Mission Planning Software Design
5 December 2006

Bryson Lee
blee@slac.stanford.edu
650-926-2866

Purpose and Definitions

- Mission Planning == Process by which the LAT team operates the instrument via the MOC.
- Recall the three week cycle:
  - GSSC defines “preliminary science timeline” (where to look)
  - MOC uses pointing profile to schedule TDRSS contacts
  - IOC’s define instrument timelines and real-time operations requests
  - MOC uploads the final ATS definition and starts it running
- All operations must be specifiable by in terms of three types of text files defined in the Operations Data Products ICD:
  - Timeline Files: define commands to be placed in the onboard Automated Time Sequence (ATS)
  - Upload-definition Files: provide the content of files to be uploaded to the onboard file system
  - PROC requests: instruct MOC personnel to execute pre-defined scripts during real-time contacts.

Mission Planning Software Functions

- **Scheduling service: Planners (people) define various activities to be performed at particular times via the Mission Planning GUI application.**
- **Translation service: Mission Planning software maps "logical" activities into specific output product instances, for example:**
  - **Physics Data Acquisition → ATS commands**
  - **Charge Injection Calibration → ATS commands and/or PROC requests**
  - **Configuration Changes → Upload-definition files + PROC requests**
- **Bookkeeping service: As data products are received, instrument responses / output are reconciled with requested activities.**

