Title: SRDP Architecture	Authors: Rafael Hiriart	Date: 3/27/2018
Document No. 530-SRDP-XXX-XXX		Revision: 0.02

Science Ready Data Products

System Architecture

Project 530

PREPARED BY	ORGANIZATION
Rafael Hiriart	NRAO DMS, Software Architect

APPROVALS	ORGANIZATION	SIGNATURE
	Organization, Title	
	Organization, Title	
	Organization, Title	

Change Record

VERSION	DATE	AUTHOR	NOTES/CHANGES
0.01	04/09/2018	Rafael Hiriart	Initial draft.
0.02	06/29/2018	Rafael Hiriart	Incorporating changes from CoDR.

Contents

Introduction

This document describes the SRDP system architecture. The project has decided to develop the architecture following a Model Based System Engineering process based on the SysML standard. We believe this process permits a more rigorous approach for analyzing requirements and designing a system that satisfies them. It also facilitates the work of updating the architecture as the project elaborates requirements following the rolling wave approach. On the other hand, this approach also means that the structure of this document can be a bit unusual. All the contents of these document have been generated from a SysML model and reflect closely the model organization. In SysML, the Package element is used to organize the model, so this document describes the architecture as a sequence of package sections that follow the hierarchical organization of the model. After a description of each package and its contents, a series of annotated diagrams describe several aspects of it.

An integral part of this effort is the traceability of requirements to the system elements that originate them, and the definition of test cases that verify that the system fulfills the requirements as specified. We plan to create these artefacts from the model, but they are not yet available at the current stage of the project.

In fact, the process of designing the system is on-going, and this document describes a snapshot of the current design. At this point, the architecture has been mostly based on the System Concept document [530-SRDP-014-MGMT]. This document provides a high-level, conceptual overview of the system, from where formal stakeholder (or Level 0) requirements have been extracted [530-SRDP-015-MGMT]. These will be refined into system requirements (Level 1) and system element requirements (Level 2), a process that hasn't yet started. Consequentially, the level of detail of the system architecture matches the level of detail of the System Concept document, and further elaboration will proceed as requirements are refined.

On the other hand, this architecture also gathers the experience of other SRDP-related systems in NRAO:

- The standard NRAO pipeline has been generating calibration tables for all VLA observations since 2015.
- The VLA Sky Survey has been generating calibration and imaging products since August 2017.
- The NRAO Archive already provides product search and retrieval interfaces.

These reference systems inform this architecture, although care has been taken in incorporate their functionalities and experience without constraining the design too much on existing implementations.

Reference Documents

1. **530-SRDP-014-MGMT:** SRDP System Concept, Kern, SRDP Req. Comm., Revision 1.0.
2. **530-SRDP-015-MGMT:** SRDP Stakeholder Requirements, Treacy, Revision 1.0.
3. **530-SRDP-010-MGMT**: System Engineering Management Plan, Treacy, Kern, Revision 1.0.

Glossary

ALMA:	Atacama Large Millimeter Array
CASA:	Common Astronomy Software Application
CSV:	Comma Separated Values
NAASC:	North American ALMA Science Center
NMASC:	New Mexico Array Science Center
SDM:	Science Data Model
SRDP:	Science Ready Data Products
TAC:	Telescope Allocation Committee
VLA:	Very Large Array
VO:	Virtual Observatory

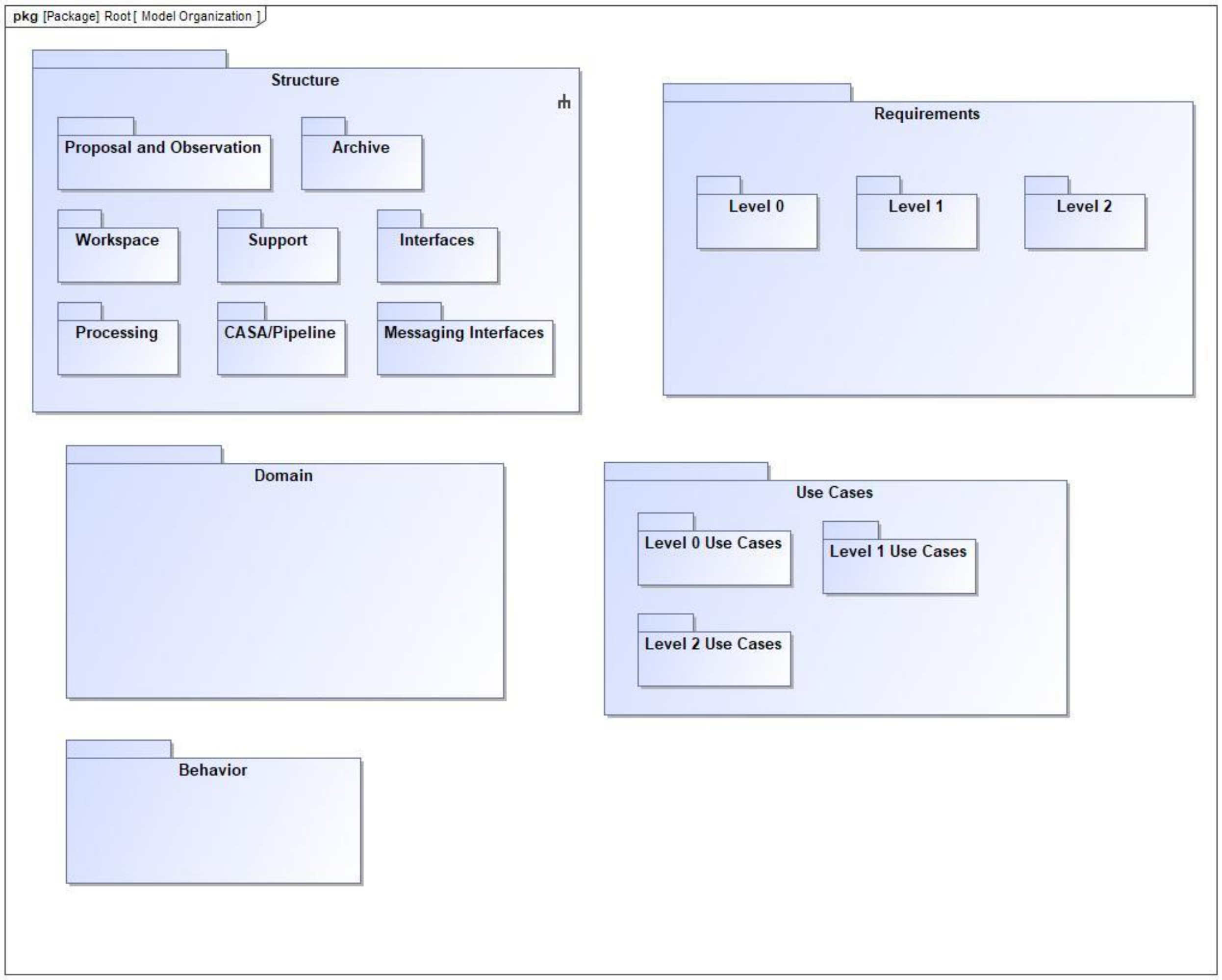
Document Organization

The structure of this document follows closely the SysML model organization. Each package is described first, followed by one or more diagrams. The model is also available online at http://update/me/please. This online report can be consulted for changes that have been introduced in the design since this document was generated, and to see additional diagrams that were not included in this document.

Diagram: Model Organization

This diagram shows the organization of the SysML architecture model. The root level packages are:

- Requirements package. This package contains the project requirements, imported from 530-SRDP-015-MGMT. As specified in 530-SRDP-010-MGMT, Level 0 requirements correspond to StakeHolder Requirements, Level 1 requirements are System Requirements, and Level 2 requirements are System Element Requirements.
- Use Cases package. This package describes functional requirements, specified as use cases. This package also contains interaction diagrams (activity, sequence, state) related with the use cases. These diagrams are also classified by abstraction level in Level 0, Level 1, and Level 2.
- Structure package. This package organizes deployable system elements, i.e., system elements that are part of the infrastructure required to implement the SRDP system. This includes servers, services, databases, libraries, pipelines, and hardware infrastructure.
- Domain package. This package contains the software elements that conform the system domain model, upon which the code is constructed. They are typically mapped into software elements such as classes and relational tables in subsequent, more detailed iterations.
- Behavior package. This package documents additional dynamic behavior, supplementing the interactions that can be found in the Use Cases package.



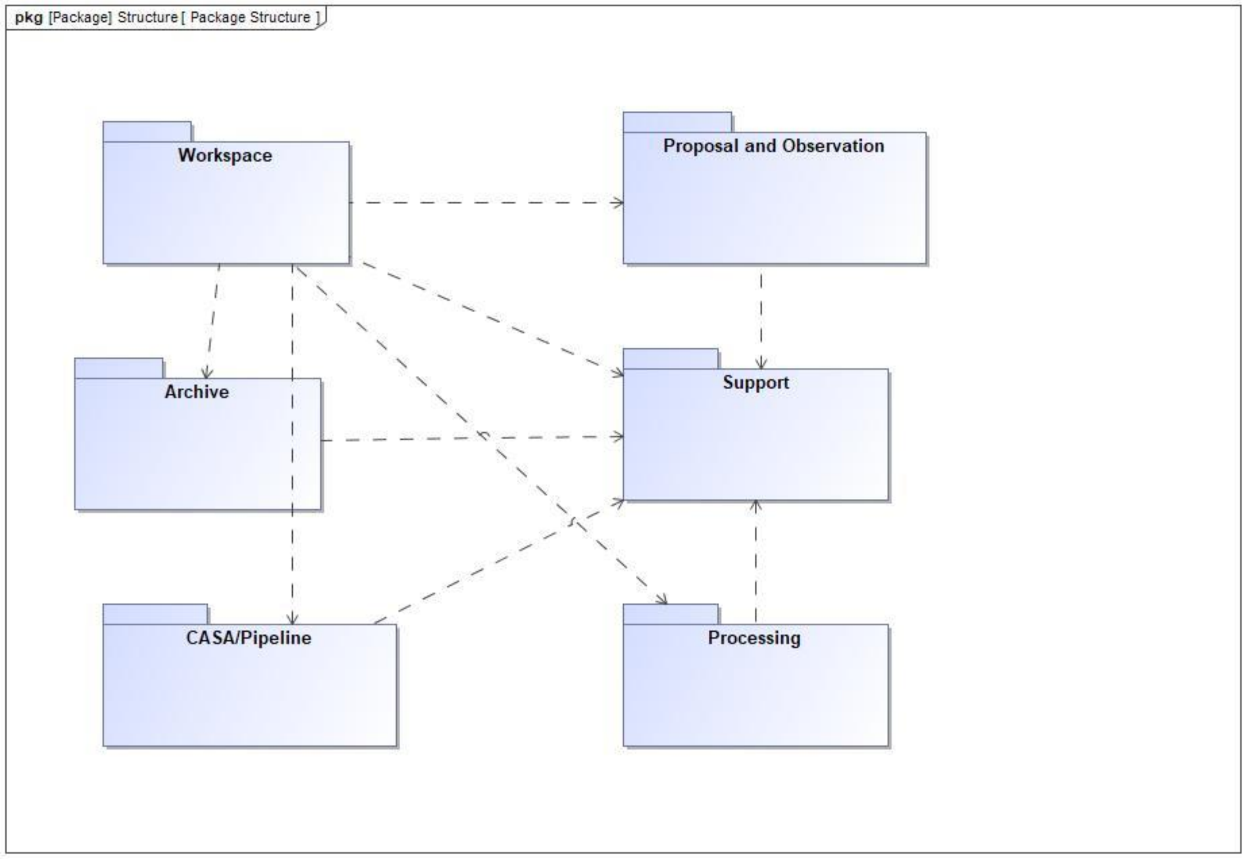
pkg [Package] Root [Model Organization]
Structure
Proposal and Observation
Archive
Workspace
Support
Interfaces
Processing
CASA/Pipeline
Messaging Interfaces
Requirements
Level 0
Level 1
Level 2
Domain
Use Cases
Level 0 Use Cases
Level 1 Use Cases
Level 2 Use Cases
Behavior

Package: Structure

This package contains the system elements that conform the system architecture. The blocks in this package represent deployable components such as servers, services, databases, libraries and hardware elements, along with their connections and interfaces. See the Domain package for the pure software data elements.

Diagram: Package Structure

This diagram shows how the elements contained in the Structure package have been organized in sub-packages in the model, along with their inter-dependencies.

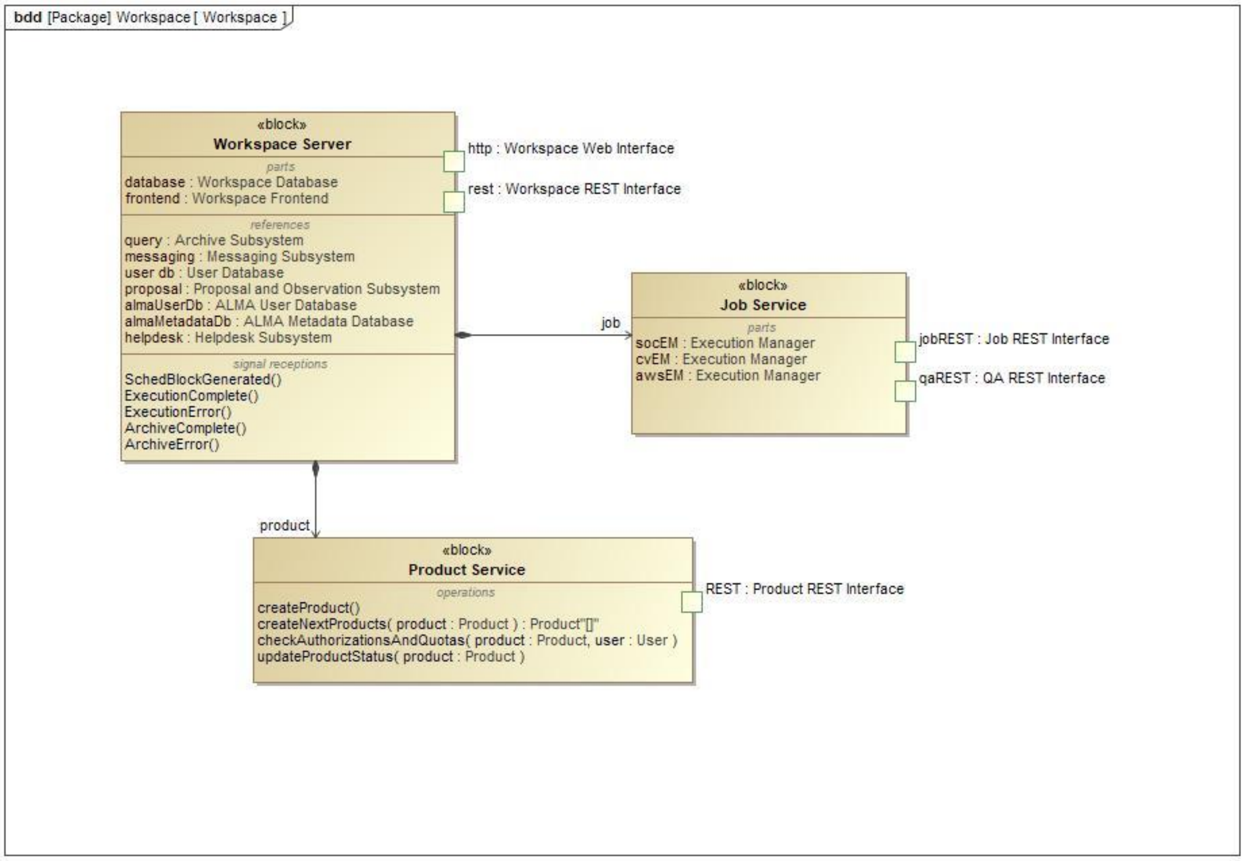


Package: Workspace

This package contains the system elements that are necessary to define the Workspace Server. The System Concept (530-SRDP-014-MGMT) and other documents use the *workflow* term to refer to multiple concepts (an actor, a part of the system, an interface, an existing component in the current implementation). To avoid confusion, this document doesn't use this term for any architectural element, but most of its functionalities are allocated either in the Workspace Server or the Proposal and Observation Subsystem.

Diagram: Workspace

This diagram shows the Workspace Server and related elements.



bdd [Package] Workspace [Workspace]
«block»
Workspace Server
parts
database : Workspace Database
frontend : Workspace Frontend
references
query : Archive Subsystem
messaging : Messaging Subsystem
user db : User Database
proposal : Proposal and Observation Subsystem
almaUserDb : ALMA User Database
almaMetadataDb : ALMA Metadata Database
helpdesk : Helpdesk Subsystem
signal receptions
SchedBlockGenerated()
ExecutionComplete()
ExecutionError()
ArchiveComplete()
ArchiveError()
http : Workspace Web Interface
rest : Workspace REST Interface
job
«block»
Job Service
parts
socEM : Execution Manager
cvEM : Execution Manager
awsEM : Execution Manager
jobREST : Job REST Interface
qaREST : QA REST Interface
product
«block»
Product Service
operations
createProduct()
createNextProducts(product : Product) : Product"[]"
checkAuthorizationsAndQuotas(product : Product, user : User)
updateProductStatus(product : Product)
REST : Product REST Interface

The elements of this package are:

Execution Manager: The Execution Manager is a software component that dispatches jobs to Processing Resources.

Job Service: The Job Service provides a REST interface for managing the specification, execution, and QA of product-generation jobs.

Product Service: The Product Service provides a REST interface for managing the definition of science-ready data products, and their status.

