needed for the SKA, it is estimated that ~11% of SDP computation will be spent in image/grid FFTs. The Imaging support document [RD04] reports efficiencies between approximately 8% and 15% of peak.
Implementation: Known implementations: CASA [RD10], AW-Imager [RD12], ASKAPSoft [RD13], WS-Clean [RD15] using FFTW library [RD08]. Whether these implementations are fit for re-use is TBD.

2.1.1.3.4. Image Arithmetic

- Add / Subtract: add / subtract two images
- Shift: shift the origin of the coordinate system (but not the tangent point of the image to the celestial sphere)
- Interpolation: transform to a denser l-m coordinate system (Or is this part of the Data Model modules?)
- Stitching: create a large image from smaller images, e.g. for mosaicing.

Variants: -
Performance: Estimated <0.1% of total computation
Implementation: -

2.1.1.3.5. Reproject

Project the image onto a new tangent plane to the celestial sphere. Equivalent to Phase Rotation in the visibility domain. This is needed for snapshot imaging, where ‘snapshot images’ are first created on planes tangent to the array and then combined (i.e. reprojected) onto a final tangent plane at fixed RA, Dec.

Variants: -
Performance: Estimated ~4% of total SDP compute
Implementation: -

2.1.1.3.6. Deconvolve

Minor cycle deconvolution algorithms, where sky components are found / fitted to the Dirty Image.

There will be sub-modules for different algorithms. Notably:

- the Multi Scale Multi Frequency Synthesis (MS-MFS) algorithm is used for deconvolution of Continuum Images,
- the Multi Scale (MSClean) algorithm is used for deconvolution of Spectral Line Cubes

Variants: There will be variants to allow for optimizations towards different accelerators (CPU, GPU, FPGA, …)
Performance: The image sizes that the Deconvolution module has to work on may be very large, see Performance Modelling [AD04]. This means we have to consider the amount of memory that is used, and facetting as an image distribution scheme. In which case we will be forced to use distributed processing components with e.g. MPI communication. This is estimated to account for about 5% of SDP computation.
Implementation: Known implementations: CASA [RD10], ASKAPSoft [RD13], WSClean [RD15] … Whether these implementations are fit for re-use is TBD.

2.1.1.4. Source Detection

The Source Detection sub-modules deal with Sky Components.

The following sub-modules transform Images into Sky Components (and vice-versa):

2.1.1.4.1. Source Finding

Find the position of a source component in an image.

Variants: There might be different approaches to Source Finding / Estimation. Note that the requirements within the ICAL pipeline are less demanding than for Scientific Analysis of images. [RD05]
Performance: Non-critical, not estimated
Implementation: Known implementations: DUCHAMP, BLOBCAT, AEGEAN, BDSF [RD05]. Whether these implementations are fit for re-use is TBD.

2.1.1.4.2. Source Estimation

Estimate the source flux, polarization, spectrum (e.g. MFS Taylor terms) and morphology for a component at a position in an image.

Variants: There might be different approaches to Source Finding / Estimation. Note that the requirements within the ICAL pipeline are less demanding than for Scientific Analysis of images. [RD05]
Performance: Non-critical, not estimated

Implementation: Known implementations: DUCHAMP, BLOBCAT, AEGEAN, BDSF [RD05]. Whether these implementations are fit for re-use is TBD.

2.1.1.4.3. Image insertion

Insert a pixelated version of a source component in an image

Variants: -
Performance: Non-critical, not estimated
Implementation: -

2.1.1.4.4. Visibility predict

Predict visibilities from Sky Components using Direct Fourier Transforms. This should support a number of component models, including:

- Point Sources
- Gaussians
- Shapelets

Note that Fast Fourier Transform Predict is also implemented, but would use (gridded) Fast Fourier Transforms followed by Degridding.

