SDP System-level Module Decomposition and Dependency View

Document number..... SKA-TEL-SDP-0000013
Document Type..... REP
Revision.....05
Authors..... P. Alexander, V. Allan, U. Badenhorst, C. Broekema, S. Gounden, F. Graser, K. Kirkham, B. Mort, R. Nijboer, B. Nikolic, R. Simmonds, J. Taylor, A. Wicenec, P. Wortmann
Release Date..... 2018-04-18
Document Classification..... Unrestricted
Status.....Released

Lead Author	**Designation**	**Affiliation**
Peter Wortmann	SDP Architecture Team	University of Cambridge
Signature & Date:	Peter Wortmann (Apr 24, 2018)	

Released by	**Designation**	**Affiliation**
Paul Alexander	SDP Project Lead	University of Cambridge
Signature & Date:	Paul Alexander Paul Alexander (Apr 24, 2018)	

Table of Contents

List of Abbreviations

CSP	Central Signal Processor
LFAA	Low Frequency Aperture Array
SDP	Science Data Processor
SKA	Square Kilometre Array
SRC	Science Regional Centre
TM	Telescope Manager

1. Primary Representation

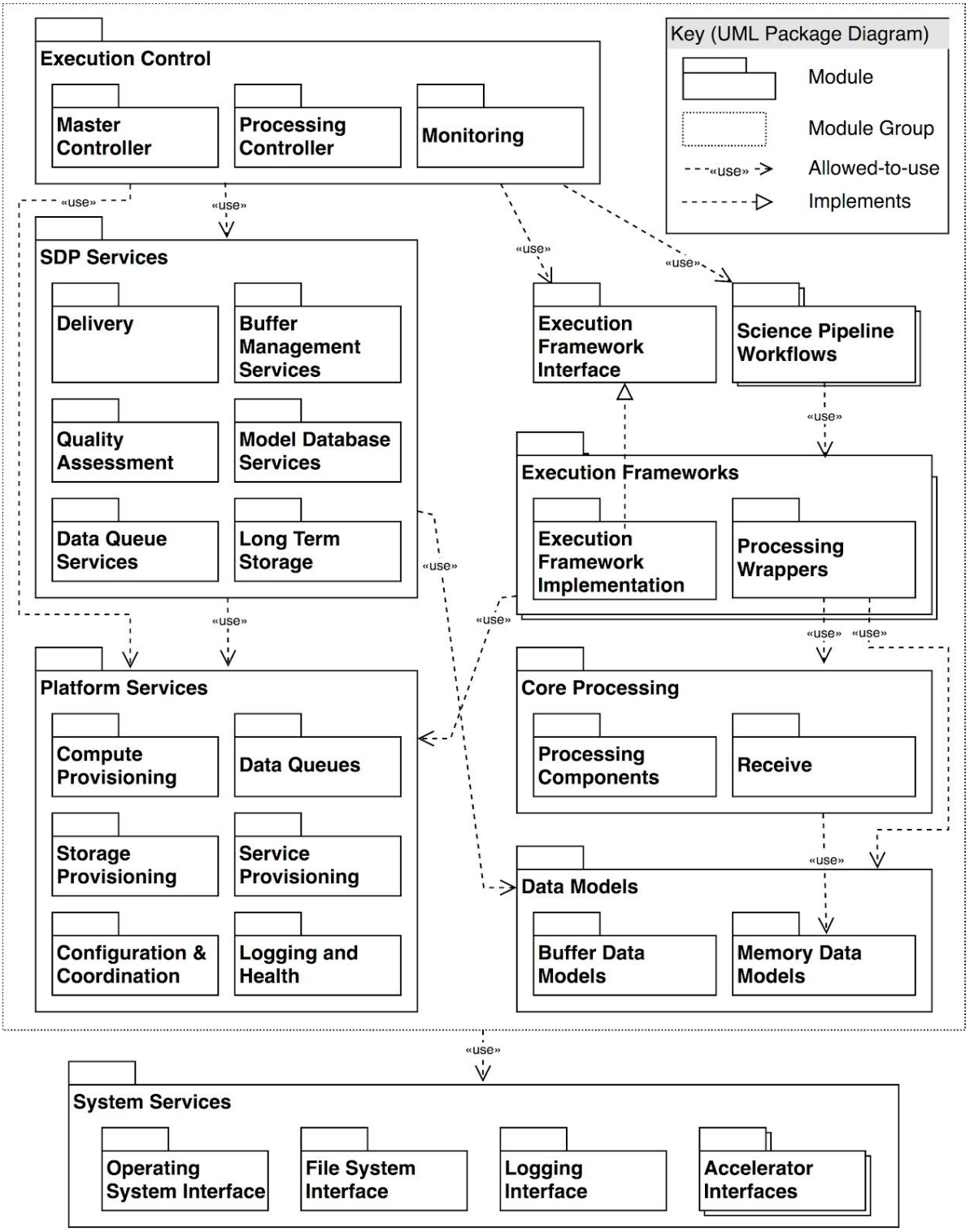

Figure 1: Primary representation of the SDP in terms of “Allowed-To-Use” and module decomposition style view