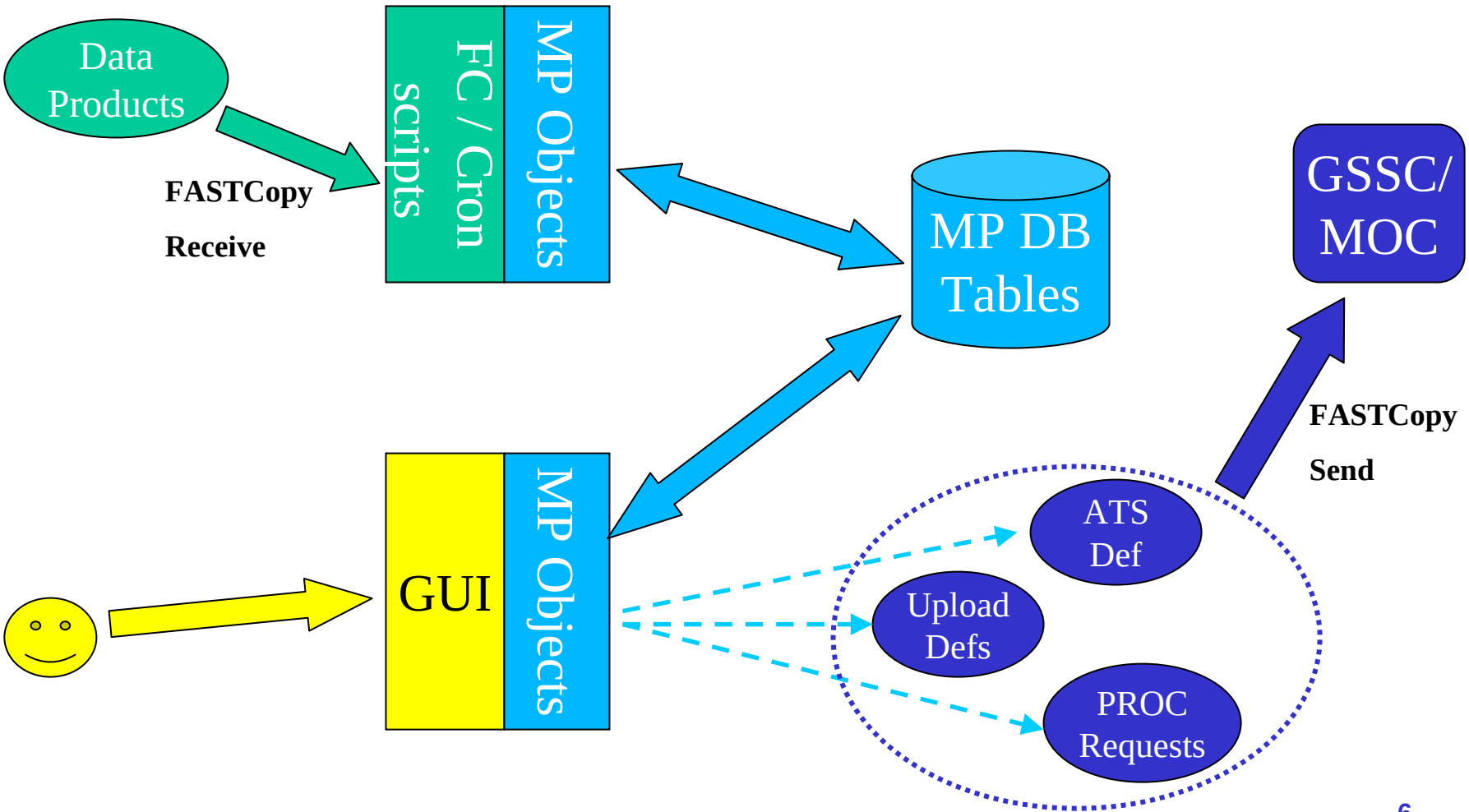
Activity Planning

- Operator (planner) schedules activities based on constraints of contacts, orbital events, science timeline.
- "Business rules" in planning software insert information into appropriate lower-level tables:
  - Acquisitions insert start/stop commands into command tables.
    - MOOT provides the keys to request specific onboard configurations.
    - FMX resolves logical keys to onboard file names for use in forming commands.
  - Config changes insert upload-definition and PROC-request records
    - MOOT identifies what to upload, and FMX supplies the content
- Output products are created by selecting records from lower-level tables based on timespan associated with a given Mission Week.
  - The resulting files are bundled and transmitted to GSSC via FASTCopy.

Activity Reconciliation

- Activities have states (Planned, Scheduled, In Progress, Complete)
  - Overall activity state is determined by the states of the individual products for that activity.
- Downlinked data products are examined to reconcile planned vs. completed activities
  - Command-response telemetry updates ATS entry requests and file-commit operations.
  - Context records in LSE datagrams provide start/end times for data associated with particular acquisitions.
- PROC requests may be reconcilable via web access to MOC's mission planning tool and/or by log files the MOC sends to us.
- Reports can be generated by querying for records with a particular state.

Data Flows


Data Products
FASTCopy
Receive
FC / Cron scripts
MP Objects
MP DB Tables
GSSC/ MOC
FASTCopy
Send
GUI
MP Objects
ATS Def
Upload Defs
PROC Requests

Implementation Technologies

- Object model is implemented in Python – interoperates with other parts of the Flight Operations S/W codebase.
- RDBMS platform is Oracle 10g
- cx_Oracle Python driver (www.cxtools.net)
  - Appears to be better maintained than DCOracle2
- SQLAlchemy – Python SQL toolkit and Object-Relational Mapper (www.sqlalchemy.org)
  - Define “table” objects that can be “autoloaded” from an existing DB
  - Declare “mappers” that associate user-defined Python objects with a table or tables
  - Framework “figures out” relationships to construct composite objects from groups of related tables
  - Concept of “Session/Unit of Work” allows the user to manipulate mapped objects programmatically, then “flush” information back to the database, with the framework performing the proper DML operations in the right sequence.
- 4Suite XML package for Python (www.4suite.org)
  - DOM implementation, MarkupWriter object for XML generation

