ALMA Phasing Project Update to TelCal Design

ALMA-05.11.62.01-0001-C-DSN

2014-04-7

Prepared by:	Organization Role	Date and Signature
H. Rottmann W. Alef	**MPIfR!** **MPIfR!**	
Approved by:	Organization Role	Date and Signature
Authorized by:	Organization Role	Date and Signature

Change Record

Version	Date	Affected Section(s)	Author	Reason/Comments
A	2013-04-24	All	H. Rottmann	CDR Draft
B	2013-06-18	1.3	G. Crew	4-digit seq numbers
C	2014-04-07	All	H. Rottmann	Updated for ALMA-10.8

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Purpose

This document describes the implementation for the required changes to the TelCal (Telescope Calibrations) subsystem in the frame of the ALMA Phasing Project (**APP!**). For a detailed description of the existing TelCal system please consult documents [RD1] and [RD2]. Chapter **??** will review the basic architecture of the existing TelCal system. In chapter **??** we will briefly describe the strategy and the general concept for phasing the **ALMA!** array. Chapter **??** will focus on the implementation details within the TelCal subsystem.

1.2 Reference Documents

The following documents contain additional information, are referenced in this document, and should be consulted for further, more detailed information.

Table 1.1: *Reference Documents*

Reference	Document Title	Document ID
[RD1]	Telescope Calibration Design Document	COMP-70.75.00.00-002-I-DSN
[RD2]	TELCAL IDL Documentation	COMP-70.00.00.00-70.75.00.00-A-ICD
[RD3]	ALMA Software Science Requirements and Use Cases	ALMA-SW-0011
[RD4]	Phasing Algorithm for the Phasing System	not assigned
[RD5]	ALMA Phasing Project Update to Corr/Control Design	ALMA-05.11.61.01-0001-A-DSN

1.3 Acronyms

ALMA Atacama Large Millimeter/submillimeter Array

APP **ALMA!** Phasing Project

MPIfR Max Planck Institute für Radioastronomie

Chapter 2

TelCal Architecture

This chapter gives a brief overview of the general TelCal architecture. It outlines the main components and their responsibilities within TelCal. The picture presented here is in no way a complete description of the TelCal subsystem but focuses on the aspects relevant to the TelCal modifications within the **ALMA!** phasing project (**APP!**). For a comprehensive discussion of the TelCal subsystem please see the original design document ([RD1]).
The TelCal subsystem performs three major tasks which are handled by different TelCal subpackages:

- Reading the data and metadata: handled by `TelCalDataManager`
- Processing the data: handled by `TelCalResults`
- Publishing the results: handled by `TelCalPublisher`

2.1 TelCal package structure

The TelCal system is structured in various subpackages. The following list displays the packages that are relevant in the context of the planned **APP!** extensions of TelCal. Figure **??** sketches the basic TelCal architecture.

- **TelCalDataManager** This package is responsible for reading and filtering of the bulk data streams and the associated metadata. It also controls the workflow of the various calibration tasks that TelCal performs. The `ReceivingDataManager` **ACS!** component runs persistently as it must receive and react on incoming events from the notification channel. The `ReceivingDataManager` component reads the channel-averaged-data stream and obtains the associated scan metadata from the `DataCapturer` component. The scan intent encountered in the scans metadata determines which calibration processing (if any) will be done by TelCal. The TelCalDataManager package contains the `ReceivingDataManager` and the `ParameterTuning` interface. The `ParameterTuning` interface is used to pass additional parameters (not included in the **ASDM!** metadata) to TelCal.
- **TelCalResults** The TelCalResults package contains the engines that implement the algorithms for the various calibration tasks which TelCal performs. This package also contains complimentary libraries (e.g for accessing **ASDM!** data) and the python scripts to be used for offline calibration and plotting in **CASA!**.
- **TelCalPublisher** This package is responsible for publishing calibration results and informing other system components that new results are available (through notification channel events). TelCalPublisher contains the `GetTelCalResults` interface for publishing the calibration results.