Elements are units of implementation referred to as modules. “Allowed-to-use” relationships are shown as arrows annotated with “use”; “Is-part-of” relations between modules are shown

by nesting; “implements” realises the interface. More detailed explanations of the meaning of elements and relations are described in the Element Catalogue.

Reading guide: The module view is split into three layers. Execution Control at the top is responsible for providing the top-level control interfaces and coordinating the more loosely coupled services and workflows at lower levels. Execution Control is split between the Master Controller, which is responsible for starting and maintaining services, and the Processing Controller, which handles scheduling and execution of processing. At the bottom sits System Services, which offers interfaces that might get used throughout the SDP architecture.

The middle layer is split into two pillars, with the left side showing service modules and the right side showing SDP processing modules.

- For the service modules, these are again split into two layers, with SDP Services providing SDP-specific functionality such as Quality Assessment, Delivery and Model Databases. Platform Services offers more general-purpose services such as infrastructure for Compute or Service Provisioning, maintaining Configuration and Coordination information, and aggregating Logging and Health Information.
- For the processing modules, considerable effort has been put into decoupling modules vertically: Execution Frameworks are modules that execute distributed Science Pipeline Workflows, controlled via a generic Execution Framework Interface. Processing Wrappers encapsulate Processing Component and Receive modules from the next layer, which perform the actual processing.
- Processing Components and Receive are only allowed to depend on Data Models and System Services. They are meant to be implemented as purely functional components with minimal implicit dependencies on other parts of the system. This is to ensure modifiability, testability and encapsulation of performance concerns.
- Data Models define data formats and provide their implementations, and will be used throughout the architecture. Those are further split into Buffer and Memory Data Models to reflect the difference in performance guarantees.

The only interfaces available to all modules are System Services, which offer general-purpose interfaces to the Operating System, Storage as well as Logging and Accelerator facilities.

2. Element Catalogue

This section is organised as a dictionary where each entry is an element of the Primary Representation.

2.1. Elements and Their Properties

This section explains the elements of the Primary Representation, identifies particular properties and highlights their functionality. As this view is about modularisation, we especially attempt to identify where we can encapsulate certain **Expertise** in order to ensure

buildability. This might open up certain **Implementation** options such as leveraging off-the-shelf solutions to improve affordability as well.

2.1.1. Execution Control

Groups all modules related to the control and monitoring of the SDP. This especially handles top-level resource allocation and process (services / execution framework) orchestration.

2.1.1.1. Master Controller

Tracks status of the SDP. Handles provisioning of compute resources to SDP Services, most of which will be static configuration of the platform.

Expertise:	System operation
Implementation:	In-house

2.1.1.2. Processing Control

Manages Processing Block execution according to resource availability. The Workflow Engine coordinates SDP services and execution frameworks to execute Science Pipeline Workflows.

Expertise:	Observation/processing planning and resource management
Implementation:	In-house, possibly utilising off-the-shelf scheduling and resource allocation solutions

Additional Functionality:

- The **Subarray & Processing Block interfaces** implement TM's subarray and processing block control interfaces, maintaining information received from TM in the coordination database.
- **Resource scheduling:** Assign buffer and compute resources to scheduling blocks and processing blocks. This function will make use of the SDP resource model, which is part of the SKA Core Software (see context diagram in Section 3).
- **Workflow Engine:** Manages high-level execution of Science Pipeline Workflows. See Science Pipeline Workflow sub-module for more details on steps required by such workflows, but this especially includes:
    - Workflow (step) scheduling
    - Execution Framework control
    - Buffer/Long Term Storage control
    - Model Database control
    - Data Queues control
    - Quality Assessment control
    - Delivery control

2.1.1.3. Monitoring

Collects information about global operational status and service health using information provided by Platform Services and SDP Services. This especially concerns following logging and other system health information and filtering out information relevant to the Telescope Manager.

