Minutes of CSDT meeting #2, RAL 22nd February 2001

Ref: FIRST/FSC/MOM/0175
Date: 26th February 2001
Issue: 1.0
Author: Steve Guest

Participants:
Johannes Riedinger (ESA/HSCDT)
Stephane Veillat (ESA/HSCDT)
Hassan Siddiqui (ESA/HSCDT)
Kevin Galloway (ESA/HSCDT)
Jon Brumfitt (ESA/HSCDT)
Jean-Jacques Mathieu (ESA/HSCDT)
Peer Zaal (SRON/HIFI)
Do Koster (SRON/HIFI)
Erich Wiezorrek (MPE/PACS)
Ekkehard Wieprecht (MPE/PACS)
Rik Huygen (KUL/PACS)
Steve Guest (RAL/SPIRE)
Sunil Sidher (RAL/SPIRE)
Neal Todd (IC/SPIRE)

Agenda
It was agreed to add discussions on proposed changes to the CCM.

Minutes of Last Meeting
To save time it was agreed to refer to the comments made by ErW in his mail of Feb 6th. The issue of a CCB was raised for clarification. The understanding is that there will be no formal CCB for the time being.

Open Actions
CSDT#1-03: SG stated the answer to the action and undertook to distribute it in writing.

Common Development Environment
JB presented the status of tools for common development. The presentation is attached.

JR raised the issue of Together/CVS integration. JB explained that there is a security bug in Together leading to an unencrypted password being stored. JB & EW pointed out that they prefer to use CVS separately from Together anyway. JB also recommended using home-made make files rather than the ones generated by Together and to use “javac *.java” rather than dependencies as it is more efficient. Other tools outside of Together are also needed.

RH asked about the availability of accounts in the system. The response was that everybody has read/write access to the CVS tree. ErW & EkW commented that they were unaware of the presence of the packages outside the FIRST tree. JB proposes to control the naming of packages & subsystems.

Following discussion of whether to submit packages as JAR files or put them into CVS, JB proposed the former (drawing on experience from XMM). HS suggested that the package could also be automatically uploaded into CVS. Files checked into the system should also follow Javadoc conventions.

SG expressed concern that a number of tools appeared to be extremely Unix-specific. There is a desire to also be able to do development on PCs running Windows.

RH & EkW asked how we are to handle items of common use and who is responsible for them (JB). Discussion followed on whether it is time to start s/w coordination meetings, which would be coordinated with CSDT meetings. How can communication be improved in general? Ideas included a mailing list and a web site.

Building Block Design
JB presented some possible ways of implementing command Building Blocks.

ErW stated that scientists and engineers want to use a script to define an observation. There need only be one BuildingBlock class and the constructor could read a script to generate sub-nodes. There is a complication in the duration calculation since whereas some instrument commands have a defined time, some are not pre-defined in the MIB. If something changes in the script it is necessary to regenerate the entire tree. A “dirty flag” could be introduced for something that needs to be recalculated. The tree would only be made persistent when frozen ready for the MOC. There is also a need for access to calibration objects during script execution. This could be done with an extra statement in the language.

JB felt that Jython was “not a clean solution” and that defining a language was surprisingly straightforward. It was also felt that it would be better not to be tied to an external product. On-the-fly compilation introduced security issues which would have to be checked. **It was agreed to implement the scripting solution.** Scripting language requirements are in the CUS URD and Use Cases. Versioning of scripts is also needed. **EkW took an action to circulate ADASS ideas on similar systems.**

MIB Ingestion
HS presented the MIB ingestor status.

ErW remarked that the MIB is not a BLOB on ingestion, but a collection of ASCII tables. JR pointed out that ICD#1 is pending from the EGSE working group and that the tables for Herschel need to be filled in.

All instruments were in agreement that there is a need for both IA and QLA to access h/k data from the MIB. Versioning should make it possible to use any previous MIB. The MIB ingestion needs inputs from the various clients (CUS, QLA, IA) as to what services they require. **Each instrument took an action to provide this by March 2nd.**

IA/QLA framework

Prior to JJ's presentation, EkW was invited to say a few words about the work he had been doing in this area. He had started with some ISOCAM data and algorithms and converted the IDL to Java. Jython with plotpy was tried but lacked image visualisation. The freeware packages VisAD and DISLIN were then tried. These were better, especially DISLIN, but do not provide all the functionality users expect. The position of RSI with respect to IDL is still being monitored. JJ asked "why IDL?", to which DK responded that is was because of its programming and display capabilities. EkW stated that he was looking for contacts on technical issues. It was agreed to set up an IA implementation group comprised of EkW, JJ, JB, PZ, SG. JB proposed a technical note. There followed some discussion and the needs and priorities of IA versus QLA. SG said that the QLA was the priority for SPIRE but the needs of IA undoubtedly impacted the CCM and so it needed to be thought about. EkW indicated that he had primarily been looking at IA. The IA/QLA working group would analyse the requirements on the core model. **EkW took an action to arrange a kickoff meeting for this group. SG took an action to forward the details of the forthcoming SPIRE QLA workshop to the group.**

JJ then presented the status of his work on the IA/QLA framework. This prompted a discussion on whether ProductTM is the correct name. **RH took an action to provide a diagram and glossary of terms to KG.** All three instruments stressed the need to access data as a stream as well as by building block. This need is complementary and not incompatible with JJ's work – both approaches are needed.

TM ingestion

KG presented the status of his work on TM ingestion, which was generally well received. KG asked whether it would be useful to have a direct link bypassing the database between the TM ingestor and server. It was thought that it probably would be useful. It was clarified that the instrument data frame processing engines are needed.

AOB

There was insufficient time available to discuss the proposed changes to the CCM. SG asked what the mechanism (process) was for changing the CCM. Proposed changes are to be submitted to the CSDT and JB will respond with his analysis. **JB took an action to respond to the inputs to the CCM that had been provided.**

DoNM: Wednesday 28th March at ESTEC.

Action Items

AI#	Actionee	Action	Due date
CSDT#2-01	EkW	Circulate ADASS ideas on scripting languages.	asap
CSDT#2-02	SPIRE PACS HIFI	Provide inputs as to what services CUS, QLA, IA require from the MIB ingestor.	March 2nd
CSDT#2-03	EkW	Arrange IA/QLA working group kickoff meeting.	asap
CSDT#2-04	SG	Forward details of SPIRE QLA workshop to IA/QLA working group	asap
CSDT#2-05	RH	Provide diagram and glossary of terms to KG	asap
CSDT#2-06	JB	Respond to proposed changes to the CCM	March 26th

Appendix 1 – Agenda

Start 09h00- End 15h30

09h00
Introduction - Agenda
Comments on last MoM (CSDT#1)
Review of open actions

09h45
Common development environment (JB)
- what is already in place
- what will be put in place/when as part of WP 23400

10h15
break

the following agenda items correspond to on-going HCSS WPs. WP 24414 CUS/MIB
interface is not explicitly reported as Mark Thomas cannot join the meeting
however major issues wrt the WP are expeced to be discussed under the BB
design point, (see below).
Issues , unless trivial, will not be solved at the meeting, however the
meeting should help common understanding of the issues and agree on how to
towards solutions.