Workspace Database:

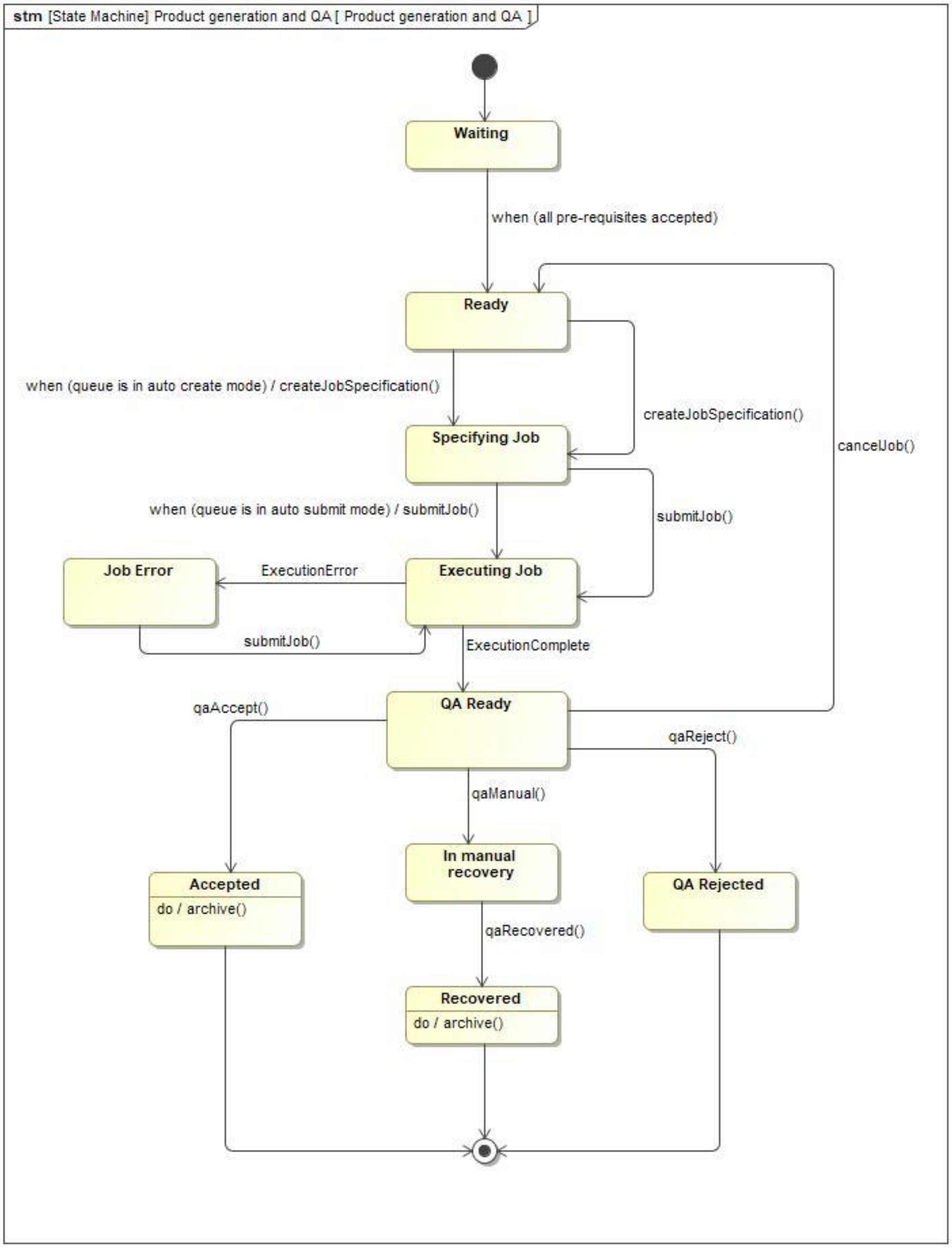
The Workspace database stores the data entities necessary to implement the services provided by the Workspace Server. Following the architecture of the VLASS Manager, it will probably consist in a relational database implemented in Postgres. See diagram Product Generation Structures for an overview of the objects it will be required to manage.

Workspace Frontend: The Workspace Frontend provides a Web application interface for the Workspace Server. Following the architecture of the VLASS Manager, it will probably be implemented as a JavaScript AngularJS application.

Workspace Server: The Workspace Server provides services for managing the definition of science-ready products, and the execution of the processes required to create them. It provides both a Web application interface and a scripting REST interface.

Diagram: Product generation and QA

This diagram presents the states a product assumes during the processes involved in creating it and performing its quality assurance. This is a behavioral diagram associated with the Workspace Server, presenting the effects of its method calls and signal receptions.



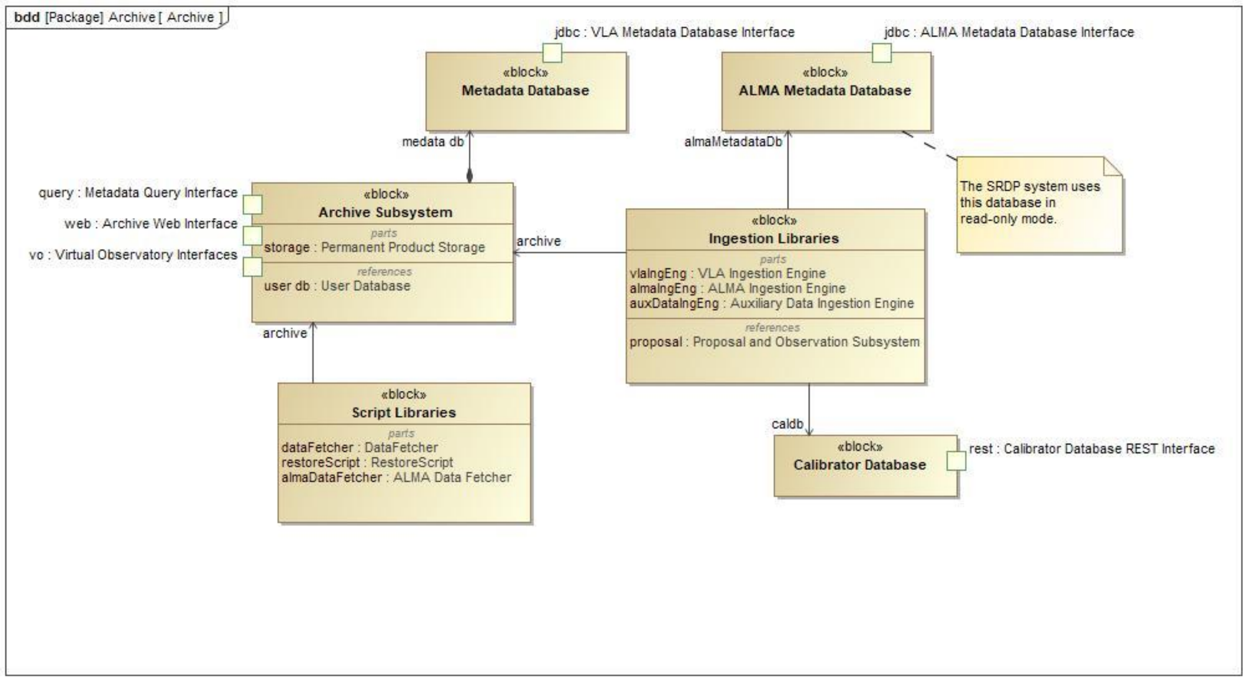
stm [State Machine] Product generation and QA [Product generation and QA]
Waiting
when (all pre-requisites accepted)
Ready
when (queue is in auto create mode) / createJobSpecification()
createJobSpecification()
cancelJob()
Specifying Job
when (queue is in auto submit mode) / submitJob()
submitJob()
Job Error
ExecutionError
Executing Job
submitJob()
ExecutionComplete
qaAccept()
QA Ready
qaReject()
qaManual()
Accepted
do / archive()
In manual recovery
QA Rejected
qaRecovered()
Recovered
do / archive()

Package: Archive

This package contains system elements related with storing and providing access to science-ready data products after the process of generating them has completed and they have been accepted by QA.

Diagram: Archive

This diagram shows the system elements contained in the Archive package. In general, data products are stored as a one or more data files in Permanent Product Storage, and meta-data describing these products in a database. Ingestion code extracts the required metadata, applies transformations when necessary, populates the corresponding database rows, and copies data files into permanent storage.
For ALMA, as the production of standard calibration and imaging products is not in the scope of SRDP, the meta-data is retrieved from the ALMA Metadata Database, and integrated into the Metadata Database.
The Calibrator Database stores flux curves and other calibrator parameters, which need to be included as part of the auxiliary data during ingestion.


bdd [Package] Archive [Archive]
jdbc : VLA Metadata Database Interface
«block» Metadata Database
jdbc : ALMA Metadata Database Interface
«block» ALMA Metadata Database
medata db
almaMetadataDb
The SRDP system uses this database in read-only mode.
query : Metadata Query Interface
web : Archive Web Interface
vo : Virtual Observatory Interfaces
«block» Archive Subsystem
parts
storage : Permanent Product Storage
references
user db : User Database
archive
«block» Ingestion Libraries
parts
vlaIngEng : VLA Ingestion Engine
almaIngEng : ALMA Ingestion Engine
auxDataIngEng : Auxiliary Data Ingestion Engine
references
proposal : Proposal and Observation Subsystem
archive
«block» Script Libraries
parts
dataFetcher : DataFetcher
restoreScript : RestoreScript
almaDataFetcher : ALMA Data Fetcher
caldb
«block» Calibrator Database
rest : Calibrator Database REST Interface

The elements of this package are:

ALMA Data Fetcher: The ALMA Data Fetcher is a command line utility that retrieves data products from the ALMA Archive. This is currently provided by the ALMA software, although some level of adaption with a wrapper may be necessary to integrate it with the rest of the system.

ALMA Ingestion Engine: This software component retrieves the pertinent fields for an ALMA data product from the ALMA Metadata Database, and populates the respective fields in the Metadata Database, which integrates information for both the VLA and ALMA data products.

ALMA Metadata Database:

The ALMA Metadata Database holds meta-data for ALMA data products.

ALMA actually has several meta-data databases: a quick-look relational database populated by the online system during an observation, an Oracle XML document-based database populated at the end of an observation, and relational database populated periodically from XML documents by the "Harvester". This system element refers to this last "harvested" database, which is designed to provide a scientific point of view of the ALMA products. The ALMA Ingestion Engine accesses a copy of this database in the NAASC in a read-only manner.

Archive Subsystem:

The Archive Subsystem incorporates several servers that store and provide access to data product data and meta-data. This subsystem provides three interfaces: an HTTP application (a.k.a. the Archive Access Interface, or AAT), a query REST interface, and a Virtual Observatory (VO) interface.

Currently, the Metadata Query Interface is implemented by Apache Solr.

Auxiliary Data Ingestion Engine: This is a software component that retrieves and ingests auxiliary data for an observation.

Calibrator Database: The Calibrator Database manages calibrator parameters, such as flux curves, and polarization properties.

DataFetcher: The DataFetcher is a command line utility that retrieves data products from the Archive Subsystem.

Ingestion Libraries: This component aggregates several software components necessary to ingest data and meta-data into the Archive Subsystem.

Metadata Database:

The Metadata Database holds the meta-data for both VLA and ALMA science-ready data products.

See the diagram Archive Metadata Database for a description of its structure.

Permanent Product Storage: The Permanent Product Storage is the final repository for the SRDP data products. It is currently an NGAS storage server.

RestoreScript: The RestoreScript restores a calibrated measurement set. It retrieves the raw data and the calibration tables, and invokes the CASA Pipeline to restore the calibrated MS.

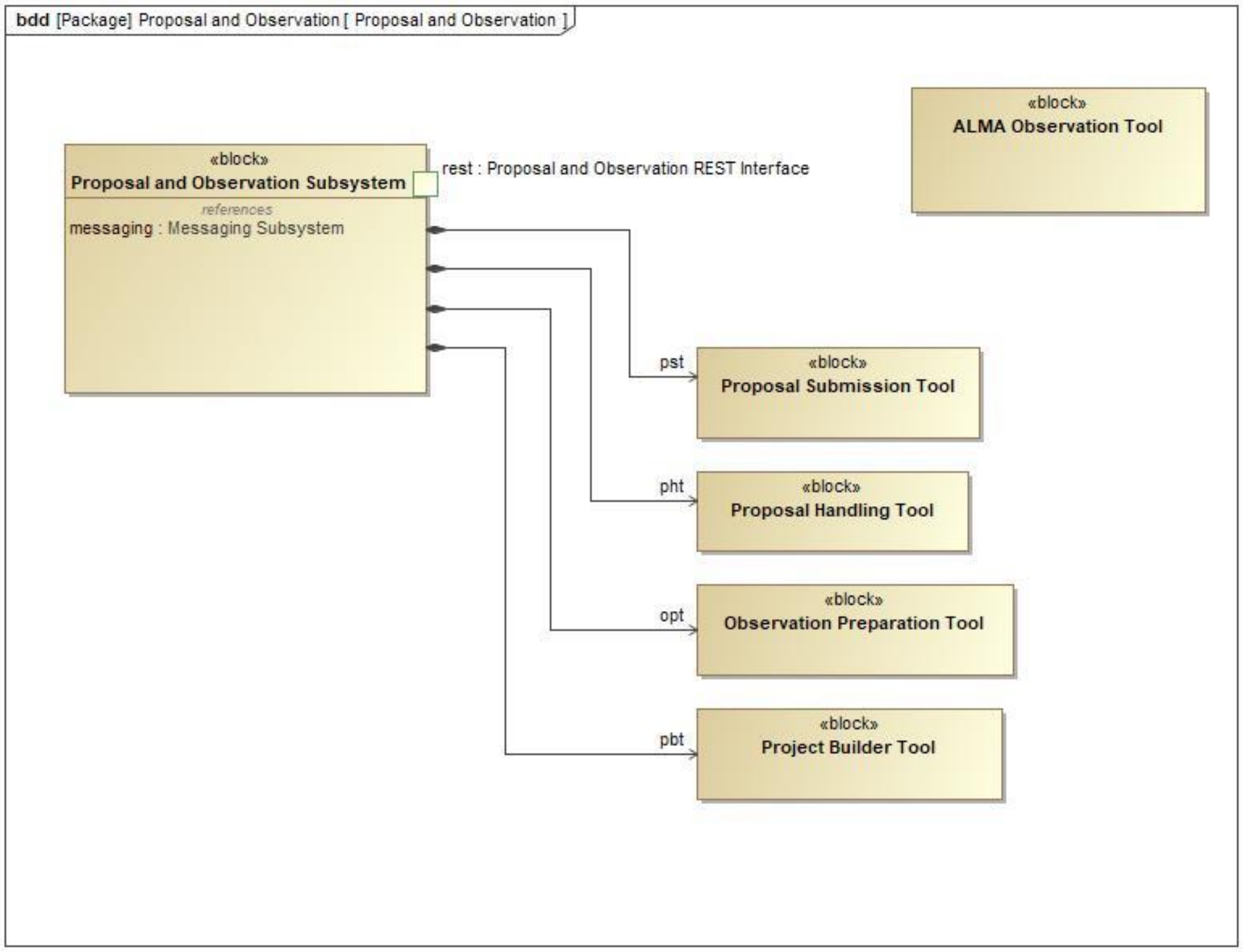
Script Libraries: This library aggregates several scripts that are necessary to fetch data products and restore calibrated measurement sets.

VLA Ingestion Engine: This is a software component that ingests a VLA data product into the Archive.

Package: Proposal and Observation

This package contains system elements related with the processes involved in submitting and managing observation Proposals, defining Scheduling Blocks, and performing observations, both for VLA and ALMA. It is worthwhile noting that there is currently an on-going project to refactor and modernize the VLA components, which is in the requirement elaboration stage.

Diagram: Proposal and Observation


bdd [Package] Proposal and Observation [Proposal and Observation]
«block»
ALMA Observation Tool
«block»
Proposal and Observation Subsystem
references
messaging : Messaging Subsystem
rest : Proposal and Observation REST Interface
pst
«block»
Proposal Submission Tool
pht
«block»
Proposal Handling Tool
opt
«block»
Observation Preparation Tool
pbt
«block»
Project Builder Tool

The elements of this package are:

ALMA Observation Tool: The ALMA Observation Tool manages both Proposals and Scheduling Blocks. Scheduling Blocks are defined in two phases: phase 1 Scheduling Blocks are used for Telescope Allocation Committee activities, and phase 2 Scheduling Blocks are using to perform observations.
ALMA makes use of the concept of Observing Unit Sets, which group data that are intended to be processed together. Although a similar concept is defined for the VLA, it has never been used for this purpose.

Observation Preparation Tool: The VLA Observation Preparation Tool is a Web application that allows to define Scheduling Blocks.

Project Builder Tool: The VLA Project Builder Tool is a Web Application that generates Scheduling Blocks from a Proposal. It currently generates only the skeleton of a Scheduling Block, which is not observable. It is used as a starting point by the PIs to define the Scheduling Block manually.

Proposal and Observation Subsystem: This subsystem collects all the functionality of the VLA proposal and observation components. As explained above, these components are being re-designed, so it is not clear yet if they will continue to be separate applications, or could be integrated into one, as is the case of the ALMA-OT. For the purpose of this architecture, they are referred collectively as the Proposal and Observation Subsystem.

Proposal Handling Tool: The Proposal Handling Tool is a Web application used by the Telescope Allocation Committed to execute simulations, manage the reviewing process, assign priorities, and award telescope time.