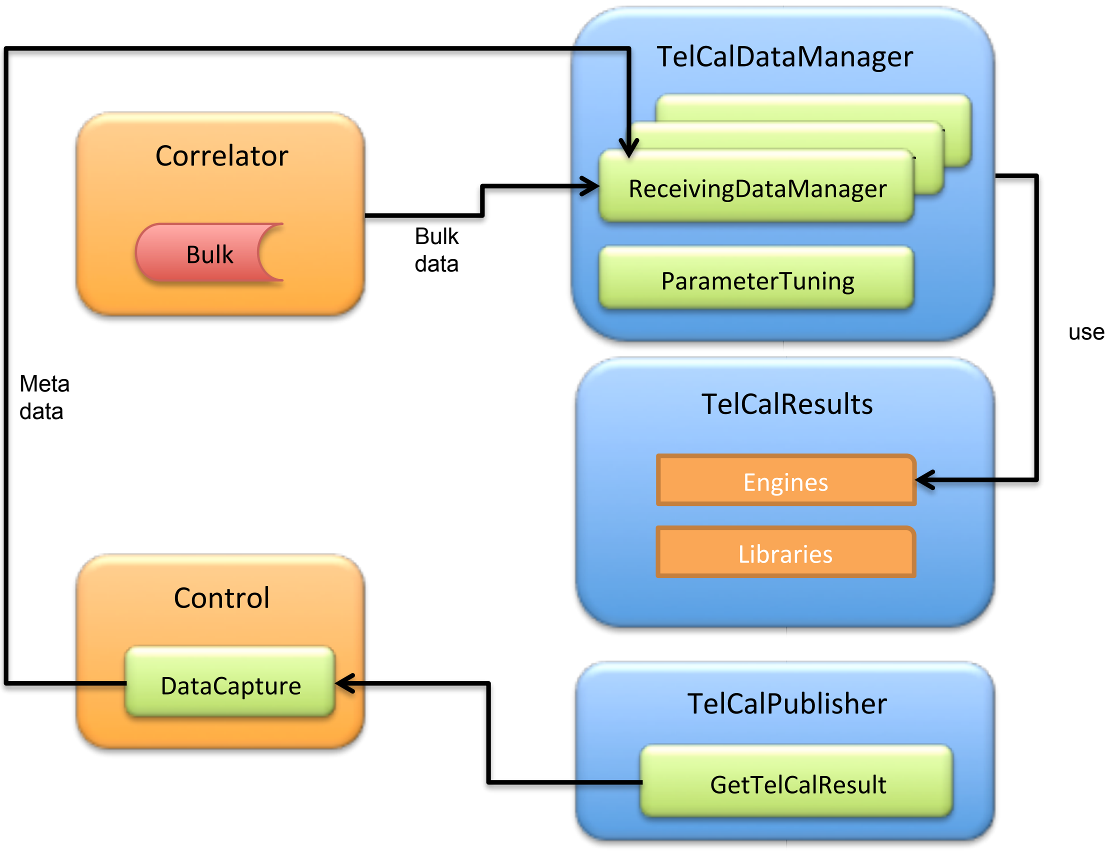

Figure 2.1: *A sketch of the TelCal structure and its relationship to other packages. Note that this figure does not lay out the complete TelCal architecture but focuses on the parts that are relavant for the **APP!** developments.*

2.2 TelCal modes of operation

The TelCal subsystems supports three modes of operation:

- **Online mode** The online mode is the default mode of operation where TelCal is started up as a subsystem of the **ALMA!** software. The TelCal workflow is controlled by consuming and issuing notification channel events. In the online mode TelCal uses real-time bulk and meta data.

- **Offline mode** In the offline mode python scripts can execute non-realtime calibration tasks from (real or simulated) data stored on disk. The offline mode is mainly used in the development process in order to do code unit testing as well as to verify the adopted algorithms.

- **Simulated online mode** This mode is a hybrid mode between the online and the offline mode. Here TelCal is started up as a standalone subsystem (without external components being present). The workflow within TelCal in this mode can be driven by simulated notifications events. The data consumed is stored on disk. This mode is useful for testing the workflow logic within TelCal.

Chapter 3

Phasing loop concepts

3.1 Overview

Phasing of the **ALMA!** array will operate on two different time-scales. The "fast" loop operates on time-scales of a second and applies the **WVR!** and residual delay corrections. The "slow" loop operates on the timescale of a subscan duration (2-30 s) and determines the residual antenna phase corrections in order to bring the phased array sum into coherence. The fast loop is handled by the correlator and is described in details in the **APP!** Corr/Control design document[RD5]. The slow loop is handled by the TelCal subsystem and its concepts will be outlined in the following sections.

Figure **??** displays the sequence diagram for the slow part of the phasing loop. From TelCal's perspective the phasing calibration starts when the `ReceivingDataManager` receives `ScanStarted` event. It then parses the scan intent and, in case a relevant intent is found, initializes the appropriate objects (e.g. `Scan`, `Result`) needed for the phasing task.

Upon receiving the `ScanProcessed` event the `ReceivingDataManager` can access the channel-averaged-data (with the fast loop corrections already applied) and obtains the associated metadata from the `DataCapturer` component via the `getFinalMetaDataQL` interface.

After reading the data and the metadata the actual phasing processing starts. Due to the asynchronous nature of the fast and the slow phasing loop, part of the scan data send by the correlator has phasing solutions from the previous scan already applied. These data have been flagged by the correlator in order to signal to TelCal that this portion of the scan must not be used in the determination of new phasing solutions. In a first step TelCal will select the unflagged portions of the channel-averaged scan data and will time-average these. The phase solver then determines phase corrections for each subband and station so as to bring the phased array sum into coherence (details of the solution algorithms are discussed in sections **??** and **??**. If the phasing source has extended stucture the phase solver will need to compute the source model as part of the solving processing (see section **??**). Finally a quality estimate will be calculated as will be described in section **??**.

