Atacama
Large
Millimeter
Array

ALMA Software

ALMA Project Data Model

Design Document

COMP-70.60.00.00-005-A-DSN

Version: A

Status: Approved
2009-07-22

Prepared By:		
Name(s) and Signature(s)	**Organization**	**Date**
Alan Bridger	UKATC	2009-07-22
Approved By:		
Name and Signature	**Organization**	**Date**
Brian Glendenning xxx	NRAO Xxx	Yyyy-mm-dd yyyy-mm-dd
Released By:		
Name and Signature	**Organization**	**Date**
xxx	Xxx	yyyy-mm-dd

Change Record

Version	Date	Affected Section(s)	Author	Reason/Initiation/Remarks
Adraft	2009-03-27	All	A. Bridger	First Version
A	2009-07-22	1.3,2,4.1,4.2,4.3	A. Bridger	Updated following CDR7 review

Table of contents

1 Purpose

This document describes the purpose and overall structure of the ALMA Project Data Model (APDM). The APDM describes the data structures that are used to capture the information supplied at the proposal submission stage, those required to drive observing with ALMA, and also the structures that track progress against that observing.

The APDM is one of a number of Data Models within the ALMA Software System. Each such Data Model acts as the definition of a Data Interface between two or more of the ALMA Computing subsystems. Its definition and maintenance is under the control of the subsystem responsible for producing instances of the interface. Obviously the definition has to be made in conjunction with the consumers of the data (other subsystems), as would be the case for functional interfaces.

Section 2 of this document summarises some background information about ALMA Data Models.

Section 3 summarises the context of the APDM.

Section 4 describes in more detail the current structure of the APDM, and tries to provide some of the rationale behind the design.

For complete detail of each element and attribute (including documentation) it is best to read the XML schemata (or the UML model), but these are large and difficult to navigate (and therefore understand) in a printable document. Thus for the moment it has been decided not to include in this document the complete schemata. However, these may be found online in the OBSPREP wiki (in the ALMA Computing Wiki). The top level link to these is:
http://almasw.hq.eso.org/almasw/bin/view/OBSPREP/PublishedVersionsOfTheAPDM

Currently the APDM is still changing with each release of the ALMA Software System, this document and the publications found at the link above will be updated with each release. The current schemaVersion of the APDM is “8”, and the ALMA software release to which it relates is R6.1.

This is a work in progress that represents the current state of the ALMA Project Data Model.

1.1 Glossary

The ALMA Software Glossary [Glossary] contains a complete glossary.

Some key terms, although contained in [Glossary] are repeated here for reading clarity, and in addition some new terms peculiar to the APDM or its use are included.

ACS	ALMA Common Software – A software framework for all ALMA software.
BO	Business Object – used in this subsystem to describe domain objects one level up from primitive storage classes
HLA	High Level Analysis – separate ALMA activity coordinating high level design.

OT	Observing Tool (usually specific to ALMA, sometimes generic)
Package	Major Component of Subsystem
SB	Scheduling Block or SchedBlock – unit of observing for ALMA.
Subsystem	Subsystem of the ALMA Software System
UML	Unified Modeling Language – Standard design or modeling language for software
XMI	XMI Metadata Interchange – an openly defined interchange format that UML models can be saved in.
XML	EXtensible Markup Language – Text based language for describing structured documents

1.2 Applicable Documents

[1] [Architecture] , Software Architecture & High-Level Design, ALMA-70.15.00.00-001-J-GEN J.Schwarz et al.

[2] [SSR], Science Software Requirements and Use Cases, ALMA-70.10.00.00-002-L-SPE, Robert Lucas et al.

[3] [ACS] ALMA Common Software Architecture, COMP-70.25.00.00-002-F-DSN, G.Chiozzi et al.

[4] [Versioning] Schema Versioning Proposal, A. Wicenec et al.

1.3 References

[5] [Glossary] ALMA Software Glossary, COMP-70.15.00.00-003-E-GEN, J. Schwarz et al.

[6] [AlmaDataModel], Developing The ALMA Data Model (Draft), J. Schwarz et al.

[7] [ObsPrepDesign], COMP-70.60.00.00-001-N-DSN, ALMA Software Observation Preparation Subsystem Design Document, Alan Bridger et al.

2 Data Model Basics

As mentioned in section 0, the ALMA Project Data Model is one of a number of Data Models in the ALMA Software system. The ALMA standard language used to describe these models is XML, they are all described by XML Schema. However, for some of the models the source for the model is UML. A code generation process then creates the XML Schemata, and software APIs, from this UML source. The APDM is one of these models. It is beyond the scope of this document to go deep into the details of this process, but a brief description of how it applies to the APDM will help provide a basic background for the reader. A full description of the process may be found in [AlmaDataModel], with some further information in [Architecture].

For the ALMA project the source of the APDM is a UML model stored in XMI format in the ALMA CVS repository (in the module ICD/HLA/APDM). The code-generation framework oAW is used to parse this UML model and then create XML Schemata and a Java class library API used by the observing preparation subsystem.

A summary of the process is as follows:

1. The APDM is defined in UML using a subset of UML concepts (the code generation framework does not support all of UML)
2. The Model is saved to the XMI interchange format.
3. The code generator takes the XMI stored model and generates from it a series of XML Schema definitions
4. It also generates a class library in Java that directly reflects the UML structure of the APDM. These are the classes described later as the <element>Data classes.
5. Finally a second code generation framework, castor, takes the XML schemata as input and creates Java binding classes that allow serialization of the java objects created using the above API.

Clearly this process creates two sets of Java APIs that could be used to access the XML structures. The reason for this is partly historical, but also provides the developers with much greater control over the functionality of these classes (specifically the <element>Data classes). The relation of these is not particularly relevant to the aims of this document, but can be found described in [obsPrepDesign]. Briefly the classes generated at stage 4 can be extended straightforwardly allowing the developer to add functionality and intelligence. Object serialization and storage is delegated to the classes derived from the schemata at stage 3. Given time this architecture would be simplified.

The main purpose of this document is to describe the APDM content and its XML schemata.

Supporting models and software provide a variety of “valued added” functions, including standard pairings of values and unit, classes for standard physical quantities, etc.

Two further important ALMA data model concepts should be covered in this background description: The very important concept of the “entity” and the useful concept of the “identifiable part”.

A class in the UML model may be labelled with the <<entity>> stereotype. This instructs the code generation process to make this class a “top level” class, which will receive its own XML Schema definition. This forms the basis for the complete XML documents used in the ALMA system. Entity schemata include special entity structures, which allow the documents to be referenced and uniquely identified. This structure is described further in section 4.1.2. Within an entity further data structures can be created from UML composites (“by value”). Outside the entity only references to it can be made.

