

FSA Integration Partner

United States Department of Education

Federal Student Aid



EAI Enhancement Report

Task Order #117

Deliverable # 117.3.1

Version 1.0

August 22, 2003



Table of Contents

1	INTRODUCTION	3
1.1	SUMMARY	3
2	GENERIC WEB INTERFACE.....	3
2.1	CONNECTION POOLING	4
3	MQSI RETIREMENT	6
3.1	EAI TRANSFORMATION ENGINE	7
4	APPENDIX A – TECHNICAL DOCUMENTATION.....	10



1 Introduction

1.1 Summary

The EAI team completed three development efforts which improve FSA's ability to implement and support integration using the EAI technology. They are:

- Generic Web Interface (section 2)
- Connection Pooling (section 2.1)
- MQSI Retirement (section 3)

Each of these efforts were designed, developed, and tested by the EAI team. They were then migrated to the Inter-System Test (IST) environment and used in testing done by our trading partners. After completion of testing in the IST environment, they were migrated to the Production environment.

This document describes the Generic Web Interface with Connection Pooling, as well as the revised interfaces to support MQSI Retirement.

2 Generic Web Interface

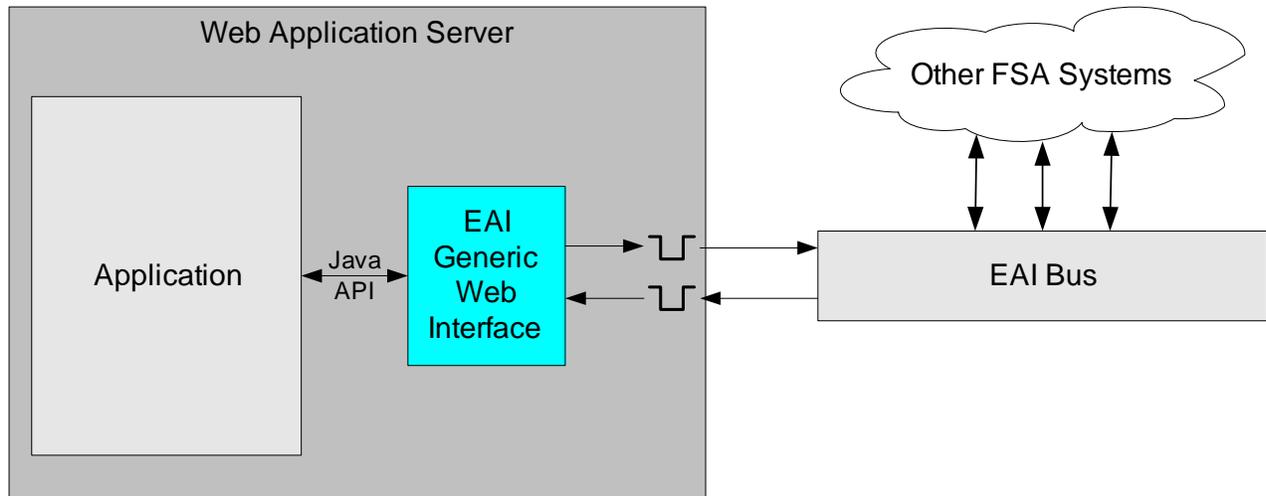
Integration Partner web application teams requiring messaging capability do not often have developers with the skills and knowledge to quickly begin using the EAI architecture. The EAI team needed to provide application teams with a way to take advantage of the capabilities of the EAI architecture without having to learn its technical aspects. To decrease web application development time, improve the quality of web application interfaces, and make interface support easier and more cost effective, the EAI team developed a reusable and scalable interface to the EAI architecture which web application developers can easily use.

The EAI Generic Web Interface is the reusable component that allows Integration Partner developers to access EAI messaging capabilities using a Java application program interface (API). It consists of a set of Java classes, listed in [Appendix A](#), that are available to be incorporated into, and used by, an application to send information and receive responses from another application using the EAI Architecture.

The interface is designed to be simple to learn and easy to use. Application developers do not need to know about the underlying transport mechanism, which is the WebSphere MQ messaging software. No knowledge of the communications infrastructure is required. Developers simply supply the request for information through one of the Generic Web Interface Java APIs defined in the [IEAI interface](#), and the corresponding information is returned.



