A SOFTWARE ARCHITECTURE FOR IN-FLIGHT ACQUISITION AND OFFLINE SCIENTIFIC POST-PROCESSING OF LARGE VOLUME HYPERSPECTRAL DATA

Jason Brazile

Remote Sensing Laboratories
Dept. of Geography, University of Zurich
jason.brazile@geo.unizh.ch

Peter Kohler and Simon Hefti

Netcetera AG
Zurich, Switzerland
{peter.kohler,simon.hefti}@netcetera.ch

ABSTRACT

The European Space Agency (ESA) is sponsoring a joint Swiss/Belgian/German initiative to design, build, and deploy an airborne dual prism dispersion pushbroom imaging spectrometer known as APEX (Airborne Prism EXperiment) to support earth observation applications at a local and regional scale. APEX has been designed to acquire 1000 pixels across track (covering 2.5-5 km, depending on flight altitude) with a maximum of 300 spectral bands simultaneously in the solar reflected wavelength range between 400 and 2500 nm [1]. This coverage of the Visible/Near Infrared (VNIR) and Shortwave Infrared (SWIR) spectrum in addition to a rigorous pre-flight and in-flight calibration and characterization process enable the resulting data to be used in a wide variety of applications including snow, vegetation, water, mineral, and atmospheric analysis and monitoring [2].

We discuss the design and ongoing implementation of the software architectures for both in-flight data acquisition and offline level 1 scientific data processing which are based in large part on the extendible and embeddable features of the Tcl scripting language to allow for rapid prototyping, hardware simulation and control, automated testing, and modularization and integration of proprietary software components as well as foreign domain-specific languages.

We additionally describe design trade-offs and architecture modifications that were made to better fit this programming model (event driven vs. threaded programming, "fat" calls vs. "thin" calls, etc.) in order to keep as much program logic as possible at a simpler higher level. In cases that require strict performance requirements, we describe how we built prototype architectural components and in some cases compared them with versions in C to help determine feasibility and ensure that memory and processing resource budgets could be met.

Keywords: Embedded Control, Multi-Domain Integration, Scientific Processing

1. INTRODUCTION

The planning and specifications for the APEX instrument started in 1997 with a feasibility study [3] and proceeded through a scientific performance definition and an industrial design phase. The current construction phase of the instrument began in 2002 and is planned to be final in early 2005, when the first end-user flight campaigns are scheduled.

The core of the spectrometer consists of a beam splitter separating incoming light into the VNIR (380-1000nm) and SWIR (930-2500 nm) wavelength ranges where the beams are spatially and spectrally re-imaged on independent, co-registered detector arrays. The detectors both resolve 1000 spatial pixels across track and more than 150 spectral rows each, which are then summarized via reprogrammable on-chip binning into a maximum of 300 spectral rows for both detectors. A subset of the relevant specifications is shown in table 1.

Specified Parameter	**Value**
Field of View (FOV)	$\pm 14°$
Instantaneous Field of View (IFOV)	0.48 mrad
Flight Altitude	4,000 - 10,000 m.a.s.l.
Spectral channels	VNIR: ≈ 140 SWIR: ≈ 145
Spectral Range	400 - 2500 nm
Spectral Sampling Interval	400 - 1050 nm: $<$ 5 nm 1050 - 2500 nm: $<$ 10 nm
Spectral Sampling Width	$<$ 1.5 $\times$ spectral sampling interval
Scanning Mechanism	Pushbroom
Storage Capacity On Board	$>$ 300 GByte
Dynamic Range	12 . . . 16 bit
Positional Knowledge	20% of the ground sampling distance
Attitude Knowledge	20% of IFOV
Navigation system, flight line repeatability	± 5% of FOV

Table 1. Selected APEX Specifications

The data processing requirements of the APEX instrument can be logically separated into two somewhat independent components – on-board data acquisition, and offline scientific post-processing. The first section of this paper addresses the architecture requirements and subsequent design and implementation used for in-flight data acquisition, which mainly concerns itself with collection and integration of data from various sources and arranging for it to be stored in a safe and timely manner.

