AIPS++ Software Design: Telescope Data Handling

Last changed on $Date: 2003/07/16 03:46:22 $

T. Bottomly

Introduction

This is a design document intended to describe both the object and functional models relating to the input processing of telescope data in the AIPS++ system. This is, by no means, a final design document; suggestions for improvement, change, etc. are welcomed.

The following approach was taken in deriving the design diagrams:

1. An object model was developed for VLA Archive Tape Processing. The model did not include attributes or methods for the defined classes.
2. The object model was "generalized" to handle the following types of telescope data sources: the real-time system, a telescope system simulator, and other data media on which telescope data resides. Again, the model lacked class attributes and methods.
3. Functional diagrams were developed for the VLA Tape Processing case.
4. The object model (VLA-specific) was updated to reflect the methods and attributes made apparent during the creation of the functional diagrams.
5. Functional diagrams were developed for the "general case".
6. The object model (general) was updated to reflect the functional diagrams.

This document describes the design for the general case followed by that for the VLA.

References

Related documents include the following:

- *Object-Oriented Modeling and Design*; Rumbaugh, Blaha, Premerlani, Eddy and Lorenson, Prentice-Hall, 1991.
- *AIPS++ Software Design – Analysis and Design of Major AIPS++ Subsystems*; Hjellming and Glendenning, 1993.
- *AIPS++ Implementation Memo #109, An Overview of AIPS++ Design*, Hjellming and Glendenning, 1993.
- *VLA Synchronous System Archive Tape Format*; Sowinski and Bottomly, 1993.

1 Class Design – General

The following diagram describes the class structure related to a *DataSource* – a source of measurements for a *MeasurementSet* and data for a *TelescopeModel*.[1]. The different "kinds" of *DataSources* include: the data media written by a telescope data system, a real-time data stream from a telescope system, and a telescope system simulator.

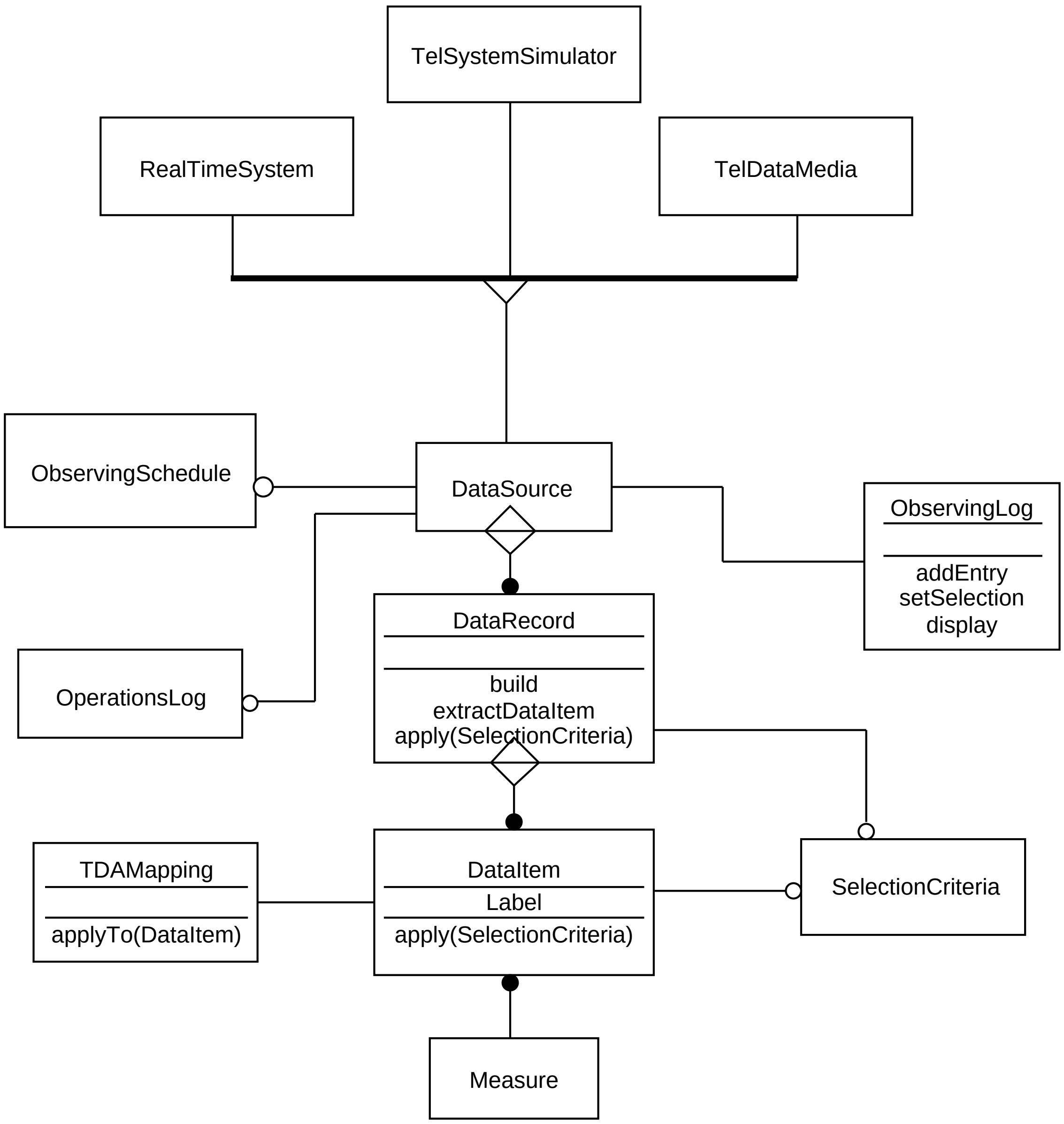
Associated with each *DataSource* is an optional *ObservingSchedule* which reflects the parameters chosen for an observation. The second prototype does not include any processing of the *ObservingSchedule*, but it could be used in the future for a comparison between scheduled observation parameters and the results of the corresponding observation run. The *ObservingLog* contains results of the observation run in the form of a time-based summary of the measurements and telescope behavior from a *DataSource*. Log entries are generated when there is a significant change of parameters in the input data stream.