Note that collection of health and logging information is mainly implemented in Platform Services as the (likely off-the-shelf) Logging and Health sub-module. This module implements the SDP-specific filtering and decision making depending on that data.

Expertise: System operation

Implementation: In-house

2.1.2. SDP Services

Groups modules that provide domain-specific SDP services to support processing. As this is a data-driven architecture, this especially concerns maintaining data items around the execution of Science Pipeline Workflows: Buffer Services and Data Queue Services maintain the two main data exchanges of the SDP architecture. Model Databases, Data Preparation and Delivery Services as well as Long Term Storage manage long-term data stores serving and/or getting updated by processing.

2.1.2.1. Delivery

Assembles final data products and delivers them to the observatory or regional centres. Maintains the Science Data Product Catalogue using data from Scheduling Blocks and the Science Data Model produced by Model Databases. Enables Data Product discovery and manages outgoing data transfers to Science Regional Centres.

Expertise: Distributed systems, IVOA services, Science Data Model

Implementation: In-house integration of open source components

2.1.2.2. Quality Assessment

Handles assessment of partial results from Science Pipeline Workflows to allow previews into the quality of the final data products. Might involve some analysis of the data.

Expertise: Science pipeline operation

Implementation: In-house

2.1.2.3. Long Term Storage

The persistent background storage to the Buffer. Provides long-term storage for data products. Management of storage lifecycle might overlap with Buffer Management Services.

Expertise: Data preservation

Implementation: Off-the-shelf with custom control

2.1.2.4. Model Database Services

Produces the Science Data Model from the Global Sky Model database and the TANGO connection to TM. Handles queries and updates to the Sky Model database.

Expertise: Science Data Model

Implementation: In-house, likely utilising off-the-shelf databases internally

2.1.2.5. Buffer Management Services

Arranges storage of input data for workflow execution as well as Data Products. Manages

lifecycle of storage instances across buffer and long time storage. Implements Data Island Interface for applications to access storage.

Expertise:	Data lifecycle management
Implementation:	Mixed in-house and off-the-shelf (might overlap with long term storage)

2.1.2.6. Data Queue Services

Handles initialisation and management of Data Queues as required for execution of Science Pipeline Workflows. Performs real-time forwarding of alert and calibration data to TM.

Expertise:	Real-time telescope operation
Implementation:	In-house

2.1.3. Platform Services

Groups modules that provide lower level, non-domain specific computing platform support services. It is expected that it will be implemented mostly using configured off-the-shelf modules. It especially provides the implementation of resource and service Provisioning (providing storage resources, databases on request, etc.), Data Queues (real-time data streaming), Configuration and Coordination, and Logging.

This module implements the components documented in the Platform C&C view [RD04].

2.1.3.1. Compute Provisioning

Provides the compute infrastructure to SDP modules to run applications in certain environments. This will encompass mechanisms for software defined infrastructure management, hardware configuration and management, deployment automation as well as inventory management.

The software environments provided will at minimum provide the System Services interfaces. Compute resources are provisioned by Master Controller and Processing Controller components, which then assign them to either SDP Services or Processing Components.

Expertise:	Cloud or Distributed Computing, Data Centre Administration
Implementation:	Mostly off-the-shelf (OpenStack, hosted inside OpenStack managed infrastructure or alternatives)

2.1.3.2. Storage Provisioning

Provides the storage infrastructure to SDP modules, which may be of a block, file or object nature. Most, but not necessarily all, large-scale provisioning of storage resources will involve Buffer Management Services, as other services might be allowed to request their own work space. High performance multi-tier storage will be used to provide the Hot/Cold Buffer spaces.

Expertise:	Cloud or Distributed Computing

Implementation:	Mostly off-the-shelf - OpenStack (or customised) Service, hosted inside OpenStack managed infrastructure, or alternatives

Additional Functionality required for provisioning of:

- “Hot buffer” storage - performance driven
- “Cold buffer” storage - capacity driven
- Object storage - flexibility
- Block Storage - reliability driven for compute images
- Distributed storage orchestration - such as File System as a service for scalability

2.1.3.3. Service Provisioning

Provides a set of standard services to the SDP that are expected to be uniformly useful to harmonise across the SDP. These may be, inter alia, high performance and resilient memory resident key object stores, DBMS as-a-service, NTP and DNS.

Expertise:	Cloud or Distributed Computing, Data Centre Administration
Implementation:	Mostly off-the-shelf - OpenStack (or customised) Service, hosted inside OpenStack managed infrastructure, or alternatives

2.1.3.4. Data Queues

Real-time service for streaming data between workflows and services. Used for loose coordination between executing workflows (e.g. global calibration) as well as publishing calibration solutions, alerts and Quality Assessment data.

