Component-Oriented Design Studies for Efficient Processing of Hyperspectral Infrared Imager Data

Ray Garcia, Maciek Smuga-Otto

Space Science and Engineering Center, University of Wisconsin - Madison

Background

• The Geosynchronous Imaging Fourier Transform Spectrometer (GIFTS) instrument combines high spectral resolution soundings associated with Fourier Transform Spectrometers (FTS) with high spatial and temporal resolution, creating three-dimensional near-real time views of atmospheric radiance, temperature, water vapor, and winds.

• The volume and rate of data sent down by such instruments poses a considerable design challenge for the ground processing system: to keep up with the data in real-time while maintaining a flexible architecture. This software must be largely reusable for validation and product delivery for future instrument systems such as the GOES-R Hyperspectral Environmental Suite (HES).

• The University of Wisconsin Space Science & Engineering Center is working with NOAA to overcome the long-term challenges of these terabyte-per-day scale instrument systems, applying and extending domain knowledge on distributed, real-time, and object-oriented software systems in order to maximize the positive impact of future imaging FTS instruments in meteorological remote sensing.

System Requirements

• Throughput (bits per second) of a full-scale system must meet or exceed that of the instrument. GIFTS produces 128 x 128 pixels x 6144 samples x 16 bits / 11 seconds = 139MByte/s uncompressed data.

• Reproducibility ensures that data products from the system have established truth value by allowing data to be traced back to its origins.

• Latency (seconds) of system must meet user requirements for timeliness. GIFTS data must be processed to atmospheric profiles and meteorological products (Level 2) within 15 minutes of arrival.

• Reliability (percent) describes the acceptable loss of system function over a time period.

• Robustness - the system handles likely component failures without incurring downtime.

• Testability - the system components can be proven prior to final system integration.

• Maintainability - the design can be adapted to changes in implementation, limited changes of requirements, and changes in staffing.

Use Cases Supporting Data Processing

(listed by component)

Use Cases for System Controller ('Master')
Accept metadata tokens for observation data from Input Delivery component, and supplementary data updates from the Reference Database.
Match observations with supplementary data and pipeline configurations to create jobs.
Assign jobs to Pixel Pipeline instances using cost estimation heuristics.
Dispatch work orders to subsystem components in order to complete jobs.

Use Cases for Pixel Pipeline ('Worker' nodes)
Accept observation data, supplementary data and job requests.
Sequence algorithm stages to generate products from observations.
Forward products to Output Delivery component, send audit metadata to Audit Database.

Use Cases for Reference Database
Reliably retain ephemeral supplementary data for the symbols needed by the Pixel Pipeline.
Distribute reference data structures to another component upon request.
Permit updating of reference symbol entries and values, with configuration control for each symbol.
Permit creation of new reference symbols within the Reference Database, to support additional pipelines with alternate sets of algorithms.
Notify System Controller and Audit Database of updates to reference symbols.