Additionally associated with each *DataSource* is an optional *OperationsLog*. This is a collection of entries made by operations personnel or the real-time telescope data system during an observation run. An observed problem with antenna power and corresponding time is one example of the contents in a log entry. Data could be flagged during input processing based on the entries in this log. This is another item which is not considered in the second prototype.

Each *DataSource* is made up of many *DataRecords* which in turn are made up of many *DataItems*. A *DataItem* is associated with a *Measure* as the input stream is processed. This association represents at least two diffent kinds of *DataItems*: those which are "control" or "context" items, i.e., define record size or structure; and those which contain the data related to the observation run, i.e., measurement values or instrument-related data. The former type of data is used in parsing a data stream, the latter type is placed into a *TelescopeDataAssociation*.

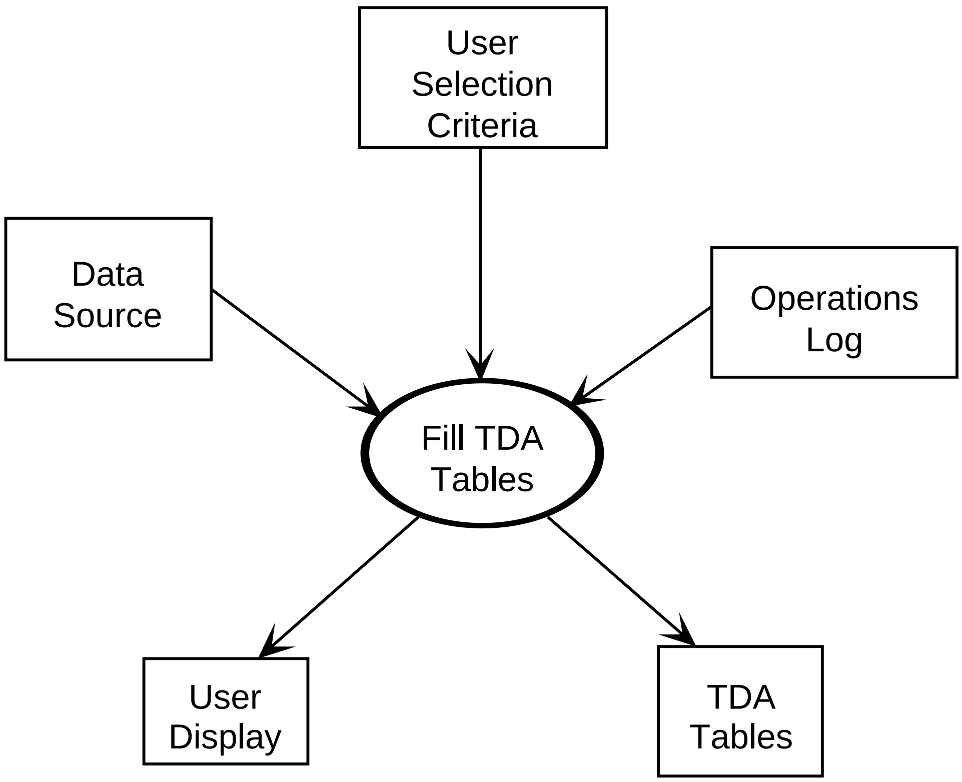
A *DataRecord* may or may not require decoding before *DataItems* are extracted. In the general case, it is assumed that decoding is unnecessary. Selection Criteria can be applied to a *DataItem* or a *DataRecord*. Finally, the selected *DataItems* are placed in *TelescopeDataAssociation* tables via the *TDAMapping*.