Expertise:	Data streaming
Implementation:	Off-the-shelf

2.1.3.5. Configuration & Coordination

Highly available database storing SDP configuration information. Used for coordinating platform-wide behaviour, such as service registry/discovery, orchestration as well as providing a consistent picture of overall SDP state. Should support scalable notifications on configuration changes (possibly by providing guaranteed messaging).

Expertise:	Cloud or Distributed Computing, Data Centre Administration
Implementation:	Off-the-shelf

Additional Functionality:

- **Configuration:** Resilient distributed key/value store with safe transactions. Must be able to provide a guaranteed notification/messaging mechanism, such as ability to trigger on state changes or a message queue.
- **Service registry/discovery:** Tracks the location of SDP services, providing clients the ability to access them.

- **Health checks and resilience**: Perform application health checks (such as heartbeats), configured per application when it gets provisioned. Can be configured to restart applications automatically to allow autonomous recovery. Where recovery is not possible or fails, provides notifications to allow detection of failure states (i.e. by Master Controller or Processing Controller).
- **Authentication and Authorization, Secrets Management:** Access to the database should be authenticated in order to support robust multi-tenancy. Provides clients with tokens required to access resources.

2.1.3.6. Logging and Health

Collects and aggregates health information from all SDP infrastructure, such as compute, storage and networking. Provides fine-grained views into the system health both for local operators as well as to Execution Control modules. Note that in contrast to Monitoring (in Execution Control), this module is not expected to implement action based on this information.

Expertise:	Log and health metrics aggregation
Implementation:	Off-the-shelf

2.1.4. Science Pipeline Workflows

Data-driven pipelines configured by Processing Blocks, expressed in terms of a language as executed by the Workflow Engine (see Processing Controller). Specifies the necessary steps to execute a data-driven workflow, which will typically involve:

- Configuring the Buffer to provide required Data Islands
- Creating Data Queues for dynamic data
- Initialising Quality Assessment to generate appropriate metrics
- Using Model Database to extract a Science Data Model
- Employing Execution Framework instances to execute workflows on available processing resources. Note that this might involve starting multiple instances in parallel to allow top-level parallelism.
- Update the Science Data Product Catalogue in Data Preparation and Delivery

See section 2.4.4f in the Operational System C&C view [RD02] for behaviour examples.

Expertise:	Radio astronomy workflows
Implementation:	In-house

2.1.5. Execution Framework Interface

Provides a common interface to manage the interaction between Processing Controller and Execution Framework implementations. This will consist of a way for the Processing Controller to instantiate the Execution Framework for a Science Pipeline Workflow using provisioned resources - such as compute and storage, but also services such as data queues.

Furthermore, it should allow Processing Controller code to monitor and influence processing, such as providing configuration or cancelling execution. We would likely use Data Queue and Coordination Services to implement this control.

Expertise: Distributed Computing, HPC system support.

Implementation: In-house

2.1.6. Execution Frameworks

Used for execution of Science Pipeline Workflows stages. The SDP will support multiple independently developed and maintained Execution Frameworks, even within the same workflow.

Execution Framework instances parametrised by Workflows are called Execution Engines. See the Processing C&C View [RD03].

2.1.6.1. Execution Framework Implementation

Core functionality of an Execution Framework: Handles fine-grained scheduling of assigned compute resources, organises data movement and invokes Processing Component execution through Processing Wrappers.

Expertise: Data flow implementation

Implementation: Either off-the-shelf or in-house

2.1.6.2. Processing Wrappers

Wraps Processing Components, Receive and SDP service interfaces for Science Pipeline Workflows and the Execution Framework Implementation. Should allow the Execution Framework Implementation to instantiate and execute processing graphs in a distributed way without introducing strong coupling to the actual Processing Component implementations.

Implementing this might involve light data-model specific transformations such as splitting or merging data, or caching for example Science Data Model data. The Processing Wrappers are also expected to take care of conversion between different Data Models, such as Memory Data Models and Buffer Data Models. Especially note that this means that processing storage I/O is expected to be implemented by the Wrappers in coordination with the Execution Framework, not by Processing Components.

Expertise: Data flow kernel integration

Implementation: In-house

This module is further decomposed in the Processing Component Module View [RD01].

2.1.7. Core Processing

Libraries implementing core SDP-specific data processing. The main radio-astronomy and interferometry components are implemented as Processing Components. Just-in-time handling of incoming data from CSP and LFAA is handled by Receive libraries.