When the calibration calculation has finished the `TelCalPublisher` sends the `AppPhaseReduced` event to inform other components that the phasing calibration results can be obtained via the `GetTelCalResults` interface. This is the final step of the slow phasing loop.

3.2 Basic phasing theory

Phasing of local interferometers is a well-proven standard technique implemented at various radio interferometers mostly for **VLBI!**. The assumptions and algorithms used are identical to those of self-calibration in mapping interferometry data. The main purpose is to determine phase-corrections for all antennas of an array so that the individual antenna signals can be added in a coherent way.

For an N-element interferometer a phase $\phi_{obs}(t)$ can be measured as a function of time t on the baselines between any two antennas i and j . The observed baseline phases are the sum of the structure phase of the observed source ϕ_{mod}, the instrumental phase ϕ_{ins}, the atmospheric phase

ϕ_{atm} and a thermal noise term ϕ_n specific to each baseline.

$$\phi_{obs}(t) = \phi_{mod}(t) + \phi_{ins}(t) + \phi_{atm}(t) + \phi_n(t)$$

The structure phases which have to be known with sufficient accuracy from a model, and the instrumental and atmospheric phases have to be removed from the observed baseline phases which can subsequently be added coherently.

Instrumental and atmospheric phases can be factorized by antenna so that after subtraction of the source model phases we can determine the N unknown sums of atmospheric + instrumental (or "antenna") phases. The problem is over-determined for arrays of more than two antennas and we have to solve for $N-1$ antenna phases ψ_c as we are dealing with (antenna) phase differences. The calculated antenna phases $\psi_c(t) = L \cdot (\phi_{obs}(t) - \phi_{mod}(t))$ (where L is some linear transformation) are then used to correct the observed baseline phases. Usually one antenna is designated as the reference antenna r and for a standard least squares solution we get

$$\psi_{c_i} = \frac{1}{W}\left[\sum_{i,j\neq i} w_j\phi_{ij} - \sum_{j,j\neq r} w_j\phi_{rj}\right]$$

where w_j are antenna weights which could reflect the sensitivity of each antenna. W is the sum of the weights. It should be mentioned that the antenna-based noise $\psi_n(t)$ is actually reduced by a factor $\sqrt{N}$ from the baseline-based noise $\phi_n(t)$.

3.3 Source modeling

Two cases have to be considered. In the initial implementation a point source model will be assumed and no model phases have to be considered, as they will all be zero for a point source at the phase center of the array.

For the more general case of a source extended on **ALMA!** scales, δ-components will be provided to TelCal by **VOM!** via the `ParameterTuning` interface (it is anticipated that δ-components will be provided via the VEX-schedule). For each new source to be used for phasing, the δ-components of the source model will be gridded onto an array of adequate dimension using a two-dimensional Gaussian function as often done in standard mapping software. The array will be **FFT!**ed to the UV-plane and model phases will be interpolated for each visibility to be used in the subsequent estimate of the antenna phases $\phi_{ins}(t) + \phi_{atm}(t)$. (For strongly varying model phases on long baselines (unlikely to be good for phasing) the model phases could be subtracted before the data is time averaged or such data could be discarded. In a later implementation TelCal could be modified to decide to do this for long baselines where it is necessary according to some criteria yet to be defined.)

The amplitude taper in the UV-plane introduced by using a Gaussian function for gridding the δ-components can be ignored.

3.4 Phase Solver

An antenna gain solver (simple least squares) is already a part of the `TelCalResults` package and is utilized by some of the calibration engines. It will be also used for the initial implementation for the **APP!** phase solver algorithm.

In an initial step the lease square solution will be implemented without making use of antenna weighting. A weighted solver will be implemented in a second step. In addition to the usual $1/{T_{sys}}^2$ weights it will be enabled to make use antenna weights provided by **VOM!** via the `ParameterTuning` interface. Calculation of weights would be based on the quality estimation given by TelCal and possible manual interaction by the **ALMA!** operator. For instance an antenna found to be bad can be kept in the solution with a reduced weight. Note: even antennas excluded from the phased sum should be included in the phase solver algorithm with very low weights. This will allow monitoring their performance and possibly their re-inclusion in the phased sum upon recovery. The required

matrix inversion has to be calculated only when weights change.
More sophisticated phase solvers might be implemented at a later stage depending on the experience gained during the commissioning phase of **APP!**.