[1] AIPS++ Software Design – Analysis and Design of Major AIPS++ Subsystems; Hjellming and Glendenning, 1993


TelSystemSimulator
RealTimeSystem
TelDataMedia
ObservingSchedule
DataSource
ObservingLog
addEntry
setSelection
display
DataRecord
build
extractDataItem
apply(SelectionCriteria)
OperationsLog
TDAMapping
applyTo(DataItem)
DataItem
Label
apply(SelectionCriteria)
SelectionCriteria
Measure

Data Source

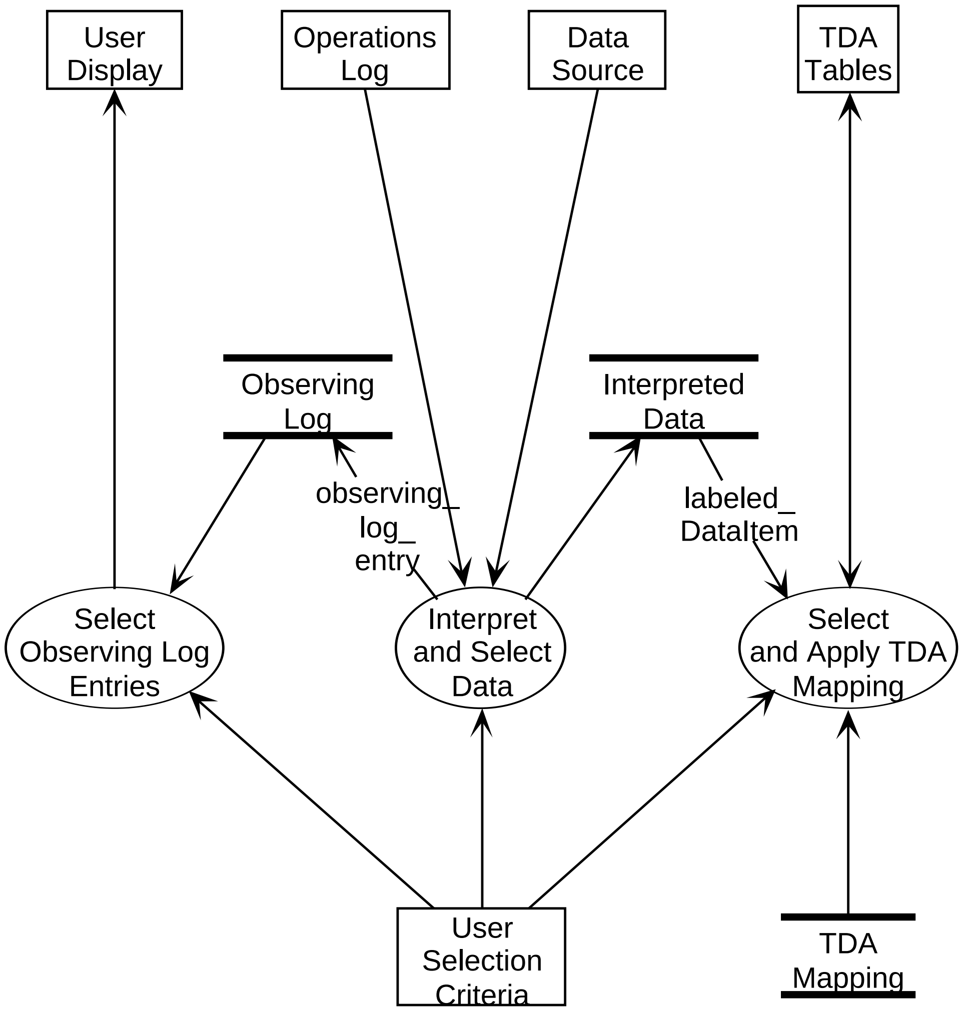
2 Functional Design – General



Process Data Source

The initial functional diagram for processing telescope data from a *Data Source* shows the context in which the processing is done. Filling of the tables which make up the *Telescope Data Association* is based on *User Selection Criteria*. Observation summary information created during the process of filling the TDA tables is made available to the user via the

User Display. An *Operations Log* may or may not be available to allow the flagging of data or the bypass of decoding input data.