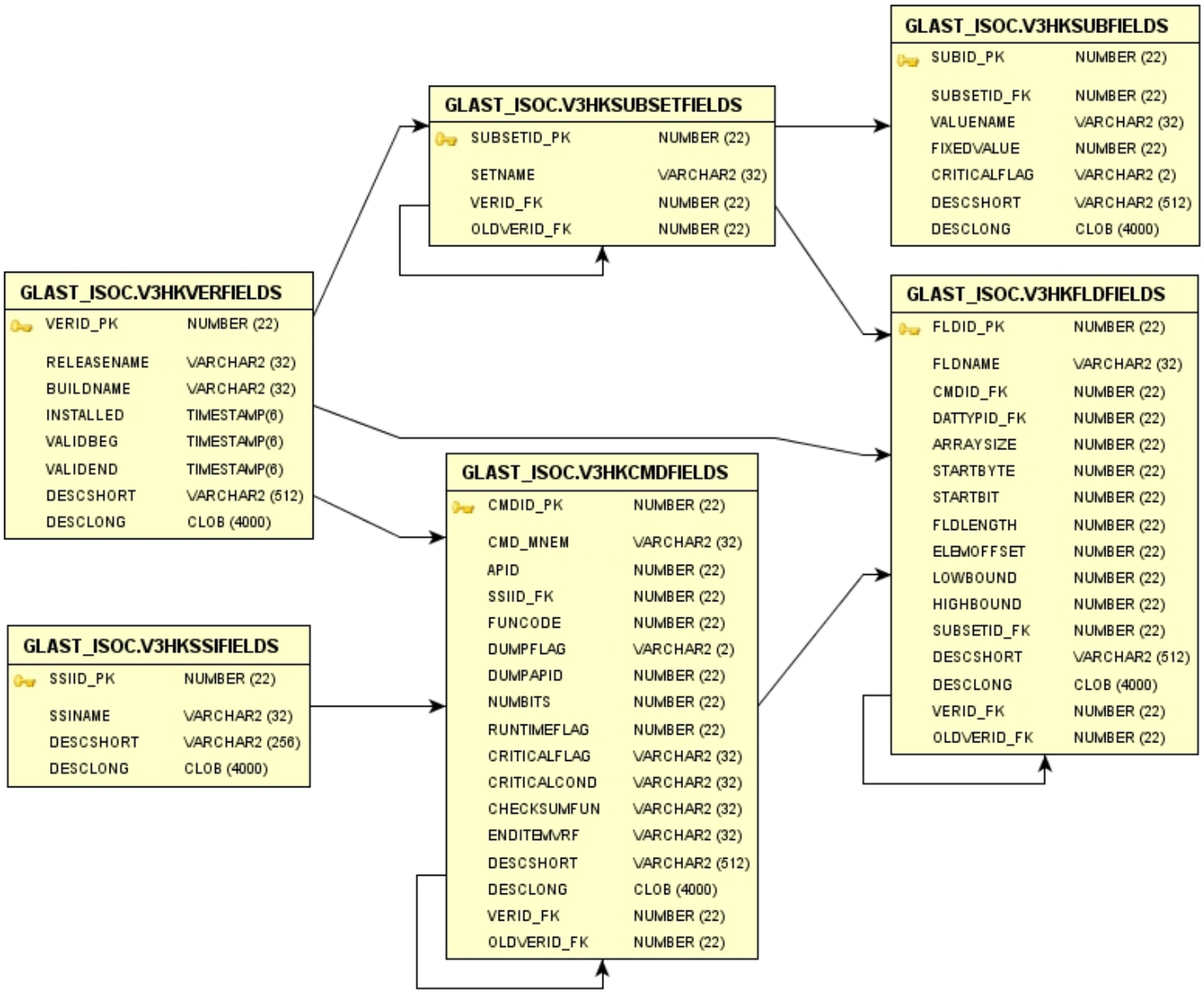
Object Model

- Four groups of tables used to define composite objects. Each object type has an associated collection
- CommandDb → collection of CmdDef objects
  - Represent telecommands defined by LAT flight software
  - Commands have one or more fields, each of which can have an enumerated set of allowed values.
- EventDb → collection of "events" defined by external entities
  - OrbEvent: SAA passage, eclipse, node crossing
  - ContactEvent: Scheduled uplink/downlink
  - ObsEvent: Pointed or survey-mode observation
  - Used to constrain activity scheduling

