JOINT FRAMEWORK PROJECT:
A FLIGHT TEST DATA PROCESSING SYSTEM

John J. Modi, Tony L. Essman, Douglas H. Brandon, Joe W. Waller,
Robert S. Hester, Fern L. Pham, and Vien X. Bui
Flight Test System and Services
Lockheed Martin Enterprise Information Systems
Fort Worth, Texas

Dan C. Green and Mark G. Kerzie
Flight Test Engineering/Instrumentation & Data Systems
Lockheed Martin Aeronautics
Marietta, Georgia

ABSTRACT

The Joint Framework Project (JFP) is an effort to conjoin the software data processing pipeline frameworks between Lockheed Martin's Flight Test Data Centers. The JFP integrates the existing Data Processing Framework (DPFW) with the Joint Enterprise Test System (JETS) data products concept of pipelines. The JFP is constructed with simple governing concepts of data pipes and filters, engineered to manage post flight and real time test data for LM Aeronautics Flight Test mission support, and the results are presented here. The JFP is an Object-Oriented dynamically configurable framework that supports LM Aeronautics Flight Test programs. The JFP uses the Adaptive Communications Environment (ACE) framework, an open source high-performance networking package, to implement the components. The joint framework project provides a real time and an interactive / background post flight test data processing environment reproducing MIL-STD 1553, ARINC 429, Pulse Code Modulation (PCM), Time Space Position Information (TSPI), Digital Video, and High Speed Data Bus (HSDB) data streams for flight test and discipline engineers. The architecture supports existing requirements for the flight test centers, and provides a remarkably flexible environment for integrating enhancements. The JFP is a collaborative effort consisting of LM Aero Flight Test software teams at Marietta, Fort Worth, Edwards Air Force Base, and Palmdale. A prototype will be presented of the JFP addressing the data specific treatment of demultiplexing, decommutation, filtering, data merging, engineering unit conversion, and data reporting. An overview of the distributed architecture is presented, and the potential for the JFP extensibility to support future flight test program requirements is discussed.

INTRODUCTION

The Joint Enterprise Test System, or JETS, is an innovative software package used to process real-time as well as post flight test data, specifically MIL-STD-1553 and Pulse Code Modulation (PCM) telemetered data. The package was originally designed to replace the legacy VAX-based systems with more powerful PC systems that over the last decade have dramatically increased throughput for high-volume data processing. The system also provides attractive pricing as well as flexibility in setting up flight test processing centers with smaller systems.

JETS works with raw test data from a data file containing acquired data from various inputs on board a jet aircraft, including data such as velocity, altitude, and other systems instrumented with data acquisition equipment. It creates manageable reports used by various engineers throughout the Flight Test community for further analysis. There are essentially two steps to get from a raw data file to the prepared reports. First, the data is converted from the raw bits to human recognizable units of measure (Engineering Units), such as feet for altitude. Second, a filtering process takes place to reduce the data down to a manageable set of parameters within a defined segment of time, or a time slice. This allows the engineer analyzing the data to obtain specific results from requested test parameters.

The post-flight processing for the current incarnation of JETS was originally based on a client-server system. A database is populated with data containing information expected from the flight tape. Specialized equipment is then used to extract a file or set of files from the tape onto an NT File System (NTFS) file for processing with JETS. The flight test engineer then invokes JETS, selects a DPD, or Data Product Descriptor, from a list taken from the database, selects the acquired file, and invokes the Data Product generation routines via the interface.

The client side of this process provides a windows-type interface with the standard menus and Graphical User Interface (GUI) controls, such as button, list and edit controls. JETS provides the tools necessary for the test engineer to create the specific boundaries for generating data products. The flight test engineer defines, from within the client, a DPD, a set of time slices, and any possible overrides to the DPD configuration, such as sample rate. The client communicates with the database server objects to display information from the database to assist the test engineer in defining the boundaries for generating the data products. The client also uses data from the database to localize, or filter, data based on requested parameters and time slices.

The server side of the process interfaces with the database and reads the flight test file or files, sending valid input back to the client. In some cases, data is regarded as invalid due to several indicators. These indicators are checked as data is read into the server so that filtering known bad data can be dropped before it is sent back to the client. This helps to speed some of the processing and helps to keep invalid information from entering into the resulting data products. The server also does the work of interacting with the database to convert data to the Engineering Units.

Though the current incarnation of JETS is a client-server based system, it can be, and in fact in some cases is used on local systems as a complete package. This increases the flexibility of its use in control rooms and flight test processing labs.