Fill TDA Tables

In this diagram the *Fill TDA Tables* process has been functionally decomposed into the following processes:

- a process which handles selection of observing log entries
- a process which interprets the input and
- a process which applies the *TDA Mapping*

Selection can occur before or after interpretation of the data.

Interpret and Select Data obtains data from *Data Source*, creating labeled *DataItem*s and their associated *Measure*s. The input data stream frequently consists of "header" and "measurement" data. The former is often "control" or "context" information which is processed internally by the *Interpret and Select* process. The latter type of data is placed into *Interpreted Data* store for later mapping into the TDA hierarchy. When applicable, *observing_log_entries* are generated.

It is intended that the internal data store *Input to TDA Mapping* be a static mapping which "directs" a *DataItem*, based on its *Label*, to the target table. Implementation of the store should be flexible enough to allow for ease of change in an input data stream with minimal impact. Determination of placement of the *DataItems* produced by the *Data Source* is the major piece of work associated with this process.

Further decomposition is ended at this point for the general case.

3 Class Design – VLA-specific

The following diagram shows the classes related to a particular set of instances of *TelDataMedia* in the general case, i.e., the *VLAArchiveTape* class. Instances of this hierarchy are different based on *FormatRevision*.

Notable differences between this and that of the general case are related to the *DataRecord* class and its methods and relationships. The functionality of *DataRecord* is represented in the following classes:

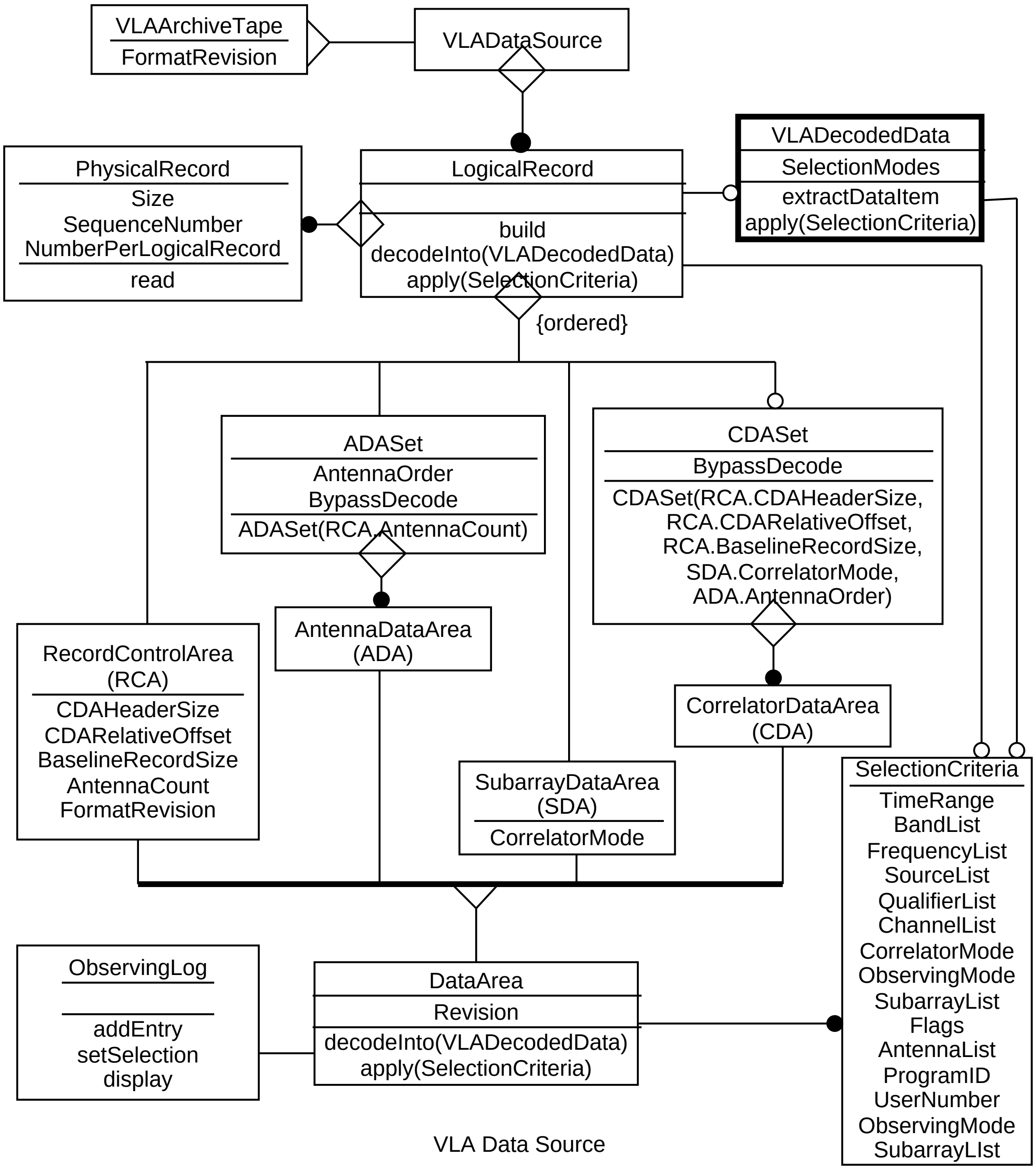
- the *LogicalRecord* class and its associated hierarchy,
- the *PhysicalRecord* class and
- the *VLADecodedData* class and its associated hierarchy (which is shown on another diagram).