10h30
CCM (WP 23200): Building Block design (JB)
- recall of TN and alternatives
- feed-back from prototyping alternatives
- issues:
  - selection of alternatives (criteria, process)

11h30
MIB ingestion (WP 24410)
- analysis and design status
- issues:
  - definition of services to be offered to client (CUS, QLA, IA)
  - handling of multiple MIB versions: outline of one proposal

12h00
lunch

13h00
IA/QLA (Ekkie)
- analysis and design status
- issues (TBD)

13h45
IA/QLA framework (WP 24310) (JJ)
- analysis and design status
- Issues (TBC with JJ)

- design alternatives (product/process , ProductStream or ??)

14h15
TM ingestion (WP 24210) (KG)
- analysis and design status
- issues:
    - definition of I/F with instrument data frames processing engines: proposals
    - understanding of I/F with IA/QLA framework, starting from the TN on
data processing from TM ingestion onward and the HGSSE interface descriptions

14h45
Conclusion:
- any reflections on the way the comment development is proceeding: improvements
- recap of actions
- next meeting

Common Development Environment

Jon Brumfitt

CSDT meeting 220201

Common development environment

Overview

- what is already in place?
- what is planned?
- current ideas

Work packages

- 23410 tools for manual builds & automatic test harnesses
- 23420 tools for auto builds + packaging / delivery mechanisms
- 27000 software coordination
- ongoing evolution

What already exists?

CVS repository

- pserver mode with one no-shell account for each ICC
- passwords have been issued

```
setenv CVSROOT :pserver:$USER@astro.estec.esa.nl:/local/repositories/FIRST_CVS
cvs login
cvs checkout fcss/foo    # e.g. subsystem 'foo'
```

Structure

model/design	UML CCM
model/domain	UML domain model
develop/together	add-on modules for Together
develop/fcss	root for Java pacakges
develop/fcss/nl/esa/first/foo	package nl.esa.first.foo

Java package structure

fcss tree - examples

nl.esa.first	evolutionary common system
nl.esa.first.foo	subsystem 'foo'
nl.esa.first.foo.gui	foo's 'gui' package
nl.esa.first.devel	development tools
nl.esa.first.util	utilities shared by subsystems
nl.esa.first.util.swing	custom Java Swing components
nl.esa.first.store.iface	prototype persistent store interface
nl.esa.first.store.vers	prototype Versant implementation of store
nl.esa.first.ccm.iface	prototype CCM interface
nl.esa.first.ccm.vers	prototype Versant implementation of CCM

This will probably evolve, but we need to start somewhere!

FSC and ICC evolutionary code is all in the nl.esa.first package tree.

Java package structure

Prototype tree - examples

If you need package names for exploratory prototypes:

nl.esa.jbrumfit	Jon's prototypes
nl.esa.kgallowa	Kevin's prototypes
de.mpg.mpe.ewieprec (?)	Ekki's prototypes

...

Simple prefix OK if you don't want to share it with others.

Note

- develop exploratory (throw-away) prototypes outside the 'first' tree
  - unique namespace for each developer
- don't check exploratory protypes in under 'fcss' tree in repository!
- proto tree can use fcss tree, but not vice versa

Together project structure

Hierarchy of Together projects

fcss.tpr	Read-only view of whole tree
nl/esa/first/first.tpr	
nl/esa/first/mps/mps.tpr	Developer's view of mps subsystem
nl/ersa/first/phs/phs.tpr	Developer's view of phs subsystem

- Each project is 'owned' by one person

Creating a Together project

- create a project for a sub-tree (e.g. nl.esa.first.foo)
- project properties:
  - package prefix = nl.esa.first.foo
  - classpath - add latest build of system / other packages
  - for now, just include source packages in project paths, read-only
- check in to CVS (recommend command line)

Manual builds

Together

- compile / run OK for simple program
- complex programs / projects need more, e.g:
  - copy resources (icons, properties) to output tree
  - enhance persistent classes
  - run code generators (e.g. javacc, jjtree)
  - generate documentation (e.g. javadoc, jjdoc)
  - support package delivery + versions / dependency
  - running test harnesses
  - generating JARs
- builds shouldn't rely on using Together

Propose to have command-line build invoked from Together menu

Manual builds

Makefiles

Not really appropriate for Java, except as simple scripts

- Java builds much faster with "javac *.java"
- javacc handles some dependencies - clashes with make
- Java imports are NOT transitive (unlike C++ #includes)

There is a basic makefile in the repository, to get started

- UNIX specific - will replace with a better mechanism

Simply include GNUmake.include and set some variables:

```
PACKAGE = nl.esa.first.foo
RESOURCES = gui/images
include ../../../../GNUmake.include
```

Package delivery

Options

- submit packages by checking them into CVS
  - use CVS tags to identify versions
  - easy to break the system (continuous integration)

- submit packages as JAR files
  - each package has a version number and dependency info (c.f. RPM files)
  - use dependencies to build latest compatible set of packages
  - run reception checks and reject invalid packages
  - use CVS more as a backup mechanism

CVS is OK to get started

Test harnesses

Each deliverable package will have a test harness

- where possible, these run automatically
- GUIs may require some manual tests
- packages must be tested before delivery

Tools

- will probably use 'JUnit' (TBC) - www.junit.org

Automatic builds

Build script

- run automatically each night
- build / integrate complete set of delivered packages
  - check package versions / dependencies
  - compile
  - generate documentation
  - run test harnesses & report on failed packages
  - make available to developers
- each build has a build number
- regular builds, although Java is better than C++ at incremental build

Note

- developers check out their own package source from CVS
- include other packages via classpath from latest (or specific) build

Building Block Design

Jon Brumfitt

CSDT meeting 220201

Overview

Building blocks are still at the conceptual level in the CCM

- We need an architectural design

The problem

- Need the flexibility to add new modes / block types easily
  - e.g. "one-off" observing modes + ILT
  - This should not require new software releases
  - Better to represent block types as objects than as classes

Investigations carried out

- TN-013 considers implementation approaches
- Prototypes to explore feasibility / implications

Design & implementation

Possible approaches

- Basic Java approach
- Jython interpreter
- Custom language + interpreter
- Compiling Java on-the-fly