Below is a simple diagram showing where the EAI Generic Web Interface fits in the over-all architecture:



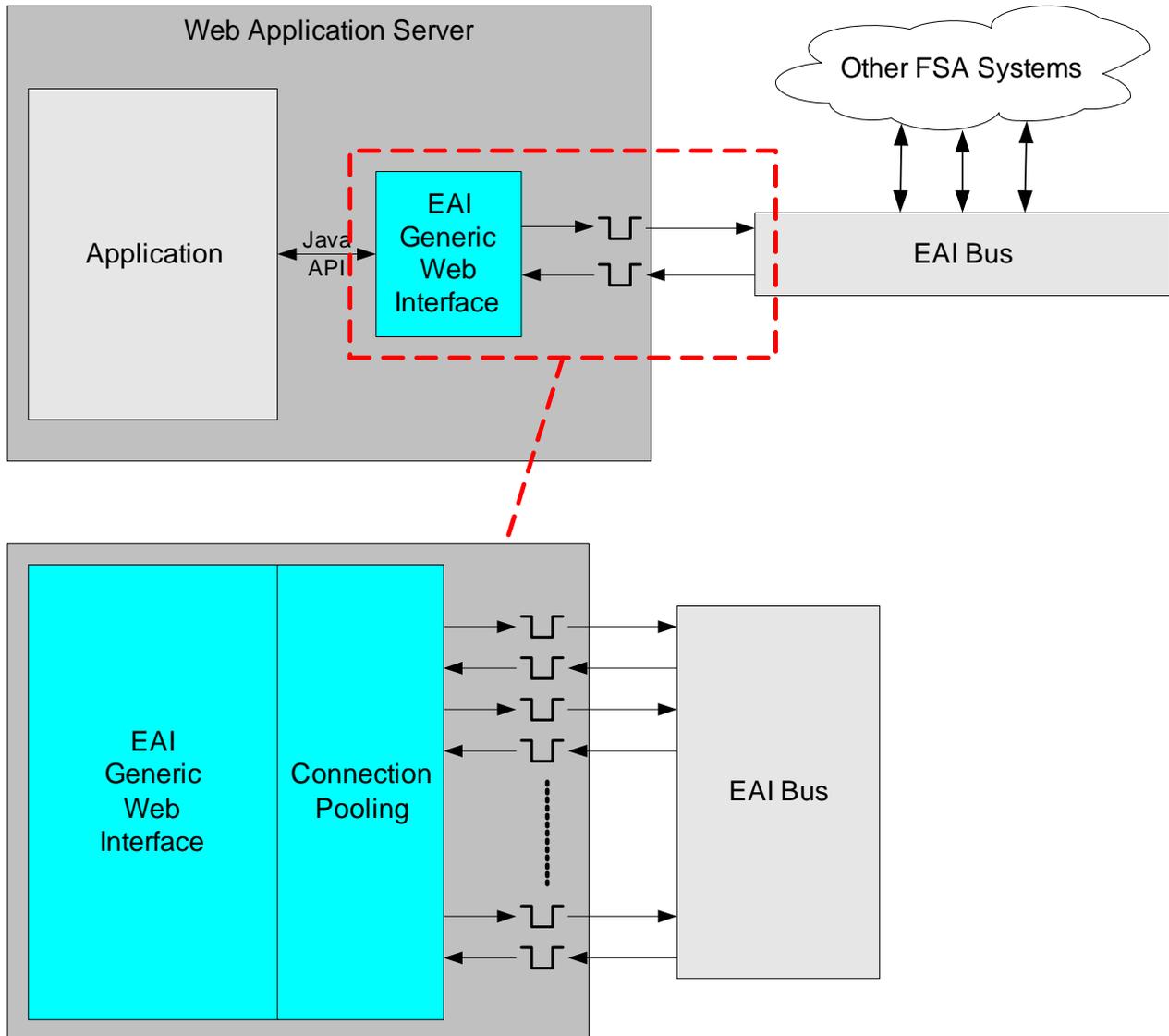
Development and unit testing of the EAI Generic Web Interface with Connection Pooling began on November 4, 2002. It was tested in the IST and Performance Test environments in November and December 2002. It was deployed to FAFSA Production on January 1, 2003 and to LOWeb Production on April 14, 2003.