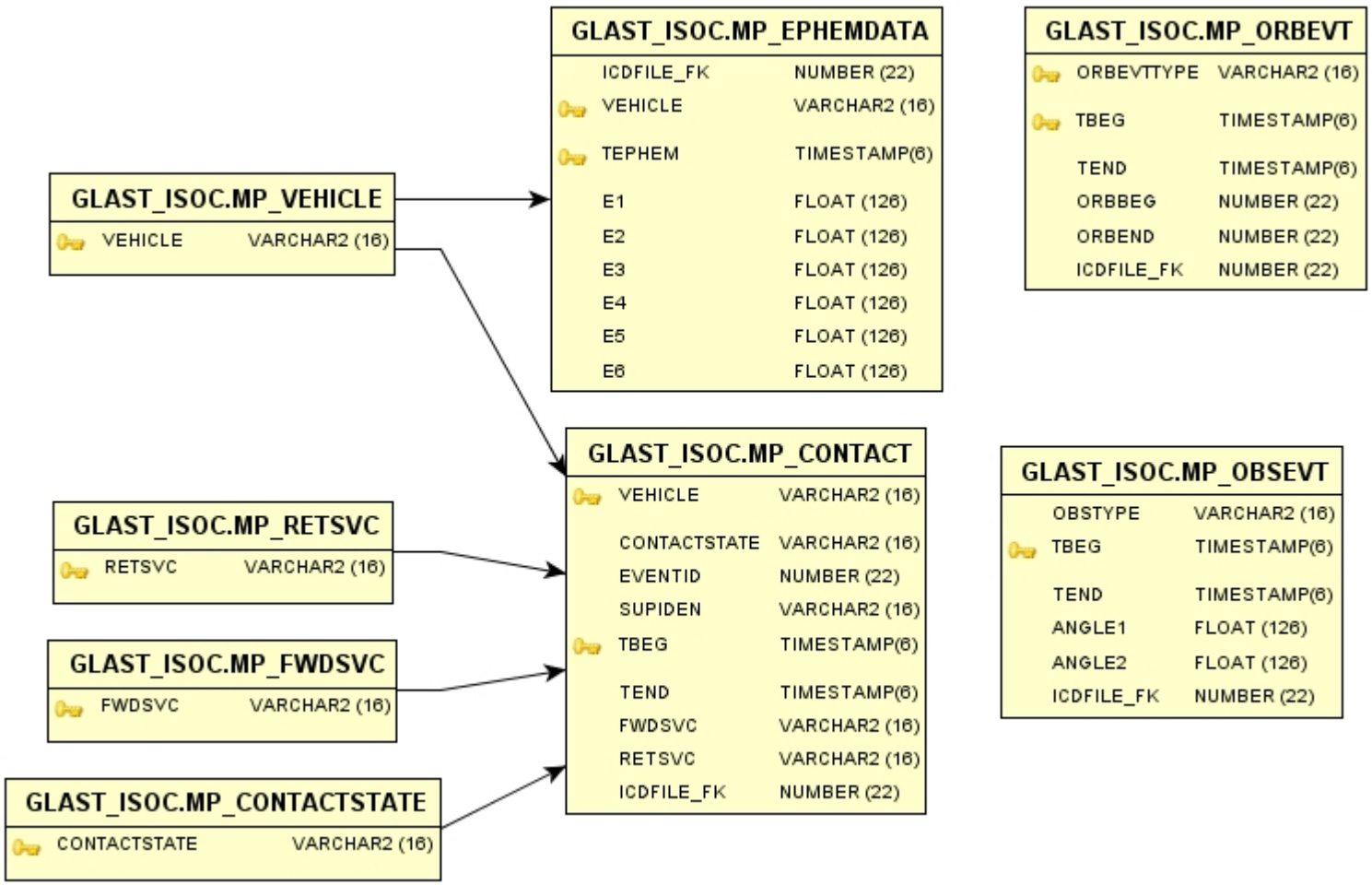
Object Model (cont'd)

- **ActivityDb → collection of LAT activities defined by planners for a specific timespan**
  - **Activity → lists of ATSEntry, ProcReq, UploadDef and Acquisition objects**
    - **ATSEntry → telecommand name + field values to be issued from the ATS, at an absolute time or triggered by an EventDb event.**
    - **ProcReq → list of ProcCall objects, with instructions to the MOC operator. Associated with a particular contact.**
    - **UploadDef → list of UploadSeg objects, each of which specifies a portion of a file to be uploaded (allows us to distribute uploads of large files across multiple contacts).**
  - **Defined list of "types" of activities equate to a user-interface dialog and specific "business rules" for creation**
    - **Current types are ATSList, Acquisition, FileLoad, ProcReq.**