The JETS Solid State Release is meant to be the next logical step in processing flight test data. It is being designed for use on distributed systems to help expedite throughput as well as flexibility in how it is configured.

The Joint Framework Project (JFP) is designed and developed to support web-enabled Object Oriented components to the Solid State Digital Data Recorder Information Technology. The JFP uses the Adaptive Communications Environment (ACE) framework, an open source high-performance networking package, to implement the components. The JFP provides a real-time and post flight test data processing system reproducing MIL-STD 1553, Pulse Code Modulation (PCM), Time Space Position Information (TSPI) Video Data, and High Speed Data Bus (HSDB) telemetry streams for flight test and discipline engineers. JFP components support the existing Lockheed Martin Data Processing Framework (DPFW) architecture and provide tools for enhancements to support unique program requirements. DPFW is a collaborative effort consisting of LM Aero Flight Test Marietta, Fort Worth and Palmdale.

The purpose of these efforts is to continually minimize the time required to provide a usable data product. The ultimate goal is to synthesize walk away data from a real-time operation and minimize turnaround from post flight data sources. Additional attributes of the JFP include data merging from multiple data sources and data types.

DISCUSSION

Lockheed Martin Flight Test Data Processing's primary responsibility is to provide low cost, timely, accurate, and reproducible flight test data reports for use in engineering test and evaluation of aircraft. To accomplish these tasks Lockheed Martin is moving away from the expensive, proprietary systems, and moving towards common and compatible software solutions. Reliability, high performance, portability, low maintenance and ease of extensibility are the key objectives.

ARCHITECTURE

Flight test data is digitized and recorded on the test article, and sometimes sent to a ground station via telemetry, during the test. This data is most often highly multiplexed into a single signal (stream) that needs to be de-multiplexed and interpreted before it can be used. The process of de-multiplexing and interpreting the data is what we are addressing with this software.

Our goal is to produce a platform independent component framework that will provide a rich toolset for doing flight test data reduction tasks which include: de-multiplexing multi-stream file formats (Chapter 10, CCSDS, etc.), PCM decommutation, decoding of specially coded data like IRIG Chapter 8 MIL-STD 1553 and ARINC 429, applying Engineering Unit calibration and the merging of dissimilar data types and formats. These components could then be used to work together to do common data reduction tasks. The Joint Framework Project is the result of this effort.

The Joint Framework Project is built upon the idea of combining pipes and filters with a high-performance multi-threaded framework. The result is a remarkably robust, resilient, and scalable system.

The notion of treating data streams as pipes has a successful pedigree in UNIX. It is a natural paradigm for processing flight-test data. Filters provide a mechanism for data transformation. The system can be thought of as a pipeline: various filters connected by pipes. Raw flight-test data enter at one end, and engineering reports come out the other.

The plumbing is provided by the ADAPATIVE Communications Environment (ACE™), a high-performance, multi-threaded framework. This is what ACE™ says about itself on the ACE™ website (http://www.cs.wustl.edu/~schmidt/ACE.html):

The ADAPTIVE Communication Environment (ACE) is a freely available, open-source object-oriented (OO) framework that implements many core patterns for concurrent communication software. ACE provides a rich set of reusable C++ wrapper facades and framework components that perform common communication software tasks across a range of OS platforms. The communication software tasks provided by ACE include event demultiplexing and event handler dispatching, signal handling, service initialization, inter-process communication, shared memory management, message routing, dynamic (re)configuration of distributed services, concurrent execution and synchronization.

ACE is targeted for developers of high-performance and real-time communication services and applications. It simplifies the development of OO network applications and services that utilize inter-process communication, event demultiplexing, explicit dynamic linking, and concurrency. In addition, ACE automates system configuration and reconfiguration by dynamically linking services into applications at run-time and executing these services in one or more processes or threads.

COMPONENTS

The Joint Framework Project (JFP) is a collaborative effort between Marietta, Palmdale, and Fort Worth flight test software teams to produce reusable components for processing flight test data. The components are based on processes atomic to flight test data processing.

Within the JFP, the high-level reusable components are called *pipe components.* Each pipe component implements a uniform interface that allows it be connected to compatible components. Flight-test data are processed by constructing pipelines using pipe components. The pipeline geometry determines how the data are processed. The following figure illustrates the various types of pipe components.