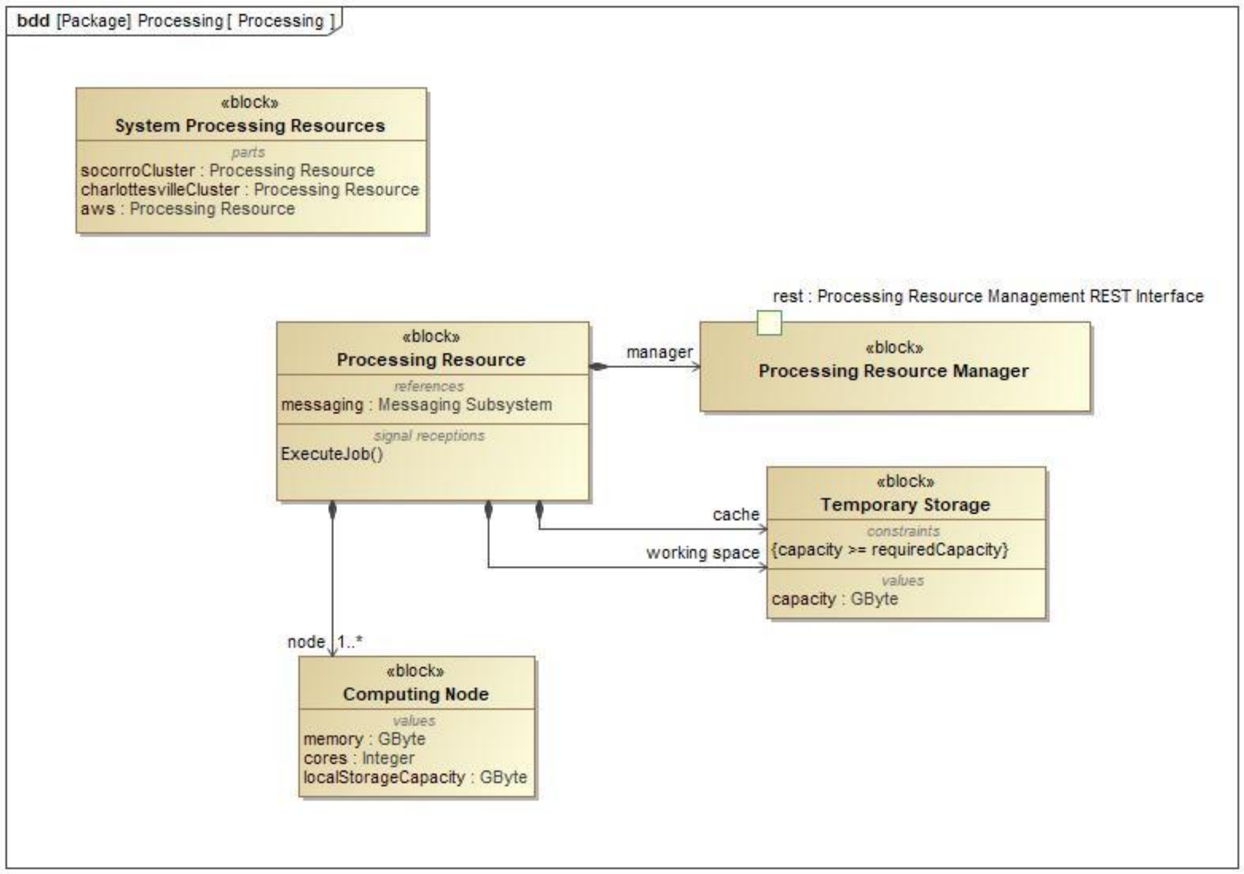
Proposal Submission Tool: The Proposal Submission Tool is a Web application through which PIs can define observation Proposals.

Package: Processing

This package contains processing resource elements such as computing nodes, storage systems, and associated management software. These resources can be local (in Socorro and Charlottesville), or located in external services such as Amazon Web Services or XSEDE.

Diagram: Processing

This diagram shows processing resource elements.


bdd [Package] Processing [Processing]
«block» System Processing Resources
parts
socorroCluster : Processing Resource
charlottesvilleCluster : Processing Resource
aws : Processing Resource
rest : Processing Resource Management REST Interface
«block» Processing Resource
references
messaging : Messaging Subsystem
signal receptions
ExecuteJob()
manager
«block» Processing Resource Manager
«block» Temporary Storage
constraints
{capacity >= requiredCapacity}
values
capacity : GByte
cache
working space
node 1..*
«block» Computing Node
values
memory : GByte
cores : Integer
localStorageCapacity : GByte

The elements of this package are:

Computing Node: This element represents a physical computer that is part of a processing cluster.

Processing Resource: A Processing Resource represents a collection of computing nodes, storage and networking resources under a management interface that allows to execute and manage processing jobs.

Processing Resource Manager: The Processing Resource Manager is a component that manages the computing resources contained in a Processing Resource. It provides interfaces to submit jobs, query the job completion status, etc. Currently the system uses Torque/MAUI and Apache OODT Workflow for this function. We have been discussing replacing OODT Workflow by special purpose REST server, which would provide the same interface to Torque/MAUI managed resources and AWS. This is under very preliminary design stages, though.

System Processing Resources: This element is included as a context element (that is, it doesn't represent an actual hardware or software system, but a pure model element), to indicate the different Processing Resources included in the system.

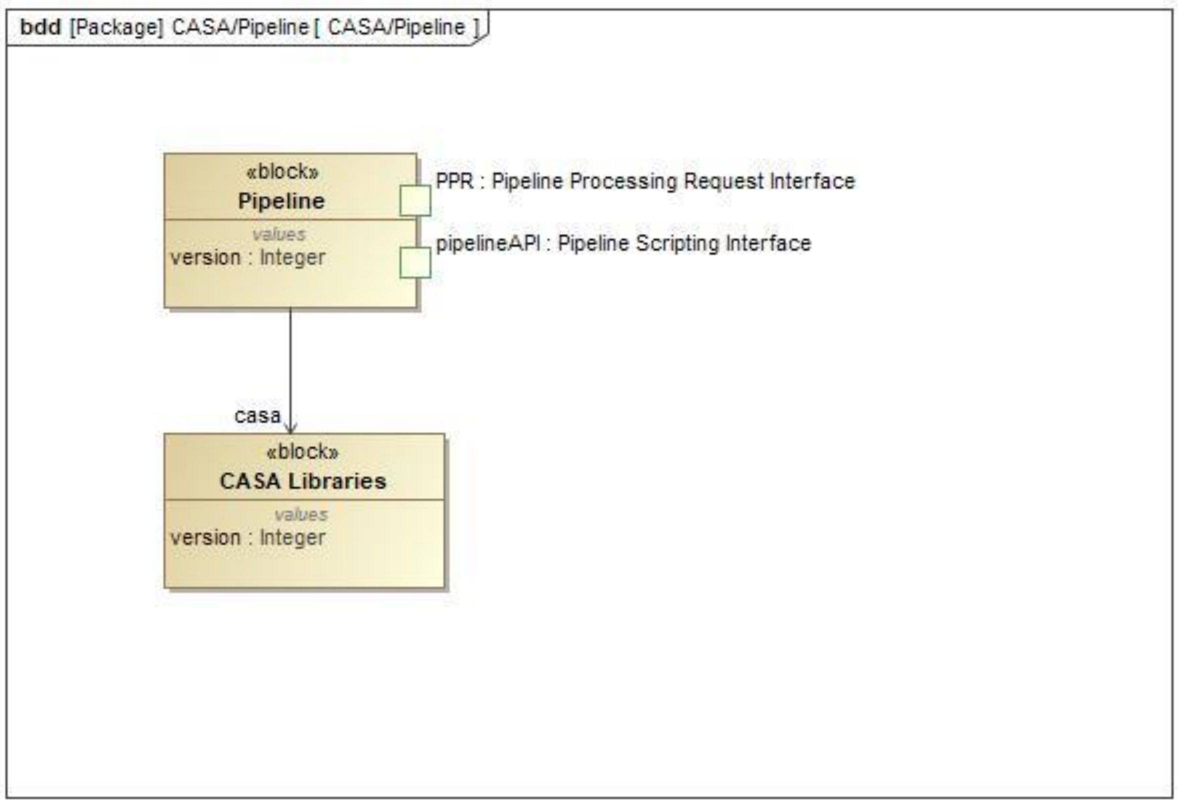
Temporary Storage: This element represents physical storage associated to processing resources. Currently, NRAO HPC systems use a Lustre filesystem connected to the cluster through an Infiniband network.

Package: CASA/Pipeline

This package contains elements related with CASA and the CASA Pipeline.

Diagram: CASA/Pipeline

This diagram shows relevant CASA and Pipeline elements.


bdd [Package] CASA/Pipeline [CASA/Pipeline]
«block»
Pipeline
values
version : Integer
PPR : Pipeline Processing Request Interface
pipelineAPI : Pipeline Scripting Interface
casa
«block»
CASA Libraries
values
version : Integer

The elements of this package are:

CASA Libraries: CASA is the Common Astronomy Software Application system, a data processing package that can process interferometric and single-dish data from the VLA and ALMA. It includes a set of C++ and Python libraries, and a processing environment based on iPython. This element refers to the libraries, which are called from the Pipeline.

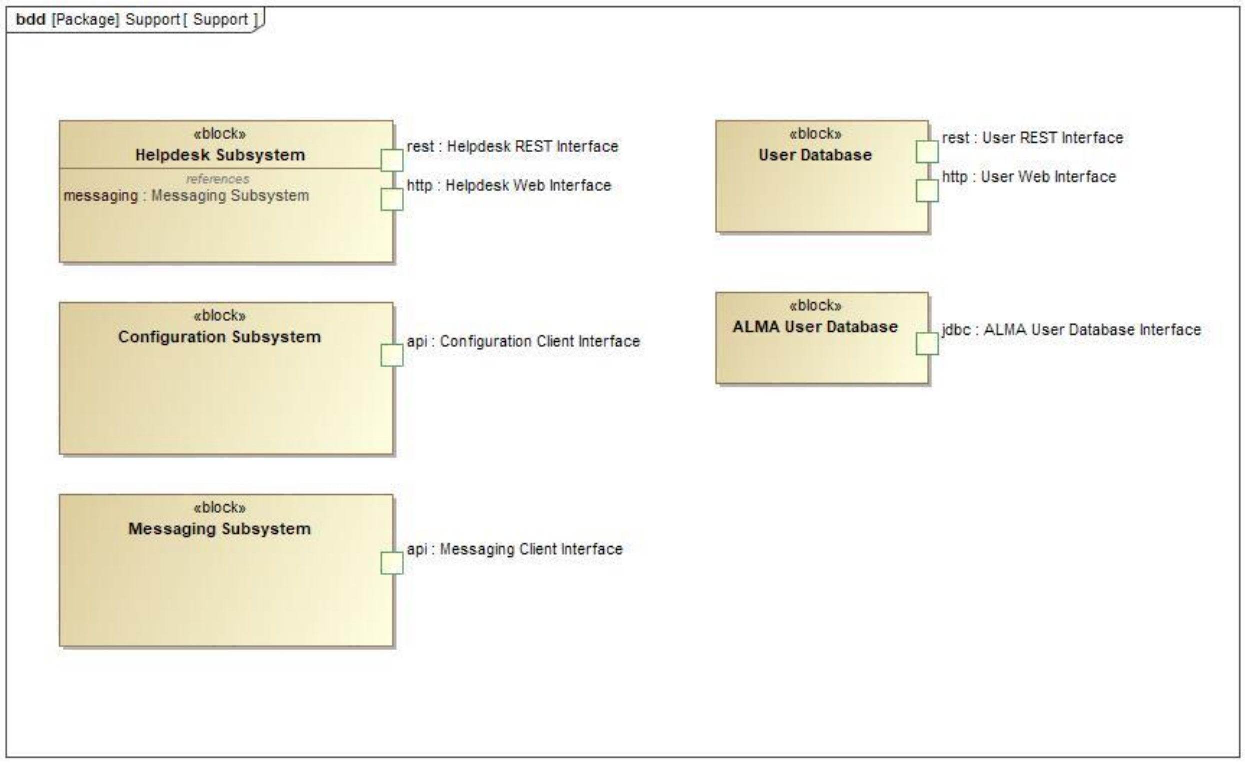
Pipeline: The Pipeline is a software package that allows automated calibration and imaging for ALMA and VLA data.

Package: Support

This package contains supporting infrastructure elements for the SRDP system.

Diagram: Support

This diagram shows the contents of the Support package.


bdd [Package] Support [Support]
«block»
Helpdesk Subsystem
references
messaging : Messaging Subsystem
rest : Helpdesk REST Interface
http : Helpdesk Web Interface
«block»
Configuration Subsystem
api : Configuration Client Interface
«block»
Messaging Subsystem
api : Messaging Client Interface
«block»
User Database
rest : User REST Interface
http : User Web Interface
«block»
ALMA User Database
jdbc : ALMA User Database Interface

The elements of this package are:

ALMA User Database: The SRDP system provides access both to the VLA and ALMA data. Currently, user credentials can be verified against the NRAO or the ALMA user databases, the choice being presented as an option in the Archive Access interface. Note that an NRAO account is necessary to make use of NRAO services. Some form of linking accounts may be necessary. The SRDP system uses the ALMA User Database in a read-only manner.

Configuration Subsystem: This subsystem manages system configuration information, such as file directories, database connection information, etc. Currently it consists of property files that define key/value pairs organized in configuration profiles, and libraries that can parse them and retrieve specific values. The system uses profiles to differentiate between local, test and production installations; as well as standard and VLASS (an example of a Large Project) installations.

Helpdesk Subsystem: The Helpdesk subsystem provides an interface to communicate with external users, notifying them of changes in their products and jobs, and following up issues. The system will most probably integrate a third-party helpdesk system, which will need to be integrated by means of the Messaging system.

Messaging Subsystem: The Messaging subsystem allows asynchronous communication between decoupled subsystems. The current implementation is based on RabbitMQ.

User Database: The User Database hold NRAO user information, including their authorization and processing, storage and data transmission quotas.

Package: Domain

The SRDP domain model. Included in this package are the entities and concepts upon which the software is constructed. Most of the contained blocks are persistent entities, which would be mapped to relational databases.

In general, the process of producing science-ready data products can be seen as consisting of three main steps:

1. **Product specification**. For the Telescope User, this process involves submitting a proposal, which is then evaluated by the Telescope Allocation Committee and possibly awarded telescope time. Based on the proposal, one or more observation Scheduling Blocks are defined. These are then scheduled and observed, resulting in raw data products. For the Archive User, the process involves defining products and computing jobs using the Archive interfaces.
2. **Product generation**. The process of executing Scheduling Blocks, which results in raw data products, and executing computing jobs that generate derived data products like calibration tables, images and catalogs. Once the products are generated, they are validated by the quality assurance process.
3. **Product curation and delivery**. Once the products have been accepted by QA, they are ingested into the Archive, from where they can be selected for subsequent post-processing and delivery.

These three steps define corresponding domain areas. For the Product generation domain see the "Product generation structures" and "Workspace structures" diagrams. For Product curation and delivery see "Archive Metadata Database" diagram. It is still too early to define the Proposal submission and observation structures. This area needs more elaborated requirements.

Diagram: Product generation structures

These are the entities involved in the generation of products.

Products are aggregated in ProductTypes. Products are also classified by inheritance (see Product Hierarchy diagram), but these two concepts are orthogonal, serving different purposes. The former is a classification useful to define the product generation, while the latter is used to control the attributes that different classes of products contain (in the Workspace server). For example, two different calibration products could have the same metadata attributes but could be generated using very different algorithms and parameters.

Both ProductType and Product are associated to Configuration objects, that control the way the generation is performed. In VLASS, these objects are general JSON documents.

ProductType and Product have Pre-requisites, which are associations with other ProductTypes and Products, respectively. This association forms a graph of types and products, that the system uses to determine the next products to create once a given product has been accepted by QA.

These structures are used to create a Job Specification, which is an entity that includes all the information necessary to create a Job Execution. When a Job Specification is created, the Configurations associated with ProductTypes and Products are transformed into ConfigurationFiles, which are serialized on disk and are parsed by the processes that create the products. An example of a ConfigurationFile is the PPR, which is an XML file that controls a CASA/Pipeline execution. The transformation of Configurations into ConfigurationFiles can be performed by a templating framework (VLASS uses Apache FreeMarker for this purpose).

A Job Execution is decomposed in one or more Job Tasks. For example, a calibration job is composed by the following tasks:

1. Prepare directories following defined naming conventions.
2. Fetch pre-requisite data products, in this case the raw data.
3. Fetch configuration files such as the PPR.
4. Execute the calibration pipeline.

5. Collect quality metrics.
6. Select products that need to be ingested and prepare job for QA.

These tasks are re-used across Job Executions.

ProductTypes are also associated with one or more ProcessingQueues. These are queues of Job Specifications that are submitted into the same ProcessingResource. A ProcessingQueue keep references to the first and last Job Specification in the queue. The Job Specification next property implements a linked list. This structure facilitates adding and removing Job Specifications from the queue, re-order them by priority, etc.