The <<identifiablepart>> stereotype may also be used in the UML model. In the schemata this causes the element so stereotyped to receive an id formed from the unique id of the containing entity, plus a further id which may be locally unique within the entity. This allows this element to be used internal to the entity structure "by reference" instead of by value, and also allows it to be referenced directly from outside the entity.

The schema definitions of entity, identifiable parts and entity references are owned by the ACS team and not part of the APDM, but they are described in this document.

Finally each XML schemata is also given a version number during the code generation process. This is inserted in compliance with the ALMA agreed schema-versioning (see [Versioning]).

3 The context of the ALMA Project Data Model

The ALMA Software system works using the concept of an Observing Project. The *Observing Project* is defined in [SSR] and throughout the ALMA software as the top level structure associated with a project resulting from a single observing proposal. In fact the Observing Project is a container for all of the information relating to a project before it begins execution. During and following execution other data associated with the project are created and held in other structures, each of which holds a reference to the original observing project. Of particular interest is the Project Status structure, which holds a record of the execution status of the Project. The ALMA Project Data Model defines the structure of both the Observing Project (ObsProject) and the execution status (ProjectStatus). Note that the ProjectStatus definition is owned by the Scheduling subsystem.

The [SSR] defines the concept of the Scheduling Block (SB or SchedBlock). The SB is the unit of observing for ALMA – any given science observation will usually be broken into many SBs, which may, or may not be executed in sequence – the ALMA Scheduling Software will decide on which SB to execute at any given time on the basis of which is the best to execute *now*. An SB has a canonical execution length of around 30 minutes.

The structure of an SB is defined as part of an ObsProject, and thus also resides in the APDM.

The [SSR] also defines the use of the ALMA Observing Tool (OT) to create both ALMA observing Proposals (Phase I) and observing definitions (Phase II). The observing definitions are ultimately in the form of SBs. Thus the OT is the producer of the information stored in the ObsProject. The Scheduler and Control system then produce the information stored in ProjectStatus. Other parts of the ALMA software system read the information in these structures.

Finally, the [SSR] outlines the use of the OT by *novices* and by *experts* and states that input should be in the form of astronomically based *Science Goals*. Thus the ALMA Observing Preparation subsystem has interpreted this distinction as a requirement to support a *Science Goal* oriented interface to the user where observing definitions are very much in terms of the science definition (the user will not need to know much about ALMA, and in particular will not need to

know about SBs), and a *System View* interface where the hardware, and how ALMA works is very much revealed (i.e. editing is carried out on SBs). A tool for generating SBs from Science Goals is provided.

4 The structure of the ALMA Project Data Model

The APDM defines the structure of the Observing Project, as an ALMA "entity" called *ObsProject*. It also defines a parallel structure called "*ProjectStatus*" (also an "entity"), which is built up with execution status information about the project as observing continues. Other entities, the *ObsProposal*, *ObsReview* and *SchedBlock*, are also defined as constituent parts of an ObsProject. One non-entity schema is generated from the APDM, *ValueTypes*, which holds the definition of various physical quantites whose use is shared by the other schemata. The entities are described separately below. The description given here is not intended to be complete, but to describe the rationale behind the overall structure. A complete description of each element, at this stage in development, is given in the URL provided in section 0 (and the Appendix).

4.1 Fundamental Definitions

Before describing the APDM structure itself a brief description of common ALMA elements used within it should be given.

4.1.1 The Schema Version

The ALMA Schema Versioning approach is described in [Versioning]. Briefly the key features are:

All potential root elements of a schema are defined with the pair of attributes schemaVersion and revision; they are mandatory to enforce the possibility to assess backward compatibility when resolving the schema location based on the information given at the instance level.

Thus at the instance level one must have e.g.:

```
<?xml version="1.0" encoding="UTF-8"?>
<RootElement ... xvers:schemaVersion="1"
xvers:revision="1.12">
...
</RootElement>
```

The first member of the pair, an integer number, here 1, must be modified every time the schema is modified with non-backward compatibility. This is the major version number. It is the responsibility of the author of the schema to state whether a modification made to the schema is backward compatible or not. Either software reading XML instance

documents must make provision for different versions, or the documents themselves must be migrated to the version required by the software. The ALMA OT takes the second approach, using xslt scripts to migrate instance documents.

The second member of the pair is (for ALMA schema) the CVS revision, here 1.12. This number is set automatically from the CVS revision number of the APDM UML model. Changes to this value should not affect compatibility.