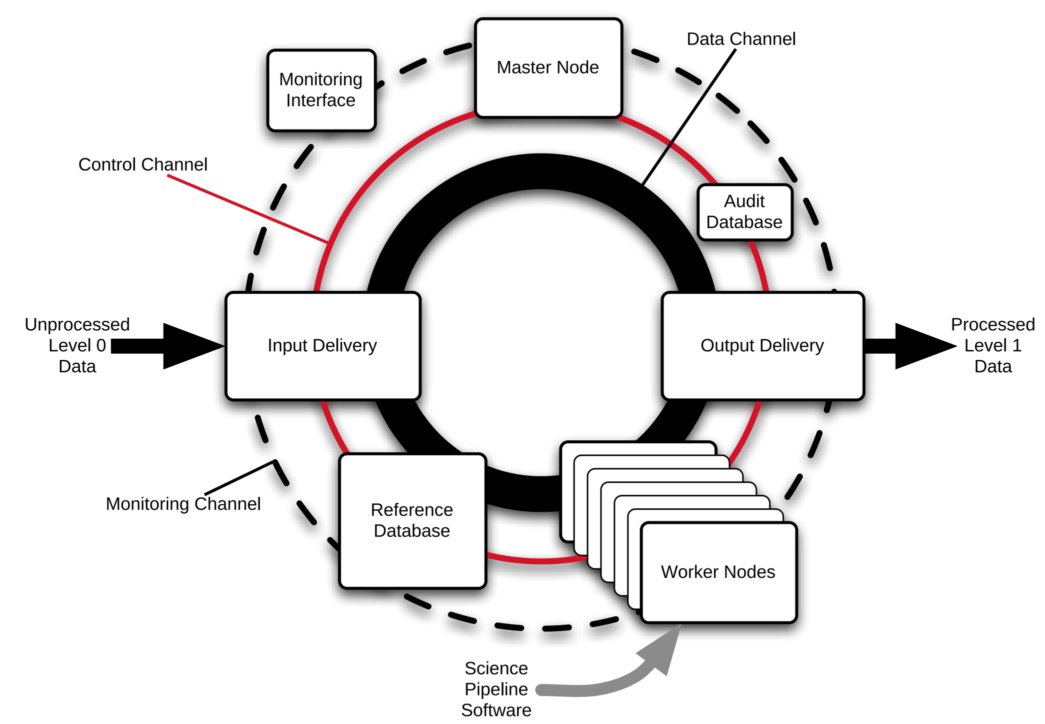
Use Cases for Audit Database
Record metadata transactions describing data processing and supplementary data updates.
Produce detailed data processing reports organized by time, components used, or products.

Use Cases for Monitoring Component
Accept asynchronous monitoring events from other subsystems.
Summarize system activity and exception status for human operator.
Log exceptions and performance statistics.

See related materials on the web at:

http://www.ssec.wisc.edu/gifts/noaa/l0l1.html

Support for this work was provided by NOAA federal grant NAO7EC0676



System component diagram

Software Components

• The system is recursively described as a set of **capsules** exchanging **signals** obeying **protocols** through specialized **connectors**.