Basic Java approach

Each Building Block / Mode type is a new Java class

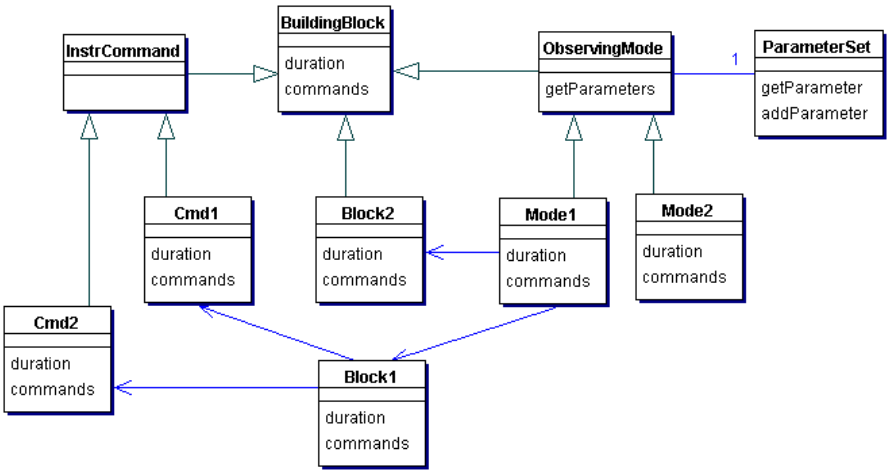
Advantages

- simple to implement for a fixed set of modes / blocks

Disadvantages

- blocks are defined by classes (code) rather than objects (data)
- requires new software releases for new modes / blocks
- need to restrict power of language
- importing MIB may require changing classes

Basic Java approach



Jython interpreter

Interpret mode / block definitions written in Jython

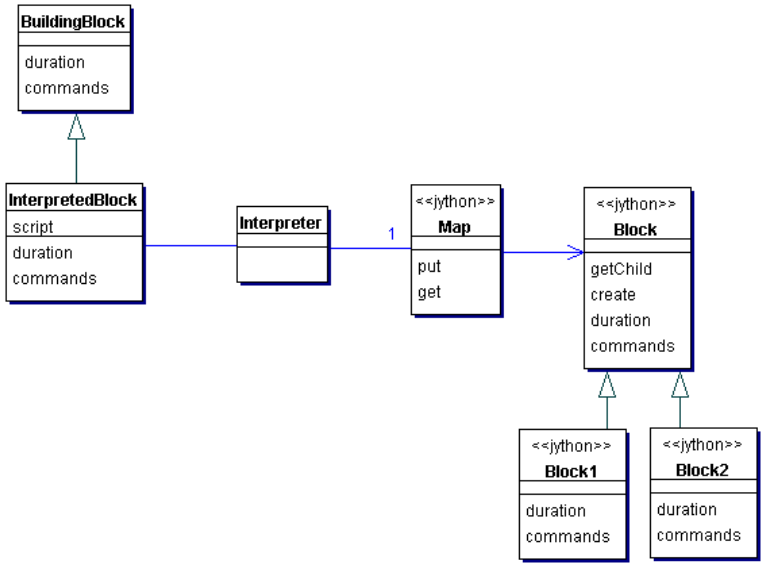
Advantages

- flexible + off-the-shelf language

Disadvantages

- integration of Java/Jython is complex at the object level
  - e.g. Java proxies for Jython objects
- cannot make Jython objects persistent directly
- poor compile-time checking => bugs & run-time errors
- block definitions are unnecessarily complex
  - e.g. methods for both duration & commands
  - implementation exposed rather than declarative definition
- performance is poor
- dependence on another technology

Jython interpreter



Jython interpreter

Example block definition

```
class MyBlock(Block):

   def duration(self,x):
      return self.getChild(0).duration(x+2)
           + self.getChild(1).duration()

   def commands(self,cmds,x):
      self.getChild(0).commands(cmds,x+2)
      self.getChild(1).commands(cmds)

   def children(self):
      return ['BlockA','BlockB']
```

Custom language + interpreter

Write an interpreter / compiler for a custom block-definition language

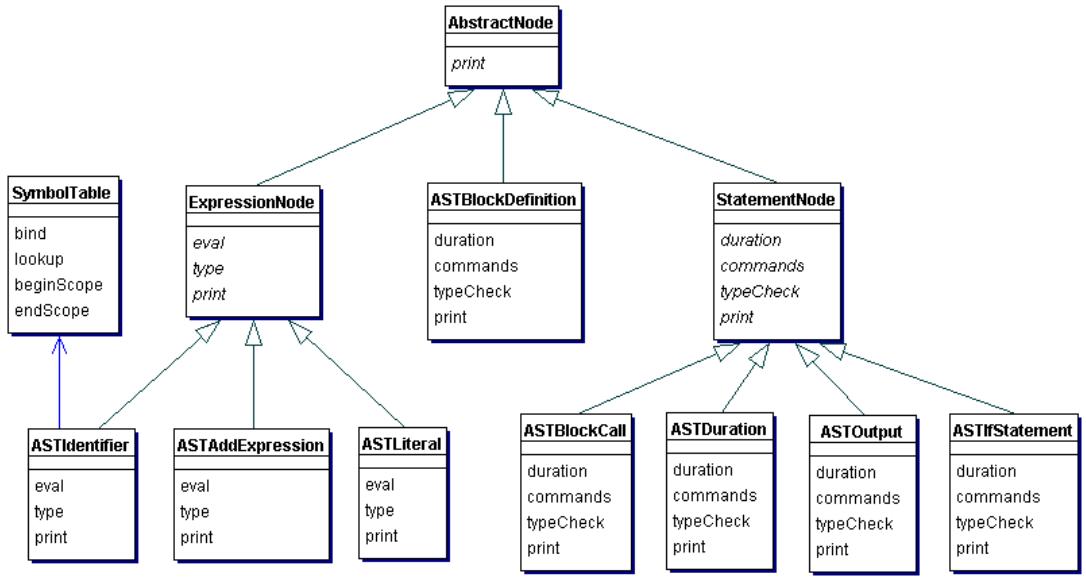
Advantages

- performance good (if compiled)
- incremental evolution - can start simple & add features
- block definitions can be cleaner and simpler
  - one definition + multiple interpretations
- better error handling
  - semantic analysis (e.g. type checking)
- written in Java - clean & portable

Disadvantages

- implementation is complex for a complex language
  - but prototype shows it is not too difficult

Custom language + interpreter



or use the Visitor pattern to separate interpretations from AST

Custom language + interpreter

Example block definitions

```
command IC1(x) {
   title("My instrument command")
   output("IC1",x)
   duration(3)
}
block Blk1(x) {
   if(x > 3) { IC1(x+2) output("offset") }
   IC2("init",x)
}
mode Mode1() {
   parameter(x,float,"offset",42,min=10,max=70)
   Blk1(x + 3)
   Blk1(0)
}
```