PIPE COMPONENTS

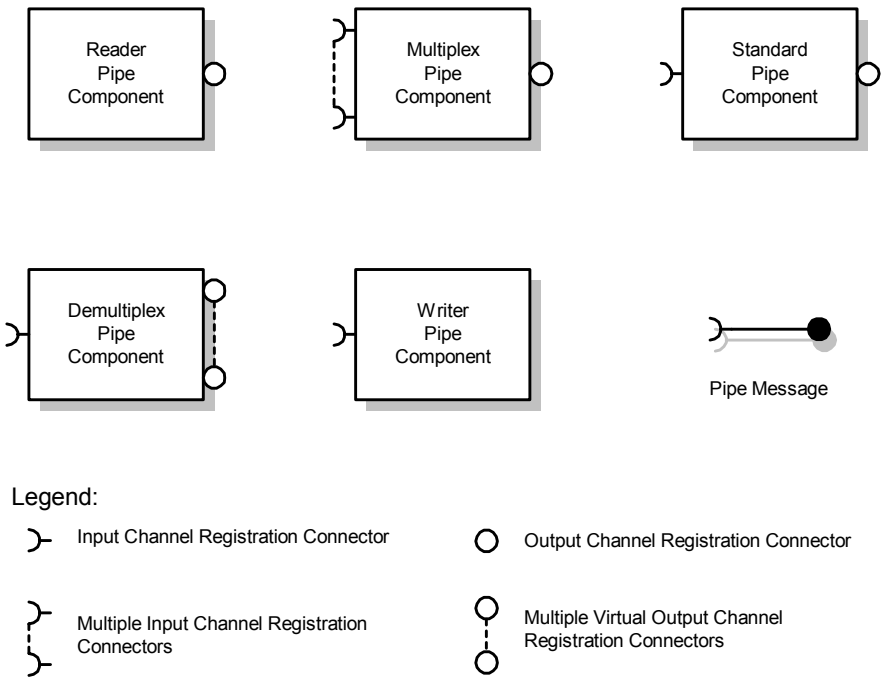

Figure 1

Pipe Components as illustrated in Figure 1:

- Pipe Message. This component is a data structure used to pass messages between components uniformly. All components receive or output pipe messages, except for raw stream readers, which output data according to stream type.
- Reader Pipe Component. This component reads data from a stream, and outputs pipe messages to all components registered on a single output channel.
- Standard Pipe Component. This component receives pipe messages from a single input channel, and outputs pipe messages to all components registered on a single output channel. Most components fall into this category.
- Multiplex Pipe Component. This component receives pipe messages from multiple input channels, merges the messages by time in increasing order, and outputs merged pipe messages to all components registered on a single output channel.
- Demultiplex Pipe Component. This component receives pipe messages from a single input channel and outputs the message to one or more virtual output channels as mapped by configuration. The configuration maps messages to output components analogous to publishing and subscribing.
- Writer Pipe Component. This component receives pipe messages from a single input channel, and writes data to an output stream.

MERGER IMPLEMENTATION

The merger implementation is two or more input streams merged together to form a single input data flow into an EU Engine.

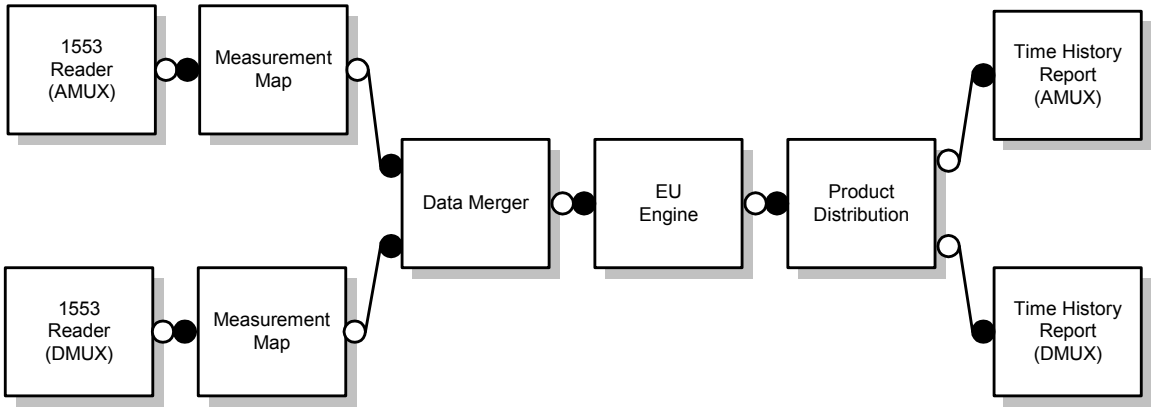

Figure 2

Figure 2 illustrates a typical merged pipeline. It has two input data sources, two measurement maps, a data merger, an EU engine to apply engineering units, a product distribution map, and two writers to produce reports. Table 1 defines each of the components of the merged pipeline.