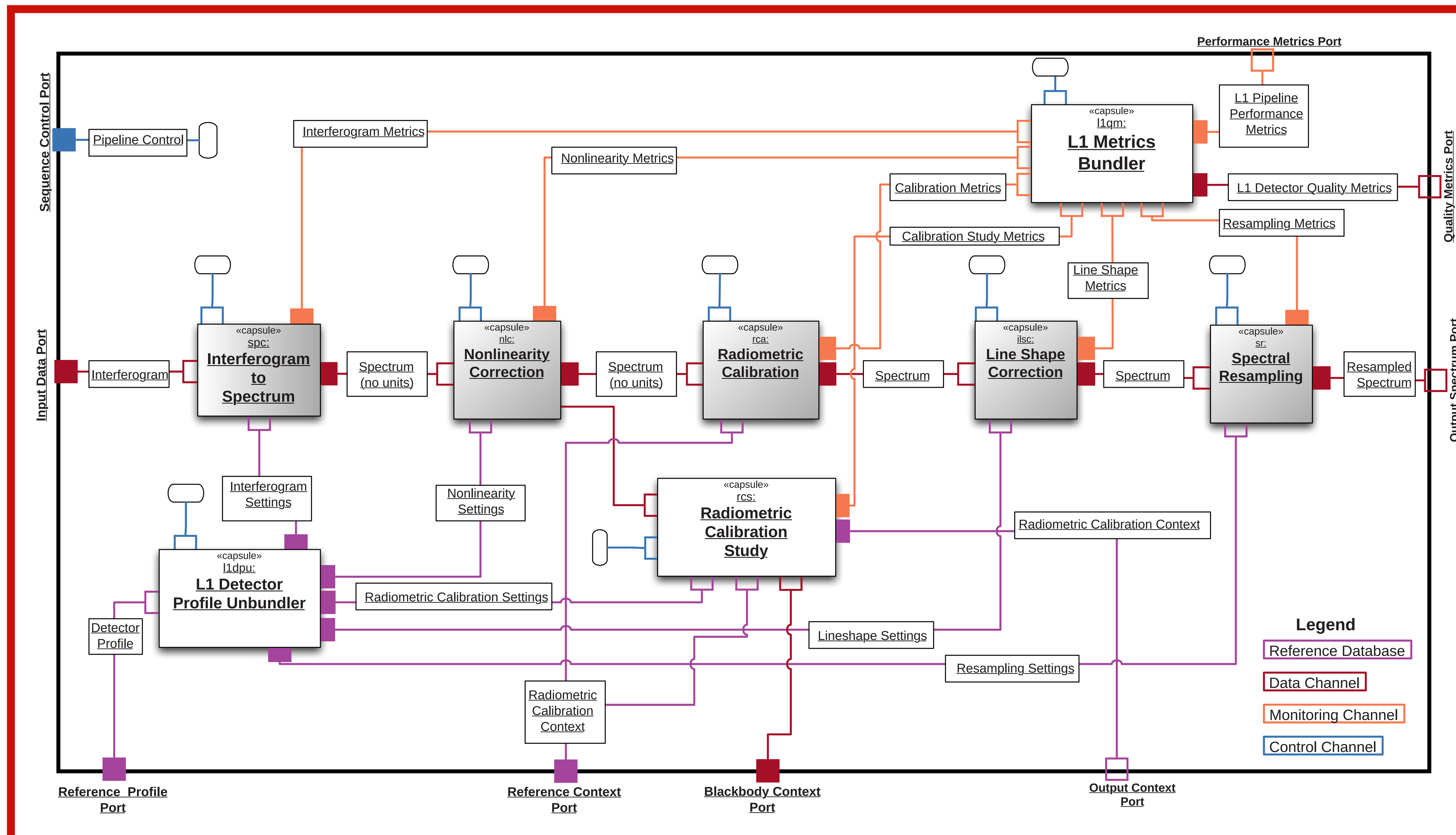
• Functional requirements decompose into capsule requirements and specifications which guarantee system behavior.

• De-coupled interfaces between subsystems, including state-flow descriptions, aid software development and testing.

• Resource budgets are balanced to meet time, space, and accuracy requirements with a tractable implementation.