Example from MT

```
observing_mode SPIRE01
   title "Point Source Photomery"
   parameter TotalIntegrationTime
   begin
      type double
      comment "Required integration time";
      maximum 1.0;
      minimum 0.0;
   end_parameter
begin
   instantiate_building_block(name=>"spire_init");
   instantiate_building_block(name=>"slew",dec=>dec,ra=>ra);
   instantiate_building_block(name=>"nod");
   the_start = startwavelength;
   if(the_start < min) { the_start - min; }
   steps = (endwavelength - start) / stepsize;
   foreach i in 1..noOfLines {
      foreach j in 1..noOfPoints {
         do {
            instantiate_building_block(name=>"chopStart",...);
            instantiate_tc(name=>"a123");
         } while requiredSignal < noise;
      }
   }
end observing_mode
```

Custom language + interpreter

Component design - integrating with the CCM

- interfaces and packages
- separate reusable parser components
- separate uplink / downlink responsibilities
- blocks, parser, clients, CUS GUI, etc
- persistence - robust persistent classes
- visitor pattern to separate phases?

Compiling Java on-the-fly

Invoke the Java compiler within program to dynamically add new classes. Store class files in the database and load with custom class loader.

Advantages

- flexible

Disadvantages

- new classes should preferably not be persistent
  - access state via persistent parent
  - simplifies schema evolution
- "messy" solution, which may reduce portability
- possible conflicts with custom class loader
- too powerful - security needs designing carefully

Parameters

Parameters are entered for an observation / observing mode

- parameter types: bool, int, float, string, etc
- default value
- constraint: range (min-max) or set e.g. {red,green,blue}
- label (for GUI)
- description (for GUI tool-tip / help)

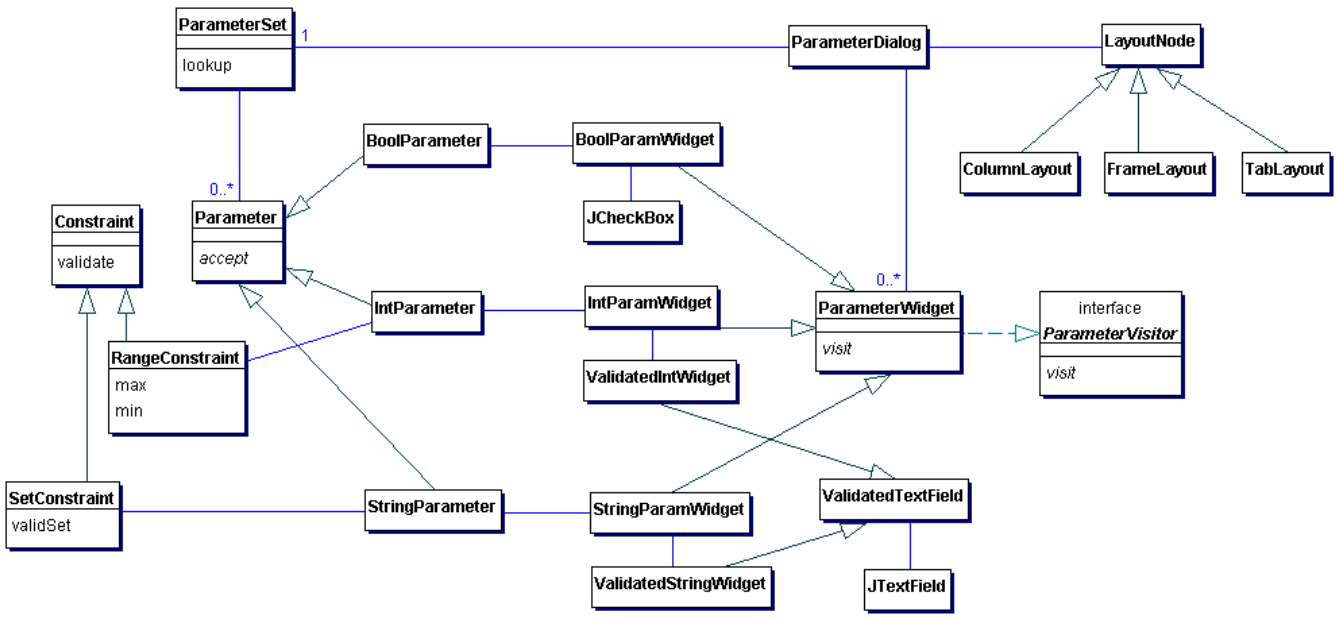
Building blocks & instrument commands have arguments

- arguments to sub-blocks may be derived (calculated)

Decouple GUI (view) from parameters (model)

- separation of concerns
- different users may want different (hierarchical) views
- Parameter dialog is useful in several sub-systems

Parameters



Conclusions

Custom language + interpreter appears to be the best

- clean design
- can be developed incrementally
- simpler block definitions
- better error handling
- written in Java
- prototyping has been straightforward

Issues

- Can we agree on the approach?
- How complex does the language have to be?

MIB Ingestion

Purpose:

- To ingest SCOS-2000 ASCII files
- To provide services to clients such as the CUS, IA and QLA