In addition every schema has a pair of version numbers assigned via the version attribute in the root element of a schema. The first of this pair is the schemaVersion as described above and the second is the CVS revision number.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema ... version="1 1.12">
  ...
</schema>
```

A word on compatibility: In general most changes to the model will not be backwards compatible, so any change should start with this assumption. Deletion, moving or renaming of items are clearly incompatible changes, and the major schema version must be changed. Addition of *optional* new items is a compatible change, the addition of required new items is not. Changes to documentation of items in the model is a compatible change, as are other non-functional changes, such as changes to diagrams illustrating the underlying model.

4.1.2 The Entity

As noted above, the concept of an entity is important to ALMA. All “entity” documents contain the common schema definition for both the Entity and the identifiable part (see below).

For practical purposes this means that all instance documents will have an element called <RootElementEntity> with several attributes the key one of which is EntityId. This is a globally unique String (obtained before storage to the Archive) by which this instance document can be identified and referenced from other documents, and retrieved from the Archive via simple queries.

4.1.3 The Identifiable Part

It is sometimes necessary to be able to uniquely find and reference an non-entity element (subnode) inside an entity document, either from within the same document, or from another document entirely. This is achieved by the addition of the attribute EntityPartId to the element. This is similar to the EntityId, but only needs to be unique inside the document.

4.1.4 The Entity and identifiable part Reference

To support the referencing of entities and identifiable parts from other documents, or in the case of the identifiable part, from the same document the EntityRef element is defined. This element has attributes identifying the entityId of the referenced document, and optionally, for the identifiable part case, the entityPartId of the subnode inside it.

4.2 The ObsProject Entity

When a potential ALMA observer uses the ALMA OT to create an observing proposal with the intent to submit it to ALMA the OT also creates an ObsProject. This is an important point: the ALMA Observing Tool operates on ObsProjects. It is particularly important during Phase II - the OT does not provide a mechanism to work on Scheduling Blocks (or other observing entities) other than as part of an ObsProject. The ALMA system as a whole requires that each SB has an associated project, so the restriction that the OT only deals with ObsProjects works towards ensuring that.

In order to hold all of the pre-observing information associated with a project the ObsProject entity class is composed of three parts: an ObsProposal, an ObsReview and an ObsProgram. These hold the information associated with the three phases of observing preparation: the proposal or Phase I, the reviewing and resultant approvals, and the observing program definition, or Phase II. *Note: the inclusion of ObsReview here, and its detailed structure, is in debate – this may change.*

Of these three structures, the ObsProposal and ObsReview are implemented as separate entities - only the ObsProgram is held with the ObsProject. This is because eventually the ObsProgram becomes the essential component of the project, and this provides ease of access for execution. The ObsProposal and ObsReview are separate entities that are actively used only during the appropriate phase.

The top level structure is shown in Figure 1 as a UML class diagram.

.