3.5 Phasing modes

The slow phasing loop can operate in two different phasing modes depending on the strength of the astronomical target source:

- **active mode** In case the program source scheduled by the astronomer is sufficiently strong it can serve as a phasing calibrator to calculate the phasing solutions continuously during the observations. New phasing solutions will be calculated and applied for every scan. Scans for which TelCal should determine phasing solutions must receive the `CALIBRATE_APPPHASE_ACTIVE` scan intent.
- **passive mode** In case the astronomical target is too weak to serve as a phasing calibrator the phasing solutions must be determined by scheduling intermittent phase calibration scans on a dedicated, strong calibration source. These calibration measurements will be done in the active mode. The phasing solutions obtained on the correlator are then applied to subsequent (possibly multiple) scans until the next phasing calibration measurement. Scans that should receive phasing solutions obtained on a previously measured phasing calibrator must have the `CALIBRATE_APPPHASE_PASSIVE` scan intent.

  In the initial implementation TelCal will be idle during passive scans. Later, TelCal could be enabled to calculate phasing solutions also on weak sources. These solutions would not be actually applied when forming the phased sum, but might be useful for monitoring phasing performance of individual antennas.

  In the passive mode the lengths of individual subscans and the frequency of measuring intermittent phasing calibrators has to be carefully adjusted to the prevailing (or typical) tropospheric conditions at the site. The parameter space for the passive mode can be tested using archived **ALMA!** data obtained under various weather conditions.

3.6 Phasing quality estimation

Different methods can be implemented to indicate the performance of an antenna in the phasing, so as to allow an operator or an algorithm in **VOM!** to exclude an antenna when forming the phased sum.

The simplest method to be implemented is to monitor the phase corrections of each antenna in **VOM!** as a function of time. After the first very few scans incremental phase corrections should be small for antennas whose instrumental phase and atmospheric conditions are well behaved.

In a first step TelCal will simply calculate the residuals to the fits and provide those to **VOM!** (as part of the `CalAppPhase` result) as an indication of the performance of each antenna.

In a later stage more sophisticated quality estimates could be realized if needed. E.g., as one of the correlator inputs will be the summed signal it can be used as a test antenna. Its signal should be N-times that of the others with the noise increased by $\sqrt{N}$. The amplitude gains on all other antennas should be equal, so that pointing errors, focus errors, and decorrelation for large baselines can be detected. In case different quality methods will be implemted the `ParameterTuning` interface will allow to select the desired method. For details of the currently available quality methods please see Appendix **??** .

3.7 Slow loop timing

The slow loop timing is sketched in Fig. **??** for operation in the active phasing mode. TelCal slow loop processing begins upon reception of the `ScanProcessed` event from `DataCapturer`. At this point channel averaging has been done and the correlator has advances already to processing the next scan. For a comprehensive discussion of the timing issues previous to TelCal please consult the appendix of the Corr/Control design document ([RD5]).

The processing steps in TelCal consist of reading and time averaging the data, obtaining the associated metadata, possibly calculating the source model, running the least square solver and publishing the results. The time-lag introduced by these operations is dominated by the data access, which can take 1-2 seconds. The actual solving process runs on time scales of milliseconds and can be neglected. We expect the latency of the slow loop to be less than 3 seconds from the time TelCal receives the `ScanProcessed` event until it has published the new phasing solutions through the `GetTelCalResults` interface. Employing the new bulk reading system (**BDNT!**) we expect this number to decrease somewhat.

As outlined in this section the slow loop processing introduces an extra time-lag (in addition to the fast loop time-lag) between the end of a scan and the availability of new phasing solutions. As the correlator applies new solutions immediately when available, we have the situation where (except for the very first scan of a scan sequence) the first part of a scan will already have phasing solutions applied. As TelCal should not include this scan portion to calculate new solutions the correlator will flag these datapoints invalid for use in TelCal.

For completeness we show in Fig. **??** the timing diagram also for the case of the passive phasing mode.