Framework

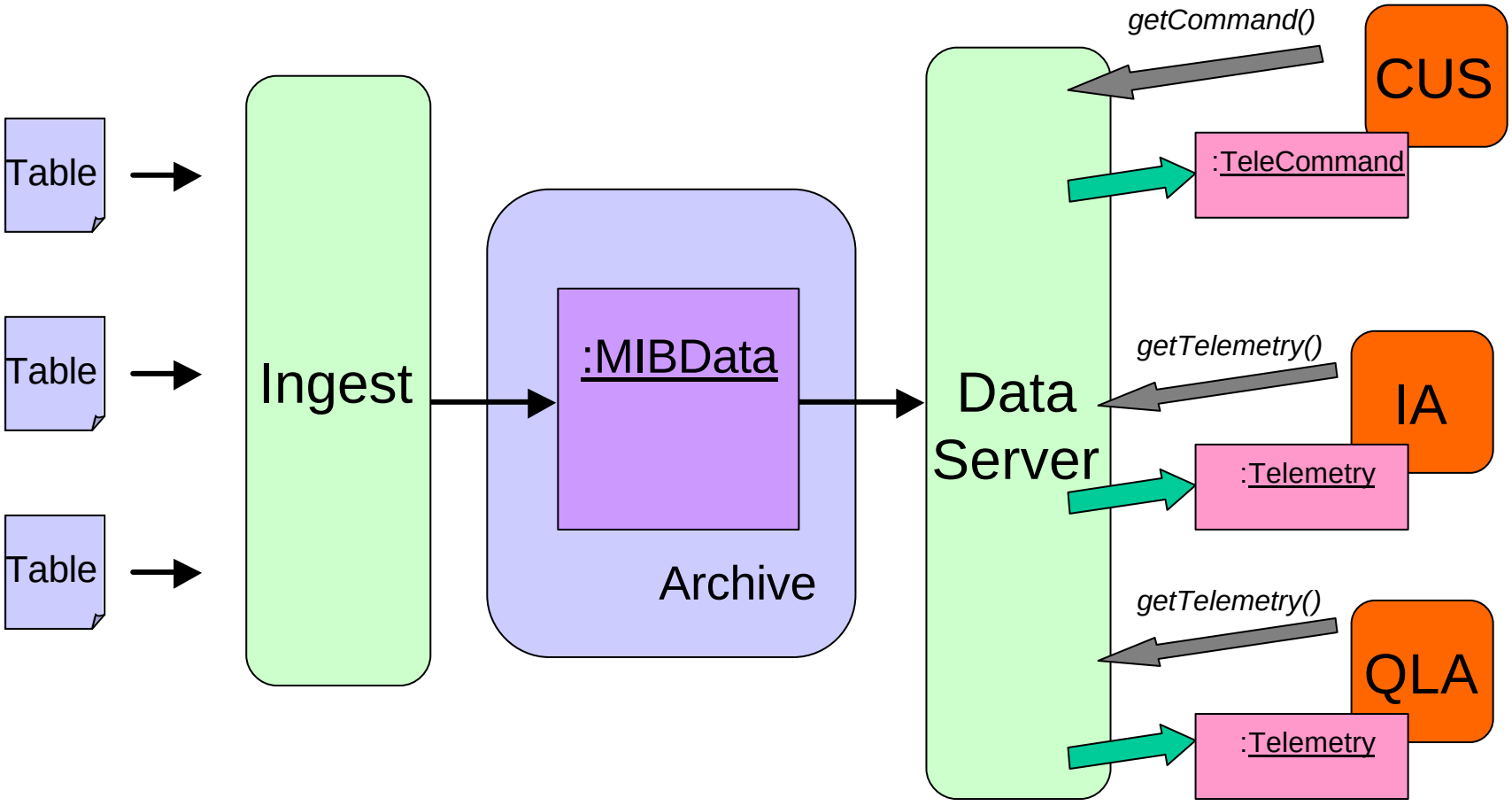
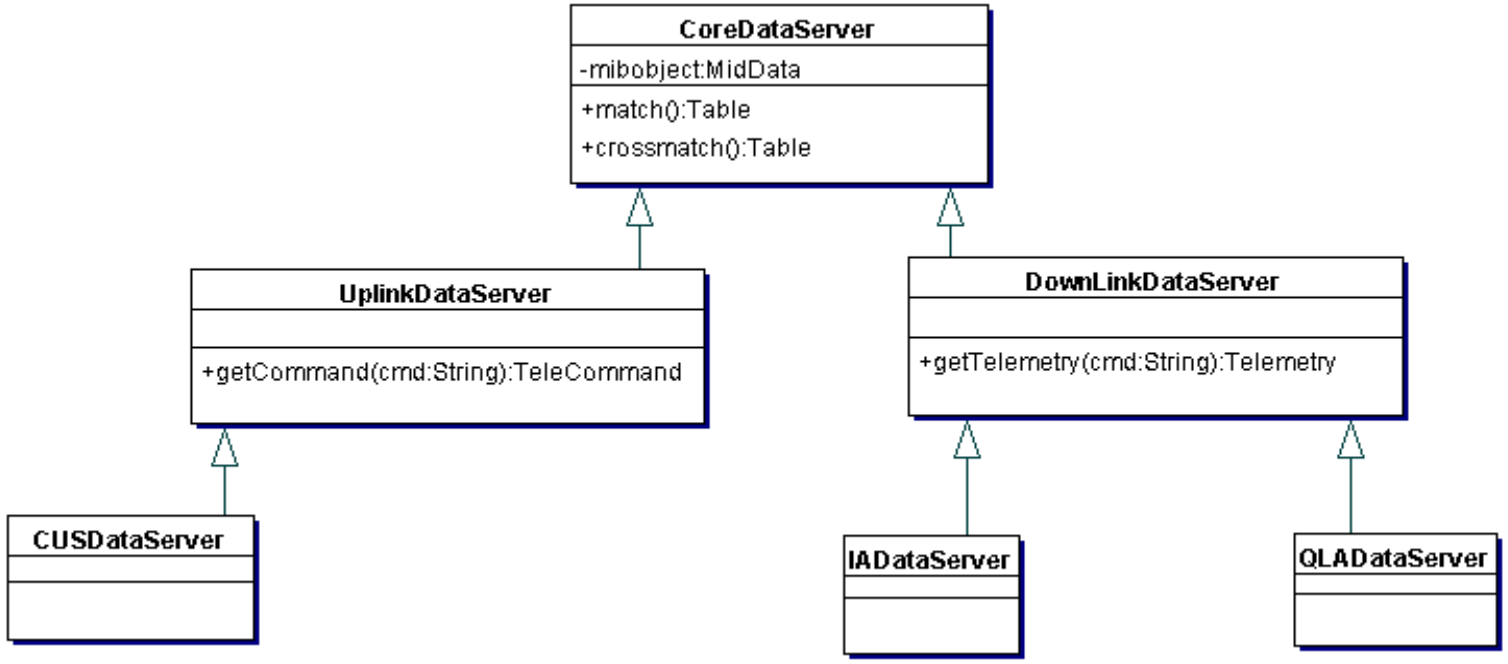
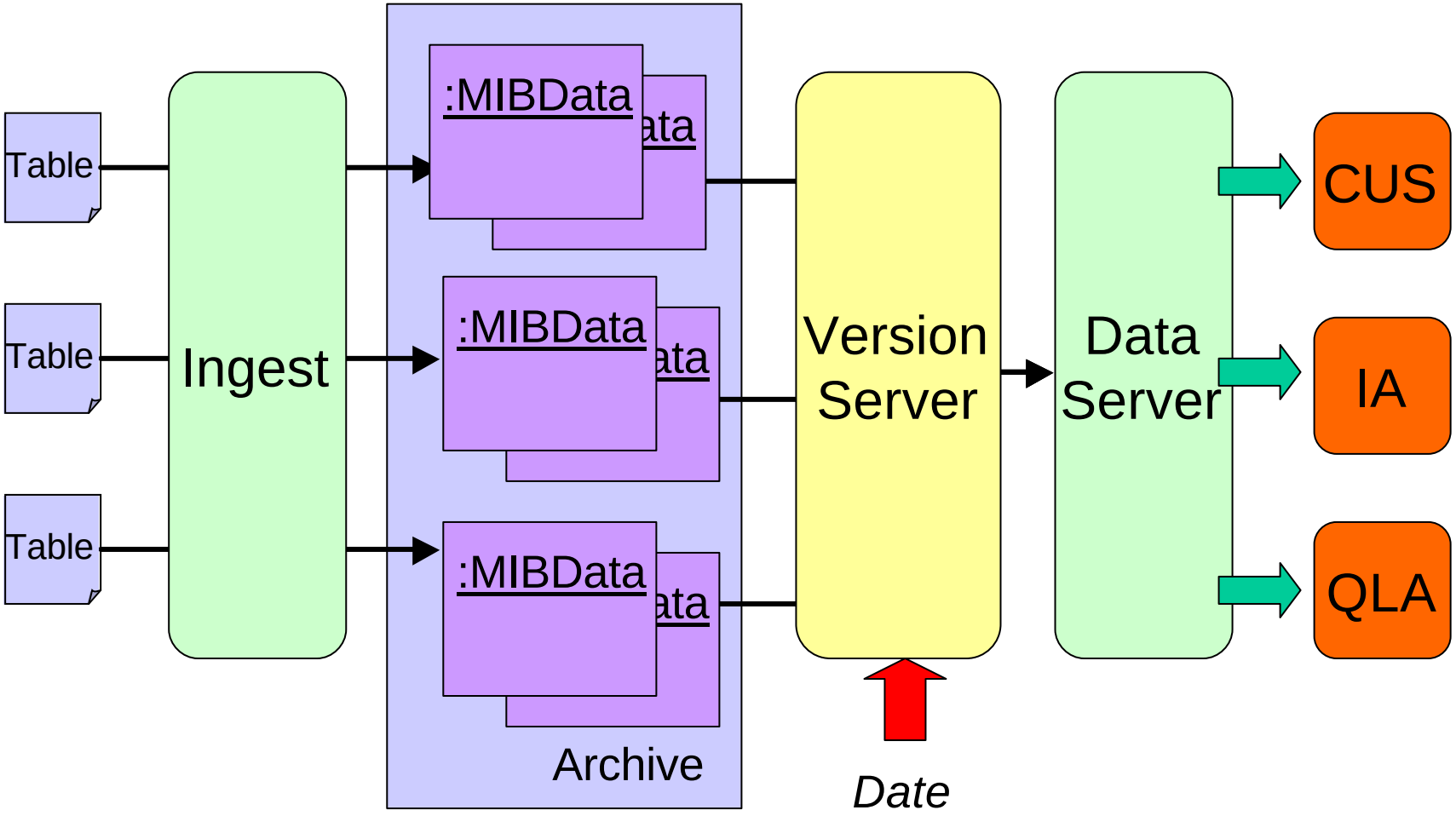