Variants: -
Performance: The Visibility Predict by means of the Direct Fourier Transform is a performance critical component, since it scales with the number of visibilities times the number of discrete source components to be treated in this way [AD04]. It is estimated to contribute about 60% of the SDP computation, yet high operational intensity might be more straightforward to realise than with other modules.
Implementation: Known implementations: CASA [RD10], LOFAR-BBS [RD11], MeqTrees [RD16], ASKAPSoft [RD13], SAGECal [RD12], AIPS [RD17]. Whether these implementations are fit for re-use is TBD.

2.1.1.4.5. Fit Sky Components

Fit Sky Component (parameters like position and flux) directly to Visibilities.

Variants: -
Performance: Not estimated
Implementation: Known implementation: LOFAR-BBS, MeqTrees. Whether these implementations are fit for re-use is TBD.

2.1.1.5. Non-Imaging

Components for Non-Imaging applications (NIP). Only the Pulsar Pipelines require (post) processing. VLBI data and Transient Buffer data do not require processing and, therefore, do not show up in the element catalog. See SDP Pipelines Design document [RD01] for details.

2.1.1.5.1. Pulsar Search

Processing components for the Pulsar Search and Single Pulse workflows.

Variants: Probably a single version suffices.
Performance: Main processing is performed by CSP. SDP only performs post-processing. For SDP these modules do not generate performance bottlenecks.
Implementation: Prototype implementations exist and can be used to construct the SDP implementation.

2.1.1.5.2. Pulsar Timing

Processing components for the Pulsar Timing workflow.

Variants: Probably a single version suffices.
Performance: Main processing is performed by CSP. SDP only performs post-processing. For SDP these modules do not generate performance bottlenecks.
Implementation: Prototype implementations exist and can be used to construct the SDP implementation.

2.1.1.6. Processing Libraries

Low-level algorithms will be split out into separate libraries, since these algorithms are used in multiple areas of the decomposition tree. Examples are:

- Functions that provide values in astronomical reference frames using physical units. E.g. as currently implemented in Casacore Measures [RD07]
- Solvers; some are currently implemented in Casacore Scimath [RD07]
    - Levenberg Marquard
    - StEFCal
    - Linear Least Squares
- FFT, e.g. FFTW [RD08], CUFFT
- Gridders
- Flaggers

Variants: For each algorithm there may be multiple variants to allow for various optimizations towards different hardware platforms (CPU, GPU, FPGA, …)
Performance: These are low-level algorithms, including some for the performance bottlenecks of SDP (e.g. FFT, Solvers, Gridders)
Implementation: In-house

2.1.2. Processing Wrappers

2.1.2.1. Processing Component Wrapper

Wrapper to make the Processing Component agnostic of the Execution Framework.

Variants: One per supported Execution Framework
Performance: No performance bottleneck

Implementation: In-house

2.1.2.2. Data Redistribution

Processing Components will work on subsets of data (e.g. sub-bands, sub-arrays or snapshots) and are not aware of the global data distribution. Therefore the data distribution task has to be handled by the Execution Framework.

Variants: One per supported Execution Framework
Performance: Data redistribution may come at a substantial performance cost. Its use should be weighed against sub-optimal performance of the Processing Components.
Implementation: In-house

2.1.2.3. Realtime & Queue I/O

Data can be exchanged in real time by two methods:

- Received from the instrument (Ingest). This data will not only be written to the Buffer, but will also made available to Real-time processing.
- Furthermore, Data Queues can be used to read and write data in real time. This can be used to exchange data with other running workflows (e.g. for cooperation on calibration solving), or to publish data about quality assessment or calibration to other components of the SDP or SKA.

In either case the wrappers need to manage the execution engine such that we can inject data arriving at real-time.

Variants: Per Execution Framework
Performance: Receive rate up to ~1 GB/s (for a case with 400 Receive nodes), Queue I/O likely <100 MB/s per node
Implementation: In-house

2.1.2.4. Buffer I/O

The principal mechanism for workflows to obtain data is by reading it from Buffer storage. This will generally involve using a Data Island's File System Interface to read and write data in Buffer Data Model Form and translate it to and from the appropriate Memory Data Model so the workflow or Processing Components can handle it.