The binding of the design to a particular *DataSource* introduces data layering specific to its structure and content.

A *LogicalRecord* on a *VLAArchiveTape* contains all the data relating to a given integration time for an observation. It can span many physical data records. Each *LogicalRecord* is made up of an ordered set of subrecords – *RCA*, *SDA*, *ADA* and *CDA* (the latter two may have multiple instances forming an *ADASet* and *CDASet* respectively). Each of these subrecords is a *DataArea* which can be decoded and can have *SelectionCriteria* applied.

The *Revision* of a particular kind of *DataArea* is defined by the *FormatRevision* of the *VLAArchiveTape* currently found in the *RCA*. Forcing tape and data area revision numbers to be identical dictates the need for a set of *DataArea*s for each *FormatRevision* even though some *DataArea*s may not have changed. If the revision numbers are allowed to differ, however, then any given *DataArea* revision might be found in a number of different tape revisions, thus avoiding a proliferation of instances of a given *DataArea* which has not changed. It is suggested that the latter be implemented, since changes are usually localized to *DataArea*s; in this case, a mapping between *FormatRevision* and *DataArea Revision* must be implemented.

The items explicitly named as attributes of the *DataArea*s in the diagram are those that determine the sizes and structures of other records, and those which determine whether decoding might be bypassed. The number of *ADA* records in an *ADASet*, for example, is based on the *AntennaCount* found in the *RCA*. Therefore, the constructor (*ADASet*) is dependent on *RCA.AntennaCount*.



VLA Data Source

SelectionCriteria which determine whether other *DataArea*s are decoded or not are applied at the *LogicalRecord* level, which in turn are applied to its components. For example, the *TimeRange* criterion is applied at the *LogicalRecord* level, which is then applied to the *SDA*. If the *DateTime* in the *SDA* falls outside the selected *TimeRange*, the *ADASet* and *CDASet* need not be decoded; hence, the capability to set the *BypassDecode* mode in each of these classes. *SelectionCriteria* which can be applied before data is decoded follow:

- time range
- source and source qualifier
- correlator mode
- observing mode
- subarray
- program ID
- user number

The current set of criteria applies to data in the *SDA* only. The design is left general enough to allow selection to be applied to any *DataArea*.

It is also the case that some *SelectionCriteria* are applied after the data is decoded; i.e., selection by:

- channel
- flag
- antenna
- frequency
- band

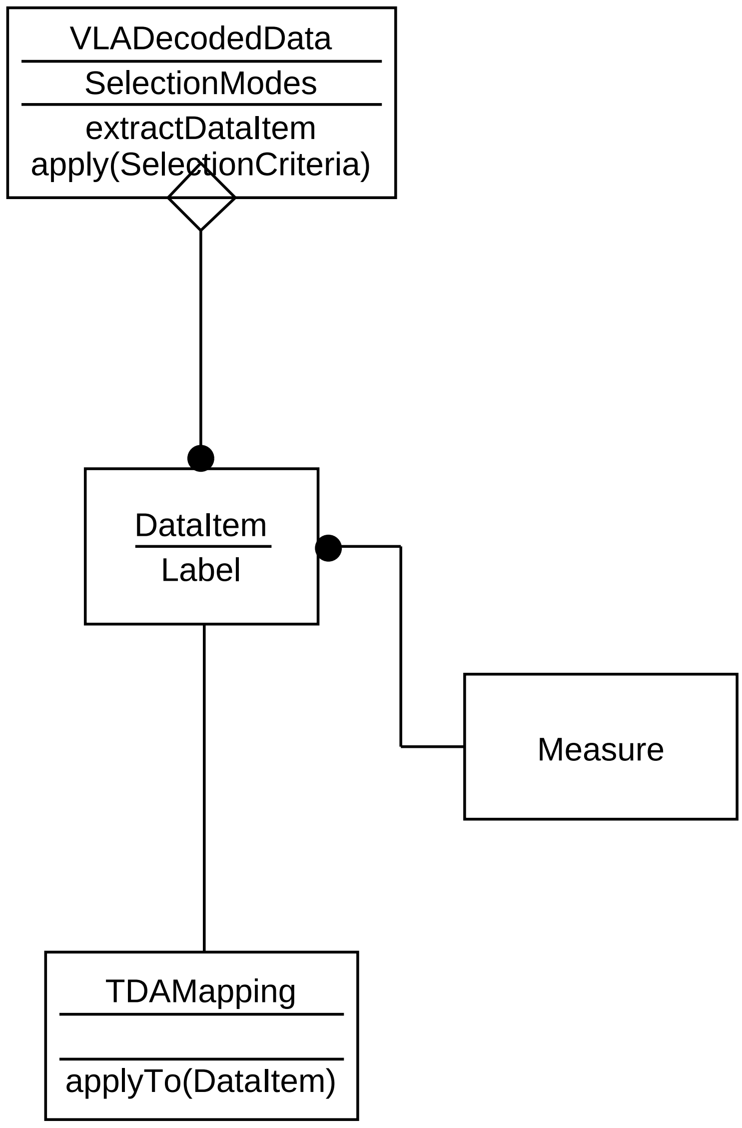
In these cases, the appropriate *SelectionModes* are set in the *DecodedData* area before extraction of *DataItem*s.

All decoded telescope data from a *LogicalRecord* is placed into the *VLADecodedData* area (not necessarily in the same form as the logical record). This method is used instead of one in which each *DataArea* is decoded into its own "decoded instance" for the following reasons:

- The *DataArea* information is not self-contained (e.g., the *DateTime* value in the *SDA* applies to the data in both the *ADASet* and the *CDASet*). In general, a *DataItem* contains entries present in more than one *DataArea*.
- It is desirable to avoid the inclusion of a method to extract each piece of data in a *DataArea* thus avoiding a dependence between the object interface and the form of the data. A goal in this design is to hide this dependence in the *decodeInto* method which is inherited by all *DataArea*s.

This design decision is subject to change, based on the efficiency of implementation.

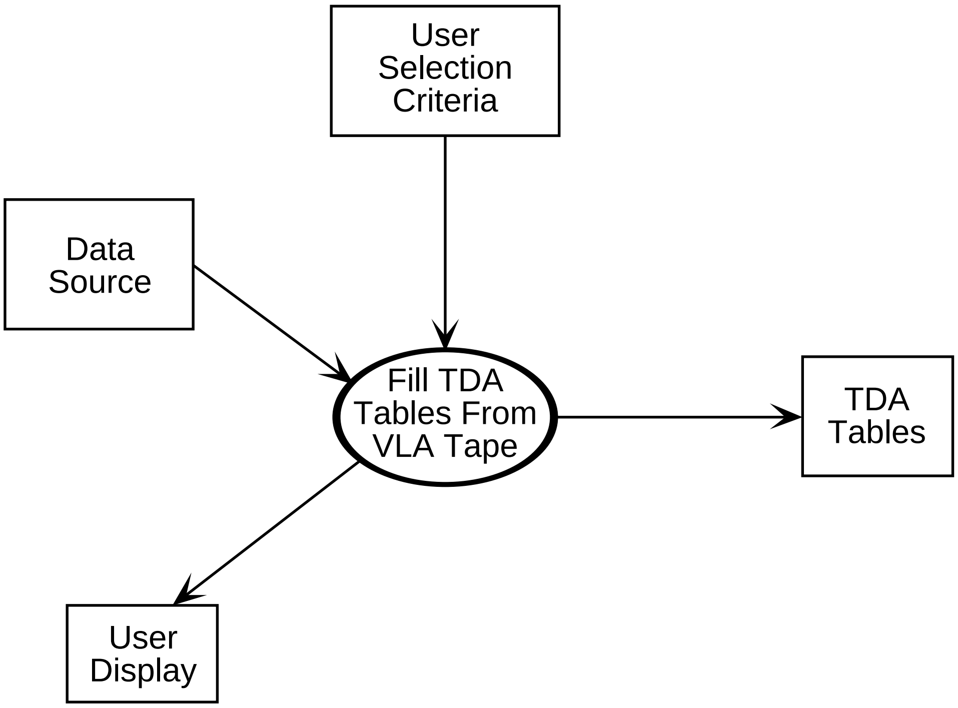
The next diagram specifies the class hierarchy of *VLADecodedData*. The decoded set of data is made of of many "labeled" *DataItems*, each of which has an associated *Measure*, and *TDAMapping*. The relation of *DataItems* to *SelectionCriteria* is not shown in the VLA case, since it is probable that all *SelectionCriteria* have been applied during the decoding process or upon extraction from *VLADecodedData*.