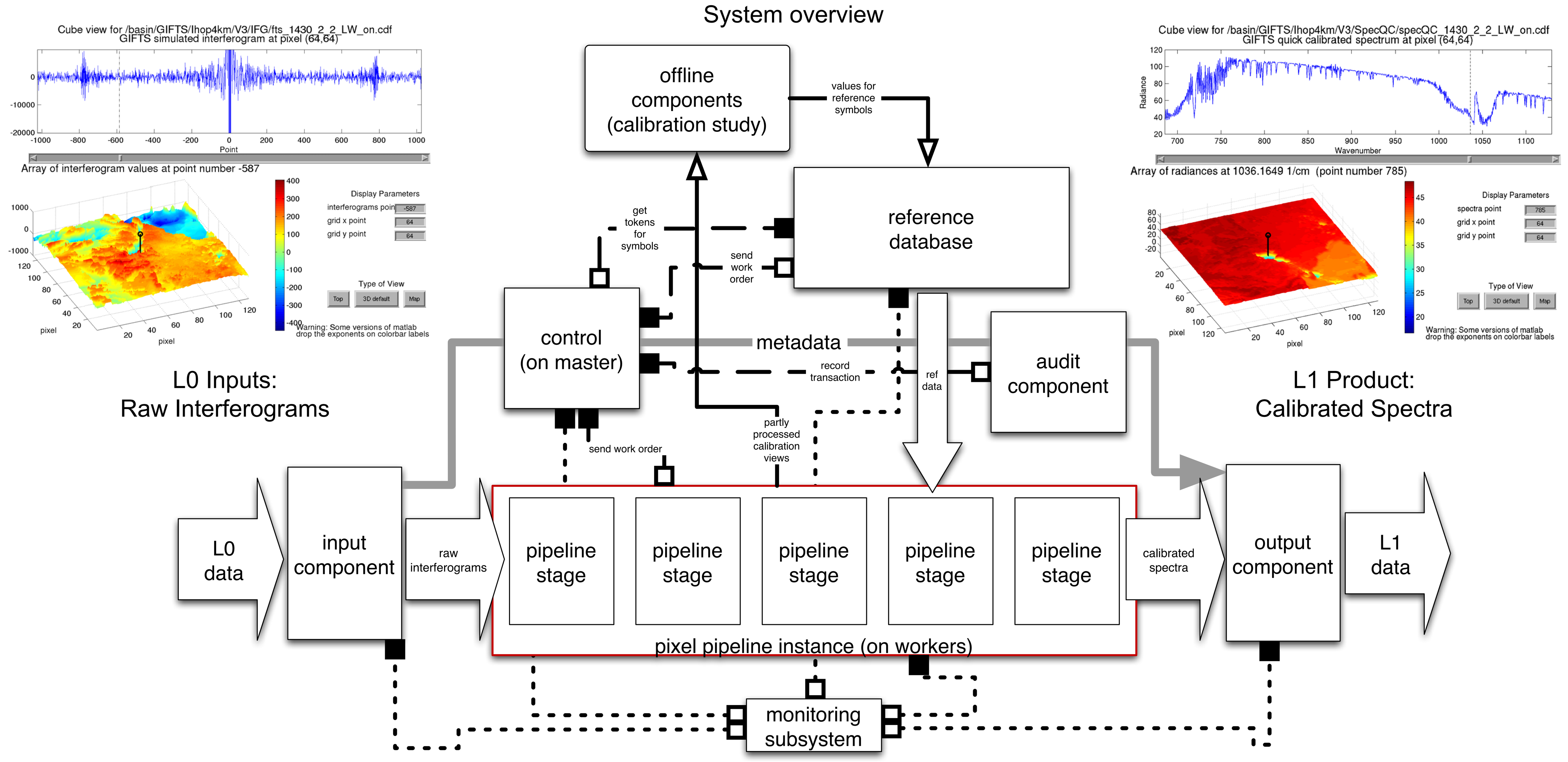
• The **UML-RT** diagrammatic vocabulary permits design decisions to be communicated independent of implementation technology.

• Component methodology is ideal for distributed software design of time-constrained systems.



Detailed UML-RT Component Diagram of L0-L1 Processing Pipeline (per pixel)