This is followed by a section on the requirements of scientific post processing which mainly involves how the collected raw data and previously acquired instrument characteristics can be correlated and calibrated into well-defined scientific units (i.e. at-sensor radiance in SI units, traceable to a certified standard (e.g. NIST, NPL)).

Then follows a section on the development process and how it has also influenced the design and implementation of the APEX system and application software. The final section summarizes and concludes with the current outlook.

2. IN-FLIGHT DATA ACQUISITION

APEX in-flight data acquisition is somewhat challenging from a data processing point of view, as the collection of raw pixel data from the optical unit needs to be simultaneously correlated with high precision data coming from separate timing, positioning, and inertial data collection instruments and written in real time to on-board disk storage. Additionally, in-flight instrument calibration is performed between flight strips, requiring control of supporting hardware such as the calibration lamp, filter wheel, and mirror which are all to be centrally and automatically directed and monitored by the operator via an on-board computer. These data flow and bandwidth requirements are illustrated in figure 1.

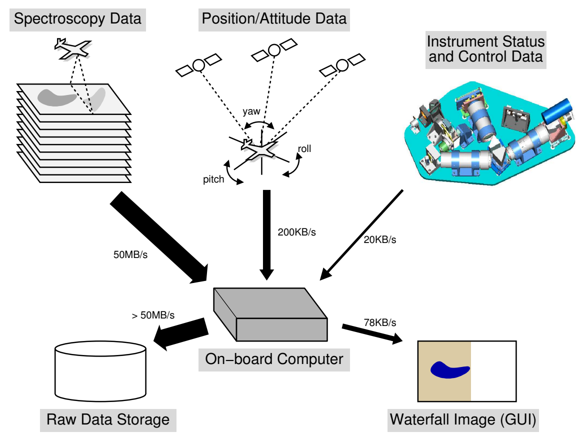

Fig. 1. Data Acquisition Requirements

Initial system analysis indicated that there were 3 critical engineering problems that needed to be addressed to ensure successful data takes:

Data Throughput Optical data flows into the system at large volume and needs to be merged in memory by the processor with incoming "housekeeping" data flowing into the system from other I/O channels. This merged data needs to written to disk in a safe and timely manner - a rate which exceeds single disk sustained write data rates.

Device Interrupt Latency Although incoming "housekeeping" data is not high volume, it does arrive at high frequency via a serial interface that is typically interrupt driven. The system can never be so busy with other tasks as to miss servicing one of these interrupts before the next interrupt arrives, otherwise data is lost.

Time Synchronization There are multiple independent devices (optics, GPS, orientation, etc.) taking data samples that need to be correlated and rectified with the notion of a common clock.

2.1. Bandwidth/Throughput

The data path between the optical unit and the on-board computer is the most difficult path to analyze/predict since it depends on many different factors. The following items have to be considered:

- Twice the optical data rate is needed on the PCI bus since the data is first transferred into on-board memory and then again from memory to the disk controller.
- In order to achieve PCI transfer rates close to the theoretical maximum, DMA (direct memory access) and long data bursts have to be implemented. This allows devices to read and/or write directly to on-board memory without needing control of the on-board CPU - allowing I/O operations to "overlap" with computation or with one another.
- DMA transactions on a PCI bus can be interrupted by I/O and event requests. To achieve maximum throughput, such interruptions have to be minimized e.g. by limiting the number of PCI devices on the same bus and/or by limiting the rate at which interruptions occur.
- A CPU board has a certain maximum I/O capacity which depends on the CPU, the I/O chipset and the type and speed of the main memory. All running applications as well as the OS (operating system) itself already use a part of that I/O capacity. If the OS and/or the applications generate heavy I/O or memory traffic, the throughput of the PCI subsystem might be affected.

In order to address the above items, a CPU board was chosen which includes dual processors, up to 4GB DDR RAM, two integrated ethernet interfaces, and - most critically - three independent PCI buses. This allows us to place the optic data PCI board and the PCI disk controller board each on a bus by themselves - using the third bus for all other peripheral I/O. The PCI hub of this board's I/O chipset would theoretically allow 500MB/s bandwidth for each of 2 PCI buses during critical data transfer, which should satisfy the 65MB/s per device requirement with capacity to spare.