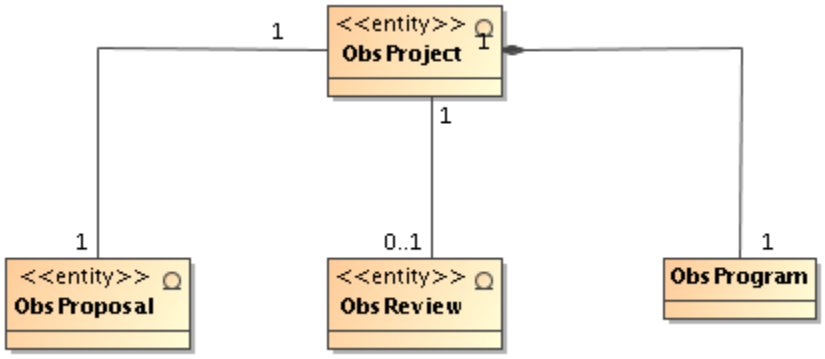

Figure 1. ***The top level structure of an Observing Project.***

Note: UML class diagrams are used throughout this document to display structures in the APDM,. For ALMA the UML model is the ALMA Project Data Model, and the derived XML Schemata are the standard implementation for persistence and information exchange. Diagrams reflecting the XML Schemata may be found in the web link mentioned in section 1, and some are provided in the Appendix.

4.2.1 The ObsProgram

As noted above the Observing Program (ObsProgram) part of the Observing Project holds all the information created during Phase II, or program preparation.
The ObsProgram consists of an *Observing Plan* (ObsPlan) and a *SciencePlan*. The Science Plan covers the *science goal* oriented view of the observing, and is intended to contain the information that most users will use to define their observing programs. This information will persist, for the convenience of users, but will not be the definition that is used by the Observatory when carrying out the observations. Instead the latter information is contained within the ObsPlan, and covers what we call the system view. The service of Program Generation (covered in [obsPrepDesign]) allows users to create the Observing Plan from their Science Plan. It is also possible for very advanced users to ignore the Science Plan and simply create all of the Observing Plan directly.

As stated, the *Observing Plan* (ObsPlan) is the container for defining the actual observing objects. In modeling terms the ObsPlan is a role for the top level *ObsUnitSet*, heading the definition of the system view. An ObsUnitSet contains a collection of Scheduling Blocks or more ObsUnitSets, along with objects defining the preconditions, performance and calibration requirements, and flow control that apply to that collection.

In the model the Scheduling Block and ObsUnitSet are implemented in terms of an abstract class, the *ObsUnit.* A Scheduling Block is a type of ObsUnit. An ObsUnitSet is a type of ObsUnit that contains either other ObsUnitSets or SBs. The SBs represent the

end points of structures and contain the information required to execute a unit of observing. This recursive composite pattern allows arbitrary hierarchies to be built up and is thus very powerful. However, it is expected that in practice most of these hierarchies will be shallow. Apart from acting as a "container" for other ObsUnits the ObsUnitSet provides two other key functions:

1. It holds a structure for describing execution dependencies between the ObsUnits it contains. This allows constraints like "execute this only when these have finished", or "execute this when this or this have finished" and so on. This structure is not yet agreed with Scheduling and so is not well defined.
2. It holds the standard data processing script to be executed on the data resulting from the SBs contained within. In other words the ObsUnitSet is the trigger for data processing, *not* the SB (except for quick look processing). When all of the ObsUnits contained in an ObsUnitSet complete the science pipeline is initiated.

The SB is a separate entity, so an ObsUnitSet actually holds references to the SB entities it contains.

The Science Plan consists of a series of *Science Goals*. There can be different types of Science Goal (there is an abstract superclass) however, most true science observing will probably be implemented by one type of science goal and other types represent different forms of observatory calibration goals. The science goal is subdivided into sections designed to collect the information necessary for observing, broadly speaking: Spatial setup, Spectral setup, calibration needs and scheduling constraints. Each Science Goal is linked (by a partid reference) to an ObsUnitSet. This link is established when *Program Generation* is executed thus one science goal will create one ObsUnitSet (which may of course contain just one SB or many, and potentially a hierarchy).