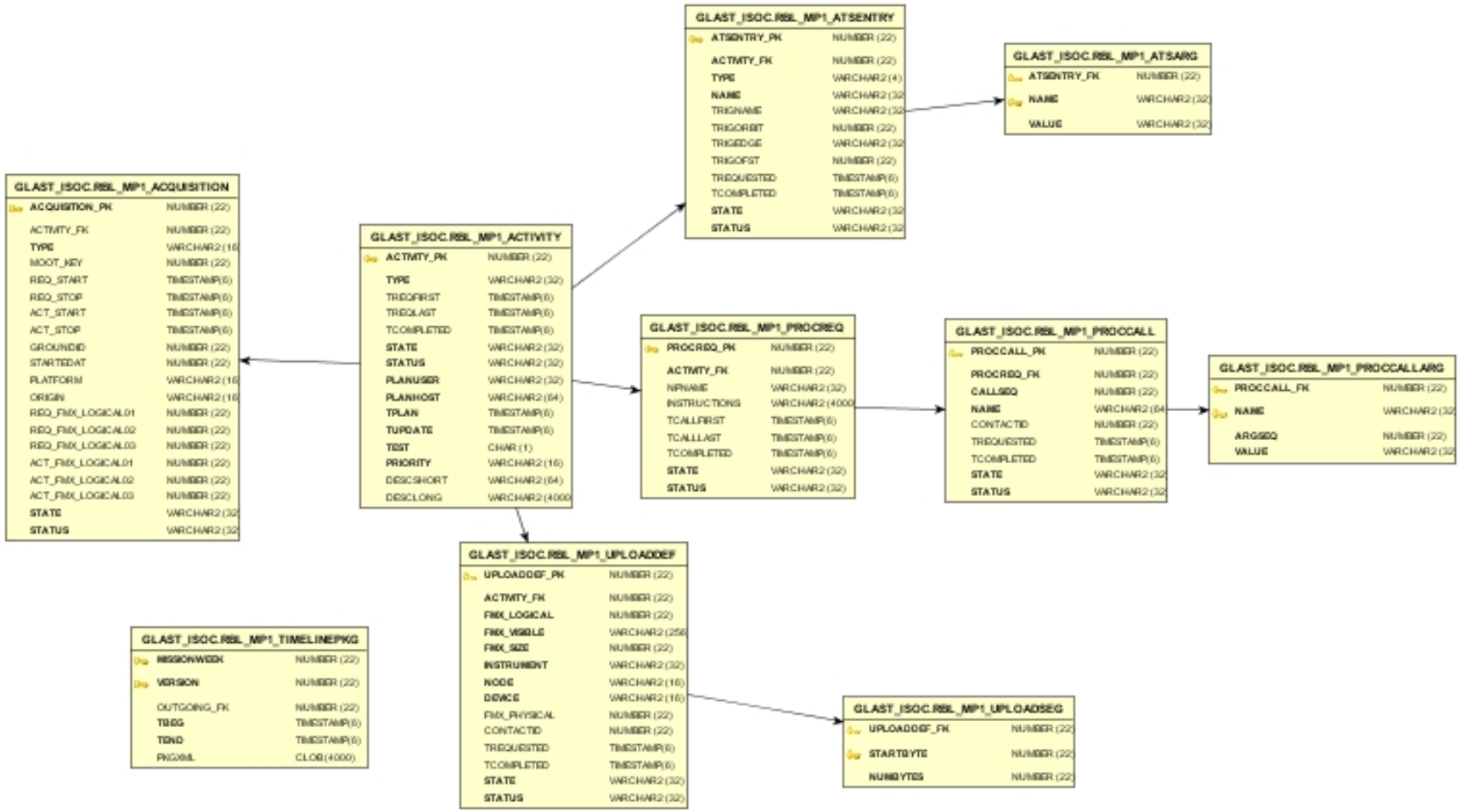
CommandDb Tables



EventDb Tables



ActivityDb Tables



From Objects to Product Files

- An ActivityDb object knows how to serialize itself to XML.
  - An XML “timeline” document contains definitions of a set of ProcReq’s, UploadDef’s, and ATSEntry’s
- The CHS “ProductUtils” package implements code to create ICD-compliant product files from XML.
  - The TransferPackage object takes a DOM-tree representation of a timeline XML and instantiates the necessary product files.
  - Then bundles them into a compliant “timeline package”

User Interface

- Work in progress, will be implemented in PyQT. “main window” dialog shown here:
- Add/Edit operations will invoke type-specific dialogs for activity creation.