VLA Decoded Data

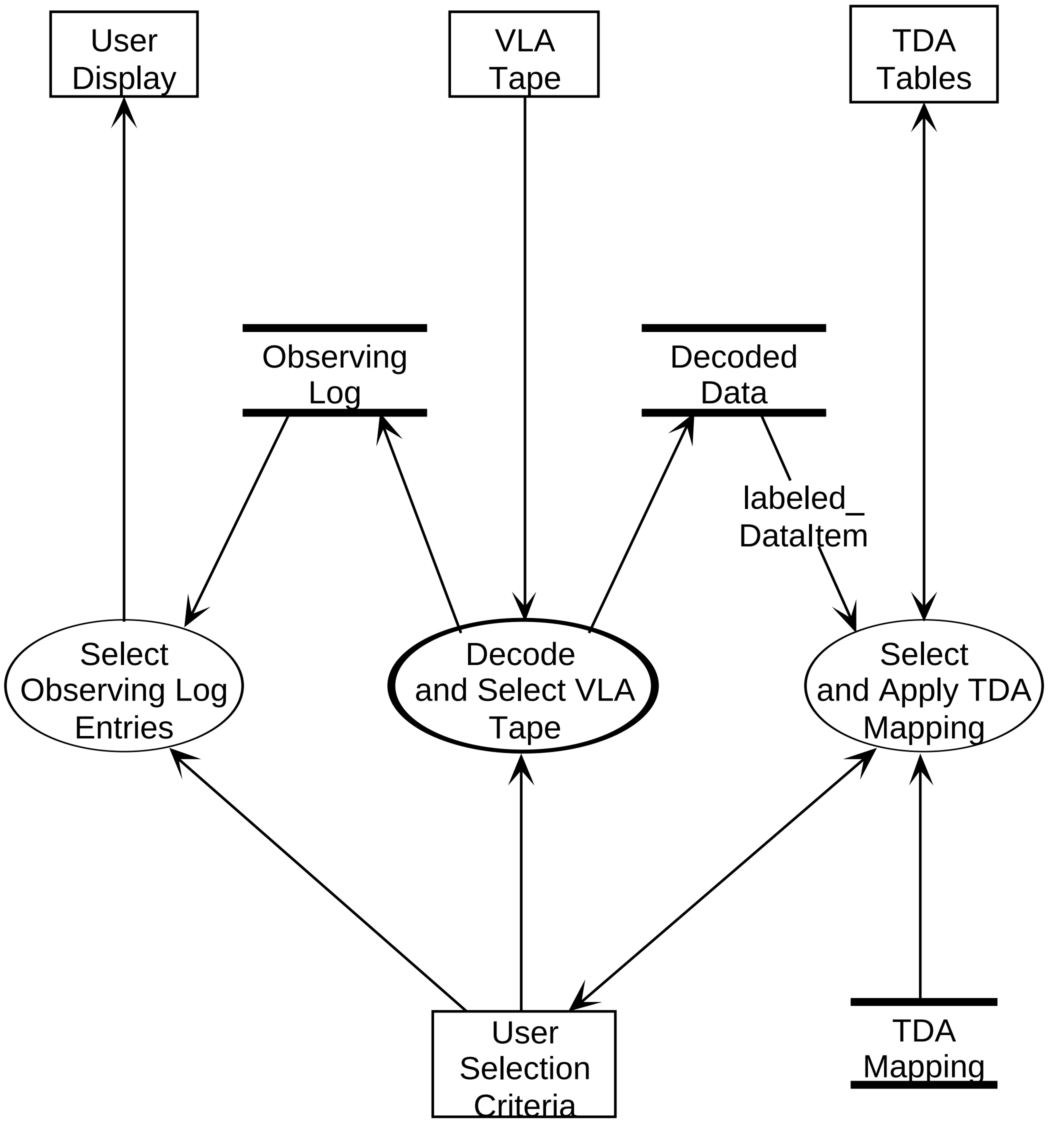
4 Functional Design – VLA-specific

There is no *Operations Log* depicted in the VLA-specific context diagram. For the VLA the log exists on paper; there is yet no automated way of processing it in the system.