Next, we looked at the sustained write performance of our disk drives. It was determined that in order to achieve the target data rate, the video data stream needed to be split up and written to multiple drives in parallel on the raw devices (i.e. no filesystem).

Armed with this information, we devised a simple experiment to test data throughput along the complete data path – two small C programs ($\approx$ 300 lines total) implementing a typical producer/consumer scenario. The producer DMA transfered data originating from an evaluation board (in place of the not yet completed optical board) on the first PCI bus to CPU shared memory. The consumer program consisted of multiple writer threads that concurrently read from the shared main memory buffer, then wrote the data through the disk controller on the second PCI bus to multiple drives in parallel. The CPU shared memory was partitioned into multiple semaphore protected buffers to allow overlapping (i.e. asynchronous) I/O.

This test was run with 5 writers to 5 disks over the entire capacity of the disks, and the throughput was measured every 5 frames. We were surprised to see such a large variation in performance depending on which area of the disk was being written to. However, the maximum performance measured was 115.1 MB/s and the minimum was 85.7 MB/s – comfortably above our target of 50 MB/s. Even better, the CPU load during the test showed 65% idle time and 35% system time (and 0% user time – no other applications were running).

When this experiment proved successful, we proceeded to implement this concept as a Tcl-based prototype for the system - using the same asynchronous I/O strategy based on shared memory and semaphore (this time, through the `svipc.so` [4] Tcl extension). We kept the single producer process (labeled *Data Acquisition* in figure 2) but used multiple processes for the N writers, instead of a single multi-threaded process. Since the N writers are created once and long-lived, there is no performance penalty in making this simplification. To complete the initial prototype, we needed only to develop our own Tcl extension (`veu.so`) for accessing the optical interface.

A final non-critical data throughput issue involved a component of the GUI. While the instrument is acquiring image data, it is required to simultaneously deliver what is known as a *waterfall image* to the operator console. This mechanism should select 3 of the 300 incoming spectral data channels and re-sample them to 8 bits per channel to represent a false color RGB (Red, Green, Blue) composite image. This allows the operator to use visual cues to verify that the mission is correctly following the intended flight line. Since this is a non-critical feature, it was determined that: (1) we can choose to only display every 1 of N lines, if desired and (2) we are allowed to use the unreliable UDP protocol (through the `udp.so` [5] Tcl extension) to transmit waterfall image data to the client since we don't care if image data is lost before it is read by the operator's browser.

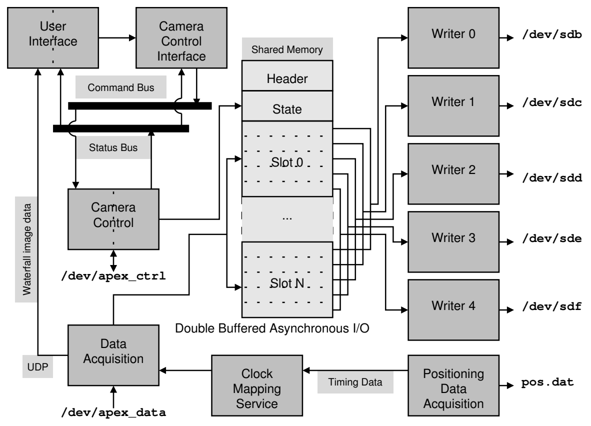

Fig. 2. Subset of Data Acquisition Architecture

We initially saw some performance degradation with the Tcl version, but were later satisfied after modifying Kelsey's memory-to-disk write routine (`shmwrite`) to do a single large write instead of multiple 4K writes.