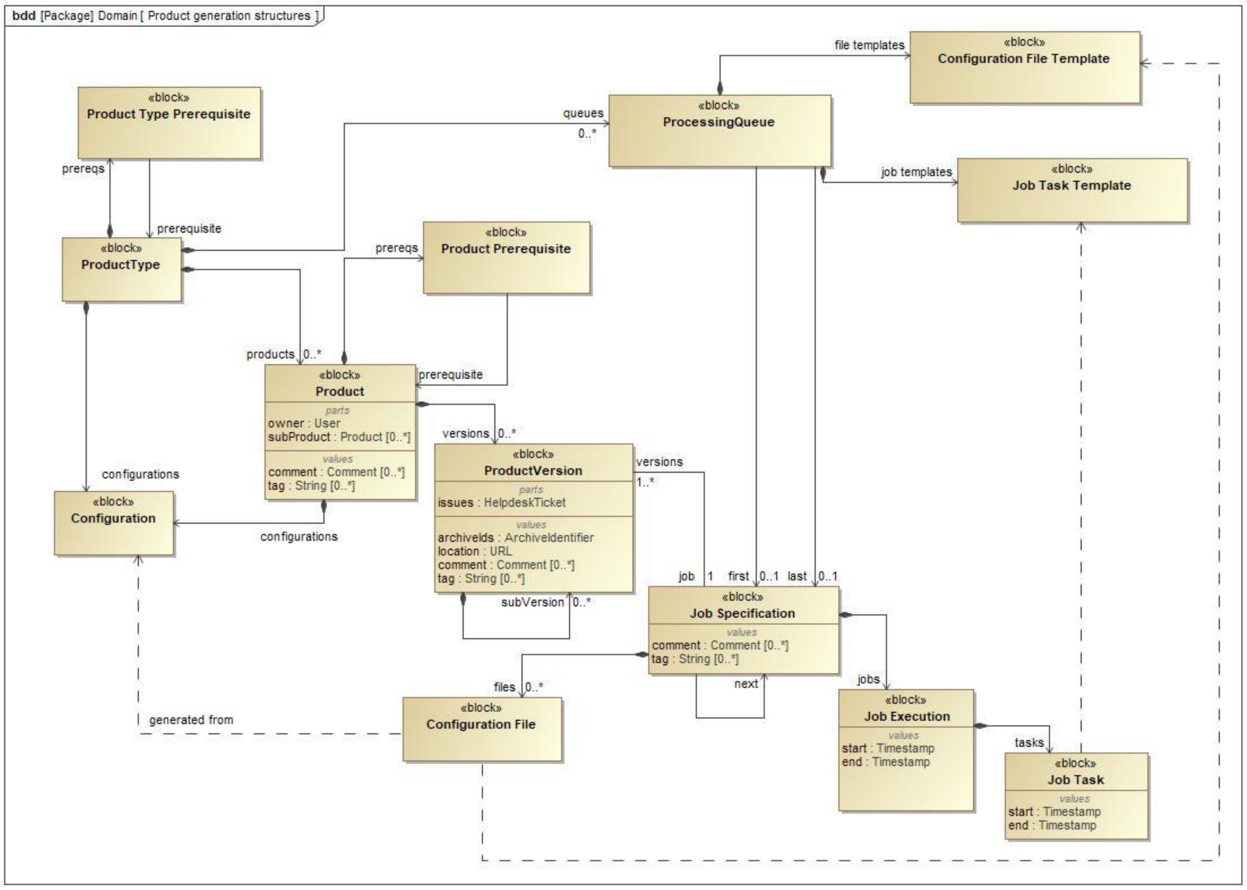


Diagram: Product Hierarchy

This diagram shows the different types of Products supported by SRDP. Each product contains multiple versions, with each one corresponding to the products generated and accepted during one Job Execution. The ProductVersion contains the archiveIds that refer to the File or FileGroup stored in the Archive Metadata Database. Note that a SchedBlock is also considered a product. In this case, its versions correspond to the ExecBlocks, i.e., the executions of the SchedBlock in the telescope. The archiveIds in the ExecBlock correspond to the SDM FileGroup.

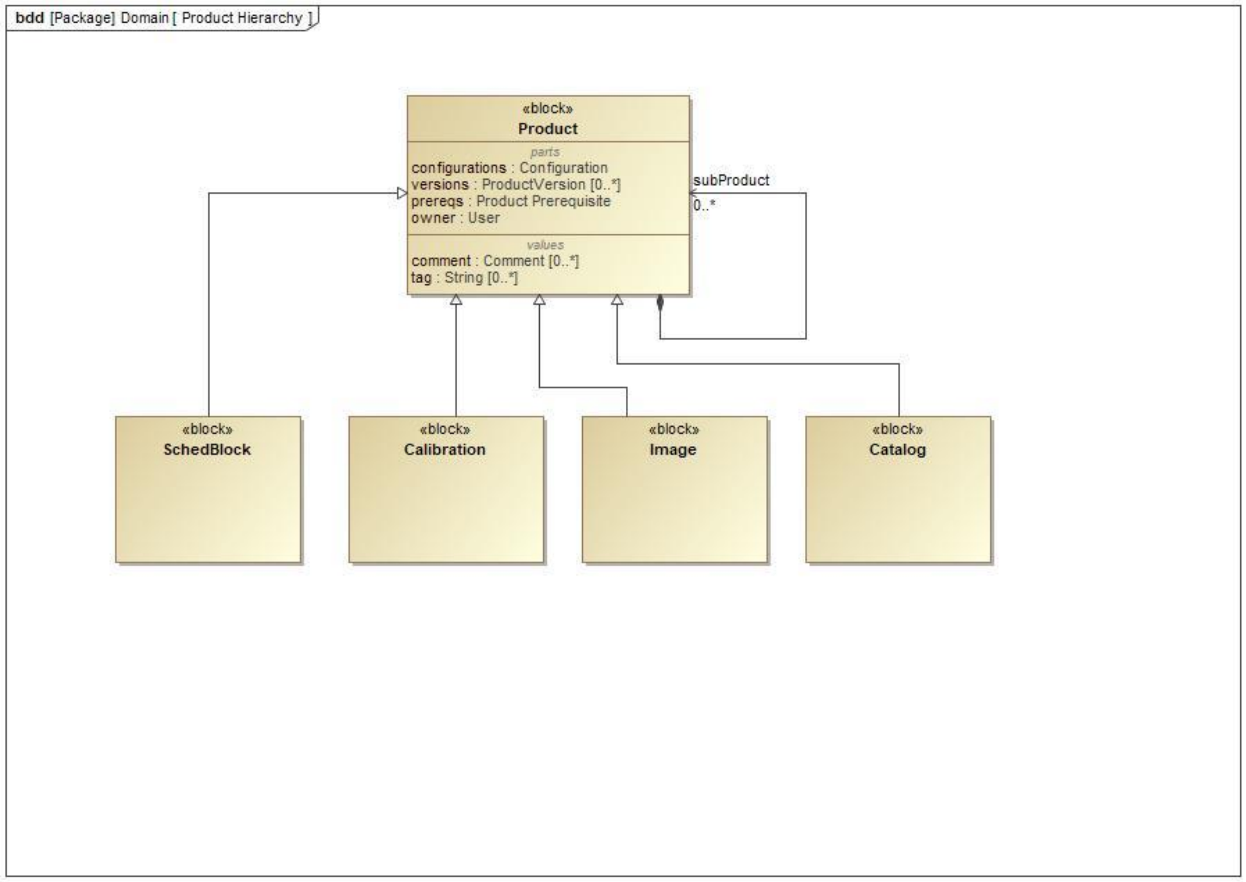


Diagram: Workspace structures

The workspace is an entity that aggregates products, jobs, and related entities for a specific user (or group of users as the User entity is a composite). It allows a user or a large project to see its relevant products and jobs in a persistent way, that is, in such a way that this information is preserved when the user logs out and comes back again later. This is necessary, as many data production jobs can take days or weeks to complete.
From the perspective of the user interface, it allows a user to store (references to) its products and invoke actions that will generate other products from them. The user interface also includes the capability of manipulating a Job Specification before it is submitted for execution, allowing the user to customize the execution parameters and configuration files.
Most of these ideas were prototyped in the VLASS user interface.

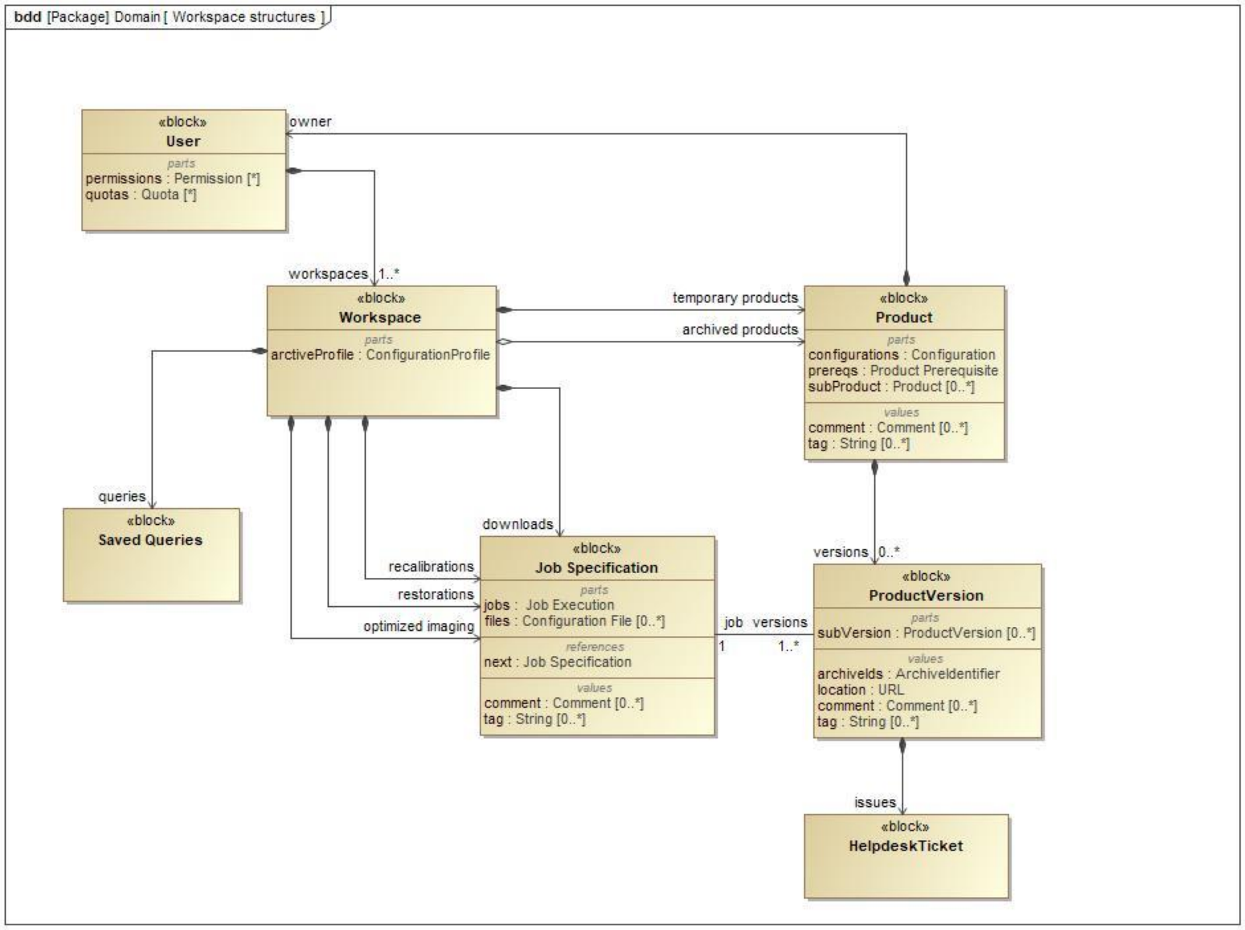


Diagram: Archive Metadata Database

The Archive Metadata Database is the final repository for curated data products once they have been accepted. It is populated by the ingestion processes, but besides this it is mostly a read-only database. It has two primary goals:

- Keep the necessary metadata in order to select products from queries.
- Organize products in hierarchical categories and keep references to their physical location in the permanent storage system.

As shown in the diagram, the FileGroup and File structures organize files in a hierarchical path-like structure akin to a filesystem, with FileGroups being similar to directories. In the current system, the permanent storage system is NGAS, which provides only a flat namespace for files.

Files and FileGroups have product-dependent metadata tables associated with them. The metadata table design is driven by the specific queries that the query and VO interfaces are required to support.

File and FileGroups are identified by their respective ID fields. These are used by the rest of the system (e.g., the Workspace Subsystem) to maintain references to them.

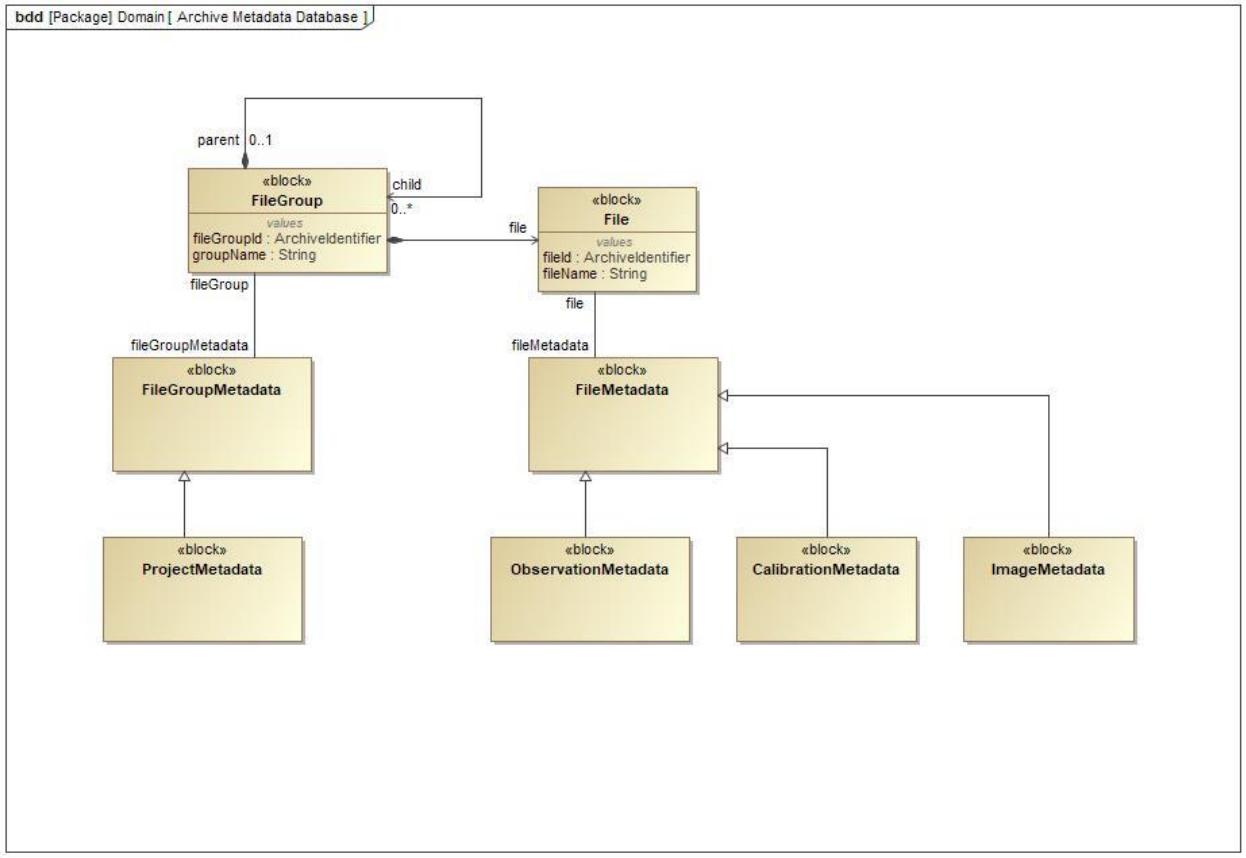


Diagram: User structures

This diagram shows elements that hold information related with a User. This is an area that is still in early stages of definition, so only a few concepts are shown. Users are grouped in UserGroups, which are also Users. This is required to support collaborative projects, including Large Projects. Each User has one or more associated Workspaces, Permissions and Quotas.