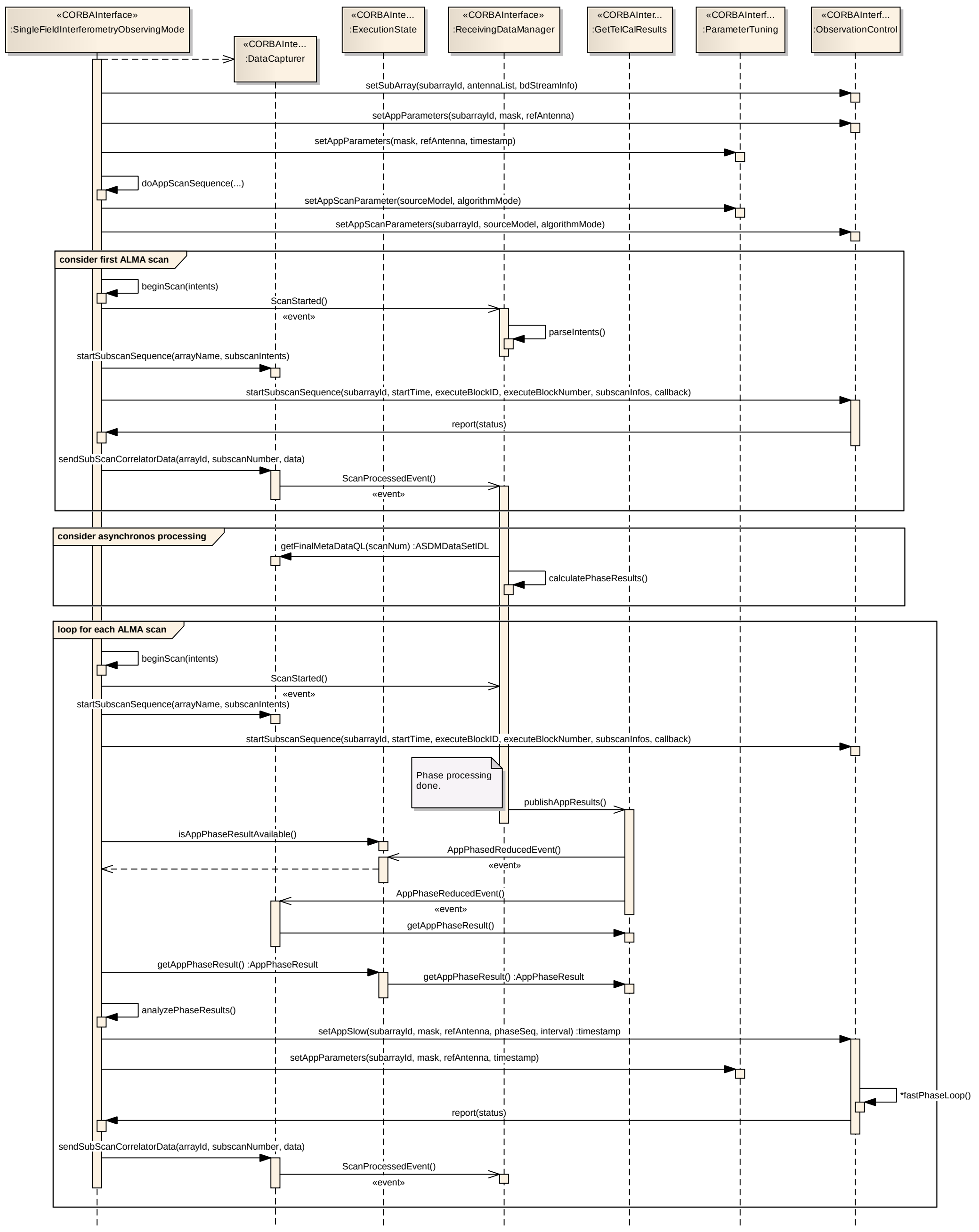
«CORBAInterface» :SingleFieldInterferometryObservingMode
«CORBAInte... :DataCapturer
«CORBAInte... :ExecutionState
«CORBAInterface» :ReceivingDataManager
«CORBAInter... :GetTelCalResults
«CORBAInterf... :ParameterTuning
«CORBAInterf... :ObservationControl
setSubArray(subarrayId, antennaList, bdStreamInfo)
setAppParameters(subarrayId, mask, refAntenna)
setAppParameters(mask, refAntenna, timestamp)
doAppScanSequence(...)
setAppScanParameter(sourceModel, algorithmMode)
setAppScanParameters(subarrayId, sourceModel, algorithmMode)
consider first ALMA scan
beginScan(intents)
ScanStarted()
«event»
parseIntents()
startSubscanSequence(arrayName, subscanIntents)
startSubscanSequence(subarrayId, startTime, executeBlockID, executeBlockNumber, subscanInfos, callback)
report(status)
sendSubScanCorrelatorData(arrayId, subscanNumber, data)
ScanProcessedEvent()
«event»
consider asynchronos processing
getFinalMetaDataQL(scanNum) :ASDMDataSetIDL
calculatePhaseResults()
loop for each ALMA scan
beginScan(intents)
ScanStarted()
«event»
startSubscanSequence(arrayName, subscanIntents)
startSubscanSequence(subarrayId, startTime, executeBlockID, executeBlockNumber, subscanInfos, callback)
Phase processing done.
publishAppResults()
isAppPhaseResultAvailable()
AppPhasedReducedEvent()
«event»
AppPhaseReducedEvent()
«event»
getAppPhaseResult()
getAppPhaseResult() :AppPhaseResult
getAppPhaseResult() :AppPhaseResult
analyzePhaseResults()
setAppSlow(subarrayId, mask, refAntenna, phaseSeq, interval) :timestamp
setAppParameters(subarrayId, mask, refAntenna, timestamp)
*fastPhaseLoop()
report(status)
sendSubScanCorrelatorData(arrayId, subscanNumber, data)
ScanProcessedEvent()
«event»

Figure 3.1: *Sequence diagram of the slow phasing loop the interaction between VOM and TelCal. Fragments highlight the first (unphased) scan, the asynchronous phase processing (overlapping with scans) and the loop for every subsequent scan.*