2.2. Device Interrupt Latency

A second critical engineering problem to be addressed was the servicing of high rate hardware device interrupts. The particular problem was that we expected high rate *housekeeping* data (25 times per second) to arrive over an RS-422 serial interface – a device typically serviced via an interrupt mechanism. There were two reasons to be wary of this issue. The first is mentioned above – we expected to make use of DMA and large burst I/O to fulfill our bandwidth/data throughput requirements and servicing device interrupts at a high rate is detrimental to this goal. The second reason to be concerned was our stated goal of trying as much as possible to keep all implementation in high level scripting code. One well-known scripting rule of thumb is to avoid scripted code in inner loops and critical performance sections - these things are better implemented in lower level code appearing as higher level primitives (i.e. ”fat calls”) at the scripting level.

While the software architects were considering how this tricky situation could be resolved – maybe by experimenting with a complicated polling device driver – an elegant solution was proposed by one of the hardware engineers. Apparently they had enough FPGA processing and memory budget remaining from the custom optical interface card that they could implement an RS-422 interface with a large on-card buffer and a programmable interrupt trigger. Even more importantly, the high rate housekeeping data could be rerouted through the optical interface masquerading as an additional fake spectral channel. This is beneficial not only for data latency and throughput reasons but also because it associates housekeeping meta data directly with the frame to which it belongs thereby reducing part of the clock synchronization problem. The only remaining data coming through the RS-422 to generate interrupts is now low rate control status i.e. command responses resulting from state change requests sent by the on-board computer to the instrument.

2.3. Time Synchronization

The third critical engineering problem to be addressed was how time synchronization should be handled between the various independent data delivery components. One of the synchronization problems was addressed above by implementing an intelligent RS-422 port. The two remaining timing-related components that need to be synchronized are the now unified optical data and the position and orientation data. In order to obtain meaningful and reproducible data, the spectral information and the position information have to be synchronized (and also serve as a necessary pre-condition for automated ortho-rectification during scientific post-processing).

In the optical component, the camera sends a line sync event to the optical interface card at the start of every image line acquired. It was determined that the best way to obtain low latency high resolution timing data was to tag the data directly in the optical interface card as it arrives from the camera (rather than via the on-board computer). This allows tagging of line sync events within a few nanoseconds. The implementation of this timer is kept simple by using a 64 bit free running counter driven by a quartz-stabilized oscillator. This clock tag is then transferred to the host computer along with the optical data.

It should be noted, however, that there is a certain delay between the optical measurement on the sensor itself and the reception of the corresponding pixel on the optical interface card. This delay is caused by the exposure time, A/D (analog to digital) conversion, serializing and de-serializing of the pixel data, encoding and decoding for the physical layer of the optical link as well as the actual transmission time through the fiber. This delay, as well as its variation has to be characterized during inter-mission calibration and used in scientific post-processing.

The positioning component is connected via ethernet to the on-board computer. The data obtained from the positioning component is already tagged with two different timestamps which can be chosen from four different sources: internal clock, GPS time, UTC time, or *user time*. The latter is calculated by adding a fixed offset to the internal clock. This offset can be set by the on-board computer sending a `time sync` message to the positioning component.

To summarize, optical data is tagged by the optical interface card's clock whereas positional data is tagged with GPS or its own internal clock. It is these two clock bases which must be synchronized. Three different methods for doing this were investigated and in order to improve reliability, it was ultimately decided to implement two of them concurrently. This requires little additional effort since most of the steps of these two methods are the same. These processes are illustrated in figure 3.

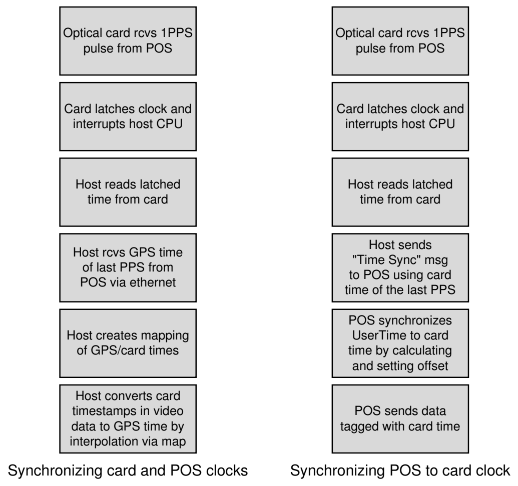
Optical card rcvs 1PPS pulse from POS
Card latches clock and interrupts host CPU
Host reads latched time from card
Host rcvs GPS time of last PPS from POS via ethernet
Host creates mapping of GPS/card times
Host converts card timestamps in video data to GPS time by interpolation via map
Synchronizing card and POS clocks
Optical card rcvs 1PPS pulse from POS
Card latches clock and interrupts host CPU
Host reads latched time from card
Host sends "Time Sync" msg to POS using card time of the last PPS
POS synchronizes UserTime to card time by calculating and setting offset
POS sends data tagged with card time
Synchronizing POS to card clock