2.1.7.1. Processing Components

Library of domain-dependent radio astronomy and interferometry algorithm implementations consuming and producing data according to Memory Data Models.

Should implement a standardised Processing Component interface to make them as much as possible agnostic of how they are used in Science Pipeline Workflows or what Execution Framework is calling them.

For reproducibility, processing components should be referentially transparent, which is to say that output data should be entirely determined by input data. Exempt of this are problems like floating point rounding errors, which due to compilation differences and non-deterministic parallel execution are hard to control for.

Expertise:	Radio astronomy algorithms
Implementation:	In-house, some third party

This module is further decomposed in the Processing Component Module View [RD01].

2.1.7.2. Receive

Handles incoming data from CSP and LFAA (for transient buffer data). The received data will be both written into the Buffer for off-line processing as well as handed off directly to Real-time Processing Pipelines (such as fast imaging or real-time calibration).

Note that Receive will be started independently of Real-time Processing that uses its data. This means that Receive will provide a custom data streaming interface, which Processing Wrappers in real-time processing will have to implement in order to obtain real-time data.

Expertise:	Networking, radio astronomy algorithms
Implementation:	In-house

2.1.8. Data Models

Groups the definitions and implementations of data representations, data formats, and appropriate utility code for SDP data. It includes support for versioning and conversion between data formats and versions. It is composed of Buffer Data Models (formats for e.g. visibility, pulsar candidates, pulsar timing, and transient data in the Buffer) and Memory Data Models (in-memory data representations for processing components and queues).

2.1.8.1. Buffer Data Models

Definitions of data representations of data inputs and products in the buffer as well as utility code for reading, writing and interpreting it. This may include intermediate Data Products not visible outside pipelines. Note that this especially includes the in-buffer representation for the Science Data Model, which this module should provide an interface for that is similar to database queries in terms of flexibility and scalability.

Note that support for legacy Processing Components will likely require support for a large number of Buffer data models, even for the same type of data (e.g. measurement sets).

Therefore physical data models should be uniquely identified and versioned. Conversion to and from different Buffer Data and Memory Data Models should be possible.

Expertise: Radio astronomy data models

Implementation: In-house, possibly involving legacy libraries

This module is further decomposed in the Processing Module Component View [RD01].

2.1.8.2. Memory Data Models

Data representations used by Processing Components and Data Queues. Meant to be used as a high-speed way to interface processing components and pipelines with each other. Might also include utility code as appropriate.

Memory Data Models will be SDP-specific, but given that they will be used across a lot of Processing Components and SDP services, modifiability should be kept in mind. Extensible binary formats (Protocol Buffers? TBD) should be used where possible, and as with Buffer Data Models, memory data formats should be identified and versioned to make it possible to check Processing Components for compatibility.

Expertise: Radio astronomy algorithms and data models

Implementation: In-house

This module is further decomposed in the Processing Module Component View [RD01].

2.1.9. System Services

Common interfaces in use by all SDP modules. Implementations are going to be managed by Platform Services. As changes to these modules might impact the entire system, this should be restricted to well-established standard interfaces.

2.1.9.1. Operating System

Unix-like Operating System interface, offering baseline functionality such as memory and process management.

Expertise: Operating systems, standard libraries

Implementation: Off-the-shelf

2.1.9.2. Storage Interface

Provides a common interface for access to storage. For example, Platform Services will be expected to provide a POSIX-like file interface to Buffer data.

Expertise: Cluster file system or object stores

Implementation: Off-the-shelf or customised

2.1.9.3. Logging Interface

Provides the capability for processes to emit logging information for tracking down system defects, root cause analysis, resource overconsumption, etc. This module implements a generic interface that decouples the generation of logs from aggregation (see Logging and Health in Platform Services).

Expertise:	Logging systems, debugging
Implementation:	Off-the-shelf

2.1.9.4. Accelerator Interfaces

Provides access to high-performance computing capabilities. This may involve standard parallel libraries such as OpenMP and OpenCL, but also more specialised libraries such as CUDA.

Expertise:	HPC and HTC
Implementation:	Off-the-shelf

2.2. Relations and Their Properties

The primary presentation uses two types of relations: The “allowed-to-use” relationship and module decomposition.

A “use” relationship implies that the implementation of a module closely depends on the implementation of another module. The “Views & Beyond” definition is that a module A “uses” module B if it cannot be implemented correctly without B also being correct. This means that modules communicating with each other do not necessarily “use” each other. Instead we can choose to “use” a common data model / protocol module that decouples them from the concrete implementation of the other. Note that at this level of decomposition, “use” relationships are not prescriptive, so we use “is allowed to use” relationship instead.