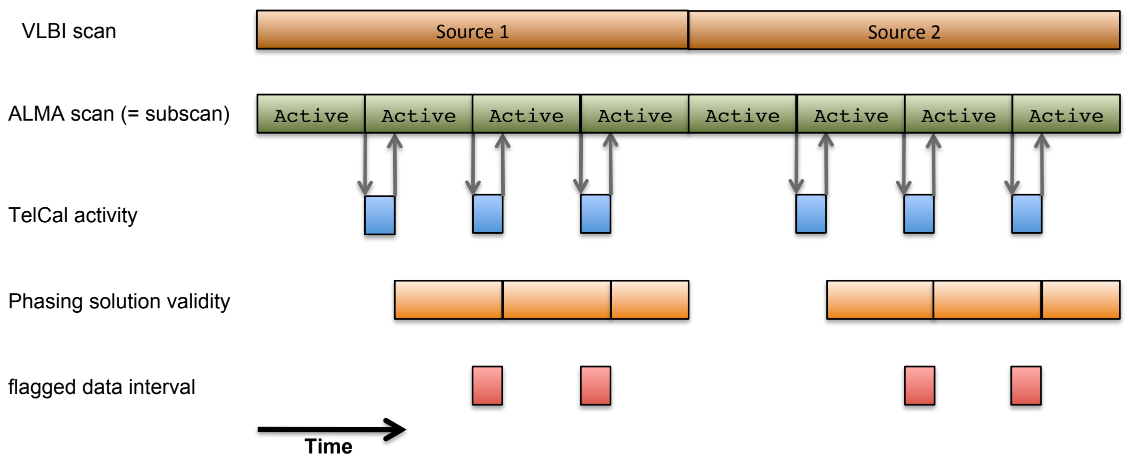

Figure 3.2: *Timing of the slow phasing loop in case of operating in the active mode.*

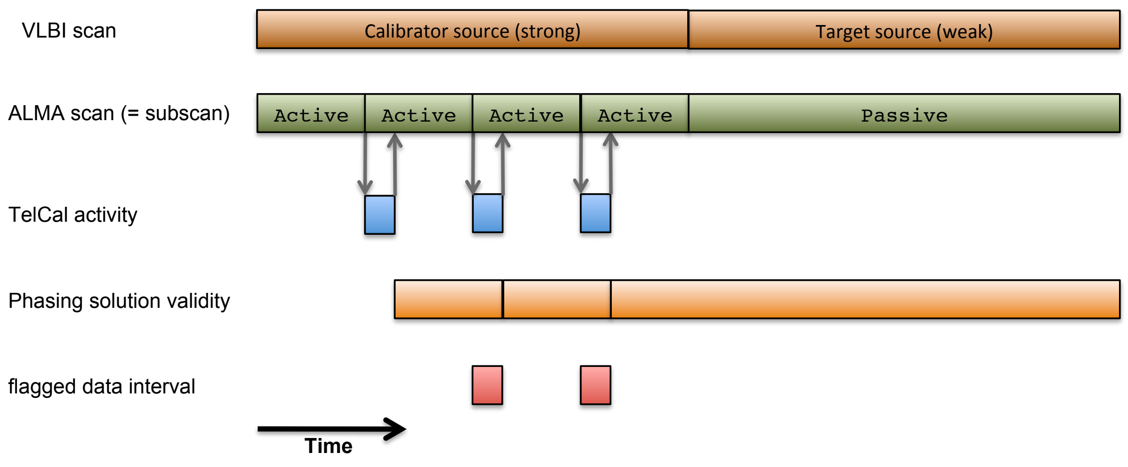

Figure 3.3: *Timing of the slow phasing loop in case of operating in the passive mode.*

Chapter 4

Implementation details

In chapter **??** we have reviewed the basic architecture and the general concepts of operation of the present TelCal subsystem. It is important to note at this point that the work within **APP!** can be fully realized within the current architecture by simply extending the existing interfaces and components.
In the following sections the required changes to the various TelCal packages will be described in more detail.

4.1 TelCalDataManager package

4.1.1 Event consumers

Currently all TelCal action is controlled by consuming the `ScanStarted` and `ScanProcessed` events from the notification channel. This is true also for the new phasing calibration logic to be implemented by the **APP!**. A few modifications need to be made to the relevant event consumers:

- `ScanStartedEventConsumer` This consumer prepares the `Scan` object which will be used in the actual calibration processing. Initialization is driven by the encountered scan intent and therefor `ScanStartedEventConsumer` needs to be instructed to also react on the new scan intents introduced for the phasing calibration `CALIBRATE_APPPHASE_ACTIVE` and `CALIBRATE_APPPHASE_PASSIVE`. Specifically these intents must be included in the call to `Scan::addScanIntent()` within the `ScanStartedEventConsumer::newScan()` method.
  Another task of `ScanStartedEventConsumer` is to read additional calibration parameters from `ParameterTuning` and attach them to the `Scan` object. This is handled within `ScanStartedEventConsumer::setParameterTuningParameters` which needs to be modified in order to handle the extra parameters required for the phasing calibration (see **??**).