Fig. 3. Time Synchronization

2.4. Other features

Once the above three critical performance problems were satisfactorily addressed, it was determined that the system could indeed be mostly implemented at the scripting level. It was at this time that some additional scripting components were defined:

Message Bus Since it was determined that multiple processes would be interested in sending commands to the instrument and multiple (possibly different) processes would be interested in receiving instrument status data from the instrument, the concept of a message bus was developed. Any process wishing to listen to messages on a bus `subscribe` to a particular topic and then receive (asynchronously) any messages `publish`ed on that topic. The implementation makes use of event handlers to process asynchronous messages.

Runtime Configurabilty In order to maintain configuration flexibility, it was decided to implement instrument command sequences (e.g. cool to 70deg K, set filter wheel to position 3, begin recording) as scripts. The flight management system can for example automatically trigger these scripts when it reaches preprogrammed waypoints during a data take. By making these sequences scriptable, the full flexibility of the system is always available to the operator.

Regression Tests Regression tests have been implemented for individual components throughout the development of the system. Additionally, once camera operation became scriptable, it enabled system wide regression tests to also be scriptable.

2.5. Performance

Complete system wide throughput has not been measured since the initial scripted proof-of-concept was validated and put in place. It is planned that the regression testing framework can also be used to maintain a regularly runnable throughput performance test.

The goal of such a test would be to identify the upper limit of the data rate the final system is able to feed through from the detectors to the disks, while doing correctly all the modifications on the data block as described above (merging housekeeping data, rectifying timestamps, extracting waterfall images). Therefore, performance measurements have to determine that upper bound and its governing parameters, while ensuring that data integrity and order is guaranteed.

As outlined above, the on-board computer must buffer data coming from multiple sources and write them to disk via multiple writers. This is a data flow problem similar to determining how long it takes to fill a bathtub when it has a leak.

Note that both incoming and outgoing data rates may vary over time, e.g. due to user programmable frame rate changes or even differing disk zones during writing. The governing parameters therefore are:

incoming data rate governed by DMA block size and PCI bus business

outgoing data rate governed by bus speed and write capacity of the disks

CPU load governed by merging, extracting, rectification tasks on the buffer itself as well as other running applications and the operating system itself.

Therefore, the performance measurements should measure the incoming data rate and use that as the independent variable for all following measurements:

- Measure the outgoing data rate depending on:
  - the incoming data rate (i.e. by DMA buffer size, assuming that we can control the PCI load, since it is a dedicated PCI bus)
  - the size of the buffer (i.e. the number and sizes of slots in shared memory)
  - CPU load
- Determine the failure rate for that same parameter set, calculated from:
  - any deviations of data order on the final system
  - difference between sent-in and written-to-disk data blocks

3. SCIENTIFIC POST-PROCESSING

It is expected that individual flight campaigns will collect data on the order of 100s of GB that need to undergo an offline chain of data correction and characterization processes based on previously acquired and in-flight calibration parameters [6]. This processing chain includes conversion of raw data values into SI units, bad pixel replacement, and correction of smear, straylight, smile and frown anomalies. Higher-level processing is also planned, such as correction of at-sensor radiance values to ortho-rectified ground reflectance considering atmospheric and geometric effects [7], [8]. However, this higher level processing will be addressed in a later phase of the project and the current phase needs only ensure that the parameters and data required for that level of processing are produced and made available.