Table
Table
Table
Ingest
:MIBData
Archive
Data Server
getCommand()
CUS
:TeleCommand
getTelemetry()
IA
:Telemetry
getTelemetry()
QLA
:Telemetry

DataServer Class Diagram

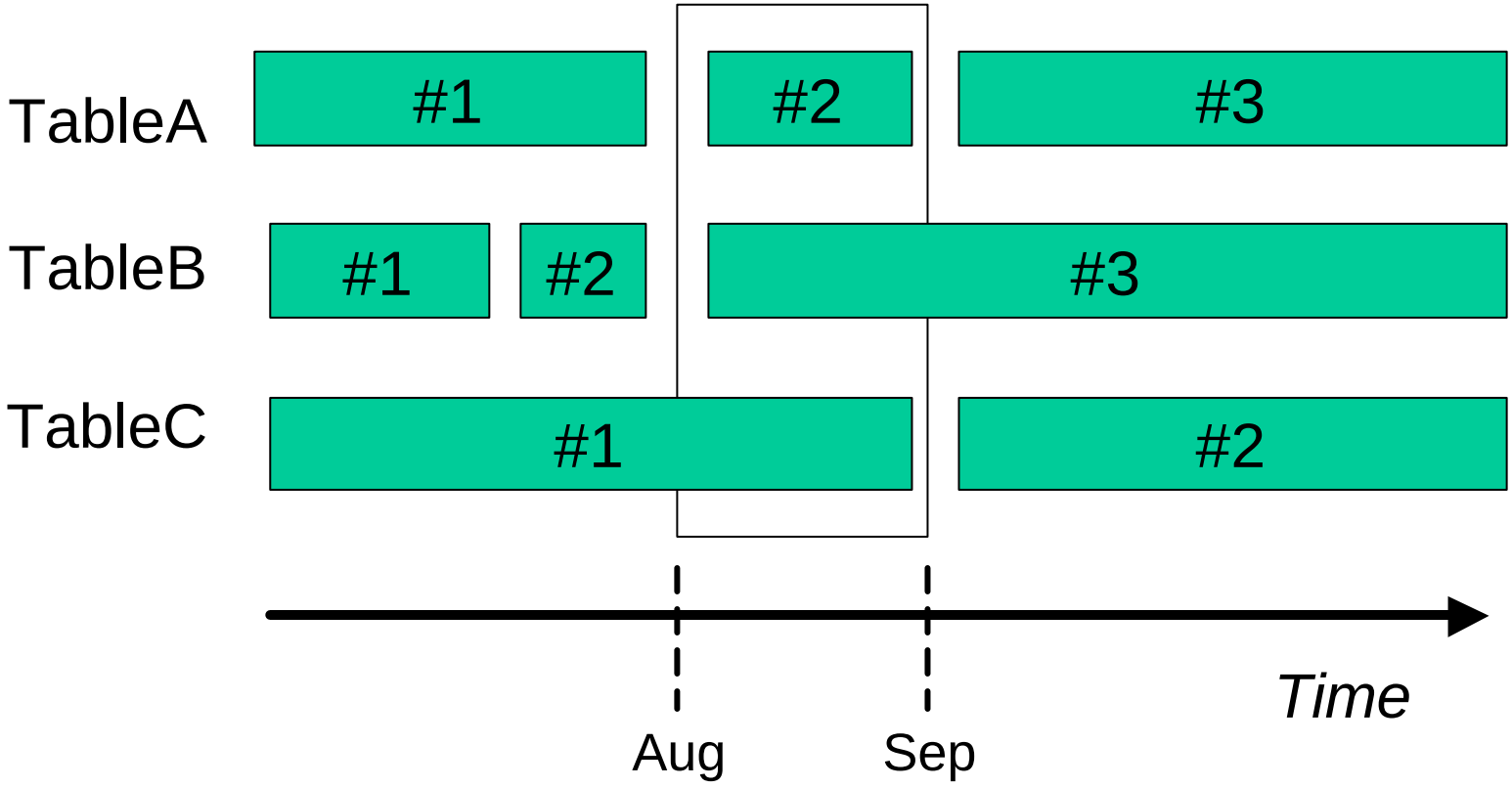


Framework (with versioning)



Choosing a Version

- Tables will be stored in archive with initial/final validity dates.



