Lessons Learned
Implementing FIMS 1.0

Ian Hamilton, Tony Vasile
Signiant
Outline

• Introduction
• Background
  • Framework for Interoperable Media Services (FIMS)
  • The FIMS Test Harness Project
• Lessons Learned
  • Test Framework Selection
  • SOAP and REST Support
  • Using Jobs
  • Using Queues
  • Using Profiles
• Summary
Introduction

How can we assemble a system that ingests and prepares content for playout and non-linear delivery based-on flexible interaction with capture, transcode and transfer products from multiple vendors?
Background

Framework for Interoperable Media Services (FIMS)

- Joint AMWA/EBU Initiative
- Vendor-Neutral Common Framework for Implementing Interoperable Media Services
  - Based on Service Oriented Architecture (SOA) principles
  - Interoperable, Interchangeable and Reusable Media Services
Background

*Media Specific Requirements*

- **Long Running Processes**
  - Asynchronous Communication
- **Resource Management**
  - Queuing, Infrastructure Services
- **Media Bus**
  - Media Storage and Movement
Background

FIMS Reference Model
Background

FIMS Resource Model
Background

FIMS Test Harness Project
L1: Test Framework Selection

Service Consumer

Callback Binding

Notification

Service Provider Binding

Job

Queue

Capture

Transfer

Transform

Test Cases .x
ml

Test Data .x
ml

SOAP RPC

REST HTTP
L2: SOAP and REST Support

SOAP

Client

<soap:Envelope>
  <soap:Body>
    <transferRequest>
      <transferJob>
        <resourceID>{jobGUID}</resourceID>
        <priority>low</priority>
        <profiles>
          <profile xsi:type="TransferProfile">…</profile>
        </profiles>
      </transferJob>
    </transferRequest>
  </soap:Body>
</soap:Envelope>

Server

<soap:Envelope>
  <soap:Body>
    <captureAck>
      <transferJob>
        <resourceID>{jobGUID}</resourceID>
        <status>running</status>
        <serviceProviderJobID>{SPJobID}</serviceProviderJobID>
      </transferJob>
    </captureAck>
  </soap:Body>
</soap:Envelope>

REST

Client

PUT /job/{jobGUID}
HOST:{svrHost}
Content-Type:text/xml;
...
+xml version="1.0">
  <transferJob>
    <resourceID>{jobGUID}</resourceID>
    <priority>low</priority>
    <profiles>
      <profile xsi:type="TransferProfile">…</profile>
    </profiles>
  </transferJob>
  </captureAck>
</soap:Body>
</soap:Envelope>

Server

HTTP 1.x/200 OK
Content-Type:application/xml
...
+xml version="1.0">
  <transferJob>
    <resourceID>{jobGUID}</resourceID>
    <status>running</status>
    <serviceProviderJobID>{SPJobID}</serviceProviderJobID>
  </transferJob>
</soap:Body>
</soap:Envelope>
L3: Working with Jobs
L3: Working with Jobs

- ERP Orchestration
  - ERP Job ID
  - Job ID

- Media Orchestration
  - Job ID
  - Transcode Job ID
  - Transfer Job ID

- Transcode
- Transfer
L4: Working With Queues

[Diagram of a queue with attributes and relationships]
L4: Working With Queues

<table>
<thead>
<tr>
<th>Job</th>
<th>Priority</th>
<th>FinishBefore</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job 1</td>
<td>Urgent</td>
<td>12-12-01 11:00</td>
</tr>
<tr>
<td>Job 3</td>
<td>High</td>
<td>12-12-02 11:00</td>
</tr>
<tr>
<td>Job 2</td>
<td>Low</td>
<td>12-12-03 11:00</td>
</tr>
<tr>
<td>Job 4</td>
<td>Low</td>
<td>12-12-04 11:00</td>
</tr>
</tbody>
</table>

Execution Order

Job Priorities
- Immediate
- Urgent
- High
- Medium
- Low
L4: Working With Queues
L5: Working With Profiles
Summary

• The FIMS Test Harness Provides an Easy Way to get Started with FIMS
  – Working with the resource model
  – Exercising calling patterns and bindings

• Applying SOA to Media Helps Unlock the Full Potential of File-based Workflows
  – Seamless interaction between products from multiple vendors
  – Lower implementation costs and improved business agility
Thank You

Ian Hamilton
Signiant, CTO
ianh@signiant.com

For more information on FIMS and the Test Harness Project:

Lessons Learned Implementing FIMS 1.0
Ian Hamilton and Tony Vasile

Abstract
The Framework for Interoperable Media Services (FIMS) is a joint initiative of AMWA and the EBU. The objective of FIMS is to standardize service interface definitions for implementing media related operations in a Service Oriented Architecture (SOA). As one example of how FIMS can be used, a system that ingests and conforms content for playout and non-linear delivery can leverage FIMS to flexibly interact with media capture, transfer and transform products from multiple vendors. As one benefit of using standard interfaces, system components can easily be added, updated and removed in response to changing requirements and demand. This paper discusses lessons learned while implementing FIMS 1.0.
Introduction

The Framework for Interoperable Media Services (FIMS) is a joint initiative of AMWA and the EBU. The objective of FIMS is to standardize service interface definitions for implementing media related operations in a Service Oriented Architecture (SOA). The FIMS 1.0 Specification defines Capture, Transfer and Transform service interfaces and future FIMS updates will define additional interfaces.