A simplified block diagram of the planned processing is illustrated in figure 4. The data acquisition process described in the previous section produces the top four components on the left side in the *Raw Data* column. The lower two components are produced during inter-mission calibration of the instrument which takes place in a laboratory known as the *Calibration Home Base.*

At this point all of the raw data is still present in the on-board computer and needs to be transferred to the off-line processing and archiving facility (PAF) computer. During this data transfer, quick consistency checks are made, and some simple constant-time operations can be performed such as bad pixel detection as well as generation of a high-resolution composite RGB pseudo-color *quicklook* image. During this data download phase, some intermediate files are created for use during scientific data calibration processing phase which ultimately produces what is known as the level 1B data product.

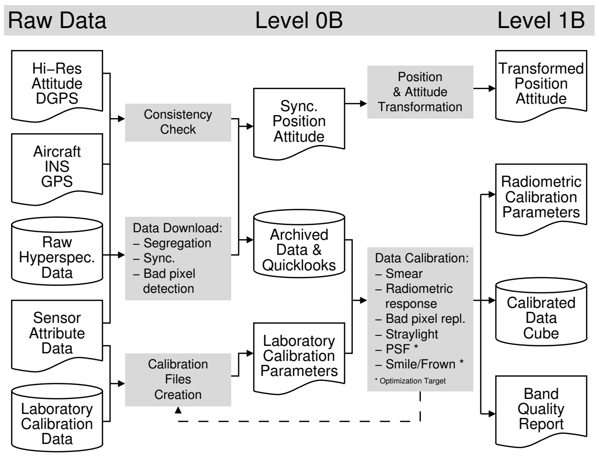
Raw Data
Level 0B
Level 1B
Hi–Res Attitude DGPS
Aircraft INS GPS
Raw Hyperspec. Data
Sensor Attribute Data
Laboratory Calibration Data
Consistency Check
Data Download: – Segregation – Sync. – Bad pixel detection
Calibration Files Creation
Sync. Position Attitude
Archived Data & Quicklooks
Laboratory Calibration Parameters
Position & Attitude Transformation
Data Calibration: – Smear – Radiometric response – Bad pixel repl. – Straylight – PSF * – Smile/Frown *
* Optimization Target
Transformed Position Attitude
Radiometric Calibration Parameters
Calibrated Data Cube
Band Quality Report

Fig. 4. Post Processing Requirements

3.1. Key Algorithms in Foreign Language

Essential post-processing algorithms for hyperspectral data calibration and analysis are usually developed by scientists using special purpose high-level data modeling languages such as TMW's MATLAB and RSI's IDL. Because many of these correction and calibration algorithms are an active topic of research, it was strongly desired to leave scientifically sensitive processing algorithms in their original modeling language as much as possible to facilitate peer-reviewed validation as well as the ability to easily incorporate updates according to the latest advances. But while these languages are often ideal for their domain, they often don't provide convenient interfaces to interesting external software components such as relational database management systems, web servers, and cluster framework libraries.

However, these modeling languages do often provide a C programming interface to their internals allowing them to be embedded into other applications. This is certainly true in the cases of MATLAB and IDL.

A small experiment was developed to investigate the feasibility of embedding an IDL interpreter inside a Tcl interpreter in order to allow program logic to be developed in Tcl, while allowing scientific algorithms to be processed in IDL.

The experiment involved developing an `idl.so` Tcl extension which allowed creation of an interpreter and invocation of commands as strings, and combining it with the `websh.so` [9] Tcl extension in order to implement an interactive web interface to an IDL command line. The user was able to enter arbitrary IDL commands into a form entry screen which was processed by the embedded IDL interpreter and the results formatted in HTML for display in a web browser.

When this experiment proved successful, additional commands for importing and exporting numerical arrays to/from IDL were added to the `idl.so` extension in order to be able to directly pass data to other Tcl extensions.

3.2. Additional Components

The proof-of-concept mentioned above was extended to allow access to an RDBMS (relational database management system) by way of the `mysql.so` [10] Tcl extension to allow for browsing and/or querying the hyperspectral data archive. It was then extended with the `tdom.so` [11] to allow for parsing, manipulation, and writing of XML based meta data for information that is expected to be shared with other (external) processing systems.