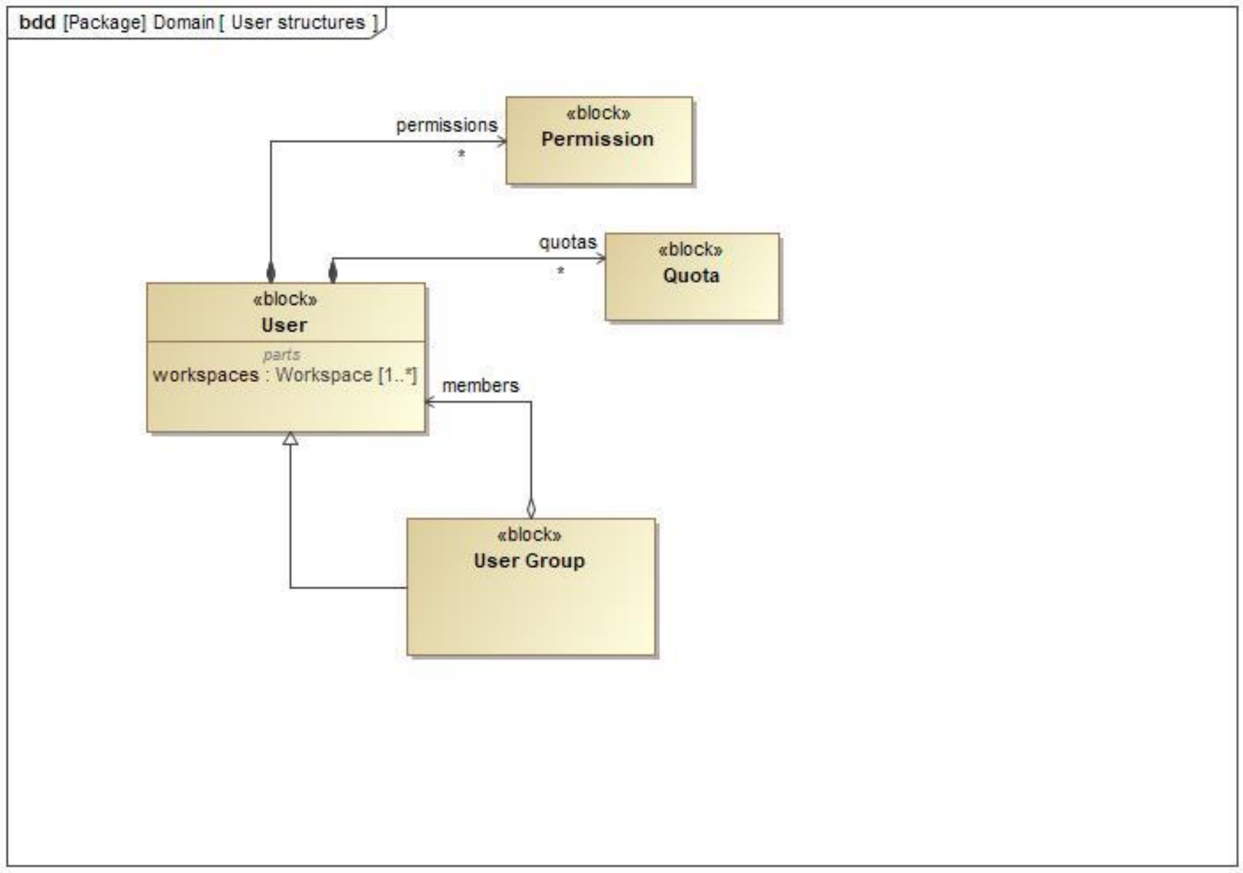
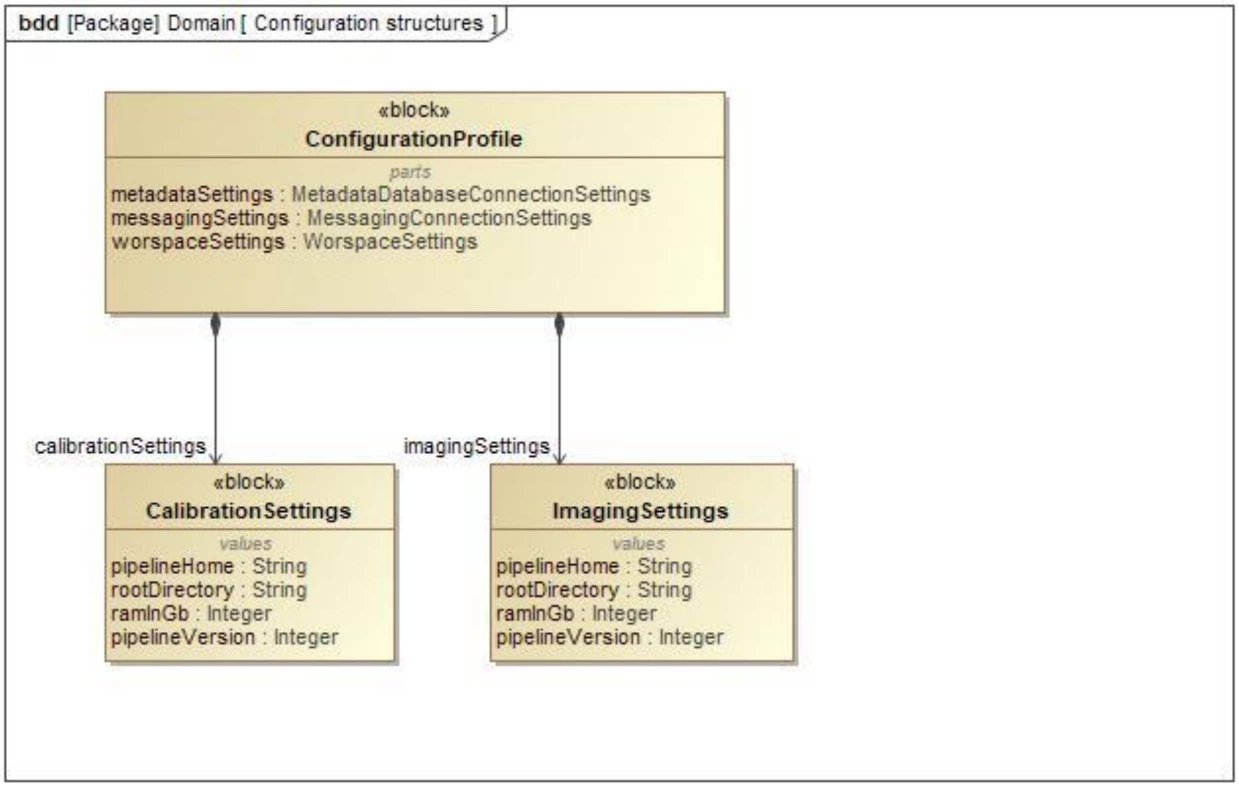


Diagram: Configuration structures

This diagram shows the system configuration structures. Currently the system uses property files (which contains key/value properties, with the keys organized in namespaces following Java package conventions) and libraries to parse and retrieve individual properties. This constitute the CAPO (Configurations for Archive, Pipeline and Other components) system. Only a few properties are shown, the rest will be added as the SRDP use cases are developed in detail.



bdd [Package] Domain [Configuration structures]
«block»
ConfigurationProfile
parts
metadataSettings : MetadataDatabaseConnectionSettings
messagingSettings : MessagingConnectionSettings
worspaceSettings : WorspaceSettings
calibrationSettings
imagingSettings
«block»
CalibrationSettings
values
pipelineHome : String
rootDirectory : String
ramInGb : Integer
pipelineVersion : Integer
«block»
ImagingSettings
values
pipelineHome : String
rootDirectory : String
ramInGb : Integer
pipelineVersion : Integer

Package: Interfaces

This package contains the interface elements. These are modeled as Ports attached to Blocks in the model. At the current stage of the project, the interfaces have been identified, but their operations have not been defined in detail yet. They will be specified as interactions diagrams are drawn based on the use case specifications. Note that this package contains synchronous interfaces, for asynchronous (messaging) interfaces, see the Messaging Interfaces package.

The elements of this package are:

ALMA Metadata Database Interface: This interface allows to submit queries and retrieve results from the ALMA Metadata Database. It is a Java Database Connectivity (JDBC) API, restricted to be a read-only interface (i.e., only SELECT statements are made).

ALMA User Database Interface: This interface allows to retrieve information about ALMA user accounts. Currently the ALMA Oracle user database is accessed through the Java Database Connectivity (JDBC) API.

Archive Web Interface: This interface is a Web application that provides basic search and delivery interfaces for anonymous users. Services that require user registration are provided by the Workspace Server.

Calibrator Database REST Interface: This is a REST interface for the Calibrator Database.

Configuration Client Interface: This interface allows to access system configuration data stored in property files. It consists in Python and Java libraries that read and parse the files corresponding to a give profile.

Helpdesk REST Interface: This is a REST interface that allows the interaction with the Helpdesk subsystem, exposing functions such as ticket creation, ticket information retrieval, ticket updating, etc.

Helpdesk Web Interface: A Web Application to access the functionalities provided by the Helpdesk Subsystem. The system will most probably integrate a third-party helpdesk system, that would provide this application.

Job REST Interface:

This interface allows to manage product-generation Jobs. It provides methods to define and create Job Specifications, submit Job Executions, etc. See Product generation structures diagram to see the associated entities.

Messaging Client Interface: This is an interface through which a component can send messages, subscribe a certain type of message, and register a method to receive them.
It will be probably implemented as a wrapper over the chosen messaging technology, which currently is RabbitMQ. This wrapper can provide a type-safe interface and take care of configuration details, for example, getting the user ids and password for RabbitMQ from the Configuration Subsystem.

Metadata Query Interface:

This interface allows the submission queries that specify selection criteria over data product metadata fields and return a collection of selected products.

Currently, this interface is implemented directly by the Solr query interface, which uses Lucene query syntax.

Pipeline Processing Request Interface: The Pipeline Processing Request interface is based on an XML file (PPR.xml) that is parsed by the pipeline. The file contains data elements such as project information, the PI name, the Measurement Set, etc.; and processing elements about the pipeline tasks that will be executed and their parameters. The processing elements is known as the "pipeline recipe".

Pipeline Scripting Interface: The Pipeline Scripting Interface is an API based on the pipeline tasks, which are constructed using CASA tasking infrastructure. In CASA, tasks are constructed declaratively in XML, which is used as the source to generate Python wrappers. Tasks behave as normal Python libraries but provide better integration with the CASA interactive environment.

Processing Resource Management REST Interface: This is the managing interface for a Processing Resource, allowing to list currently executing jobs, submit new jobs, cancel jobs, etc. Note that the system will need to integrate different kinds of Processing Resources: Torque/MAUI managed clusters, Amazon Web Services, etc. The Processing Resource Manager defines a common API to manage them.

Product REST Interface:

This is a REST interface that allows the system to manage product information. It provides methods to create new products, retrieve product information, update product status, look for dependent products, etc. See Product generation structures diagram and Product Hierarchy diagram to see the associated entities.

Proposal and Observation REST Interface: This interface provide access to the data managed by the Proposal and Observation subsystem. Currently the underlying tools (PST, PHT, OPT and PBT) don't have scripting interfaces, but implementing them is part of a general refactoring project.

Proposal and Observation Web Interface: This interface provides a Web application to access the data managed by the Proposal and Observation subsystem.

QA REST Interface: This interface allows to manage the QA process. It provides methods to review the resulting products, including the pipeline Weblog and generated quality metrics, comment jobs, "Accept & Archive" the products resulting of a Job, reject a Job, comment products, etc. This REST interface, along with the UI implemented in the Workspace Frontend, is the primary interface for interaction with the data analysts responsible for QA. Note that the Workspace Server also integrates the Helpdesk Subsystem, which is used to communicate with the Telescope and Archive Users.

User REST Interface: This interface allows to retrieve user information, such as her affiliations, permissions, quotas, etc.

User Web Interface: This interface allows a user to manage his account information, reset his password, etc.

Virtual Observatory Interfaces:

This interface represents Virtual Observatory interfaces.

VLA Metadata Database Interface: This interface allows to submit queries and retrieve results from the VLA Metadata Database. It is a Java Database Connectivity (JDBC) API interface. Note that this interface is only used by the contained block, the Archive Subsystem. Other components use the REST and VO interfaces provided by this component to access the data. This facilitates the maintainability of the database, reducing the scope of changes when making changes in the relational tables.

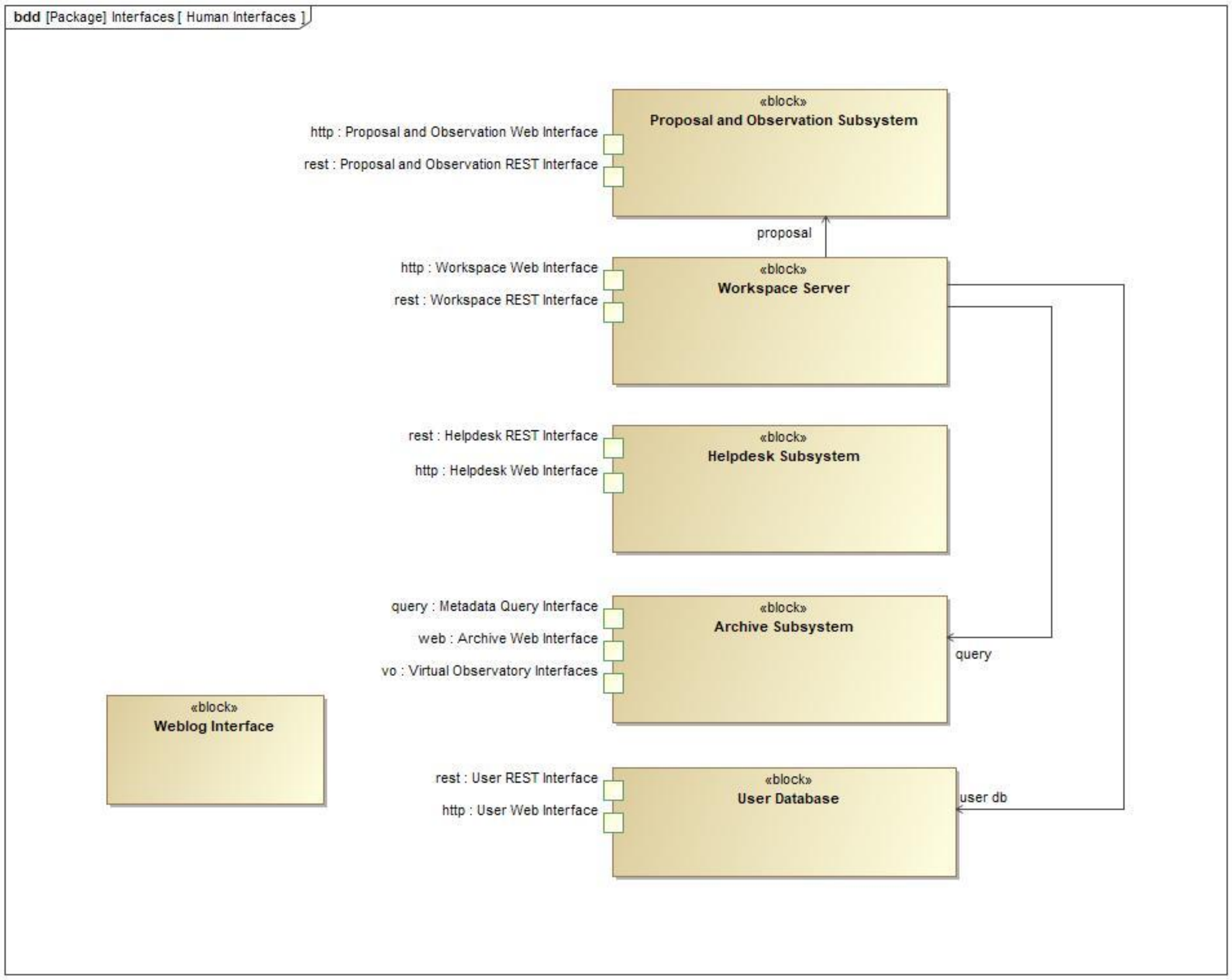
Weblog Interface: The Weblog is part of the products generated by the Pipeline. It consists of a set of generated HTML pages and plots describing several aspects of the generated data products and associated processing.

Workspace REST Interface: This is a REST application interface for the Workspace Server. It exposes the REST interfaces of its parts: the Product REST Interface, the Job REST Interface, and the QA REST Interface. It may add additional services, e.g. VLASS requires additional methods to manage the survey tiles and SchedBlock definition.

Workspace Web Interface: This is a Web application interface for the Workspace Server. It is provided by the Workspace Frontend.

Diagram: Human Interfaces

This diagram shows the subset of interfaces involved in human interactions.



bdd [Package] Interfaces [Human Interfaces]
«block»
Proposal and Observation Subsystem
http : Proposal and Observation Web Interface
rest : Proposal and Observation REST Interface
proposal
«block»
Workspace Server
http : Workspace Web Interface
rest : Workspace REST Interface
«block»
Helpdesk Subsystem
rest : Helpdesk REST Interface
http : Helpdesk Web Interface
«block»
Archive Subsystem
query : Metadata Query Interface
web : Archive Web Interface
vo : Virtual Observatory Interfaces
query
«block»
Weblog Interface
«block»
User Database
rest : User REST Interface
http : User Web Interface
user db

Package: Messaging Interfaces

This package contains signal elements, which are used to model asynchronous messages interchanged between system components. Each system element that receives a message (which usually triggers some function inside this block) has a reference to the Messaging System, and the messages it receives are annotated in the "receptions" block container. The emission of messages is documented in interaction diagrams.
At the current stage of the project, only a few messages have been defined. They will be defined as interactions diagrams are drawn based on the use case specifications.

The elements of this package are:

ArchiveComplete: This message is sent by a ProcessingResource to notify the successful ingestion of the products generated by a Job.

ArchiveError: This message is sent by a ProcessingResource to notify an error ingesting products into the archive.

ExecuteJob: This message is sent to a ProcessingResource to initiate the execution of a Job.

ExecutionComplete: This is a message that the ProcessingResource sends to notify the completion of a Job.

ExecutionError: This is a message sent by the ProcessingResource to notify that a Job Execution finished with an error.

ExecutionStatusUpdate: This is a message that the ProcessingResource sends to notify changes in the execution of a Job.

SchedBlockGenerated: This message is sent by the Proposal and Observation Subsystem to signal the generation of a Scheduling Block.

Package: Level 0 Use Cases

This package contains use cases derived from the System Concept Document (530-SRDP-014-MGMT). The System Concept document defines use cases as broad areas of related functionality. A use case in the model, on the other hand, is a concrete functional element, which is defined with the goal of producing a detailed and verifiable specification about how actors interact with the system to accomplish a specific system function. Because of this reason, some of the conceptual use cases have been broken down in more specific model use cases in this package. Similarly, areas from the System Concept document such as Large Projects, Curation and Reproducibility, and Commissioning and Validation, impose requirements over other use cases, but (so far) don't actually define use cases of their own. These requirements have been captured in the System Requirement Specification (530-SRDP-015-MGMT).

The elements of this package are:

Annotate data product reference: The Archive User selects a data product reference in his Workspace and adds a note to it.

Batch Recalibration: The Operations Staff actor uses the Workspace Server scripting interface to create re-calibration jobs for a number of selected raw data products. After this, the use case continues as described in the general Data Processing use case.

Combined Imaging: Combining data taken from multiple configurations of a particular array or telescope (including the ALMA Total Power Array) to produce quality assured images with better flux recovery.
The Archive User selects data product references and groups them in a folder. If they can be combined with minor re-gridding, the system enables the combined imaging action. The user selects this action and the system opens the Job Specification dialog, which includes the option of re-calibrating the datasets before performing the combined imaging. The use case proceeds as described in the general Data Processing use case. Ingesting the final image into the archive is conditional to having the input calibration products also stored in the archive.

Combined Imaging from Proposal: In this use case, for each configuration in the multi-configuration project, the calibration and imaging products are produced as described in the standard calibration and imaging use cases. When the last of the single configuration products is accepted, the Workspace Server checks if the proposal has specified the generation of a combined imaging product. If true, another Job Specification is created, and the use case proceeds as described in the general Data Processing use case.

Data Delivery:

The Archiver User selects data products and adds them to the delivery "basket". He can select data products from the references in his Workspace, or from the search result table. Once his basket is complete, he triggers the delivery action. The system asks for the delivery mode:

- a password protected URL that can directly accessed
- a download manager capable or starting, pausing, and resuming the download
- automated staging of data to the user's work area in either NRAO Socorro or Charlottesville cluster temporary storage (Lustre)

Additional modes of delivery such as insertion into Amazon S3, or through XSEDE frameworks will be considered as experience and user demand dictate.