The FIMS Test Harness project was initiated to promote adoption of FIMS by providing a means to independently verify FIMS implementations. Lessons learned while implementing and using the FIMS Test Harness are shared in this paper.

Why FIMS?

Moving, processing and storing media using software-centric systems deployed on commodity IT technology is an unstoppable trend in the media industry. Many early software systems, however, suffer from problems similar to those associated with monolithic hardware-centric film and video processing systems. Historically, the lack of standards for interfacing software components has resulted in expensive and inflexible custom integrations. Worse yet, many software based systems depend on watch folders to automate component interactions. This approach achieves loose coupling, but creates a whole new set of reliability and management problems.

While a detailed discussion of software architecture concepts like SOA and abstract service interfaces are out of scope for this paper, the key benefit of these concepts is reduced integration costs and improved business agility. For example, a system that ingests and conforms content for playout and non-linear delivery can leverage FIMS to flexibly interact with media capture, transfer and transform systems from multiple vendors. As a benefit of using standard interfaces to communicate, system components can easily be added, updated and removed in response to changing requirements and demand.

The FIMS reference model identifies the basic components of a SOA based system including the application layer, the process layer, the abstract service layer and the implementation layer. FIMS focuses on the Abstract Service layer which defines the interfaces the orchestration system uses to communicate with the service implementation layer.

Fig 1. FIMS Reference Model
Unlike transaction oriented business systems based on SOA principles, where services typically synchronously process small business messages, media business systems also process large media files and must account for associated requirements. The FIMS service interfaces take into account requirements specific to processing media, such as long running processes and complex resource management. The inclusion of a transfer service is itself recognition of unique media processing requirements and that getting a media asset from one location to another is not a trivial task. Asynchronous communications patterns that utilize jobs to manage state and queues for resource management are also incorporated into the interface to account for unique media processing requirements.

**What was Built**

As members of the FIMS technical board and proponents of test driven development we agreed to lead a project to design and implement a test harness for FIMS. The FIMS Test Harness can be used by vendors implementing FIMS to verify their implementations independently of integrating with third parties. Whether a vendor is implementing a FIMS service (transfer, transform, capture, etc.) or orchestrating a workflow utilizing FIMS services, the ability to perform independent verification is highly desirable.

The FIMS Test Harness provides the ability to simulate service implementations (providers) and service orchestrators (consumers). The behavior of these simulators is data driven, such that test scenarios can be implemented without coding. Test cases for the service consumer execute a sequence of commands against a FIMS service and verify the results. Test cases for service providers specify how the service responds to commands including job management and queuing behavior.

The FIMS Test Harness was used to verify interaction of our orchestration layer with mock services and to verify our transfer service implementation.

**Lesson 1: Testing Framework Selection**

One objective of the FIMS Test Harness project was to minimize the amount of new code created by leveraging an existing freely available web services test framework. Through a series of proposals and discussion amongst members of the FIMS technical board, the decision was made to use the open source “soapUI” test framework for the FIMS Test Harness.

Despite the name, the soapUI test framework provides the ability to implement tests using both SOAP and REST style interaction. While there is much debate over the merits of standards-based SOAP interactions versus conceptually simpler REST style interactions, FIMS aims to be messaging protocol agnostic. The soapUI test allows the mock service implementations and tests to focus on the message payload and not how the payload is conveyed.

One challenge with using soapUI for the FIMS Test Harness was writing fully automated tests to exercise the asynchronous calling pattern defined by FIMS. When a job is initiated via the capture, transform or transfer interfaces, callback URIs can be provided for asynchronous notification of the caller on success or fault conditions. To process the callbacks the caller
exposes callback services. SoapUI tests are easier to write for a typical synchronous communication pattern where services are called in sequence and results are verified after each call.

Verifying that notification callback services are called properly during manual testing is a simple matter of exposing notification services and observing that they are called at appropriate times with appropriate payloads; however, we wanted to support entirely automated testing. As such, we needed to implement a mechanism for a test case to register that a specific callback was required within a specific time frame.

**Lesson 2: Supporting SOAP and REST**

As previously mentioned, one of the objectives of FIMS is to be messaging protocol independent and support both SOAP RPC and REST style interactions. Accordingly, a test harness framework that supports both SOAP and REST interaction was selected. While significant effort went into ensuring that the FIMS 1.0 interface could be implemented through REST style interactions, including specification of a resource-oriented data model, FIMS doesn’t currently specify Web Services Definition Language (WSDL) 2.0 style HTTP bindings. This means that there is no standard set of URIs for identifying FIMS resources. The specification does stipulate that resource representations in REST interactions should utilize XML in the HTTP body.