It has been determined that even with the large data volumes involved, most of the basic scientific level 1 processing can be performed within the allowed time constraints using a single server class computer. However, it is possible that smile/frown and point spread anomalies that need to be corrected by re-sampling in both the spatial and spectral dimensions simultaneously might require enough processing power to merit the use of cluster processing. Investigation of the particular problems and possible correction algorithms are currently underway. In the case that processing would benefit from cluster computing, experimentation with a Tcl extension to allow access to the MPI-C (Message Passing Interface) library is tentatively planned.

The current version of the prototype kernel is illustrated in figure 5.

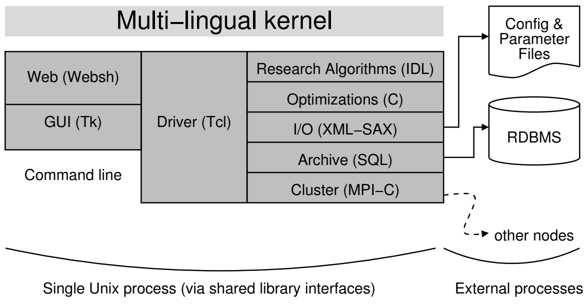

Fig. 5. APEX Post Processing Architecture

An additional ESA driven requirement of the processing system is that key calibration algorithms should be documented and that this documentation should be updated whenever the algorithm is updated. While some argue that IDL is a high enough level language to be self-documenting, this argument doesn't meet typical ESA definitions. Therefore, an IDL documentation system similar to Java's javadoc was developed to allow programmers to embed documentation inside comments in the code itself. This includes standard tags for things such as describing inputs, outputs, and side-effects. However, it additionally allows arbitrary

LaTeXcode to be embedded in a description field so that accompanying mathematical formulas can be easily expressed. Automatically generated documentation from the code is then included into an overall algorithm description document that accompanies the processing software.

3.3. Development Process

Another factor involved in the development of a software architecture for scientific computing is that many of the developers are scientists. Ideally, all software development team members would be equally able to implement all tasks independently and asynchronously – and with roughly equivalent efficiency. However it is more common to find that decomposed subproblems have an interdependence on each other, and that different scientific developers have different development strengths and weaknesses and levels of development efficiency.

One way to mitigate problems in this area and to ensure coherence in the overall design is to adopt a prototype-based iterative development model [12]. The first iteration consists of simulating program flow using a high level prototyping paradigm and subsequent iterations involve refining the simulated steps by gradually replacing them with more realistic pieces.

This is the model we prefer and has driven the architecture and design of all the software described in this paper. Trouble spots are predicted, experiments are developed to investigate them, and then prototypes are initiated accommodating the solutions to the trouble spots while making way for the remaining tasks to be filled in.

Development efficiency, which is different from and often more important than run-time efficiency is also further improved by allowing continued use of multiple development environments from the prototype phase throughout the development process. The key enabling technology for this concept is attempting to arrange automatic interoperability between these different runtime systems. This allows, for example, one scientist to develop a particular calibration algorithm independently from other team members using his most efficient language and development environment (e.g. IDL) while allowing another team member to independently develop the inner loop of a cubic convolution re-sampling algorithm in C or Fortran, and yet another developer to work on web-based form driven GUI code or SQL access to the database management system. If the resulting system has been architected to support interfaces between each of these pieces at runtime, the efficiency of the development and maintenance of the code should be high.

In this way, the development process itself can have a large effect on the software architecture design decisions that are made.

4. CONCLUSIONS AND OUTLOOK

We have discussed the analysis and requirements of system and application software for the APEX airborne pushbroom imaging spectrometer. We described how we analyzed these requirements to develop simple experiments to test critical components of a potential software architecture. Based on successful experimental results, we developed initial prototype software architectures for both onboard data acquisition as well as offline scientific post processing of hyperspectral data.