Variants: Per Execution Framework, possibly further specialised by Storage Backend type to realise the highest possible data rates.
Performance: Read rate of up to ~3 GB/s (for 1500 nodes and 10 major loops)
Implementation: In-house

2.1.3. Memory Data Models

See System-level Data Model View [AD03]. Apart from the Raw Input Data, the Telescope Model Data and Telescope State Information (which are subsets of the Science Data Model and indicated here as Science Data Model Information) are important data items for the

Workflows and its Processing Components. The relation between these two data items and the Processing Components needs to be worked out in more detail.

Variants:
Performance:
Implementation:

Data types:

- Visibilities
- Calibration Solutions
- Images
    - Image grids
    - Visibility grids
    - Convolution Kernels
- Sky Data
- NIP Data
    - Pulsar data
    - Transient Buffer data
    - VLBI data (?)
- Science Data Model Information

2.1.4. Buffer Data Models

See System-level Data Model View [AD03]. Apart from the Raw Input Data, the Telescope Model Data and Telescope State Information (which are subsets of the Science Data Model and indicated here as Science Data Model Information) are important data items for the Workflows and its Processing Components. The relation between these two data items and the Processing Components needs to be worked out in more detail.

Variants:
Performance:
Implementation:

Data types:

- Visibilities
- Calibration Solutions
- Images
    - Image grids
    - Visibility grids
    - Convolution Kernels
- Sky Data
- NIP Data
    - Pulsar data
    - Transient Buffer data
    - VLBI data (?)
- Science Data Model Information

Data types may use a Data Access Library for

- Access to Visibilities. E.g. Casacore MS [RD07]; Note the SKAO (SDP) - NRAO collaboration for the development of MSv3
- Access to Images. E.g. Casacore Images [RD07]
- Coordinates, e.g. World Coordinate System [RD09]

2.2. Relations and Their Properties

Workflows can be constructed from these low level Processing Components. It makes sense to have low-level Workflows that can be used within high-level Workflows. E.g. a Workflow for ‘Strong Source Removal’ that can be used within a Workflow for ‘Pre-processing’.

2.3. Element Interfaces

Processing Components are wrapped and their interfaces to Data Models and Execution Framework go through these wrappers. See Figure 3 (Trivial Execution Engine Example) of the Processing Component & Connector View [AD02].

2.4. Element Behavior

Processing Components only interact with Data and the Execution Framework. In principle this is flexible, allowing the Observatory to create and execute dedicated workflows from the current set of Modules. The application of this to workflows important to the SKA is out-of-scope for this document, but will be illustrated in a separate behaviour document which is to be written for CDR. The way a workflow fits within the SDP architecture is described in the Processing Module View document [AD01]. The Data Distribution aspect of workflows is also out-of-scope for this document and will be described in a dedicated Data Distribution View which is to be written for CDR.