Progress

- Prototyped initial framework.
- ERS-2 example ASCII tables parsed successfully.
- Services eg getTelecommand() prototyped.
- Need to add further services and explore versioning.
- Validation - are existing building blocks consistent?
- Optimise code

Issues

- Need **inputs** from clients (CUS, IA, QLA) as to what services these subsystems need from MIB ingestion
- Which SCOS-2000 tables are appropriate for Herschel use?

QLA/IA Framework

JJ Mathieu

22 Feb 2001

Analysis and design status

- ✓ Prototype out (uses building block and probably closer to SPG/QCP than QLA or IA)
- ✓ Looking at XML and XDF (from NASA) for exporting data in platform neutral way
- ✓ Looking at class layers (Product, NDim, Specific)

Do we agree?

- ✓ **ProductTM:** a complete measurement (ie decompressed/assembled data seen as a bit set with a header containing PUS information and BB information). A product TM cannot be used for any science (it has no structure) but contains identification data (APID, type, BBID…).
- ✓ **ScienceTM:** (aka DataFrame?) The result of applying a structure to the ProductTM and giving it a unit. The simplest form of data that can be displayed by a QLA system (not made persistent by QLA)
- ✓ ProductTM is what the TM ingestor delivers to QLA.
- ✓ ProductTM needs no BB

Do we agree?

- ✓ QLA needs Observation Execution (and BB) to give a structure to ProductTM.(What are alternatives -ie what does the framework has to support for this to happen?-)

Observation Responsibility and characteristics

- ✓ Receives raw data (ProductTM) and distribute it to BB
- ✓ Has a time span
- ✓ Links to a request
- ✓ Can create high level product (SGP/QCP)

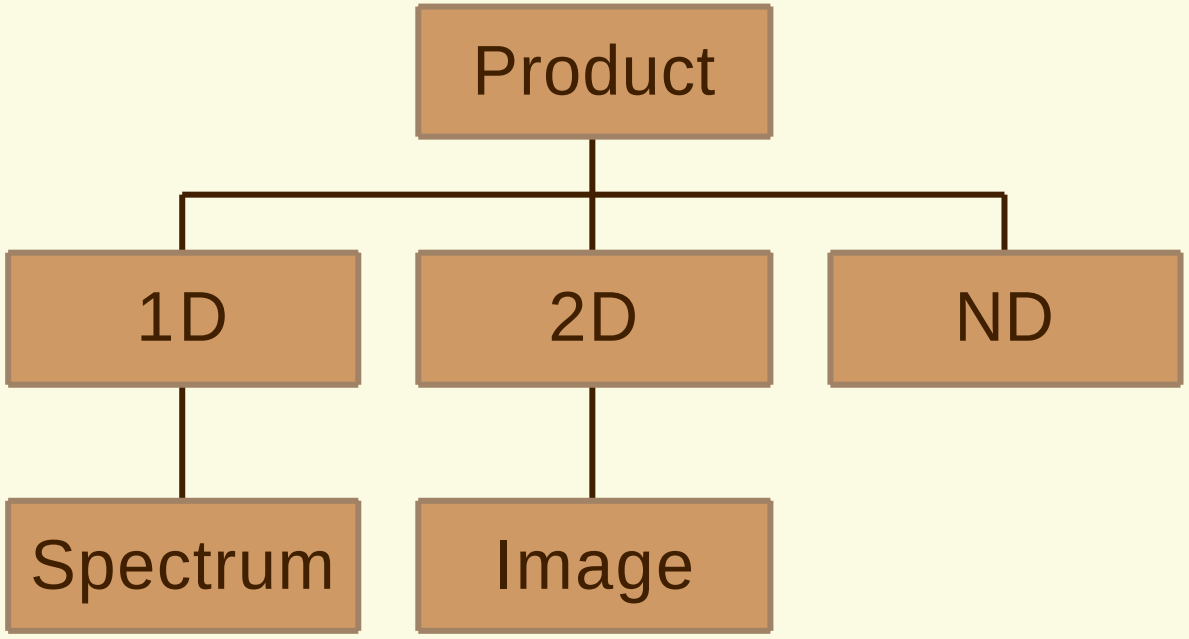
BB responsibilities and characteristics

- ✓ Transforms raw data into engineering product
- ✓ Elaborate high level product (SGP)
- ✓ Assess product quality (QCP)
- ✓ Receive raw data
- ✓ Has strict configuration control

Product characteristics

- ✓ Holds data
- ✓ Has history
- ✓ Has characteristics (unit, number of axis..)
- ✓ Can be exported into XX format
- ✓ Can be imported from YY format
- ✓ Can transform itself using a set algorithm

Products



Client properties

	QLA	QCP/SPG	ATA	IA/CA	OIA	TA
Inside HCSS infrastructure	**?**	**+**	**+**	**+**	**+**	**+**
Outside HCSS infrastructure	**?**	**-**	**-**	**+**	**+**	**+**
Interactive	**-**	**-**	**-**	**+**	**+**	**?**
Automated	**+**	**+**	**+**	**-**	**-**	**?**
RT	**+**	**-**	**-**	**-**	**-**	**-**
Consolidated	**+**	**+**	**+**	**+**	**+**	**+**
TM based	**+**	**-**	**-**	**+**	**-**	**-**
Product based	**-**	**+**	**-**	**+**	**+**	**+**
BB based	**+?**	**+**	**+?**	**+**	**+**	**-**

Future

- What do we do to progress?
  - Products group evolving to QLA/IA group?
  - Prototyping? Schedules? Priorities?
  - Suggestions?