The initial prototyping phase resulted in the development of key components that are expected to be used without major interface changes throughout the development lifetime of the software. A summary of these components is given in table 2.

scripts	
gencode	code generation from DB schema
idldoc	automatic documentation from code
testsuite	regression test suite
extended	
idl.so	IDL execution and data import/export
mysql.so†	RDBMS client
svipc.so†	Shared memory and semaphore interface
syslog.so	Interface to system error logger
tdom.so†	XML processing
udp.so†	UDP (for waterfall image server)
veu.so	data acquisition via PCI device driver
websh.so†	Web application framework
embedded	
apex	Hardware control command language
msgbus	Message bus library

†Already existing extensions

Table 2. Uses of embeddable/extendible scripting

The data acquisition software will soon be prototyped against a hardware simulator and full data acquisition system integration is planned for the end of 2003.

While the level 1 post-processor has enjoyed a more detailed design specification, its implementation is at a less finished stage of development than the data acquisition system. Many of its components are well-defined and just a simple matter of programming. However, some particular algorithms (e.g. for correction of smile/frown and point spread anomalies) are still in the investigative experimental and prototype phases. Nevertheless, the first functional delivery of the level 1 post processor is scheduled for the beginning of 2004. First end-user flight campaigns for the APEX spectrometer are scheduled for mid 2005.

5. REFERENCES

[1] M. Schaepman, D. Schläpfer, J. Brazile, and S. Bojinski, “Processing of large-volume airborne imaging spectrometer data: the APEX approach,” in *SPIE*

Imaging Spectrometry VIII, vol. 4816, (Bellingham, Washington, USA), pp. 72–79, 2002.

[2] M. Schaepman, D. Schläpfer, J. Kaiser, J. Brazile, and K. Itten, “APEX - airborne prism experiment: Dispersive pushbroom imaging spectrometer for environmental monitoring,” in *Proc EARSel 3rd Workshop on Imaging Spectroscopy*, 2003. To appear.

[3] K. I. Itten, M. Schaepman, L. De Vos, L. Hermans, D. Schläepfer, and F. Droz, “APEX - airborne prism experiment: A new concept for an airborne imaging spectrometer,” in *3rd Intl. Airborne Remote Sensing Conference and Exhibition*, vol. 1, pp. 181–188, ERIM, 1997.

[4] J. Kelsey, “Tcl interface to System V IPC facilities,” *http://www.neosoft.com/tcl/ftparchive/sorted/net/svipc-2.2.0/*, Visited July 2003.

[5] M. Miller, X. Wu, and P. Thoyts, “Tcl UDP extension,” *http://sourceforge.net/projects/tcludp*, Visited July 2003.

[6] M. Schaepman, D. Schläpfer, and K. Itten, “APEX - a new pushbroom imaging spectrometer for imaging spectroscopy applications: Current design and status,” in *IGARSS 2000*, vol. Vol. VII, (Hawaii), pp. 828–830, 2000.

[7] D. Schläpfer, , and R. Richter, “Geo-atmospheric processing of airborne imaging spectrometry data part 1: Parametric orthorectification,” *International Journal of Remote Sensing*, vol. 23(13), pp. 2609–2630, 2002.

[8] R. Richter and D. Schläpfer, “Geo-atmospheric processing of airborne imaging spectrometry data. part 2: Atmospheric/topographic correction,” *International Journal of Remote Sensing*, vol. 23(13), pp. 2631–2649, 2002.

[9] A. Vckovski, R. Brunner, and S. Hefti, “Websh: The Tcl web application framework,” *http://tcl.apache.org/websh*, Visited July 2003.

[10] H. Soderlund, G. Gulik, T. Ritzau, P. Brutti, and A. Trzewik, “mysqltcl - Tcl Mysql interface,” *http://www.xdobry.de/mysqltcl/*, Visited July 2003.

[11] J. Löwer and R. Ade, “tDOM - a fast XML/DOM/XPath package for Tcl written in C,” *http://www.tdom.org*, Visited July 2003.

[12] B. Boehm, “A spiral model of software development and enhancement,” *IEEE Computer*, vol. 21(5), pp. 61–72, 1988.