Component Name	Component Type	Description
1553 Reader	Reader Component	Reads 1553 messages from an input stream. Each instance of this component is configured to read a specific stream such as AMUX or DMUX.
Measurement Map	Standard Component	Extracts selected 1553 command words, data words, and status words from 1553 messages.
Data Merger	Multiplex Component	Merges two or more input streams in increasing order by time.
EU Engine	Standard Component	Converts raw sample measurements by applying one or more algorithms to each input sample.
Product Distribution	Demultiplex Component	Distributes measurement samples to one or more output channels as specified by configuration.
Time History Report	Writer Component	Formats and writes a time history report.

Table 1

PIPE MESSAGE TYPES

As messages flow through the pipeline they may be transformed from one type of message to another. Each Pipe Component must accept the type of message output by a component to which it is connected. Table 2 describes the possible types of messages.

Pipe Message Type	Description
MILSTD 1553 MSG	Encapsulates a 1553 message.
MINOR FRAME	Encapsulates a PCM minor frame.
ARINC 429 MSG	Encapsulates an ARINC 429 message.
MEAS SAMPLE	Encapsulates a measurement sample.

Table 2

TYPICAL PIPE MESSAGE TRANSFORMATION

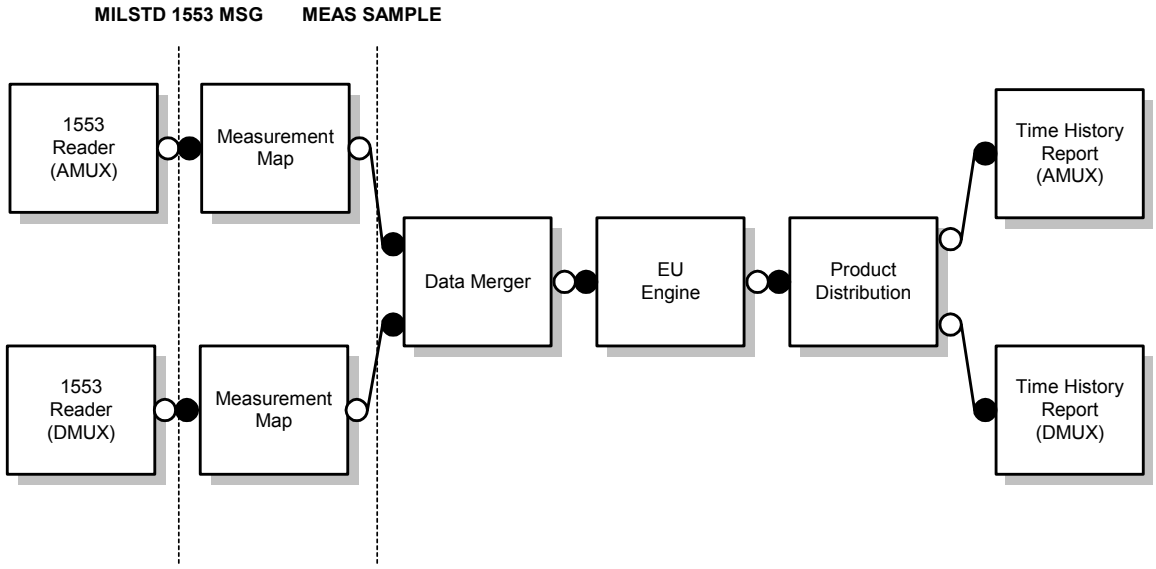

Figure 3

Figure 3 illustrates Pipe Message transformation for the pipeline depicted in Figure 2. The 1553 reader outputs pipe messages of type MILSTD 1553 MSG. The Measurement accepts pipe messages of type MILSTD 1553 MSG, and outputs pipe messages of type MEAS SAMPLE. All other components in the pipeline accept and output pipe messages of type MEAS SAMPLE.

ENGINEERING UNIT CONVERSION

The Engineering Units (EU) Engine applies engineering units to raw flight test data received from an upstream pipeline component, therein converting the raw data. The conversion is accomplished by executing one or more algorithms associated with measurement sample in a specified order. Each algorithm operates on the current value of the associated measurement. By judicious chaining of

algorithms, sample values can be synthesized from sample pieces, scaled, or otherwise operated on mathematically.