- `ScanProcessedEventConsumer` This consumer commands reading of the binary bulk data as well as of the associated **ASDM!** metadata and attaches those data to the `Scan` object.
  It then initiates the appropriate calibration engine (depending on the scan intent) and starts processing. After processing has finished it makes the calibration result available via the TelCalPublisher.
  The `ScanProcessedEventConsumer::publish` method needs to be extended in order to convert the phasing result structure into the `CalAppPhase` **ASDM!** table and publish it.

4.1.2 ReceivingDataManager

The `ReceivingDataManager` component currently provides access to the channel-averaged data, the T_{sys} measurements and the **ASDM!** metadata which is sufficient for realizing the **APP!** implementation. An implementation for the new **BDNT!** technology that will be used by **ALMA!** in the future to read the bulk data already exists. This component requires no changes.

4.1.3 TelCalAsdmResultsConverter

`TelCalAsdmResultsConverter` converts the result structures of the various calibration operations to **ASDM!** result tables. The phasing calibration to be added will have its own result structure (`AppPhaseResult`) and a corresponding new **ASDM!** result table (`CalAppPhase`). `TelCalAsdmResultsConverter` needs to be extended with a method to perform the conversion, e.g.:

```
CalibrationResult *TelCalAsdmResultsConverter::convert
    (AppPhaseResult *result, Scan *scan_p, string arrayName);
```

4.1.4 ParameterTuning

The `ParameterTuning` interface allows to get/set extra parameters for the calibration procedures. In the **APP!** case a few extra parameters will be passed to the phasing calibration via this interface:

- **quality calculation method** As outlined in section **??** the characterization of the per station phasing quality can be realized in various different ways. The `ParameterTuning` interface will be extended in order to allow the **VOM!** to select the quality estimation method to be applied by TelCal.
- **source model** For non-pointlike phase calibrators the phase solver must take into account the source model structure (see **??**). The `ParameterTuning` interface will be extended to allow passing a list of δ-components that characterize the source model to be used by the phase solver. The δ-components must have been determined prior to the **ALMA! VLBI!** observations and can be specified by the observer through the **VEX!** file (see also [RD5]).
- **antenna weights** In the first stage the phase solver will use antenna weights based on T_{sys} as described in section **??**. However, in a later stage it might be desirable to allow setting of additional weights that are based on the automatic estimations of the phasing quality and/or an interactive decision of the **ALMA!** operators. In order to allow setting of antenna weights by e.g. **VOM!** the `ParameterTuning` interface will contain an appropriate interface method.

Appendix **??** provides a list of all currently implemented parameters.

4.2 TelCalResults package

4.2.1 Engines

- `AppPhaseResult` This is a new class derived from the `Result` base class contained in TelCalResults. `AppPhaseResult` will hold the results of the phasing calibration.
- `AppPhaseScan` This is a new class derived from the `Scan` base class contained in TelCalResults. This class must implement the `Scan:process()` method which holds the actual code for performing the phasing calibration and must return an `AppPhaseResult` object.
- `Scan` This is the base class for the various derived calibration engines. The empty `::process` interface method must be implemented by the subclasses for performing the various calibration tasks. The `Scan::addCalDataRow` must be extended in order to accomodate for the new `CAL_APPPHASE` calibration type.
- `Calibration` This class serves as the factory for the various engines that perform the different calibration tasks. Based on the the scan intent the `Calibration:process` calls the appropriate `Scan:process` methods to perform the appropriate calibration action. This class must be extended accordingly in order to initiate the phasing calibration when encountering the `CALIBRATE_APPPHASE_ACTIVE` or `CALIBRATE_APPPHASE_PASSIVE` scan intents.

4.2.2 AsdmReader

The `AsdmReader` class is used by TelCal to access the scan's **ASDM!** metadata. Any updates to the **ASDM!** datamodel like the proposed addition of the two new **APP!** related tables (see **??**) will be automatically available within `AsdmReader` without the need for explicit coding. No changes required here.

4.3 TelCalPublisher package

4.3.1 `GetTelCalResults`

The **IDL!** defintion and the implementation of the `GetTelCalResults` interface must be extended to contain a `getAppPhaseResult` which should return an `AppPhaseResult` object.

4.4 ASDM

The phasing of the **ALMA!** array will require to add two new tables to the **ASDM!** datamodel: `AppParameters` to hold parameters relevant for the phasing (e.g. the reference antennna) and `CalAppPhase` for holding the results of the phasing calibration of a scan. The structure of these new tables is described in detail in the **APP!** Corr/Control design documents ([RD5]).
Additionally, new elements need to be added to these **ASDM!** enumerations:

- **ScanIntent** Two new scan intents (`CALIBRATE_APPPHASE_ACTIVE` and `CALIBRATE_APPPHASE_PASSIVE`) are needed to initiate the phasing calibration in the active and the passive mode respectively.
- **CalType** The phasing calibration will receive the new calibration type: `CAL_APPPHASE`.