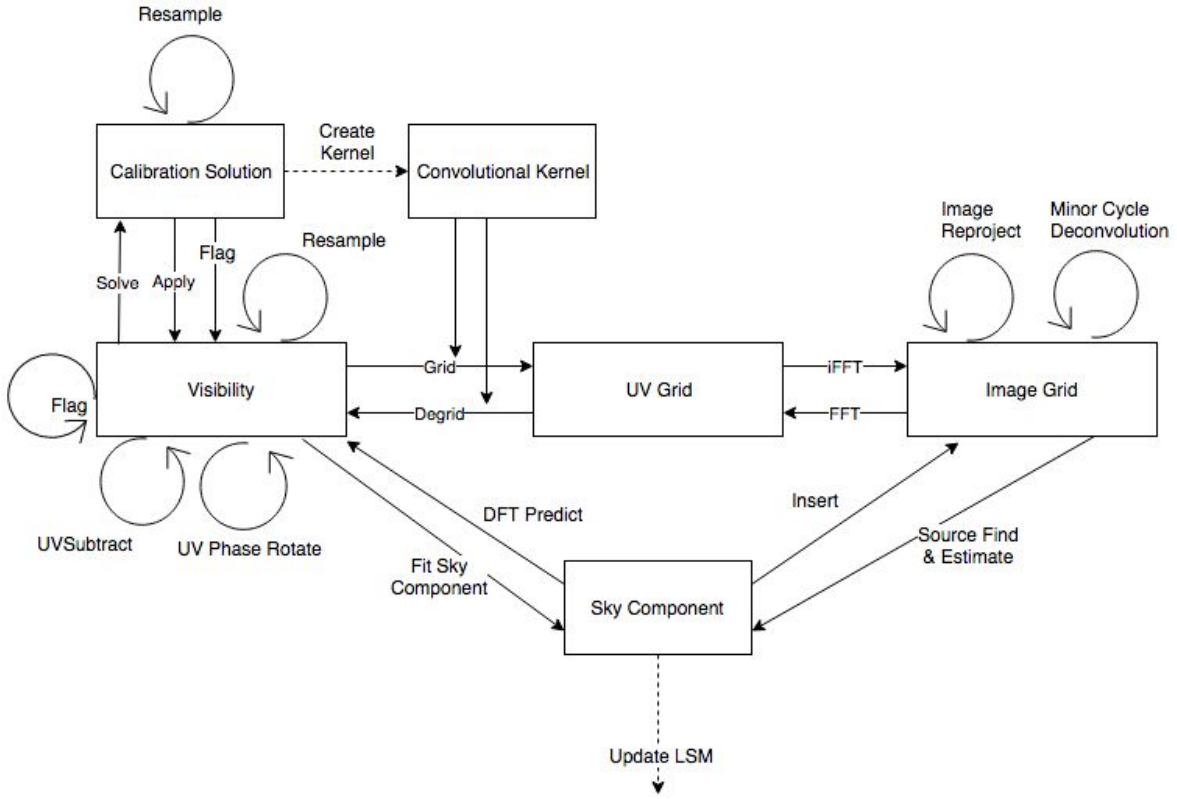

Figure 2: Data Transition diagram for Imaging applications

The module decomposition (for the imaging applications) is based on the data transitions as shown in Figure 2. As a guiding principle components are grouped together if they work on the same type of data. In this way we may achieve a 'clean' split of modules. Note that the Data Transition diagram (figure 2) is consistent with the Algorithm Reference Library (ARL) [RD06].

A Dependency Matrix mapping (Sub-) Modules onto Workflow behaviours (or Functions; [RD01]) is given below.

		F1.3 Pre-Process Visibility Data Fast	F1.4 Calibrate Real-time	F1.5 Image Fast	F1.8 Pre-Process data	F1.9 Calibrate and Image	F1.13 Process Pulsar Timing	F1.14 Process Pulsar Can-didates	F1.15 Process Tran-sient Buffer
Calibration	Visibility Arithmetic	x	x		x	x			
	Visibility Resampling	x			x	x			
	UV Phase Rotation	x	x		x	x			
	Visibility Flagging	x			x	x			
	Solution		x			x			

	Flagging								
	Application	x			x	x			
	Solving	x	x		x	x			
	Solution Operations					x			
	Solution Resampling	x			x	x			
Imaging	Image Arithmetic			x		x			
	Create Image			x		x			
	De- / Gridding			x		x			
	FFT/iFFT			x		x			
	Deconvolve			x		x			
	Reproject			x		x			
	Weighting			x		x			
Source Finding	Source Finding			x		x			
	Source Estimation			x		x			
	Image Insertion					x			
	Visibility Predict					x			
	Fit Sky Component					x			
NIP	Pulsar Timing						x		
	Pulsar Search							x	