While work is being considered within the FIMS technical board to define standard resource identifiers, we created our own set of URIs to implement REST bindings in the test harness. As such, the REST elements of the FIMS test harness are not entirely standards based and are dependent on ongoing FIMS technical board activity.

**Lesson 3: Working with Jobs**

The FIMS data model defines a “Job” object to represent operations performed on media objects. The job object captures information about the operation to be performed. The job object specifies queuing and alarming parameters like the relative job priority and target finish time. The job object also identifies the media object (called a Business Media Object or BMObject in the FIMS data model).

Each job object has an ID assigned by the orchestration system. This is the main attribute used to track the state of a job across multiple service calls including asynchronous callbacks. In the FIMS Test Harness we use job IDs to direct mock service behavior. As such, the configuration data for a mock service describes what behavior to exhibit when receiving a given job ID so that the invoking test cases can control the service behavior for given success or failure scenarios.

One job attribute that we found very useful in implementing our transport service was the service provider Job ID. The service provider Job ID facilitates loose coupling of the service provider and orchestration system job tracking implementations. The ability to link independent job IDs is particularly useful when exposing composite services that perform multiple media operations in sequence as allowed by the FIMS interface through use of a composite service profile. In this scenario, multiple orchestration systems with system specific job IDs could be involved. For example, a business-level orchestration system may interact with a media specific orchestration system, which in turn interacts with service implementations.

Another element of jobs that we found particularly useful was the ability to reference multiple BMObjects. The FIMS interface supports the concept of tracking the transitions of media objects throughout a workflow. Each service implementation can accept media
objects to process, and in turn can produce media objects. A reference to the generated media objects is added to the job object upon completion of processing by the service. This functionality allowed us to track the life cycle of media objects throughout a workflow. This functionality also makes it easier to “chain” FIMS components together, where the output from one FIMS service is the input to another FIMS service.

Lesson 4: Resource Management

The FIMS interface supports the concept of queues to manage the order jobs are processed by a service. Each Job submitted to a service has a priority and a finish by time. While queuing is not mandatory and the service specific queueing behavior is somewhat flexible, it is expected that higher priority jobs run before lower priority jobs. The finish by time is an alarming parameter that can be used to raise warnings in the event that a job will not be serviced in time.

These concepts mapped nicely on top of existing resource management capabilities in our transfer service implementation, but the service interfaces also exposed incremental capabilities useful for service maintenance. In particular a queue can be locked through the service interface so that no further submissions are allowed and the queue can drain. This is useful when decommissioning a service. A queue can also be halted through the service interface so that the existing state of the queue is maintained, but no further jobs are run. This is useful for upgrading a service. The service can maintain the existing submitted jobs for continued execution upon resumption of the service.

Lesson 5: Service Profiles

The FIMS data model specifies that capture, transform and transfer service parameters are communicated using a “Profile”. In the specific instance of a Transfer Profile the transfer Destination (or target) is specified as a Uniform Resource Identifier (URI). The media to be transferred is specified by the BMObject associated with the job.

Generically a URI can be an abstract reference to a resource (URN) or a network location for a resource (URL). In order to transfer an object from one location to another on a network, the form of the URI is constrained to a URL. URL schemes supported include ftp, http and file. A FIMS transfer service implementation only needs to support one scheme, so we constrained our implementation to file URLs in order to avoid confusion between the transfer protocol implied by the resource locator scheme and the transfer performed by the transfer service.

The BMEssenceLocator associated with the BMObject can specify a single URI, a list of URIs or a folder/container URI. A URI list can theoretically include different host references but in our implementation we treat this as a fault condition. The URI in the transfer profile conveys the target file system path where copies of transferred files are stored. Although the transfer service was designed with simple transfer protocols like http and ftp in mind, all of the mandatory parameters of our transfer service (including: transfer source host, source files, target host and target folder) are easily derived from the transfer profile and BMObject associated with the transfer job.

Non-mandatory parameters can be specified using profile extension attributes. A FIMS compliant service must not require the use of extension attributes so configurable defaults for all of these attributes are assumed in the event they are not passed as part of the profile. We used extension attributes to pass proprietary transfer template and job group parameters to our underlying transfer service. A transfer template can be configured in the underlying transfer service to control advanced file transfer behavior. The job group can be configured in the underlying transfer service to group related
jobs and control advanced job queuing behavior.

Summary

The FIMS Test Harness project leveraged theoretical and practical FIMS work performed to date and demonstrates the power of FIMS and standardized software service interfaces. Practical infrastructure related knowledge was gained by exercising both SOAP RPC and REST style interfaces and using an asynchronous calling pattern within an existing test framework. Practical experiences with the FIMS resource model were also gained, including exercising some of the less obvious elements of Jobs, Queues and Profiles.

The FIMS Test Harness project was conducted under the AMWA Intellectual Property Rights Policy. All test harness work has been contributed to the FIMS initiative for free use by members of the FIMS community with the objective of removing barriers to adoption and promoting implementation of FIMS compliant interfaces.