Module decomposition is shown by nesting. Beyond module hierarchy, this implies some degree of “allowed-to-use” relationship between contained modules even if not spelled out explicitly. Implementation decisions for these are likely going to be related. “Allowed-to-use” relationships of the top-level module are understood to propagate to lower-level modules.

The the following sections we will go through the function of individual “use” relationships in the primary representation.

2.2.1. Execution Control Relations

2.2.1.1. Uses Platform Services

Steering of the platform infrastructure in relation to SDP functionality:

1. Provision compute resources for services and processing (Master Controller, Processing Controller)
2. Manage Configuration and Coordination data, often in order to interact with other modules of the SDP architecture
3. Monitor Logging and Health data

2.2.1.2. Uses SDP Services

Starting and coordinating SDP services in relation to SDP processing:

1. Initiation of Data Product Catalogue updates using Product Preparation and Delivery
2. Setting up and tearing down Quality Assessment topics alongside processing
3. Managing data movement of Data Products to and from Long Term Storage

4. Request generation of the Science Data Model from Model Database Services
5. Organise Buffer preparations for processing or staging
6. Instantiate Data Queues as needed for processing

2.2.1.3. Uses Science Pipeline Workflows

Processing Controller parameterise and executes Science Data Workflows

2.2.1.4. Uses Execution Framework Interface

Instantiate Execution Frameworks, provisioning required resources to perform processing jobs. Monitor execution of processing as it is running. Tear down after use.

2.2.2. SDP Services Relations

2.2.2.1. Uses Platform Services

Used for coordinating work within the platform infrastructure

1. Provisioning might get used to gain access to buffer storage resources (TBD) as well as software services (such as databases).
2. Access Configuration and Coordination data
3. Use Data Queues to stream data between processing and SDP services such as Quality Assessment, Model Database Services and Data Queue Services.

2.2.2.2. Uses Data Models

Required for being able to reason about data exchanged within the SDP system

1. Buffer Data Models required by Buffer Management Services as well as Product Preparation and Delivery in order to work with Buffer data independently of processing.
2. Memory Data Models are used by Data Queue Services and Quality Assessment for run-time communication with Science Pipeline Workflows.

2.2.3. Science Pipeline Workflows Relationstate

2.2.3.1. Uses Execution Framework

Science Pipeline Workflows use the Execution Frameworks and Processing Wrappers in order to execute Processing Components.

2.2.4. Execution Framework

2.2.4.1. Uses Platform Services

Platform infrastructure must be to some degree transparent for Execution Frameworks to efficiently organise data movement for processing.

1. Might provision common software services such as databases and query locality information if available
2. Access Configuration and Coordination data
3. Data Queues are used for streaming data in and out of processing at run-time

2.2.4.2. Uses Core Processing

Processing Wrappers use standard Processing Component interfaces to ensure algorithms are as much as possible agnostic of how they are used in pipelines or what Execution Framework is calling them.