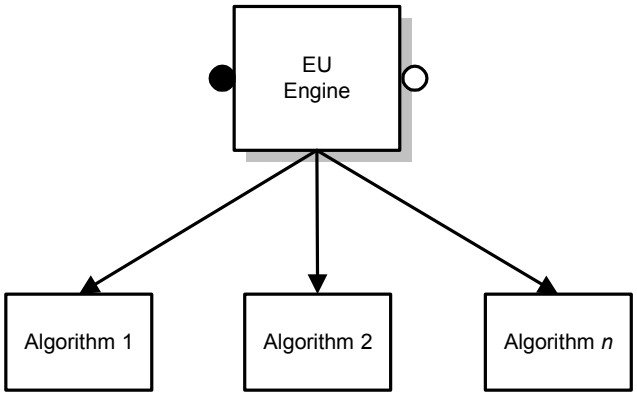

Figure 4

Figure 4 illustrates an algorithm chain within the EU Engine. The EU Engine calls one or more algorithms in a specified order to calculate a sample's EU value. The set algorithms called is referred to as an *algorithm chain*.

An algorithm chain is associated with each measurement. When the EU Engine receives a sample, it executes the algorithm chain for the associated measurement. Each algorithm accesses the current value of the sample, and may modify it. So, generally speaking, the output of one algorithm is the input to the next.

Sample output from the EU Engine does not occur unless the algorithm *OutputSample* is in the chain. Additional functionality is easily added by simply creating a new algorithm and placing it in the proper order in the chain.

Table 3 shows some of the available algorithms:

Algorithm	**Description**
Sign Extend	Extends the sign-bit of a measurement sample's count value prior to conversion to the measurement's data type.
Binary	Concatenates zero or more measurement sample counts, and converts the concatenated counts into the measurement's data type.
Linear	Calculates a sample's value using a linear equation in the form: *offset* + *X*gain,* where *offset* and *gain* are scalars, and *X* is the sample's count.
Bit Concatenation	Extracts and concatenates one or more bits from a sample's count.
Polynomial	Calculates a sample's value using a polynomial equation of arbitrary order in the form: $C_nX^n + C_{n-1}X^{n-1} + C_0$, where *C* represents a coefficient, and *X* is the sample's count.
*n*th Sample	Restricts output to every *n*th sample.
Derived Equation	Applies a previously compiled equation known to the system as a dll or shared object.
Floating Point	Converts the measurement sample's count to the *double* data type.

Algorithm	Description
Convert	
Floating Point Classify	Determines whether or not the sample's count is a valid floating pointer number.
Output Sample	Causes the sample to be output.
Trigger	Causes the algorithm chain for another measurement to be executed using the current sample's value as input to the first algorithm in the triggered chain.

Table 3

PARALLEL IMPLEMENTATION

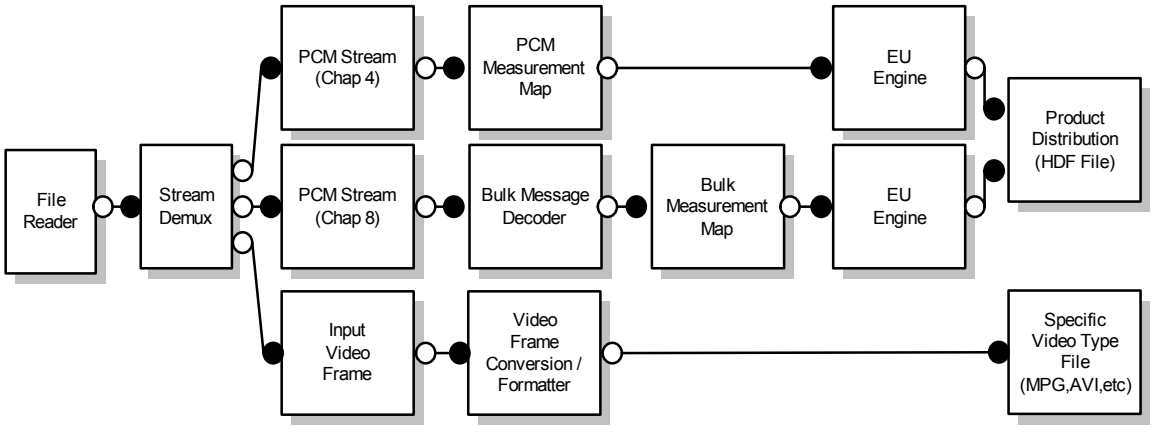

Figure 5

Figure 5 is an example of the parallel implementation. It is dynamically constructed at run time to support varying types of input streams. In this case there are three streams, a chapter 4 PCM stream, a chapter 8 MILSTD 1553 encoded stream and a video stream.