Figure 2 shows the structure of the ObsProgram, demonstrating the relationship between its parts, including the recursive nature of the ObsUnitSet and SchedBlock composition. This diagram also shows the reference between a Science Goal and the ObsUnitSet it generates.

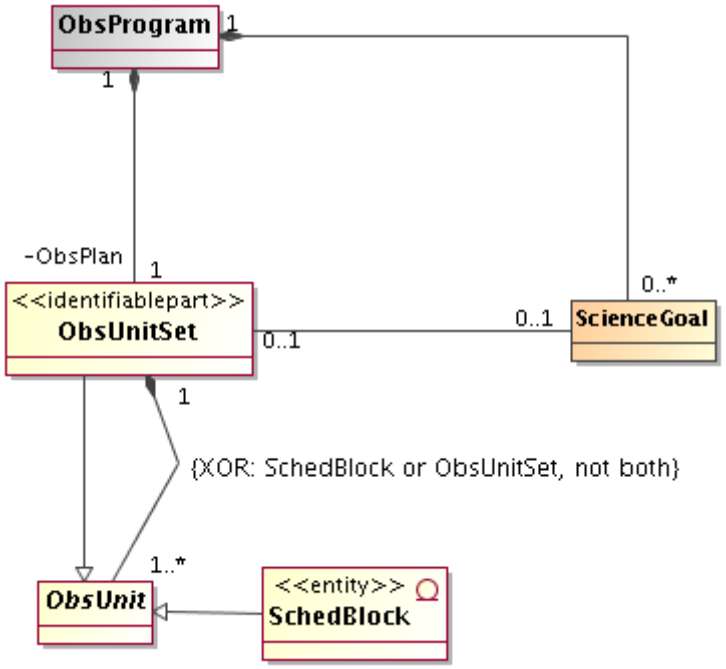

Figure 2. ***Top level items in the Observing Program***

Note that for clarity this diagram omits some of the details of implementation. In particular it omits the fact that ObsProgram is a subclass of an abstract class called ObsPhase. The same is true of ObsProposal. The reason for this is so that these two structures, representing Phase II and Phase I (respectively) of observing preparation share the Science Plan and Observing Plan structures. This abstraction has no effect on instance documents, but is found in the schemata.

The information contained within a Science Goal can be summarised as:

Spatial Setup: A definition of the region(s) of the sky to be mapped. Also includes target properties (coordinates, velocity, etc.)
Spectral Setup: A definition of a series of spectral windows to observe, and of the user information required to setup the correlator (bandwidth, resolution, polarization).
Performance Goals: The required sensitivity, angular resolutions and other performance goals.
TemporalSetup (Optional): For defining time constrained observing.
Calibration Setup: The definition of calibration requirements (goals). Currently this is not well defined and only the option of allowing the system (either the OT or the observing system) to choose for the user is allowed.

Figure 3 shows the current version of a Science Goal in some detail.

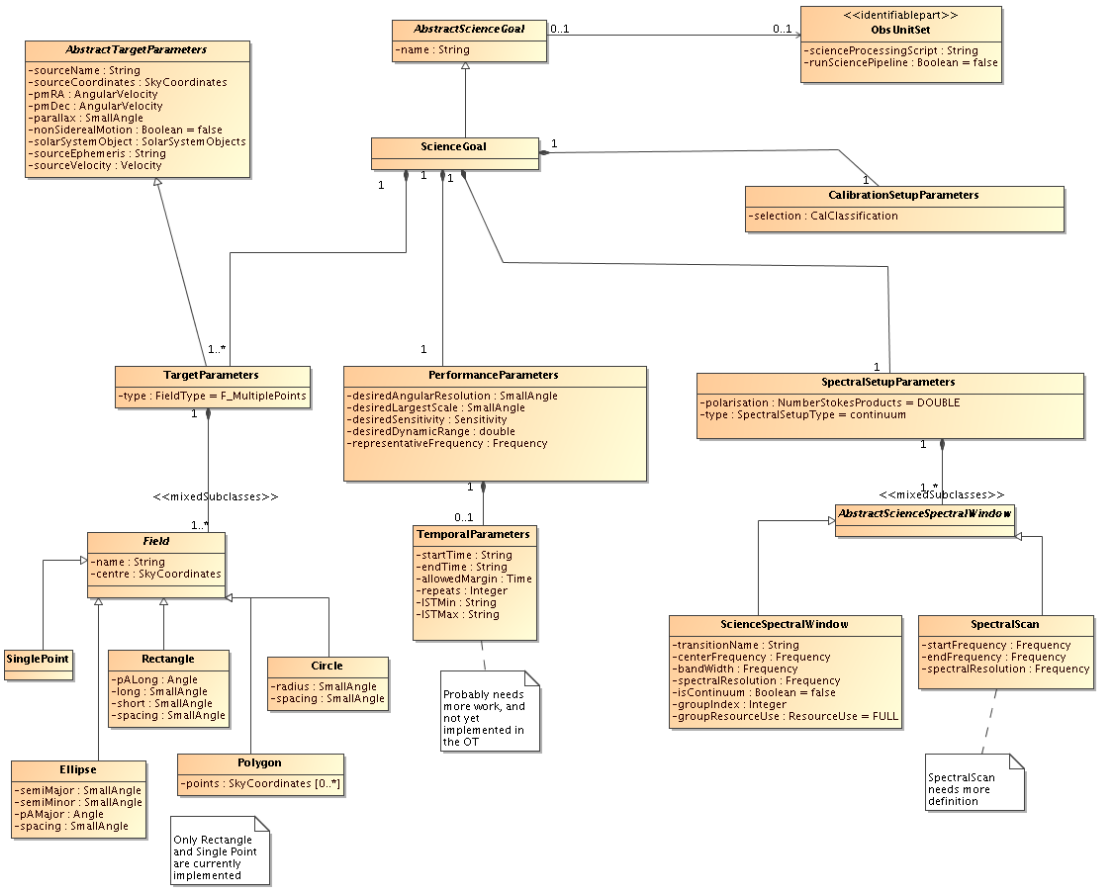

Figure 3. ***Details of the Science Goal.***

One aspect of note in this science goal definition is that it allows for *one* spectral setup (potentially of many windows), but many spatial setups (called target parameters), each of which may be a different type of map. This allows for the setup of many targets all using the same spectral setup very simply. This of course makes the assumption that the performance goals for each will be the same. Currently the ALMA OT implementation is restricted to many targets of a single pointing, or one target of a rectangular map.

4.3 The SchedBlock Entity

A *Scheduling Block* (SB) is the unit of observing for ALMA. It defines all the information that is required by the Observatory to independently execute the acquisition of a set of data and then to calibrate it as well. Since SBs are selected for execution individually they must exist as separate items in the archive and thus the SB is defined as

an ALMA entity. In general we do not expect most novice ALMA users to be exposed to the structures in an SB - only very advanced users should edit at the SB level. Changes at this level can cause inconsistencies in the definition, and may cause scheduling problems. Thus the user must know what he or she is doing to edit at this level.

Most of the structures within the SB hold information that is used either by the scheduler, to query whether or not this SB is suitable for scheduling "now", or as information that is used by the Observing Script (in the Control subsystem) to actually execute the observing (see below), and in some cases both. There are also performance goals for the SBs, to be measured against by the observing system.

So to allow the determination of "Schedulability" we find items like WeatherConstraints, a precondition that requires a current Baseline Calibration, and a ScientificPriority.