2.3. Element Interfaces

Not applicable

2.4. Element Behaviour

Not applicable

3. Context Diagram

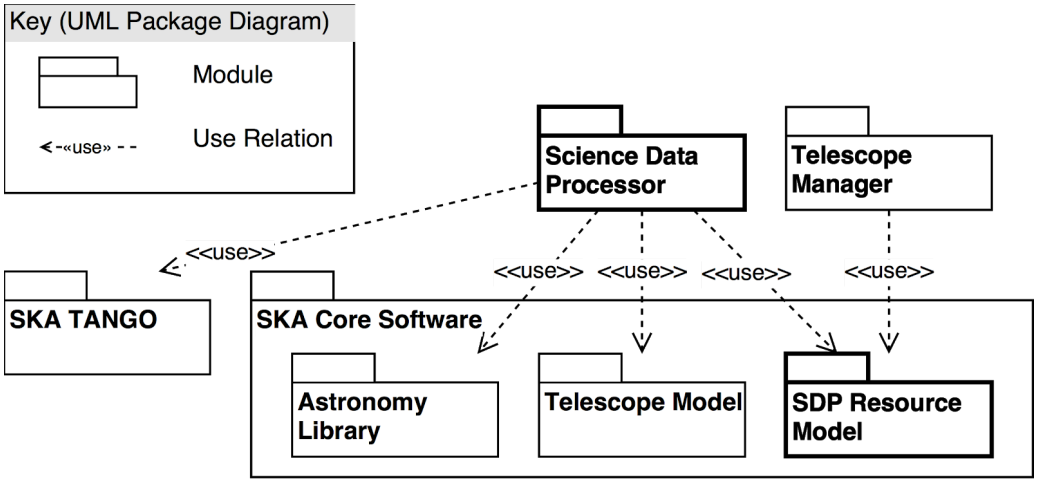

Figure 2. Context diagram showing the relation of SDP to other SKA software

The context diagram shows a selection of SKA modules with focus on the modules developed by SDP (in bold): The main Science Data Processor module, which is what the primary representation shows a decomposition for, and the SDP Resource Model module. The two external modules that SDP will "use" **and** that are outside the SDP decision making process are the "SKA TANGO" module (SKA TANGO distribution and element interfacing module) and "SKA Core Software". Specifically, SDP will use the following functionality from SKA Core Software:

- **Telescope Model:** Static (code) parts of Telescope Model information
- **Astronomy Library:** Basic astronomy/domain specific algorithms shared between telescope elements
- **SDP Resource Model:** SDP-maintained resource model shared with the Telescope Manager to facilitate scheduling and resource planning.

4. Variability Guide

The scientific pipelines being run will likely continue to evolve throughout SDP's life time. We plan for three levels of variability:

1. Workflows themselves can be exchanged and re-implemented freely. Scheduling should be set up such that it can easily be setup to run new types of workflows.
2. We would also like to avoid lock-in in terms of Execution Frameworks interpreting the workflows. The architecture should support multiple instances and types of Execution Frameworks within SDP. This is ensured by the Execution Framework Interface.

3. Finally, Processing Components should be built in a way that they can easily be exchanged. Their interface should be general enough that any processing component could be used by any Execution Framework or Science Pipeline Workflow.

This workflow variability especially allows SDP to be configured for operation in both the SKA1-Low and SKA1-Mid Telescopes using configuration changes. We do not expect that there will be other modules dedicated to either Telescope.

5. Rationale

1. The layered structure, with O/S level services at the bottom overlaid by applications and finally application supervisors is a standard structure used in distributed computing.
2. SDP Services and Processing Components are represented as the two main poles of this architecture. The reasoning is that processing is our central function, with non-processing modules filling more of a support role.
3. Workflows, Processing Components and Data Models represent the “core” of domain-dependent functionality. This makes them the primary focus of the architecture within processing; therefore they are kept at top-level.
4. Long Term Storage is viewed as “opaque” for the purpose of this module view. Proven off-the-shelf technologies exist in this space, so from an architectural standpoint these module is not enough of a risk to be considered in detail.

5.1. Constructability and Maintainability

Requirements: SDP_REQ-810 (Maintainability of Software), SDP_REQ-828 (Constructability)

The SDP system needs to be practical to build within the constraints of the construction schedule. Complexity drives cost and implementation schedule. The topology of the “module uses view” shall be used to drive the construction and implementation schedule in accordance with the dependencies. Example: If the processing components “use” execution engine module then the execution engine (v1) needs to be complete before processing component development can start.

- We have decomposed modules into functionality that requires domain-specific and non-domain-specific expertise. This should allow groups with different strengths to work together when constructing and maintaining SDP.
- The module structure suggests a number of possible architecture subsets that could be constructed and prototyped in isolation outside of the SDP. The strong isolation of processing components from the rest of the architecture allows us to both build and test Processing Components separately from the SDP, as well as to test the SDP with only “mock” Processing Components and Workflows.
- We decouple using common interfaces in a couple of places: File systems are a common interface to Buffer storage, data queues decouple real-time communication, and all remaining configuration information should go over the Coordination and

Configuration. This is a very important feature of this architecture: it strongly decouples workflows from each other (which is required for supporting multiple Execution Frameworks) as well as from the services.

- Science Pipeline Workflows are specific to an Execution Framework, and therefore we may view them as housed in a single enclosing scope. However, we want to emphasise loose coupling between these modules, as Science Pipeline Workflows will involve deep domain knowledge, while implementation of Execution Frameworks will require expertise in high-performance data flow systems.
- Up to a point the strong isolation even allows implementation of Processing Components using existing astronomical software (such as CASA). Note however that this conflicts with the requirement for Processing Components to only use SDP memory data models - introduction of “legacy” components might therefore require some modifications, such as the addition of conversion or emulation layers. This is seen as preferable to losing performance guarantees and restricting the design space for Execution Frameworks.
- The two external modules that SDP will “use” **and** that are outside the SDP decision making process are the “SKA Tango” module (SKA TANGO distribution and element interfacing module) and “SKA Core Software” that will contain basic astronomy/domain specific algorithms shared between elements. Sharing this with the rest of the telescope should minimise problems relating to data exchange with other subsystems.
- We especially expect the “SDP Resource Model” to become part of the SKA Core Software module so that it can be shared with the SKA Telescope Manager element for telescope planning purposes. Note that in contrast to other SKA Core Software modules, we would expect that SDP as an organisation would still be responsible for building and maintaining this module.