WP 24210: TM Interface: Architectural analysis and prototyping

K. Galloway

WP24210: Status

Status: On-going

TM Ingestor responsibilities/ services to be investigated/ addressed:

1. Know your instrument
2. Interface with EGSE-ILT router
3. Production of ProductTM (or DataFrames)
4. Save PUS Telemetry (TMStream) and ProductTM
5. Distribute ProductTM

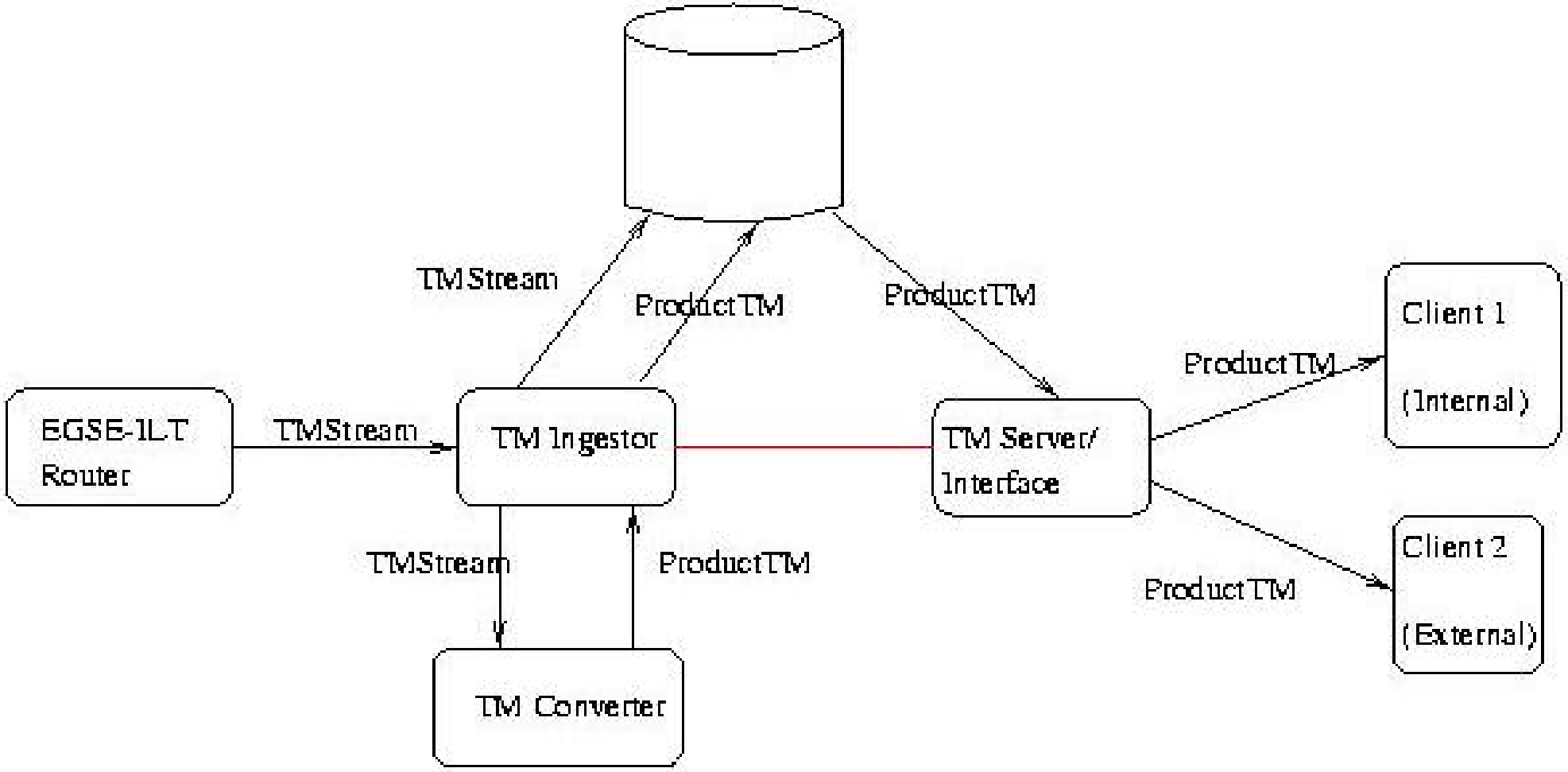
Documented in a technical note which will act as:

(1) An informal ICD between myself and the developers of the "instrument data frames processing engines". We'll decide on which interface(s) we prefer and document it here.

(2) A means for everyone to review the choices of component architectures etc.

Use case UCF-758 will be kept up to date.

WP24210: Overview of interfaces



WP24210: Know your instrument

Inputs/ Assumptions:

1. TM Ingestor will be running independently at several sites.
2. Each instrument team will have several versions of their instrument which they wish to test.
3. Only one instrument will be tested at any given time at a given site.

Solution:

The instrument is identified to the TM Ingestor before the test starts:

(I) HIFI/ PACS/ SPIRE

(2) Instrument version

The factory method design pattern can then be used when invoking the other TM Ingestor services as necessary.

WP24210: Interface with EGSE-ILT router

EGSE-ILT Router developer (Albrecht) has supplied me with:
(1) ICD
(2) A prototype router

The TM Ingestor successfully interacts with the router.

Based on TCP/IP: Java Sockets and Java I/O

Performance aspects are currently under evaluation:-

FGS-IR-4.2-50: The EGSE-ILT FCSS TM I/F shall support a data rate equivalent to the addition of the maximum instrument on-board data rate (400 kps) and the maximum TEIs data rate.

[The maximum TEIs data rate is TBD but is expected to be equal to or lower than 400 kbps]

WP24210: Production of ProductTM

1. Separate CSDT developers produce the “instrument data frames processing engines” (from WP descriptions).
2. The PACS processing engine has the most to do.

Proposed solution:

The TM Ingestor passes the science TM packets to the processing engines (no collecting).

The processing engines collect the science packets together and then produce the TMProduct.

WP24210: Production of ProductTM (cont.)

Investigated interfaces:

1. FIFO (First in first out)
   Simple collection/ container into which the TM Ingestor places the TM packets.
   The processing engine can then read the TM packets.
   The processing engine can place the generated ProductTM on another FIFO which the TM Ingestor can read.

2. Asynchronous event handling
   The TM Ingestor send the TM packets to the processing engine as events. The processing engine has a/ is a listener.
   The reverse process can be performed for ProductTM.

Documented in TN. Developers to make a decision based on performance and ease of use (for developers).

WP24210: Save TMStream and ProductTM

TMStream:

All telemetry (TMStream) is stored in containers according to:
(1) Instrument model
(2) APID
(3) packet type
(4) packet subtype

All telemetry packets which contain OBSID and/ or BBID to be associated with the observation execution and/ or building block execution.

ProductTM:
Associated with observation execution and/ or building block execution.

Associated with a ProductTMStream

WP24210: Distribute ProductTM

Is it necessary to have a direct link between TMIngestor and the ProductTM Server?

1. In early ILT there may not be OBSID and BBID associated with the TM stream (Products TN)
2. Client requests for near real-time connection

The ProductTM interface/ server

What questions will the clients (QLA) ask?
(1) Give me ProductTM as it arrives (near real-time connection)
(2) Give me ProductTM for specified time period
(3) Give me ProductTM associated with observation execution x
(4) Give me ProductTM associated with building block y