Data Processing: This is an abstract use case, outlining the common steps that the Archive User follows to process data products. For each data product reference or folder in the Workspace, the system provides the available Data Processing actions (e.g., as buttons in the UI). The user clicks in one of these actions, and the system presents a Job Specification dialog, which display the default Job parameters, which the user can override. When the user is satisfied with the parameters, he submits the Job. The system checks the user's permissions and quotas, and if these allow the submission of the Job, it is submitted. The system executes the Job. The status of the job will appear as a Job Execution object in the user's Workspace. A corresponding ticket in the Helpdesk system is created, along with a link to it in the Workspace. This link is associated with the Job Execution. The user is notified of changes in the status of the execution through the Helpdesk system. When the Job Execution completes, the Operations Staff performs quality assurance over the generated products. The user can communicate with the Operations Staff in the helpdesk ticket. If the product is accepted by QA, the product is ingested into the Archive.

Data Product Visualization: The Archive User selects the visualization action for a data product, and the Workspace FrontEnd opens the visualization interface.

Data Selection:

This is an abstract use case, representing the general idea of allowing the Archive User to select data products persistently in lists. The data product references can be annotated, tagged (with free-form tags or previously defined tags), or used as inputs in post-processing actions.

The ability of storing persistent user-defined references to data products is closely tied to the concept of workspace. See Workspace structures domain diagram, and Workspace package.

Execute Observation: The SchedBlock is scheduled for observation in the OST. The telescope operator selects and executes the SchedBlock. The observation is performed. When it completes, the online observation system sends messages to the Messaging Subsystem. These messages are received by the Workspace Subsystem, which updates the SchedBlock product. The new observation appears in the Quality Assurance interface. If it is QA accepted, the product is ingested into the archive. Optionally, the QA interface can be setup for automatic acceptance.

Export Search Results:

Exporting the result table obtained in the Web Data Discovery use case to disk, in CSV or other file formats.

Generate Scheduling Block:

When a Proposal is accepted for observation by the TAC, an operation that is performed in the PHT, the Proposal and Observation Subsystem generates a Scheduling Block in the OPT. If the Proposal is SRDP complaint, a complete and observable Scheduling Block is generated and a message is sent to the Workspace Subsystem. A new SchedBlock product is created in the Workspace, which will be the input product for subsequent calibration products. The PI can log in the Workspace server and will see his SchedBlocks along with their observation status (the status of the corresponding ExecBlocks).

See Create Scheduling Block diagram.

Generate Standard Calibration Product:

Once the raw data product has been ingested into the Archive, the Workspace server looks for dependent product types for the SchedBlock product and finds the Calibration product type. The Workspace server creates a Job Specification object in the standard calibration Processing Queue and a ticket in the Helpdesk subsystem. If the queue is in automatic mode, the Job is sent directly for execution. If the queue is in manual mode, the Job Specification can be inspected and the associated Execution Configuration files modified. In this case, the Job Specification needs to be manually submitted for execution. When a Job Specification is submitted, the system creates a Job Execution object to track the execution. The Execution Manager submits the execution to the corresponding Processing Resource (associated to the given Processing Queue). The Processing Manager submits the job to the cluster and tracks its execution. As the execution proceeds, updates are sent to the Workspace server. When the product generation job completes, the Job Execution appears in the QA interface. The Operations Staff actor performs the QA and either accepts the product or rejects it. If the product is accepted, the Workspace server triggers a new job in the Processing Resource to ingest the calibration product.

See Standard Imaging Interaction diagram.

Optimized Imaging: Generating specific science images as requested by a science user. Images will be quality assured (in conjunction with the user) and delivered to both the requestor and to the archive.

Perform spatial cutout: The user selects the spatial cutout action for an image product. The use case proceeds as described in the general Data Processing use case. The resulting spatial cutout is left in temporary storage space, and it is not ingested into the Archive.

Perform spectral cutout: The user selects the spectral cutout action for an image product. The use case proceeds as described in the general Data Processing use case. The resulting spectral cutout is left in temporary storage space, and it is not ingested into the Archive.

Persist data product query:

The Archive User selects one or more data products from the search result table and saves the query in his Workspace. The query itself (and not the results from the query) are saved in the Workspace as a Query object. See diagram Workspace structures.

Persist data product reference: The Archive User selects one or more data products from the search result table and save references to them in his Workspace. The FrontEnd Web Application also allows the user to define "folders" in his Workspace and move references between them.

Recalibration: Repeating the calibration step, either with a different version of the supporting software tools, or with additional inputs from the user. Quality assured recalibration products are stored in the archive.
The Archive User selects a raw data product and activates the recalibrate action. The system opens the Job Specification dialog, where the user can input processing parameters and the version of the pipeline, and the use case continues as described in the general Data Processing use case.

Restoration: Returning one or more measurement sets to the calibrated and flagged state they were at the end of the standard calibration process. The calibrated measurement sets might be delivered to the PI directly or serve as the initial state for other processes.
The Archive User selects a calibration product and activates the restore action. The system proceeds as described

in the general Data Processing use case. Note that in this case it may not be options for the user to modify in the Job Specification dialog.

Script Data Discovery: Searching the Archive Metadata Database using a scripting application interface. One option for this scripting interface is using ADQL, ConeSearch and other related VO protocols.

Set View Defaults:

Modifying the default values displayed in the result table of the Web Data Discovery use case.

Standard Calibration: Automatically generating science quality-assured calibration for supported observational modes.

Standard Imaging:

Automatically generating science quality assured images for supported observational modes. The objective is a homogeneous set of images in the archive to support telescope and archive users.

Once the calibration data product has been ingested into the Archive, the Workspace server looks for dependent product types for the calibration product and finds the imaging product type. The Workspace server creates a Job Specification object in the standard imaging Processing Queue and a ticket in the Helpdesk subsystem. If the queue is in automatic mode, the Job is sent directly for execution. If the queue is in manual mode, the Job Specification can be inspected and the associated Execution Configuration files modified. In this case, the Job Specification needs to be manually submitted for execution. When a Job Specification is submitted, the system creates a Job Execution object to track the execution. The Execution Manager submits the execution to the corresponding Processing Resource (associated to the given Processing Queue). The Processing Manager submits the job to the cluster and tracks its execution. As the execution proceeds, updates are sent to the Workspace server. When the product generation job completes, the Job Execution appears in the QA interface. The Operations Staff actor performs the QA and either accepts the product or rejects it. If the product is accepted, the Workspace server triggers a new job in the Processing Resource to ingest the imaging product.

See Standard Imaging Interaction diagram.

Tag data product references: The Archive User selects a data product reference in his Workspace and adds a tag to it. As the user types the tag, the system recommends other existing tags, already defined in the system, which could be Observatory tags, or Large Project defined tags.

Temporary Data Cleanup: The system periodically checks if the user-generated products stored in temporary space have been stored a period of time beyond the limits defined by SRDP policies and deletes the directories. This condition is indicated in the Workspace references. Three days before removal, the system posts a warning notification in the user's workspace and sends a notification by email.

Time Critical Observations: Accounting for time critical observations, including both triggered and target of opportunity observations. These observations modify the standard processes to decrease latency in product delivery.
If the Proposal has been flagged as time-critical, then other product generation use cases are modified. This affects

the priority of Job Executions in Processing Queues, and the parameters in Job Specifications, and the type of products the system generates.

Web Data Discovery:

Searching the Archive Metadata Database using a Web application interface. The Archive User fills up a form specifying specific values or range of values for metadata fields, and the system returns a table with the results. The user can modify the default columns displayed in the result table in the Set View Defaults extension. The user can export the result table to CSV or other file formats in the Export Search Results extension.

Diagram: Standard Calibration Use Cases

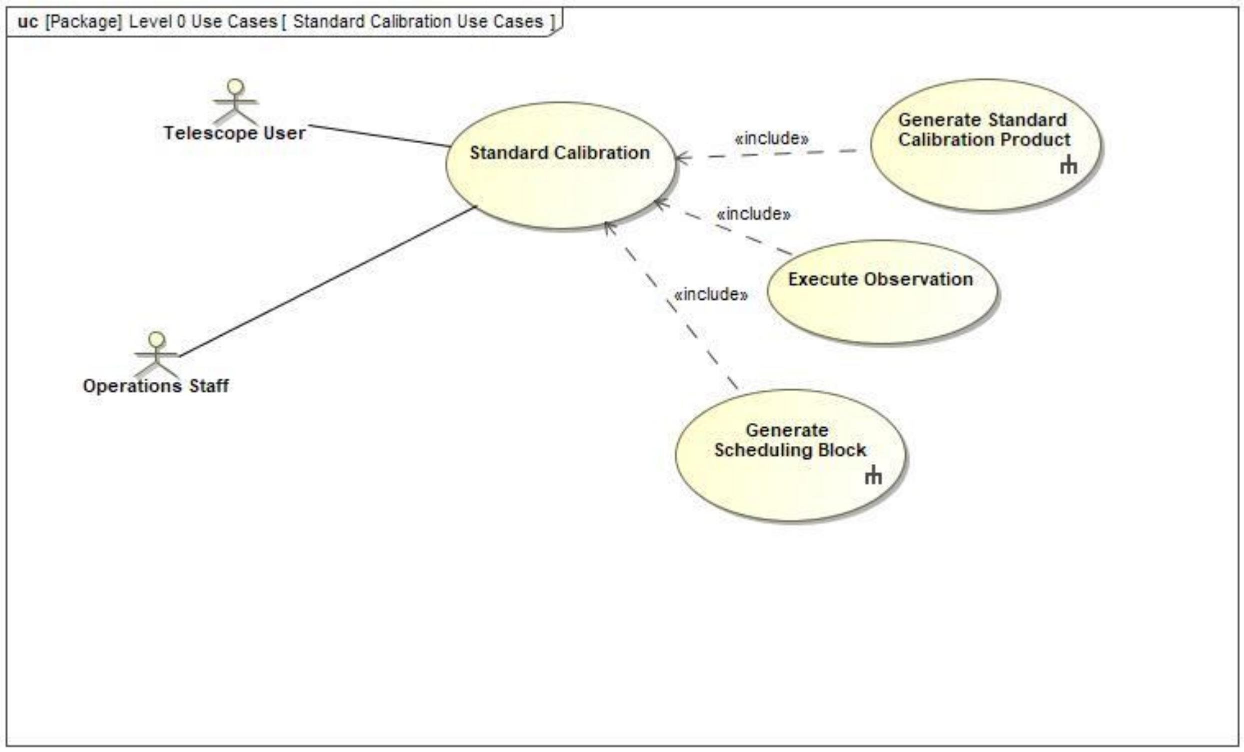

uc [Package] Level 0 Use Cases [Standard Calibration Use Cases]
Telescope User
Operations Staff
Standard Calibration
«include»
Generate Standard Calibration Product
«include»
Execute Observation
«include»
Generate Scheduling Block

Diagram: Standard Imaging Use Cases

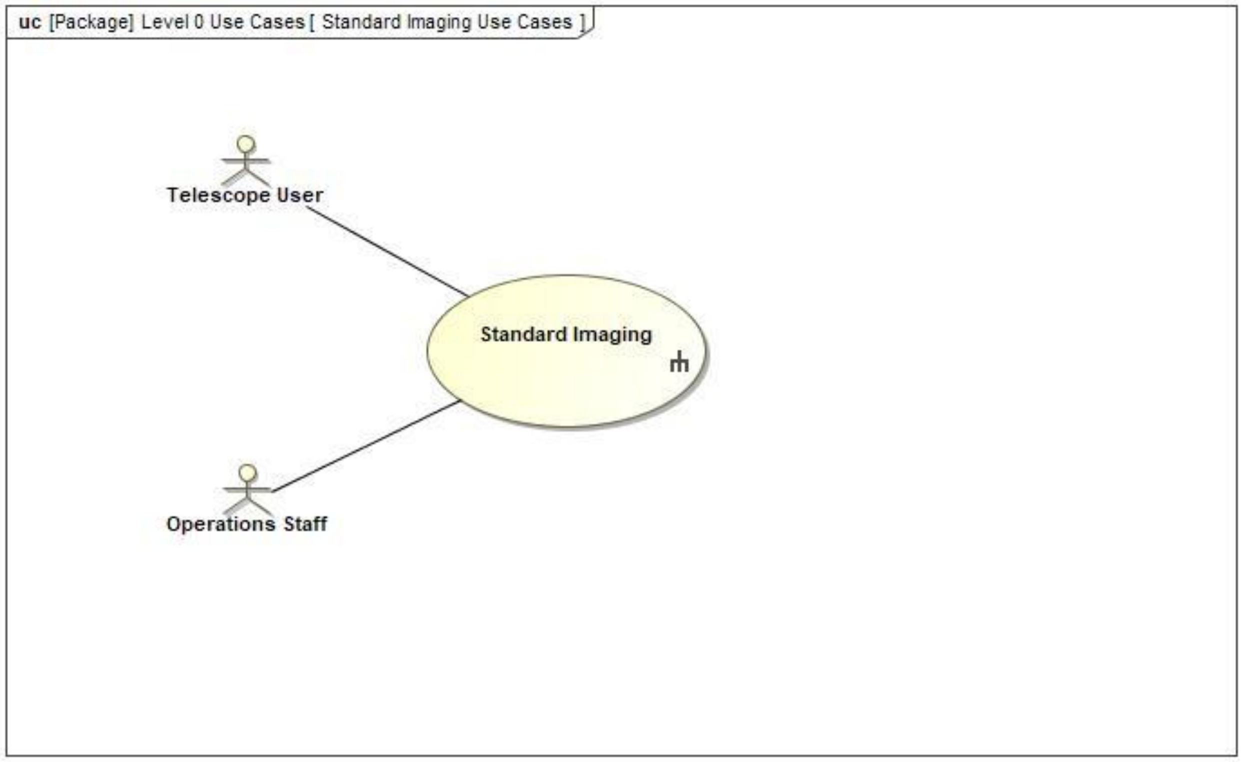

uc [Package] Level 0 Use Cases [Standard Imaging Use Cases]
Telescope User
Standard Imaging
Operations Staff