5.2. Scalability

Requirements: SDP_REQ-829 (Scalability)

- Our architecture is data driven: All services serve processing, so that we can do as much work on the data when and where it is available. Furthermore, due to taking most of the responsibility for communication out of the hands of processing components and encapsulating it in Execution Frameworks, we ensure that domain-specific functionality can not become a problem for scaling.
- Having multiple Execution Framework implementations also enables us to scale both up and down naturally: We can choose the execution engine to fit exactly the scale that we need it to run on. So for example, for small pipelines, this architecture makes it a viable choice to implement it as a self-contained process on a single node.

5.3. Performance

Requirements: SDP_REQ-826 (General Workflow / Algorithm Performance)

- For the processing modules, effort has been put into decoupling modules vertically on this side: Execution Frameworks are modules that execute distributed Science

Pipeline Workflows, controlled via a generic Execution Framework Interface. Processing Wrappers encapsulate Processing Component and Receive modules from the next layer, which perform the actual processing. See the Processing Component Module View [RD01] for details.

- The SDP needs good performance to be achieved over a wide variety of workflows and algorithms which will change over time. The level of performance that needs to be achieved (for specific workflows) needs to be balanced against general performance for other workflows and algorithms.

5.4. Reliability

Requirements: SDP_REQ-762, SDP_REQ-821-825

- Failure handling has been moved outside of specific controllers to Platform Services and the intent is to leverage standard cloud-type infrastructure - such as OpenStack, IaaS and through exercising OpenStack APIs to capture PaaS. For example, the coordination modules should to a certain point be able to check and ensure application health without involving controller modules.
- Division of “Master Controller” and “Processing Controller”: in order to split the functions which require highest uptime and reliability (Master Controller) from the function with somewhat less stringent reliability requirements but much more complexity (Processing Controller)

5.5. Portability

Requirements: SDP_REQ-812 (Portability of SDP to SRCs), SDP_REQ-816 (Portability when hardware is refreshed)

- System services are meant to abstract operating system detail away for most of the architecture by limiting the usable interface. This is meant to make it easier to port the software to different underlying operating systems.
- Normally we would expect the choice of O/S or cluster management software to be made within the SDP. However, for operation of Science Regional Centres this may mean that Platform Services will need to be re-written to support native APIs of a specific Science Regional Centre. For example, if a Science Regional Centre runs a virtualized service then provisioning may well need to be supplemented. All of this makes System Services encapsulation important architecturally.
- At this point it is not yet clear whether development tools (like a certain standard Python/SDK distribution) will be mandated by the SKA organisation. It would have to be added to the context or primary distribution as needed once this becomes more clear.
- Services are compartmentalised and may execute on separate hardware from Scientific Pipeline Workflow and Execution Framework, so hardware dedicated to processing can be refreshed independently of other infrastructure. Furthermore, kernels compatible with (or optimised for) new architectures can be introduced seamlessly either via the generalised accelerator interface, or by switching out processing component implementation used for workflows depending on the hardware used for execution.

6. Related Views

This view is decomposed further in the following views:

- Processing Module Component View [RD01]

This modules shown at this level of decomposition implement the components shown in the following component and connector views:

- Operational System C&C view [RD02]
- Platform C&C view [RD04]

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.

(no applicable documents)

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]	SKA-TEL-SDP-0000013, SDP Processing Component Module View, rev 05
[RD02]	SKA-TEL-SDP-0000013, SDP Operational System Component and Connector View, rev 05
[RD03]	SKA-TEL-SDP-0000013, SDP Processing Component and Connector View, rev 05
[RD04]	SKA-TEL-SDP-0000013, SDP Platform Component and Connector View, rev 05

8. Version History

Version	Date of Issue	Prepared by	Comments
04	2017-12-15	Peter Wortmann	Submitted for SDP M19 deliverable. This document forms part of a pack implementing the following ECPs: ECP-150007 ECP-160012

			ECP-160040 ECP-160048 ECP-160056 ECP-170031
05	2018-04-18	Peter Wortmann	Submitted for SDP M20 pre-CDR review