3. Context Diagram

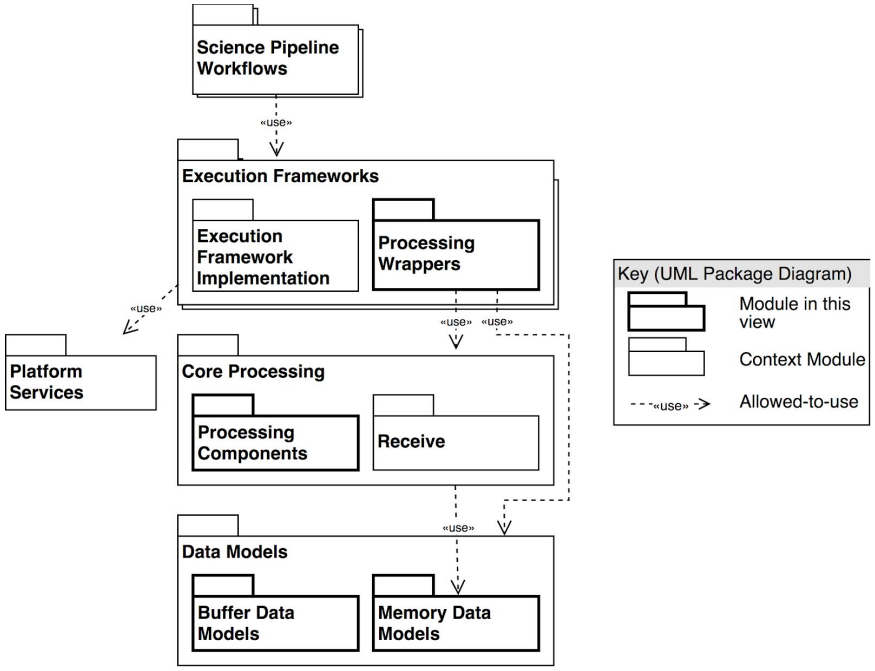

Figure 3: Workflow Modules Context Diagram

This view documents a section through processing-related modules as shown above: It ignores the execution framework implementation and Receive to instead focus entirely on Processing Wrappers and Processing Components. Therefore this view is not complete with respect to how Workflows get executed, and ignores the interfaces used to obtain real-time measurement data from the telescope.

4. Variability Guide

Processing Components are expected to be a highly variable part of the SDP architecture, therefore there are a large number of variation mechanisms:

- Using the Processing Component Interface, Processing Components are de-coupled from the Execution Engines and Workflows using them. Therefore both new as well as modified Processing Components can easily be introduced into the system.
- Through the same mechanism we can especially allow a large number of Processing Components to co-exist within the same architecture. This should allow SDP to experiment with new algorithmic developments and optimisations for accelerator architectures.
- Through usage of common data models we especially strongly suggest Processing Components to remain composable. As long as we do not allow “forks” in the data models we should be able to avoid “parallel universe” scenarios of certain sets of workflows and Processing Components become unable to share functionality with each other.
- Science Data Model information provides a flexible way to provide Processing Components with meta data, both about the observation as well as about the

Workflow. This should allow us to parametrise Processing Components in a flexible manner even as requirements change.

5. Rationale

5.1. Modifiability

The key architectural decision associated with this view is that the processing components form a reusable library that can be used by all of the workflows and execution engines supported by the SDP, and for this reason they are all organised together in a single top level module.

This is a horizontally integrated pattern, i.e., different functions of the SDP all use the same processing component module. The rationale for this are savings in construction and maintenance costs as only one, coordinate, set of modules needs to be built and maintained. This needs to be traded off against:

- organisational difficulties in specifying and building a single widely-used module
- potential impact on construction timeline critical path, and
- the ways processing components are used by workflows might become too diverse. This might lead to Processing Components getting developed for specific roles in workflows and execution engines, which would eliminate possible savings.

Furthermore, the reusability of the processing components between different execution engines drives the decision to separate out the data model in a high-level module of its own.

5.2. Maintainability

One of our key architectural drivers is that everyone needn't change their thinking, to aid maintainability: We have a set of workflows, which we can think of in procedural ways.

5.3. Scalability

To solve the problem at scale, we translate that procedural view into a data-driven view, and that is done by the workflow engine. Providing the right sort of environment to script their workflows also comes from the engine. At the lower level, to enable that, we specify that we have purely functional components, that we can string together into workflows.