Perhaps the key element of the SB is contained within one String attribute called obsProcScript (in ObsProcedure). The value of this attribute is a python script that will execute the observing sequence. This will normally be a control system command that implements a standard observing mode. It may on occasion be a complete, executable, script that has been edited by an "expert". In the ALMA OT the script name is set automatically once a standard observing mode is chosen, but user entered values, including complete scripts, are permitted if the user has sufficient privilege.

This python script has access to all of the other items in the SB. This is how the script will know where to point, how to scan and how to configure the receiver and correlator.

The other parts of an SB definition are a series of *Targets*.

4.3.1 The Target

The target is the element within an SB that defines the area to map, the spectral or instrumental setups to use, and the purpose of the observing. In fact the target element itself contains nothing except references to other elements (via a partid) that contain descriptions of these three key components. The reason for this approach is that being able to reuse these individual components among many targets was regarded as desirable. Referencing them rather than containing them makes this straightforward. This structure is shown in Figure 4.

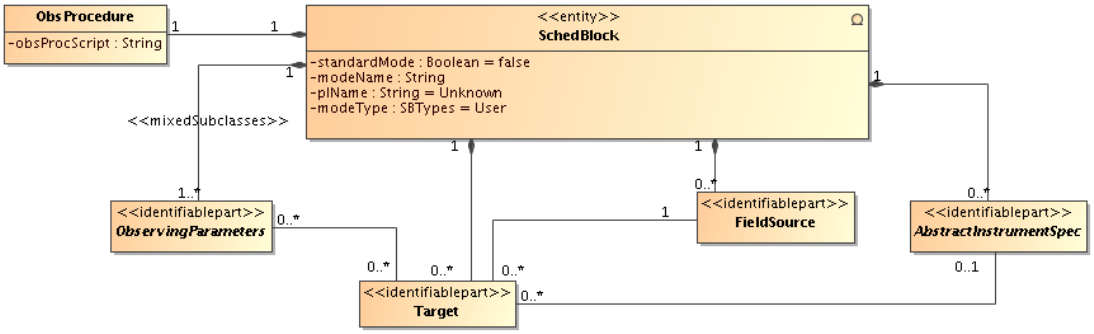

Figure 4. ***The Target definition***

The three components that make up a target are:

The FieldSource: This describes the mapping area on the sky, and the properties of the target of interest.
The InstrumentSpec: This describes the setup of the "instrument". For most purposes this amounts to a description of the front and back-end setups, and the correlator setup. However, an alternative for the instrument spec is also an optical camera setup.
A set of Observing Parameters: These represent the "intent" or purpose of the observing. This may be a science target or some sort of calibration target. More than one is allowed to support the concept that the observation of a target may have, or be used for, more than one purpose.

This concept of a target is close to the concept of a "Scan Description".

How all these targets are used in the observing is determined entirely within the python script. This provides the script writer with a lot of flexibility, but in general we expect that standard modes (with standard scripts) will be used and that these are likely to be relatively conservative – they are unlikely to require the definition of more than a few targets. In general observers will not be script writers.

One of the Targets is identified in the SB as being a *RepresentativeTarget* and is used by the observing system to monitor progress against the *PerformanceGoals* identified for this SB.

4.4 The ObsProposal Entity

This entity contains the information necessary for the Phase I process of the Observatory.

During phase I the user will define an outline of their science goals using the Science Plan – these science goals are is identical in structure to those found in the ObsProgram. In principle it is also possible for them to create an Observing Plan, using the system side definition (ObsUnitSets and SBs), again much the same as in the ObsProgram. However in the Phase I state this is not required and should be rare.

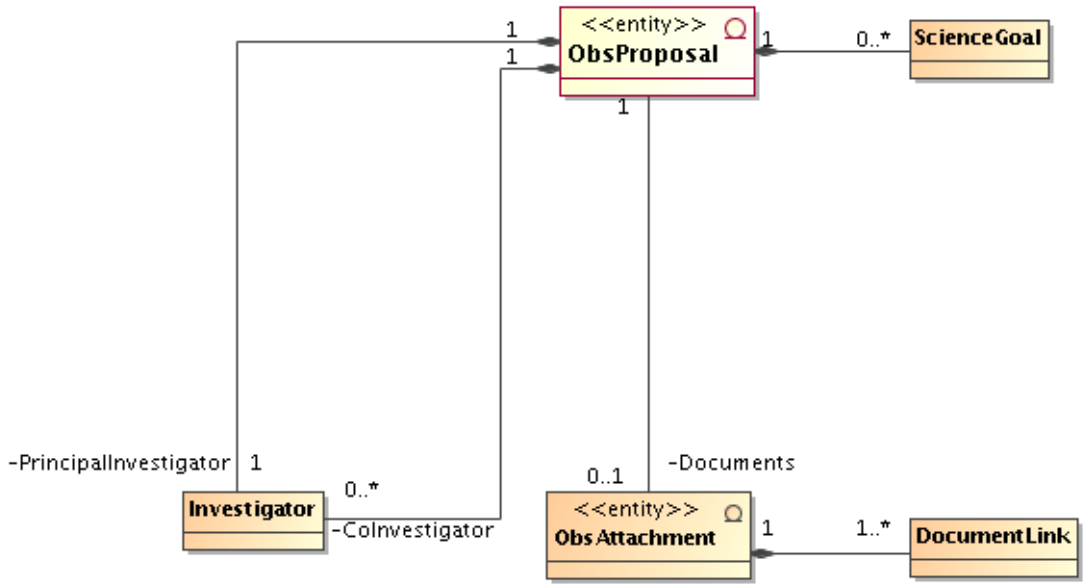

Figure 5. The Structure of the Observing Proposal

Figure 5 shows the main structure of the ObsProposal. Note, again this diagram omits details.