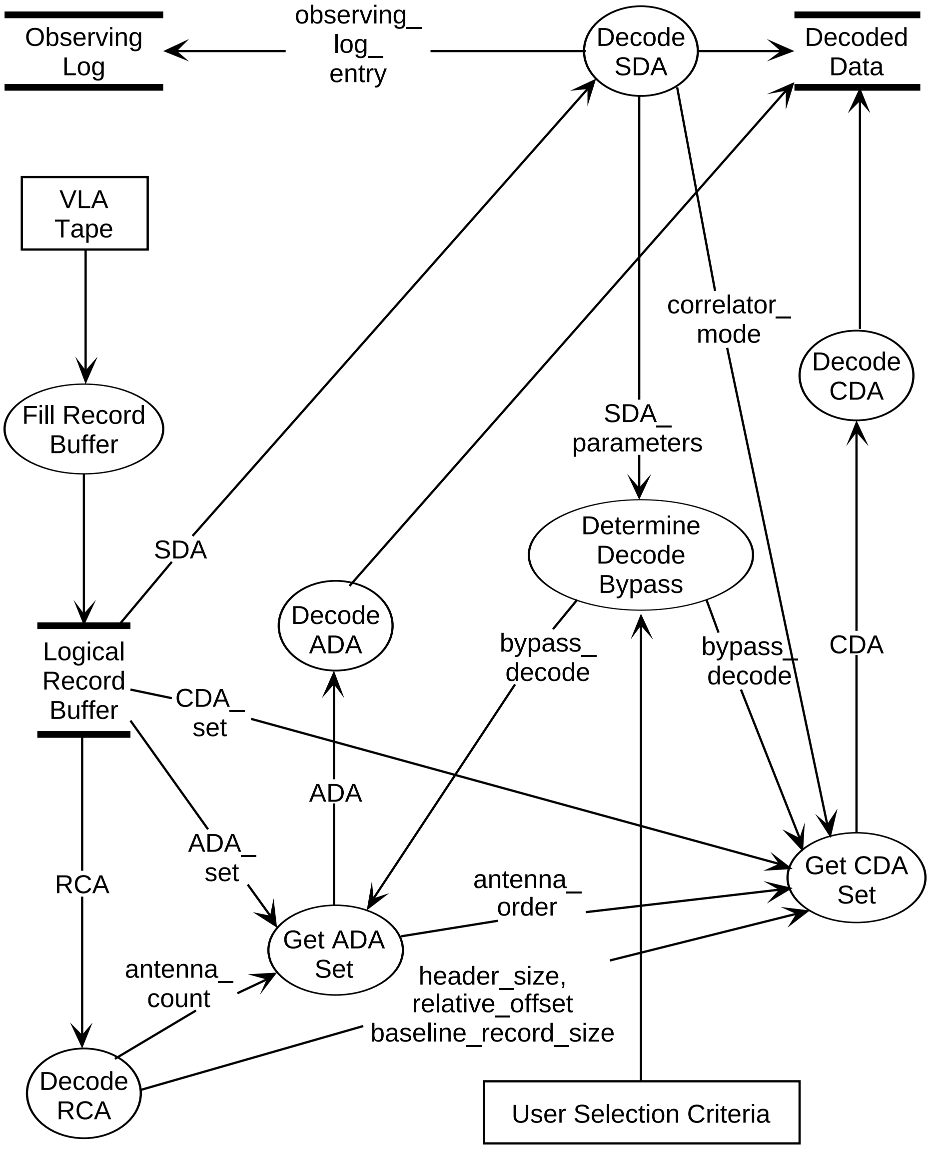
Process VLA Tape

Because the data from the VLA data stream is decoded as well as interpreted, the diagram containing the decomposition of *Fill Tables From VLA Tape* represents the VLA-specific "decoding" process. There are no other differences in the VLA case, save the omission of the *Operations Log*.



Fill TDA Tables From VLA Tape

The *Decode VLA Tape* diagram depicts the interrelationships of the RCA, the SDA, and the ADA and CDA Sets; e.g., the dependence of the CDA Set on parameters from the RCA, SDA and ADA.



Decode and Select VLA Tape

It shows that *bypass_decode* is determined by comparison of *User Selection Criteria* and the following *SDA_parameters*:

- date/time
- source name

- source qualifier
- correlator mode
- observing mode
- subarray ID
- program ID
- user number

While processing the VLA input data stream changes in the following items in the *SDA* generate *observing log entries*:

- correlator mode
- frequency
- source name
- source qualifier
- source position
- observing mode
- program ID

As is shown in this final diagram, the relationships of the VLA-specific data can be localized. It is hoped that the filling of data from other telescope sources can follow a similar design.

Table of Contents