Appendix A

Parameter Tuning

The following ParameterTuning parameters are being evaluated on the `CALIBRATE_APPPHASE_ACTIVE` and `CALIBRATE_APPPHASE_PASSIVE` scan intents.

A.0.1 Global parameters

- `appSimulationMode` [integer] activates the solution engine in simulation mode. The actual channel averaged visibilities get replaced by simulated ones depending on the simulation mode selected:

  0 simulation is off (default)

  1 simulated visibilities with $amp = 1$ and antenna phases increaseíng in steps of $20°$

  2 simulated visibilities with $amp = 1$ and random antenna phases between $-\pi$ to $+\pi$

  3 simulated visibilities with $amp = 1$ and random antenna phases between -5π to $+5\pi$

- `appDebugMode` [integer] Activates the debug mode if set to value > 0. When activated the CalAppPhase engine will print additional debug information to the ACS logs and write diagnostic data to severeal ASCII files (located in the home directory of the user running ACS):

  1. `solution0.txt / solution1.txt` contain the phase solutions for the two polarizations respectively.
  2. `fullVisiDump.txt` contains the channel averaged visibilities for all baselines and polarizations.
  3. `sigma.csv` contains the standard deviations of phase residuals per antenna.

- `phasePacking` [string] Defines how many solutions will be delivered by the phasing algorithm. Valid inputs are:

  - `ONE_PER_ANT` One phase solution per antenna, polarization and baseband. The visibilities will be time-averaged over the subscan. If the baseband contains more than one spectral window and/or mutiple channels the visibilities will be channel-averaged prior to solving for the phases.

- `phasedSumAntenna` [string] The name of the antenna to be used for the phased sum.

A.0.2 Baseband-dependent parameters

The following parameters can be set independentely for each baseband by appending the baseband name to the parameter; e.g. `phasingMode_BB_1` will set the phasing mode for BB_1.

- `phasingMode_`$baseband Name$ [string] The content of this parameter is simply passed on into the CalAppPhase table. It has no impact TelCal operations.

- `adjustToken_`*basebandName* [string] A list of text tokens separated by '+'. The content of this parameter is passed on into the CalAppPhase table. It has no impact TelCal operations. TelCal might however add these tokens to the received token list:
  - `NODATA` added if no integrations remain in the subscan for calculating phase solutions, e.g. due to a `phaseAdjTime` that is later than the end time of the subscan.
- `phasedAntennas_`*basebandName* [string] a ':' separated list of antenna names to be included in the phased sum.
- `refAntenna_`*basebandName* [string] the name of the antenna to be used as the reference antenna.
- `efficiencyAntennas_`*basebandName* [string] a ':' colon separated list of anetenna names to be used for calculating phasing efficiencies.
- `phaseAdjTime_`*basebandName* [integer] The time of the last phase adjustment made by the correlator. The time will be interpreted as nanoseconds since the 17 November 1858 00:00:00 UTC.
- `phasedArrayAdjTime_`*basebandName* [integer] The time of the last adjustement of the phased array antenna configuration. The time will be interpreted as nanoseconds since the 17 November 1858 00:00:00 UTC.

Appendix B

Quality calculation

TelCal should provide one quality estimate per baseband for each antenna. The quality values should be normalized to values in the range of $0 \leq q \leq 1$ (see also [RD5]).

B.0.3 Least square fit residuals

TelCal evaluates the residuals of the least square fit of the antenna phases to provide a quality figure of merrit. For each baseline the phase residual Δ_{ij} is calculated in the following way:

$$\Delta_{ij} = \phi_{ij} - (\varphi_i - \varphi_j) \tag{B.1}$$

where ϕ_{ij} is the observed baseline phase and φ_i and φ_j are the phase solutions obtained by the least square fit. For each antenna we now construct the standard deviation σ_i of the phase residuals:

$$\sigma_i = \sqrt{\frac{\sum_{j=1,j\neq i}^{N} \sum_{k=1}^{n_{pol}} (\Delta_{ijk})^2}{(N-1) \cdot n_{pol}}} \tag{B.2}$$

where N denotes the number of antennas in the array and n_{pol} is the number of polarizations. Note that σ_i provides a pessimistic estimate as the total phase residual occuring on the baseline is fully attributed to each single antenna of that baseline.
In the pure noise we expect $\sigma_{max} = \frac{\pi}{\sqrt{3}}$ [1]. The quality estimate q_i is now obtained by relating σ_i to σ_{max}:

$$q_i = \begin{cases} \frac{\sigma_{max} - \sigma_i}{\sigma_{max}}, & \text{if } \sigma_{max} \geq \sigma_i \\ 0, & \text{if } \sigma_{max} < \sigma_i \end{cases} \tag{B.3}$$

[1] see Thompson, Moran, Swenson, "Interferometry and Synthesis in Radio Astronomy", Section 9.3, p. 318