The detailed content of the observing proposal for ALMA is still in discussion. Thus the details in the schema must be regarded as a work in progress. The overall structure shown in Figure 5 should remain, and may be expanded depending on the result of ongoing discussions. In particular some of the detailed items that are currently attributes of the ObsProposal itself may be expanded into more complex elements. A further possibility is that a feedback structure may be added – see the next section.

4.4.1 Proposal Feedback

A normally required aspect of an observing proposal is a section covering resources required. In ALMA a user proposes for a given sensitivity, and so estimated integration time required is not a user input. In addition since the aim is to make life simple for the user, the software does not capture from the user other resources like antenna configurations required, receivers and correlator modes. Instead this information is derived from the science goal input and then presented as feedback, to the user and to the

Review Committee. As derived information these data do not appear in the APDM, because they do not need to persist. However, an element to hold them is being considered as it would simplify searches.

4.5 The ObsReview Entity

Work on defining this entity has not started in earnest. As noted earlier alternative designs may even be considered. It is also likely that this entity will be owned by the Observatory Operations subsystem.

4.6 The ProjectStatus Entity

Not considered in this document at this stage. This entity is owned by the Scheduling subsystem.

4.7 Other Schemata

Two other schemata are produced by the generation from the APDM:

4.7.1 The ObsAttachment Entity

This entity is used to provide an XML “wrapper” for binary attachments. At present it is used by the ObsProposal schema to attach binary files to an observing proposal. The binary documents themselves are attached as multi-part mime documents. The structure is shown in Figure 5.

4.7.2 The ValueTypes Schema

This schema is used to provide some fundamental physical quantities to the other schemata, for instance value-unit parings (e.g. a Frequency, with a value and unit), and other complex concepts such as SkyCoordinates.

4.8 Remaining Issues and Future Work

The APDM remains a work in progress. Despite this instances are in real use by the project for testing and verification, and will soon be used for commissioning. In particular ObsProjects and SchedBlocks have seen extensive use. Changes to these entities must be carefully controlled.

None the less a number of changes are foreseen and others under discussion. Here a brief description of the key remaining issues for each entity is listed.

4.8.1 ObsProject Issues

Items held at the top level of the ObsProject may be modified now that the interface of Projects with Operations software is being discussed.

- The value of PI here may be modified in response to operational needs.
- Version may be removed or use restricted.
- The Project Name may be merged with Title from ObsProposal

Other changes may also result.

The Science Goal definition is held within ObsProject. Changes here may include:

- Addition of a Source Flux
- Modification of the Rectangle definition to replace use of “long” and “short”
- Implementation of other mapping shapes
- Definition of the spectral scan setup.
- Possible changes to the temporal setup.

These changes would affect the use of science goals during both Phases I and II.

The ObsUnitSet definition is also held within ObsProject. Significant changes are expected here when dynamic scheduling is being defined in detail and implemented by Scheduling, as those in place are based on an old design that may change. There are also changes planned to provide an interface to the Pipeline reduction system. Two particular items known to need changes are:

- The implementation of dependency and ordering between ObsUnits, as part of the interface with Scheduling.