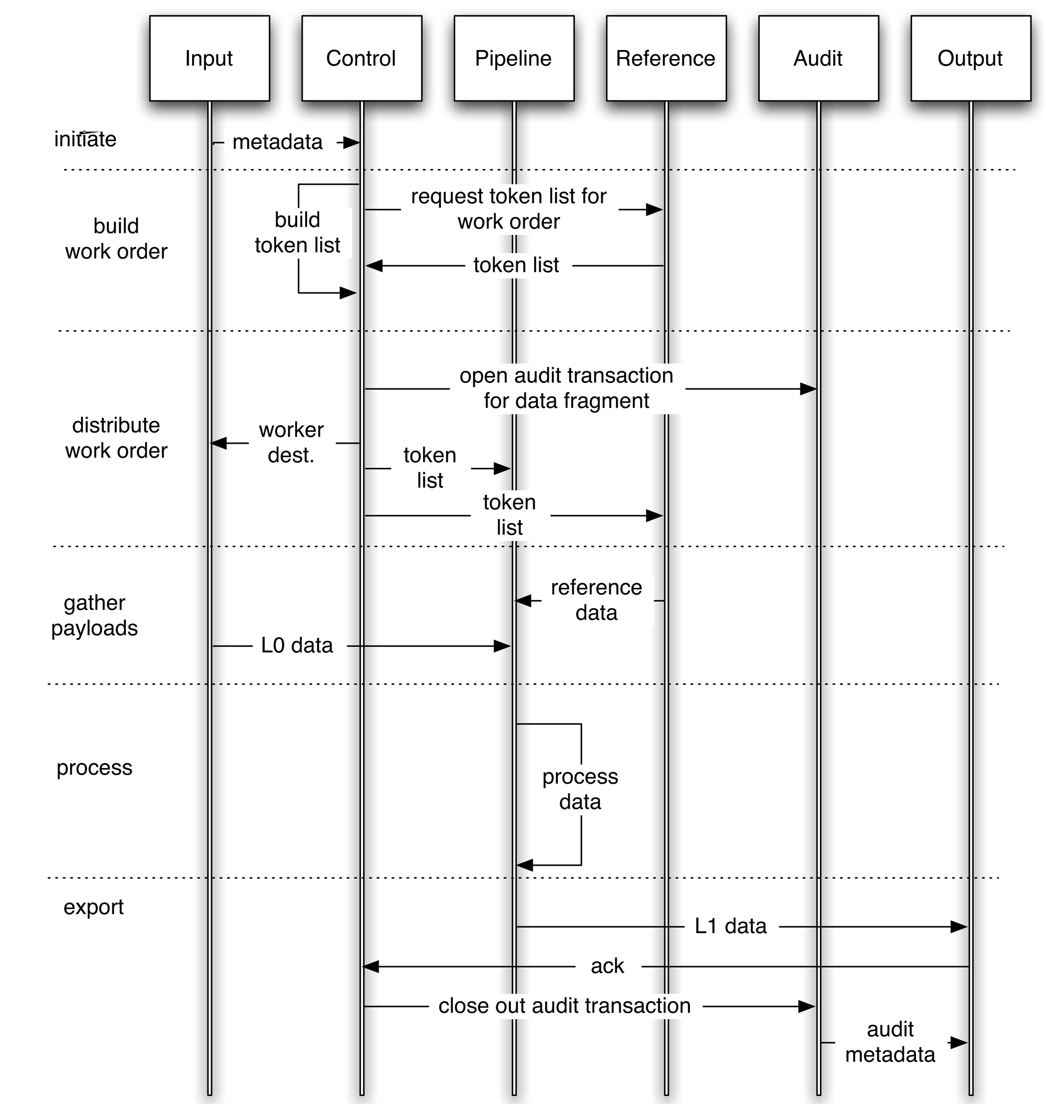
System overview



L0 Inputs:
Raw Interferograms

L1 Product:
Calibrated Spectra

Sequence Diagram for the Pixel Data Processing Use Case



Supplementary Data

• **Reference data** contains characterization and calibration parameters needed by pipeline algorithms that change with significantly lesser frequency than observation data.

• **Metadata** describes larger data entities. It provides information to the software components that is required to correctly process, store, and analyze the data it represents.

• **Audit data** is a specialized subset of metadata describing the origins and history of a product.

• **Monitoring data** is live metadata for the computing system, representing its performance characteristics and significant events such as faults or failures.

Design Considerations for Deployment on Cluster

• Distributed multiprocessor systems (i.e. clusters) are an effective solution for processing large arrays of nearly independent spectra. Ensuring that the system design and implementation be "cluster-ready" is a priority.

• Split groups of pixels off to individual worker nodes containing one or more processing pipelines.

• Cache, proxy and delegate capsules may be inserted between distributed subsystems in order to account for comparatively slow link between node CPUs.

• Anticipate that some connectors between capsules will be implemented using middleware such as ACE, CORBA, or hardware-specialized transport APIs.

• Avoid bottlenecks by ensuring that the subsystems manipulate metadata proxies of large data structures where feasible.

• Ease implementation of system robustness by gathering and checkpointing system state.

Research cluster at CIMSS