Diagram: Archive Use Cases

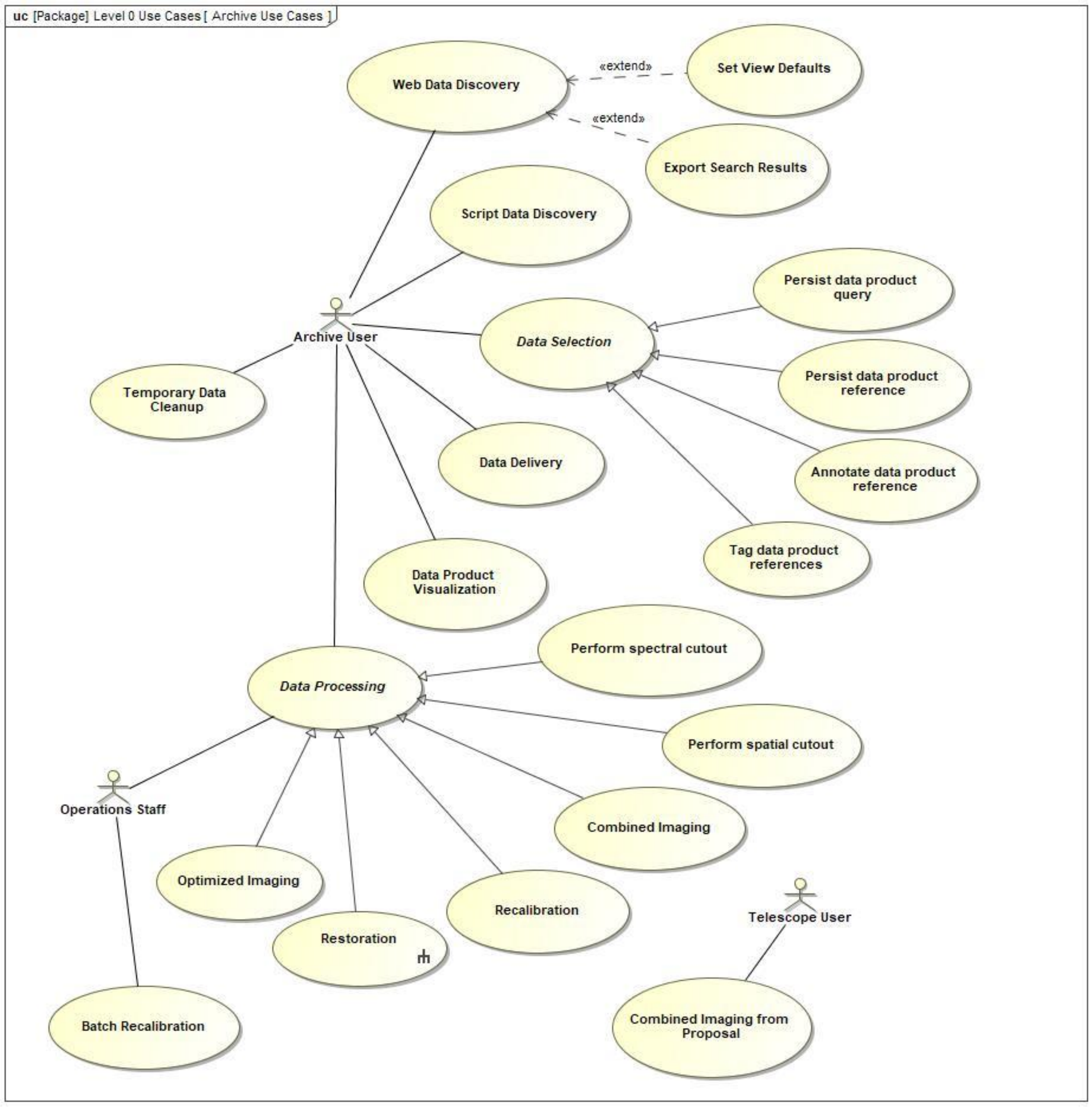
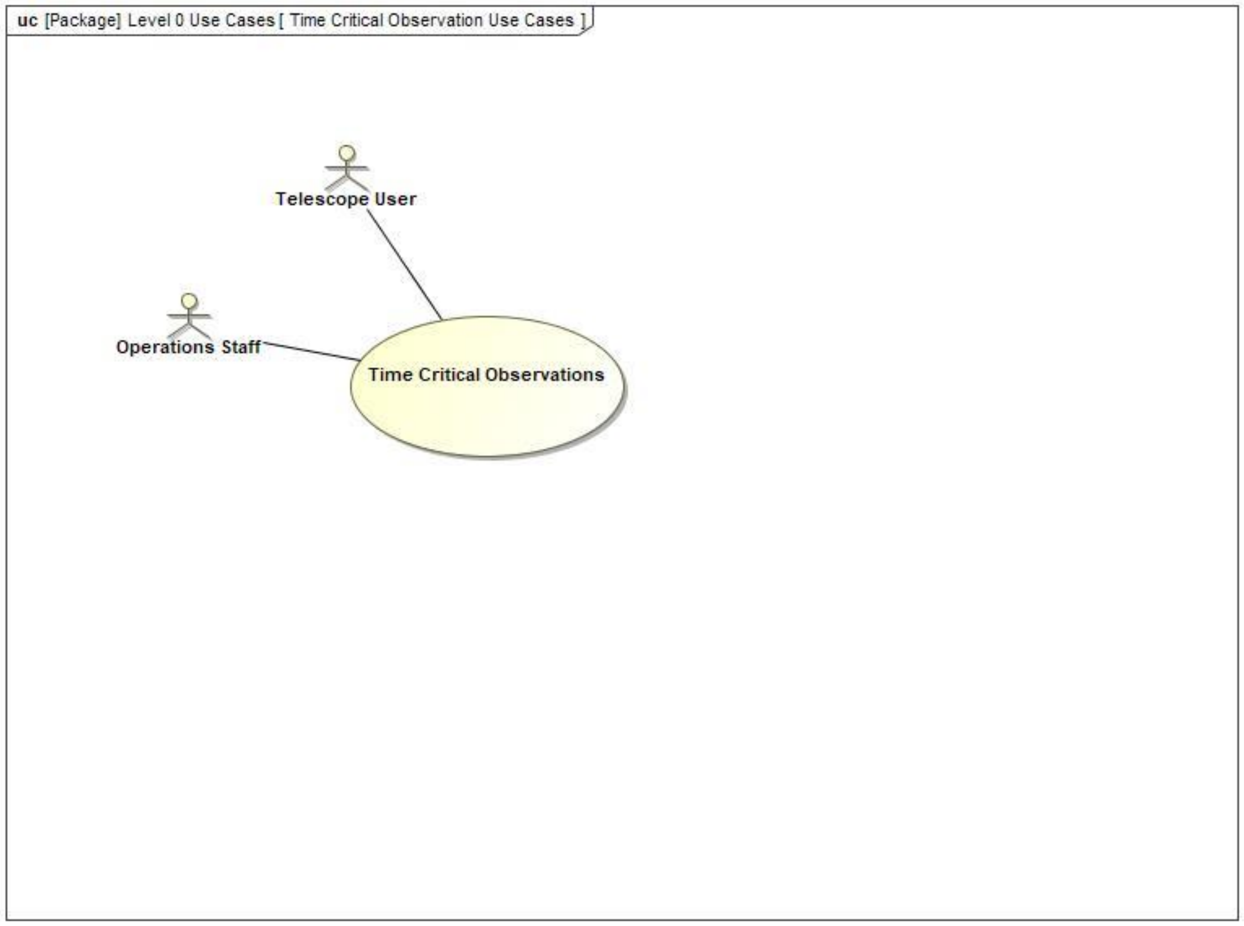


Diagram: Time Critical Observation Use Cases


uc [Package] Level 0 Use Cases [Time Critical Observation Use Cases]
Telescope User
Operations Staff
Time Critical Observations

Package: Behavior

This package contains system-level collaborations that model dynamic behavior not covered in the Use Cases package.

The elements of this package are:

Cache management: Each one of the data product generation use cases (standard calibration, standard imaging, etc.) requires the input data products files to be on the file-system in order to execute the corresponding pipeline. As retrieving or restoring the input data products are expensive operations, and to benefit from the fact that several Jobs could require the same input data products, the system will incorporate a cache. It will be necessary to manage this cache, deleting older files when the total storage occupied by the cached files reaches a certain threshold, for example. The DataFetcher component will first look in the cache for required data products. If these are not found, they will be retrieved and restored in the cache, in such a way that they are available for subsequent Jobs.

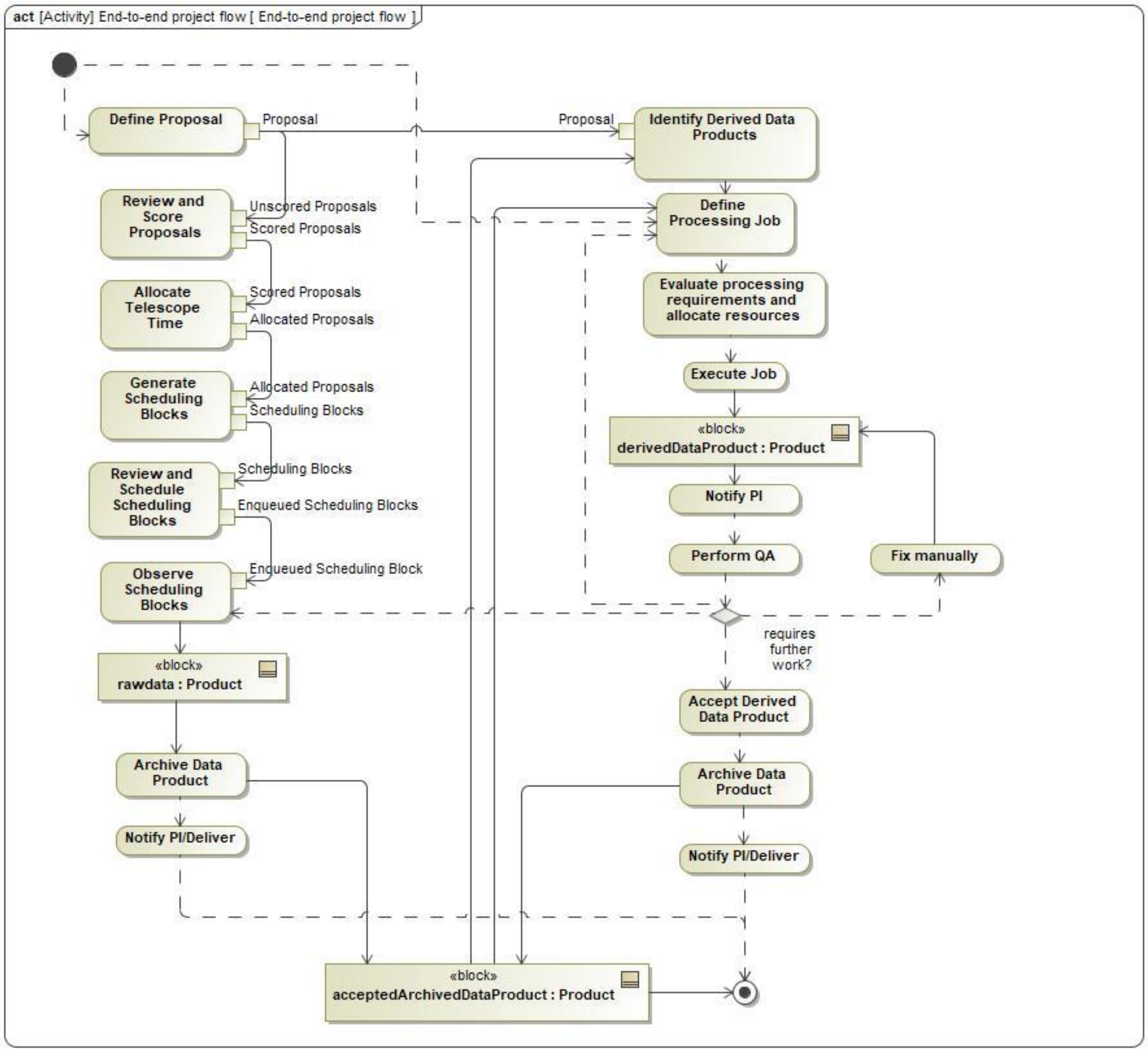
External processing:

Ingest ALMA data: The ALMA Metadata database is periodically searched for new standard calibration and imaging products, and the resultant records are imported into NRAO Metadata Database.

Overview

This section provides an end-to-end view of the important steps involved on producing data products, from the specification of products to their archival and delivery.

Diagram: End-to-end project flow

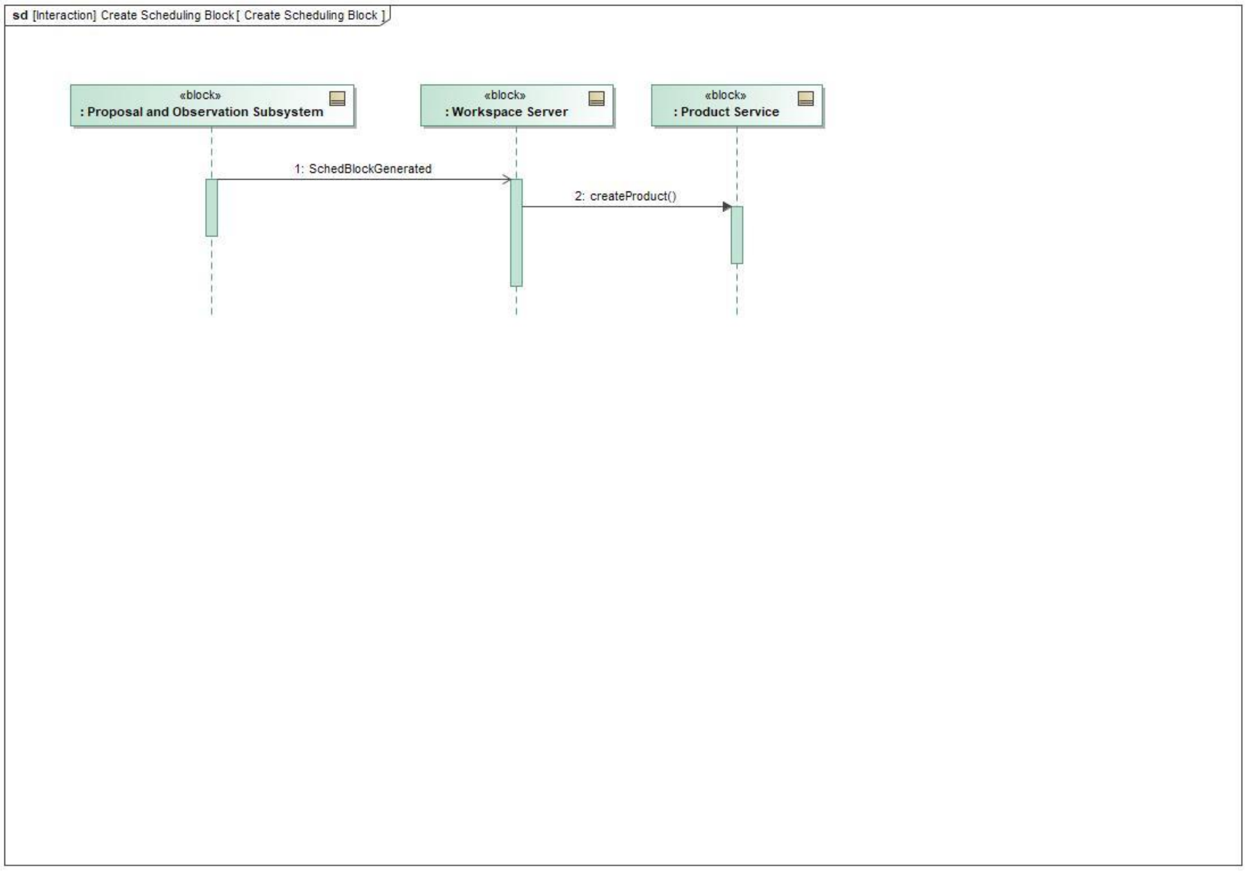


act [Activity] End-to-end project flow [End-to-end project flow]
Define Proposal
Proposal
Proposal
Identify Derived Data Products
Review and Score Proposals
Unscored Proposals
Scored Proposals
Define Processing Job
Allocate Telescope Time
Scored Proposals
Allocated Proposals
Evaluate processing requirements and allocate resources
Generate Scheduling Blocks
Allocated Proposals
Scheduling Blocks
Execute Job
«block» derivedDataProduct : Product
Review and Schedule Scheduling Blocks
Scheduling Blocks
Enqueued Scheduling Blocks
Notify PI
Observe Scheduling Blocks
Enqueued Scheduling Block
Perform QA
Fix manually
requires further work?
«block» rawdata : Product
Accept Derived Data Product
Archive Data Product
Archive Data Product
Notify PI/Deliver
Notify PI/Deliver
«block» acceptedArchivedDataProduct : Product

Interactions

This section documents details about the dynamic interactions between elements that are necessary to fulfill the system functional requirements. At this point of the project, where requirements have been specified only at Level 0, these interactions are neither complete nor specified in enough detail to proceed with the implementation.

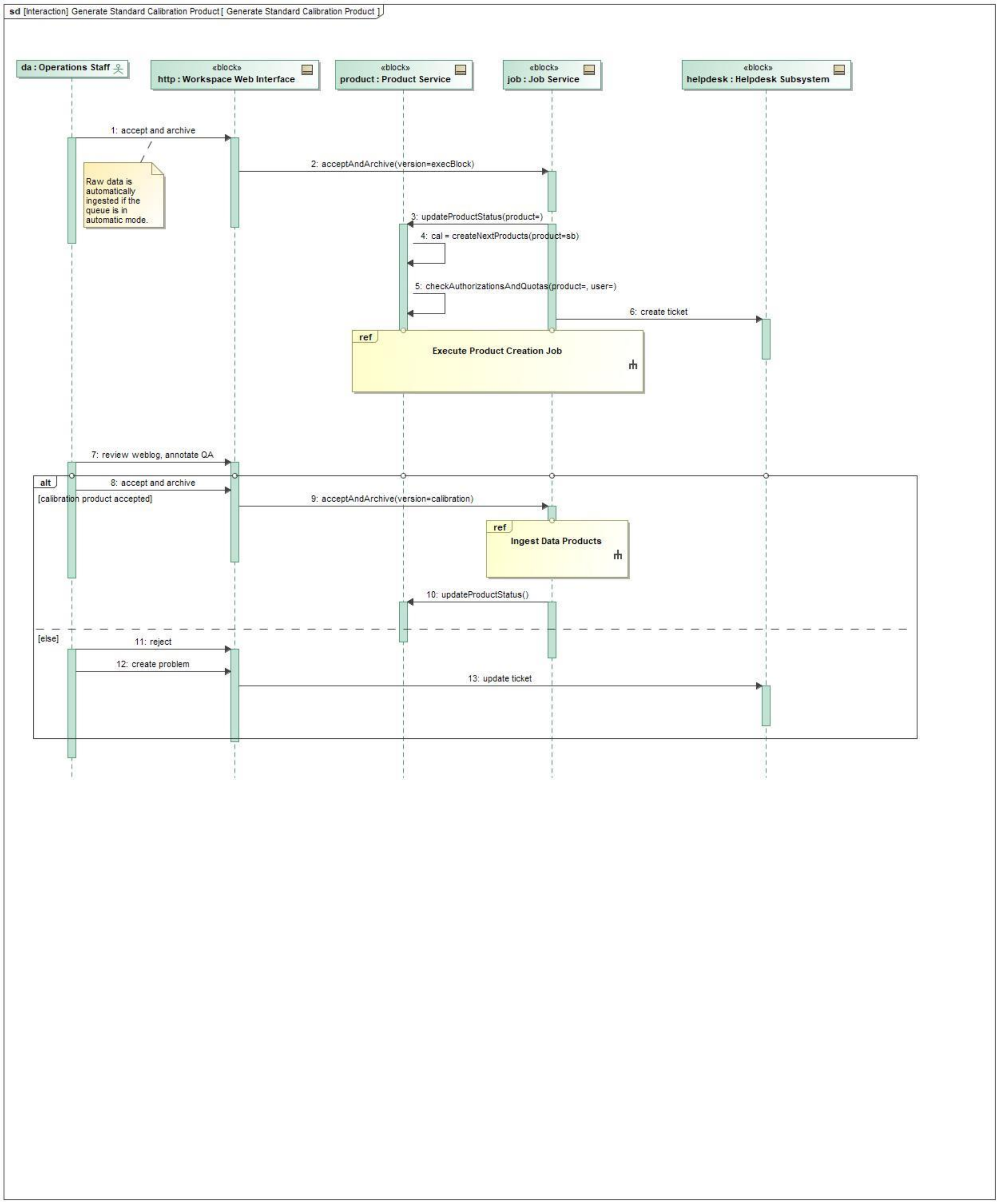
Diagram: Create Scheduling Block



sd [Interaction] Create Scheduling Block [Create Scheduling Block]
«block» : Proposal and Observation Subsystem
«block» : Workspace Server
«block» : Product Service
1: SchedBlockGenerated
2: createProduct()