- The inclusion of user-settable parameters to pass to Pipeline to provide some level of control over the data reduction process.

4.8.2 The SchedBlock Entity

This entity is comparatively stable. However, once more the interface with dynamic scheduling will probably cause changes as the current items are based on an old design

(they are not used in the current system). In addition some particular observing modes have not been fully detailed, and require additions to the current definitions. Finally changes are still likely as more complex observing is tried. A known example of this is the location of the subscan duration item.

Changes expected:

- Interface to scheduling may change
- There is no support for describing nutator setups. This is awaiting full definition.
- There is limited support for describing frequency switching (presently limited to that the correlator requires). This is awaiting full definition.
- The subscan duration (found in the SpectralSpec) is not in an optimum location. A resolution to this requires discussion with the Control and Correlator subsystems.

4.8.3 The ObsProposal Entity

This is currently under major discussion with other subsystems, the Operations team and the Regional Centres. Thus many items may change. Known and likely changes so far are:

- Removal of the PI and CoI “placeholder” items in ObsProposal (these have been replaced by the Investigator element).
- Merging of title with Project Name in ObsProject.
- Probable removal of support required, staff contact and contact person
- Changing of the allowed items in Scientific Category and Proposal Type.

However, many more changes should be expected. A requirements review in June 2009 should resolve many of these issues.

The addition of a “Feedback” element is under consideration. This would provide persistence for “derived values” from the proposal’s science goals. It would probably contain items such as:

- Antenna configurations required
- Receivers required
- Weather conditions required
- Estimated observing time required
- Correlator modes required
- Expected data rates and volumes.

4.8.4 Other Issues

There is a small implementation issue where xsd:string items that are choices from an enumerated list are *attributes* of the enclosing element, whereas all other xsd:string items are elements. This difference occurs for backwards compatibility to when the schemata were hand-written. For anyone using binding classes to access instance documents this should not be an issue. It may be addressed in the future if there is a need and if the effects on users of the APDM are small.

Many elements show as <elementName>T in the schemata. This is also an historical artifact of the code-generation. Changing this would have a large effect on the users of the APDM.

Selected Schemata Diagrams

Some selected schema diagrams are provided here for reference. For full details see http://almasw.hq.eso.org/almasw/bin/view/OBSPREP/PublishedVersionsOfTheAPDM

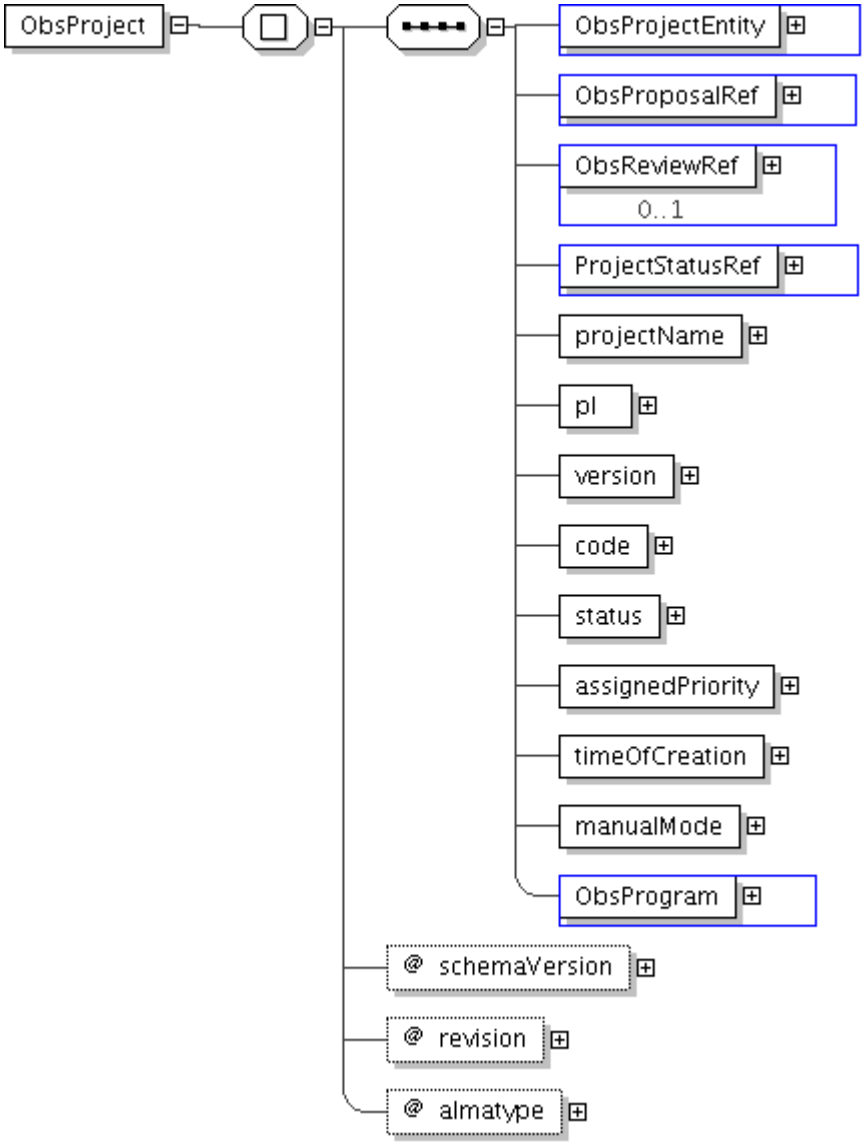

Figure 6. ***ObsProject schema***

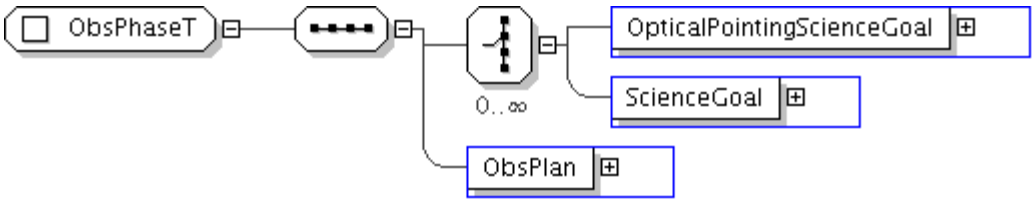

Figure 7. ***ObsPhase structure (this applies to both ObsProgram and ObsProposal)***

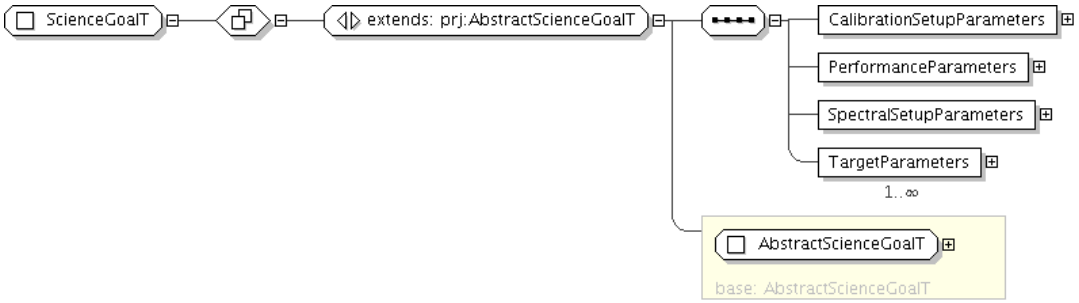

Figure 8. ***Science Goal structure***

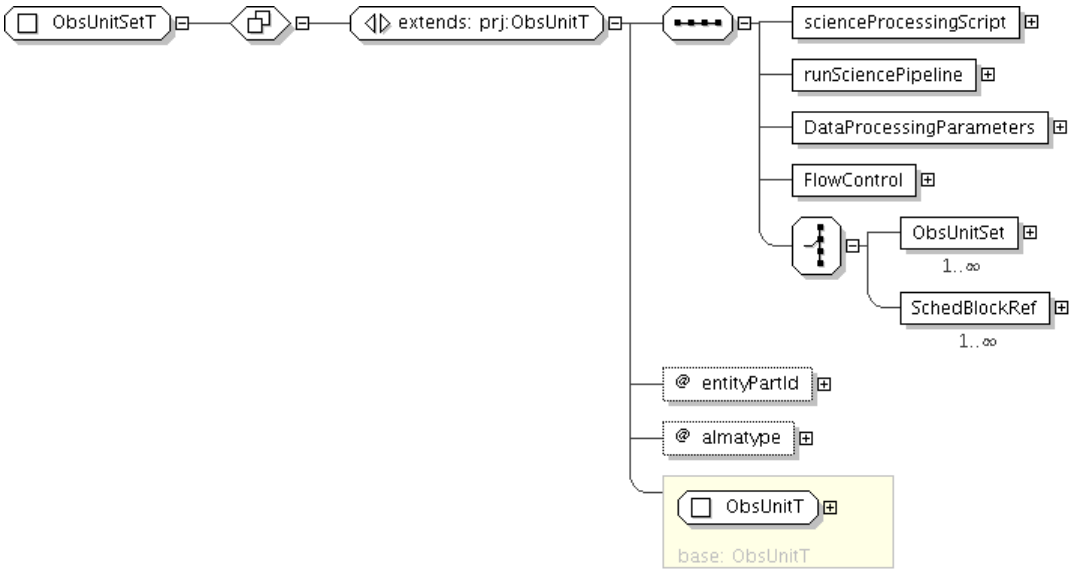

Figure 9. ***ObsUnitSet, showing relationship to SchedBlocks and other ObsUnitSets***

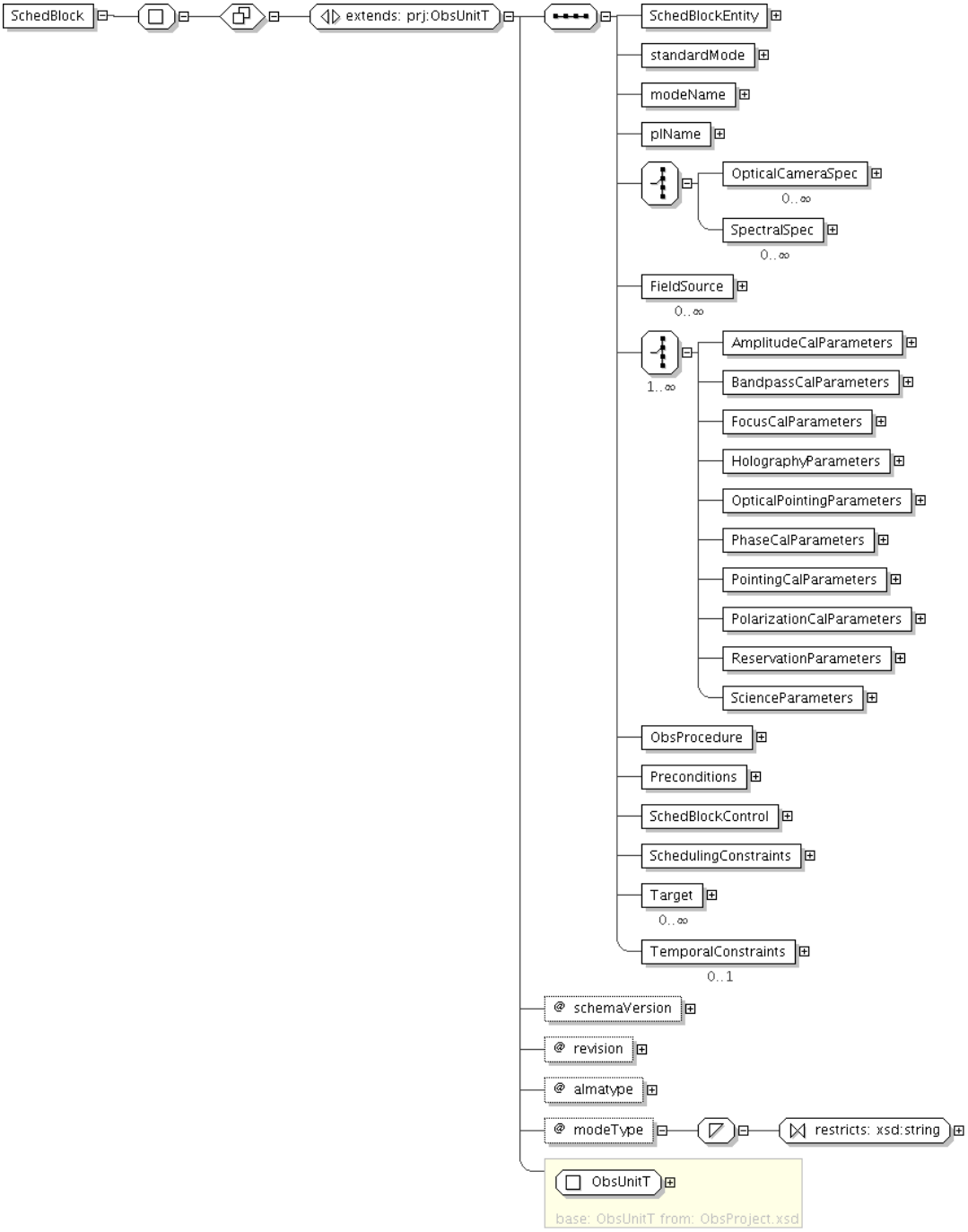

Figure 10. ***The SchedBlock schema***

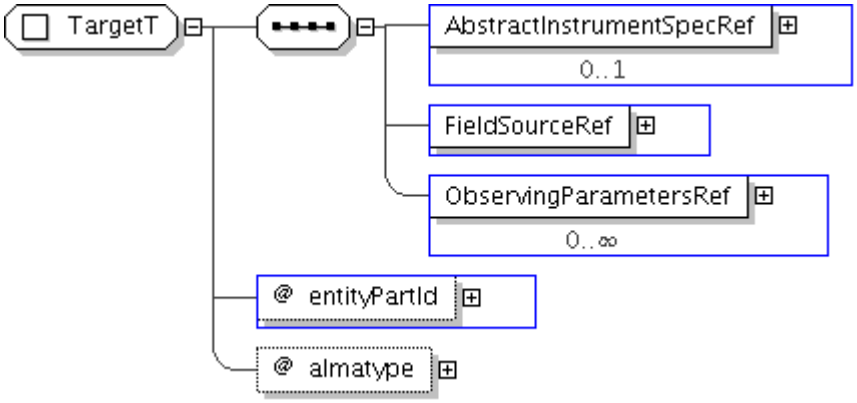

Figure 11. ***Structure of a Target***

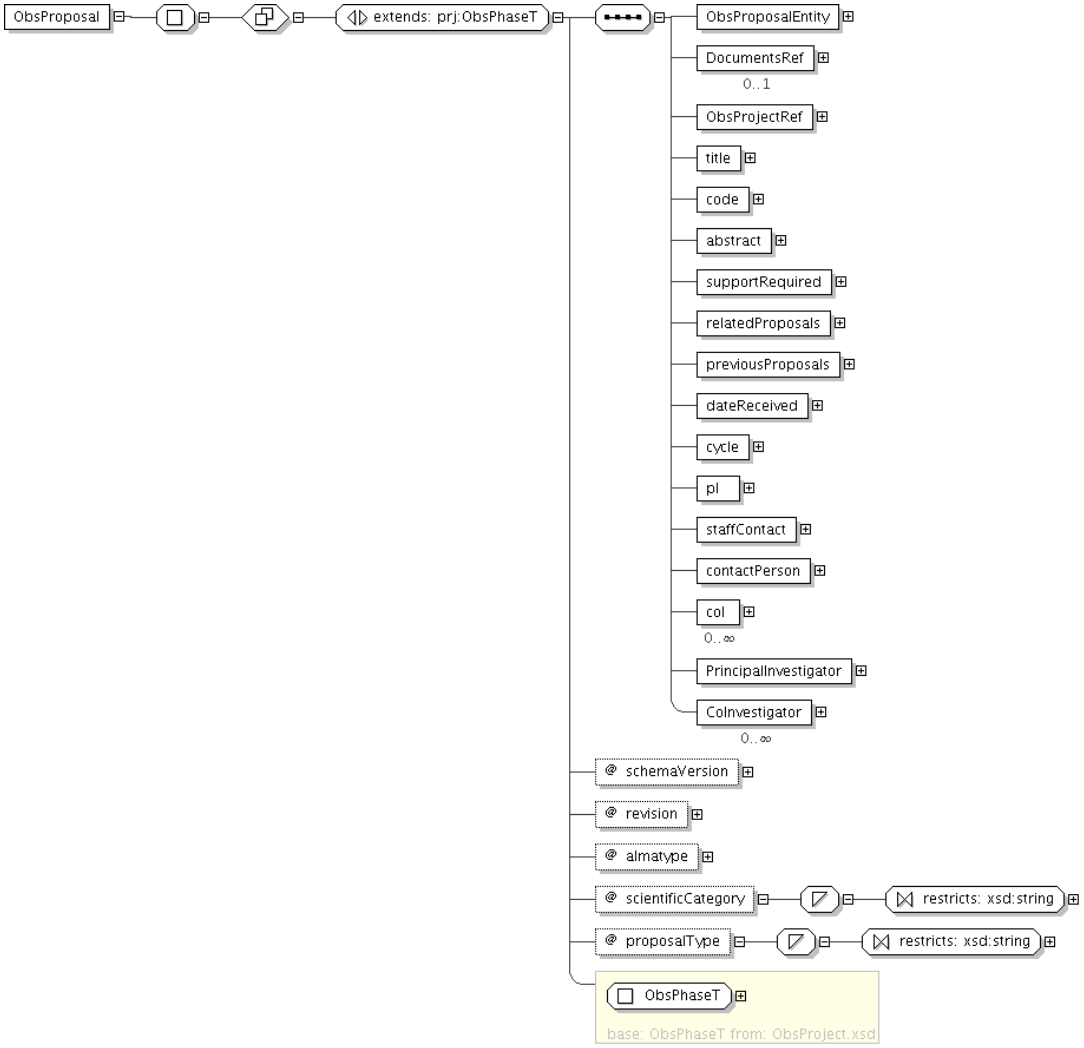

Figure 12. ***The ObsProposal Schema***

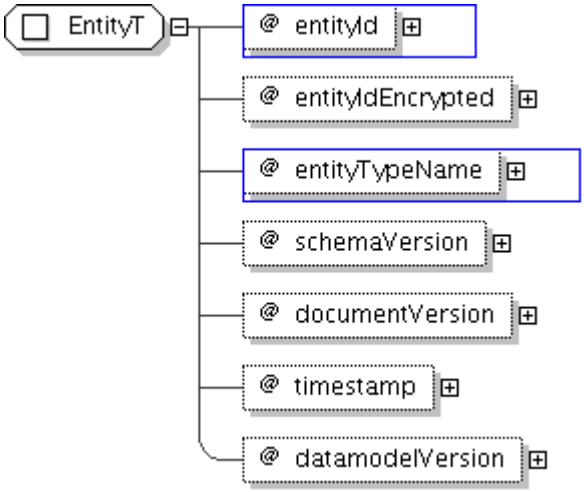

Figure 13. ***The Entity Schema***

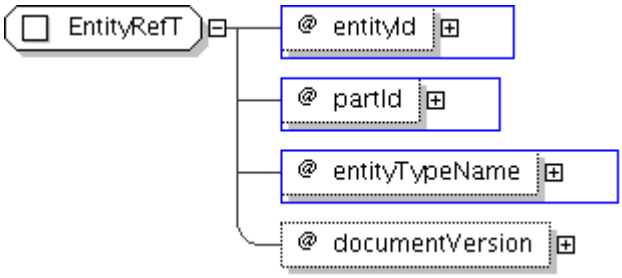

Figure 14. ***The Entity Ref schema***