Data is processed through the above example pipe lines and then stored as merged time histories in a common flight test data file (HDF5), and a video file, in this case an mpeg file. This is a common three stream setup; pipes are dynamically added and subtracted to the configuration as data request requirements change. The Product Distribution files (HDF) are used by second step analysis tools that support interactive and high quantity (background) 2nd and 3rd generation analysis and processing (air data, thrust deck, user derived parameters, frequency analysis, data file conversions, report quality plotting, etc.).

DATA PROCESSING ENVIRONMENT

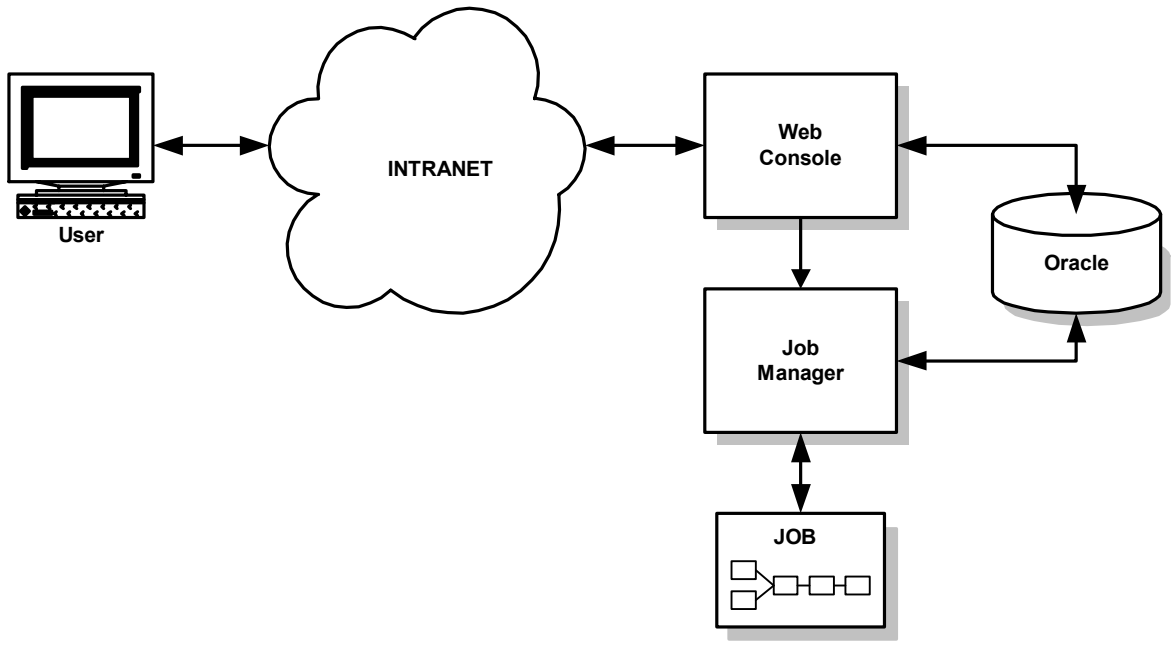

Figure 6

Figure 6 illustrates the relationships in the Data Processing Environment.

Lockheed Martin Flight Test Data Processing packages its pipelines as *Jobs*. A *Job* is a predefined collection of data products. All the information necessary to construct, configure, and execute a pipeline to produce the data products are housed in an Oracle database.

The user schedules jobs through web or desktop applications. These applications initiate the job by making appropriate entries in the database. A continually running service, called the *Job Manager*, monitors the database for submitted jobs.

The Job Manager constructs a pipeline to implement the job, and then starts the pipeline. When execution completes, the Job Manager updates the job status in the Oracle database.

CONCLUSION

The Joint Framework Project has produced a platform-independent framework of reusable components for processing flight test data. The architecture is based on combining pipes and filters with a high-performance multi-threaded foundation. The result is a remarkably robust, resilient, and scalable system.

ACKNOWLEDGEMENTS

Gregory A. Engel, Senior Manager, Lockheed Martin EIS, Lou Ellen Cole, System Integration Analyst, Lockheed Martin EIS, Jimmy R. Scott, Electronics Engineer Senior, Lockheed Martin Aeronautics, Marty M. Larkin , Electronics Engineer Senior, Lockheed Martin Aeronautics.