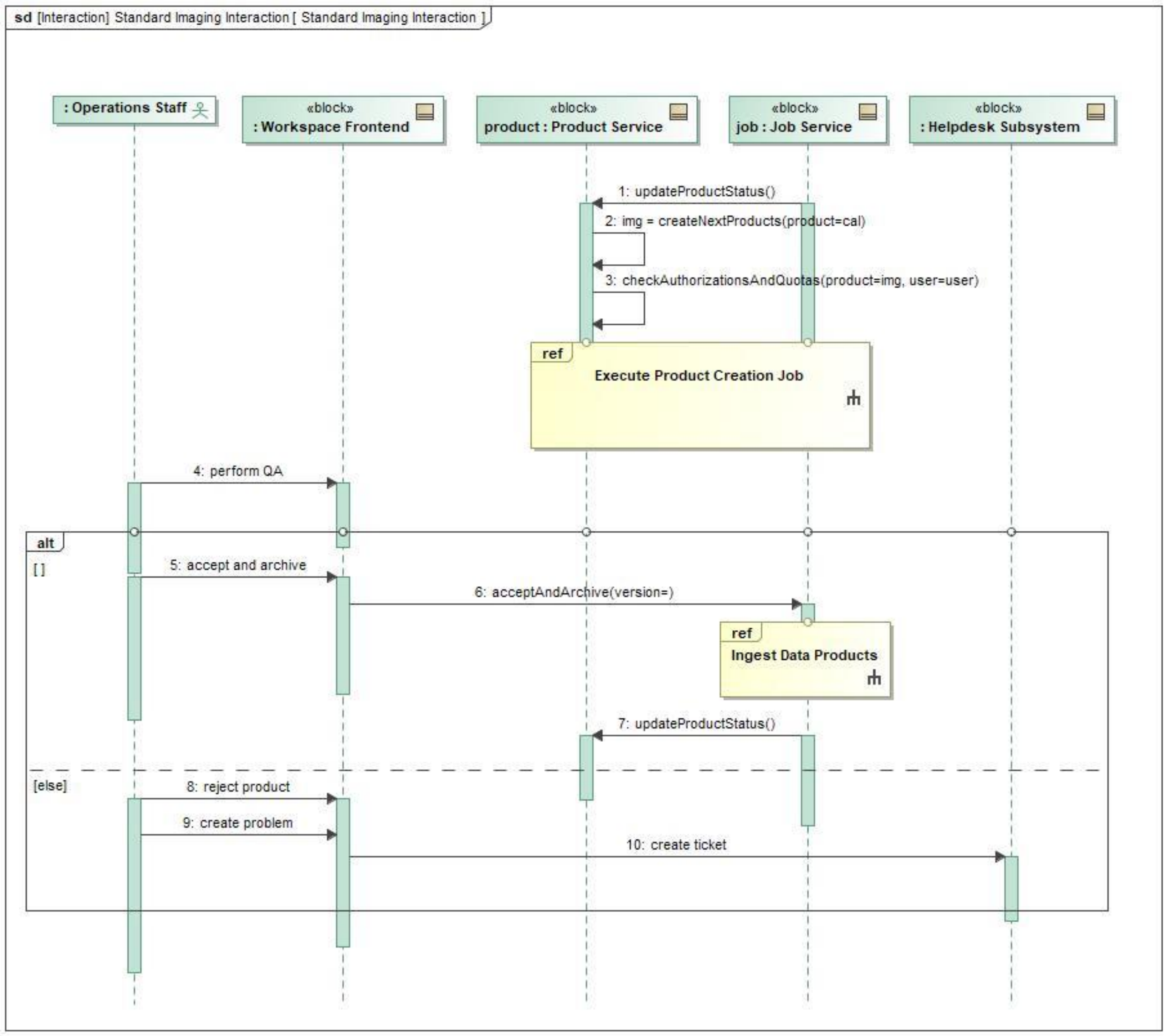
Title: SRDP Architecture	Authors: Rafael Hiriart	Date: 3/27/2018
Document No. 530-SRDP-XXX-XXX		Revision: 0.02

Diagram: Generate Standard Calibration Product



sd [Interaction] Generate Standard Calibration Product [Generate Standard Calibration Product]
da : Operations Staff
«block» http : Workspace Web Interface
«block» product : Product Service
«block» job : Job Service
«block» helpdesk : Helpdesk Subsystem
1: accept and archive
Raw data is automatically ingested if the queue is in automatic mode.
2: acceptAndArchive(version=execBlock)
3: updateProductStatus(product=)
4: cal = createNextProducts(product=sb)
5: checkAuthorizationsAndQuotas(product=, user=)
6: create ticket
ref
Execute Product Creation Job
7: review weblog, annotate QA
alt
8: accept and archive
[calibration product accepted]
9: acceptAndArchive(version=calibration)
ref
Ingest Data Products
10: updateProductStatus()
[else]
11: reject
12: create problem
13: update ticket

Diagram: Standard Imaging Interaction

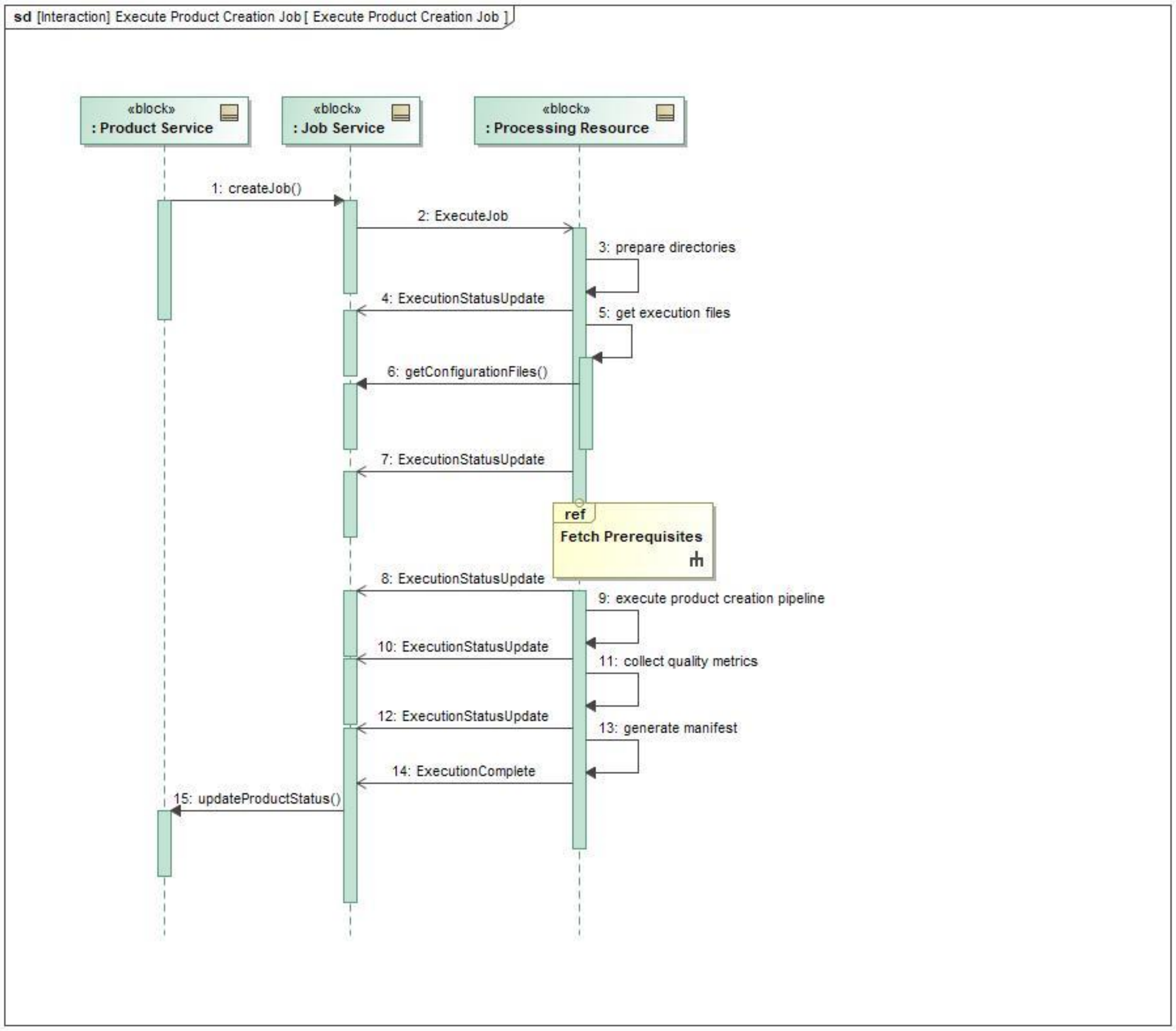


sd [Interaction] Standard Imaging Interaction [Standard Imaging Interaction]
: Operations Staff
«block» : Workspace Frontend
«block» product : Product Service
«block» job : Job Service
«block» : Helpdesk Subsystem
1: updateProductStatus()
2: img = createNextProducts(product=cal)
3: checkAuthorizationsAndQuotas(product=img, user=user)
ref
Execute Product Creation Job
4: perform QA
alt
[ ]
5: accept and archive
6: acceptAndArchive(version=)
ref
Ingest Data Products
7: updateProductStatus()
[else]
8: reject product
9: create problem
10: create ticket

Common Interactions

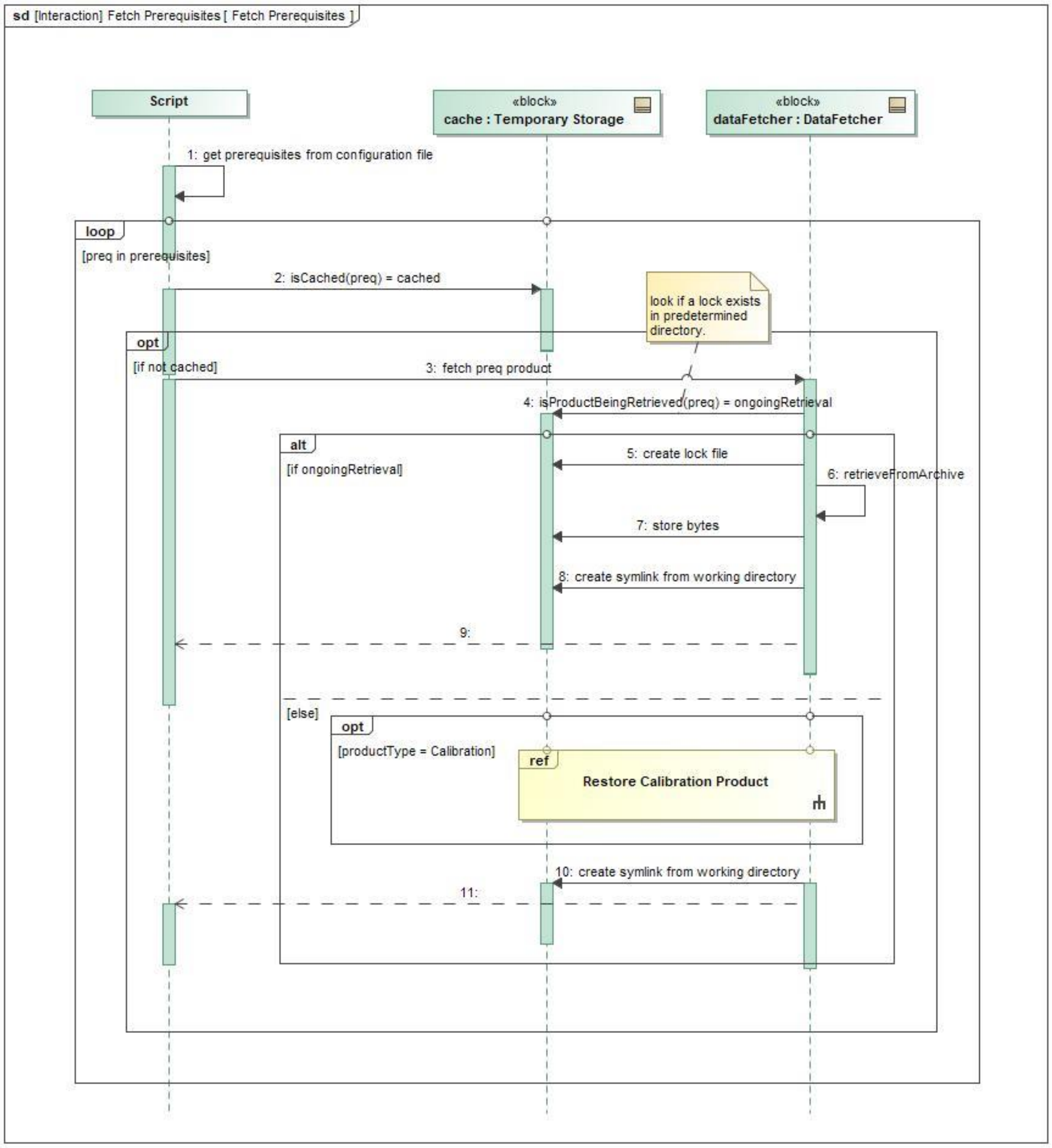
This section presents common interaction fragments that are referenced in the previous section diagrams.

Diagram: Execute Product Creation Job



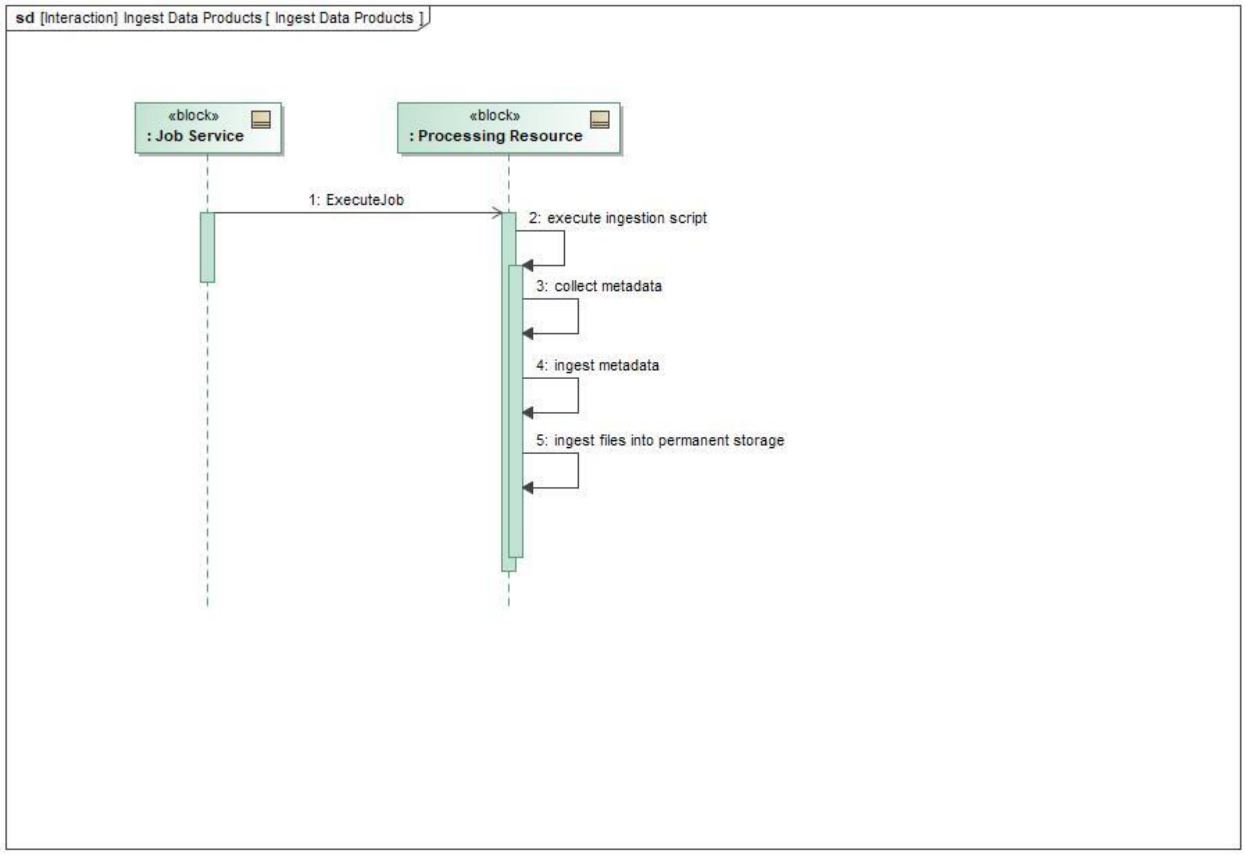
sd [Interaction] Execute Product Creation Job [Execute Product Creation Job]
«block» : Product Service
«block» : Job Service
«block» : Processing Resource
1: createJob()
2: ExecuteJob
3: prepare directories
4: ExecutionStatusUpdate
5: get execution files
6: getConfigurationFiles()
7: ExecutionStatusUpdate
ref Fetch Prerequisites
8: ExecutionStatusUpdate
9: execute product creation pipeline
10: ExecutionStatusUpdate
11: collect quality metrics
12: ExecutionStatusUpdate
13: generate manifest
14: ExecutionComplete
15: updateProductStatus()

Diagram: Fetch Prerequisites



sd [Interaction] Fetch Prerequisites [Fetch Prerequisites]
Script
«block» cache : Temporary Storage
«block» dataFetcher : DataFetcher
1: get prerequisites from configuration file
loop
[preq in prerequisites]
2: isCached(preq) = cached
look if a lock exists in predetermined directory.
opt
[if not cached]
3: fetch preq product
4: isProductBeingRetrieved(preq) = ongoingRetrieval
alt
[if ongoingRetrieval]
5: create lock file
6: retrieveFromArchive
7: store bytes
8: create symlink from working directory
9:
[else]
opt
[productType = Calibration]
ref
Restore Calibration Product
10: create symlink from working directory
11:

Diagram: Ingest Data Products


sd [Interaction] Ingest Data Products [Ingest Data Products]
«block» : Job Service
«block» : Processing Resource
1: ExecuteJob
2: execute ingestion script
3: collect metadata
4: ingest metadata
5: ingest files into permanent storage