5.4. Portability

Processing components can be used in different Workflows and by different Execution Frameworks. This is achieved by letting all processing components have the same Component Interface that interfaces with the Execution Framework by means of a wrapper.

6. Related Views

The current Workflows Module View provides more detail on parts of the System-level Module Decomposition and Dependency View, [AD01]. In particular for the Processing Components sub-module of the Core Processing Module and the Memory Data sub-module and Buffer Data sub-module of the Data Models module, see Figure 1 (Primary

Representation) [AD01]. Note that the Receive sub-module of the Core Processing module is not considered in this document, but will eventually be described in a separate document.

The System-level Data Model View [AD03], provides context for the Memory and Buffer Data Models. Note that the relation of the Processing Components to the Science Data Model (the 'meta-data') is currently not worked out. This needs attention in future updates of this document.

7. References

7.1. Applicable Documents

The following documents are applicable to the extent stated herein. In the event of conflict between the contents of the applicable documents and this document, **the applicable documents** shall take precedence.

[AD01] P. Wortmann et al., System-level Module Decomposition and Dependency View, SKA-TEL-SDP-0000013 rev 05

[AD02] B. Nikolic et al., Processing Component & Connector View, SKA-TEL-SDP-0000013 rev 05

[AD03] K. Kirkham et al., System-level Data Model View, SKA-TEL-SDP-0000013, rev 05

[AD04] R. Bolton et al., Parametric models of SDP compute requirements, SKA-TEL-SDP-0000040, revision 1C, 2016-03-24

7.2. Reference Documents

The following documents are referenced in this document. In the event of conflict between the contents of the referenced documents and this document, **this document** shall take precedence.

[RD01] R. Nijboer et al., SDP Pipelines Design, SKA-TEL-SDP-0000027, 2016-05-24

[RD02] G. van Diepen et al., SDP Memo: Receive and Pre-process Visibility Data, SKA-TEL-SDP-0000028, 2016-04-06

[RD03] S. Salvini et al., SDP Memo: The SDP Calibration Component, SKA-TEL-SDP-0000029, 2016-04-07

[RD04] A. Scaife, SDP Memo: The SDP Imaging Pipeline, SKA-TEL-SDP-0000030, 2016-04-07

[RD05] M. Johnston-Hollitt et al., PDR.02.05.04 Science Data Analysis Pipeline Reference Document, SKA-TEL-SDP-0000031, 2015-02-09

[RD06] T. Cornwell, Algorithm Reference Library (ARL) documentation; http://www.mrao.cam.ac.uk/projects/jenkins/algorithm-reference-library/docs/build/html/ARL_directory.html

[RD07] Casacore documentation; http://casacore.github.io/casacore/

[RD08] FFTW documentation; http://www.fftw.org/

[RD09] WCSLib documentation; http://www.atnf.csiro.au/people/mcalabre/WCS/wcslib/index.html

[RD10] CASA software documentation; https://casa.nrao.edu/

[RD11] LOFAR software documentation; https://www.astron.nl/radio-observatory/lofar-data-processing/software-processing-tools/software-processing-tools

[RD12] LOFAR imaging cookbook; https://support.astron.nl/LOFARImagingCookbook/index.html

[RD13] ASKAP software documentation; https://www.atnf.csiro.au/computing/software/askapsoft/sdp/docs/current/index.html

[RD14] A.Offringa et al., Post-correlation filtering techniques for off-axis source and RFI removal, MNRAS, 422, 563 - 580, 2012.

[RD15] WSClean software documentation; https://sourceforge.net/p/wsclean/wiki/Home/

[RD16] MeqTrees software documentation; http://meqtrees.net/

[RD17] AIPS software documentation; http://www.aips.nrao.edu/index.shtml

8. Version History

Version	Date of Issue	Prepared by	Comments
05	2018-04-18	R. Nijboer	Submitted for SDP M20 pre-CDR review