2.1 Connection Pooling

Some applications require the ability to send and receive large volumes of requests and responses in a short timeframe. For these applications, repeatedly opening and closing one or two connections (MQ channels) will not provide the required interface performance. They need the ability to open and reuse multiple connections in order to process large volumes of requests and responses immediately and in parallel. To meet this requirement, the EAI team added Connection Pooling to the Generic Web Interface design. Connection Pooling is part of the Generic Web Interface, therefore it does not have its own publicly defined interface.

Connection pooling provides an interface capability that reuses WebSphere MQ channels which are kept in a pool. Connection pooling is an enhancement to, and a subcomponent of, the EAI Generic Web Interface. Its purpose is to improve the response time of web interface queries and to increase the number of concurrent requests an application can perform. It enables many, simultaneous, reusable, open connections to the underlying WebSphere MQ messaging infrastructure, which eliminates the need to re-establish a connection with each request.

Below is a simple diagram showing where Data Pooling fits into the EAI Generic Web Interface:



The Connection Pooling enhancement helped FAFSA reach its performance test goals of being able to process over 100 messages per second. It processed the FAFSA Production peak in February and March 2003 of 30-40 messages per second with no issues. For more information on the FAFSA performance testing, see deliverable 102.1.5 FAFSA 7.0/PIN ITA Support Report (correspondence # 03EDU0215).



3 MQSI Retirement

WebSphere MQSI is used in the EAI architecture for transformation of messages on the EAI Bus. For example, Application A may store dates in the format MM/DD/YY, while Application B may use the format CCYYMMDD. If Application A needs to send date information to Application B, MQSI can be used to transform a date such as "08/22/03" to "20030822" so that Application A can send the date in its native format, and Application B can receive the date in its own format. MQSI is a powerful tool with many capabilities, but our requirements of it are limited to its message transformation capabilities.

MQSI uses development tools and deployment processes that are completely different from the ones that the EAI team uses for all other components of the EAI architecture. MQSI requires developers to use a tool called Control Center for transformation development and to control the MQSI Configuration Manager tool, which is used for deployment of transformation interfaces called message flows. MQSI also requires a DB2 database to operate. These tools require EAI developers to have Windows NT servers to develop message flows and to deploy the message flows to the Test environment. They also require a separate Windows NT server in the Production environment for deployment to Production.

Using these tools for Production deployments requires EAI developers to 1) export message flows from the Development/Test Configuration Manager, 2) transfer the exported message flows to the Production environment, 3) import the message flows to the Production Configuration Manager, and 4) deploy the message flows from the Production Configuration Manager. This process includes more manual steps and consequently greater possibility for errors than the deployment process for all other EAI components using FSA's enterprise configuration management tool, Rational ClearCase, for versioning and building code releases.

Another challenge with using MQSI is that the EAI team has found that developers with experience using MQSI Control Center, Configuration Manager, and DB2 are not very common. Developers with Java skills, however, are readily available. Additionally, Java can be developed on a variety of platforms including the developers' own laptops and its code can be checked into ClearCase. A Java solution can follow the build and deployment process that all other EAI components use and it can be debugged, fixed and deployed remotely.

The benefits of a commercial off the shelf (COTS) product such as MQSI are that a COTS product has gone through extensive testing, has a wide base of users, and has the dedicated support of the vendor. These benefits are balanced with the cost of licensing the COTS product and the cost of maintaining personnel with the skills necessary to support it. The risk of developing a custom solution to replace MQSI is greatly reduced by the fact that the EAI solution only attempts to replace the simple capabilities required, rather than all the capabilities offered by MQSI. The risk is also mitigated with thorough testing and by running both MQSI and the custom solution in parallel to verify that their functionality is exactly the same. The EAI team will also leave MQSI installed (but not running) in the Production environment for at least a month after the EAI

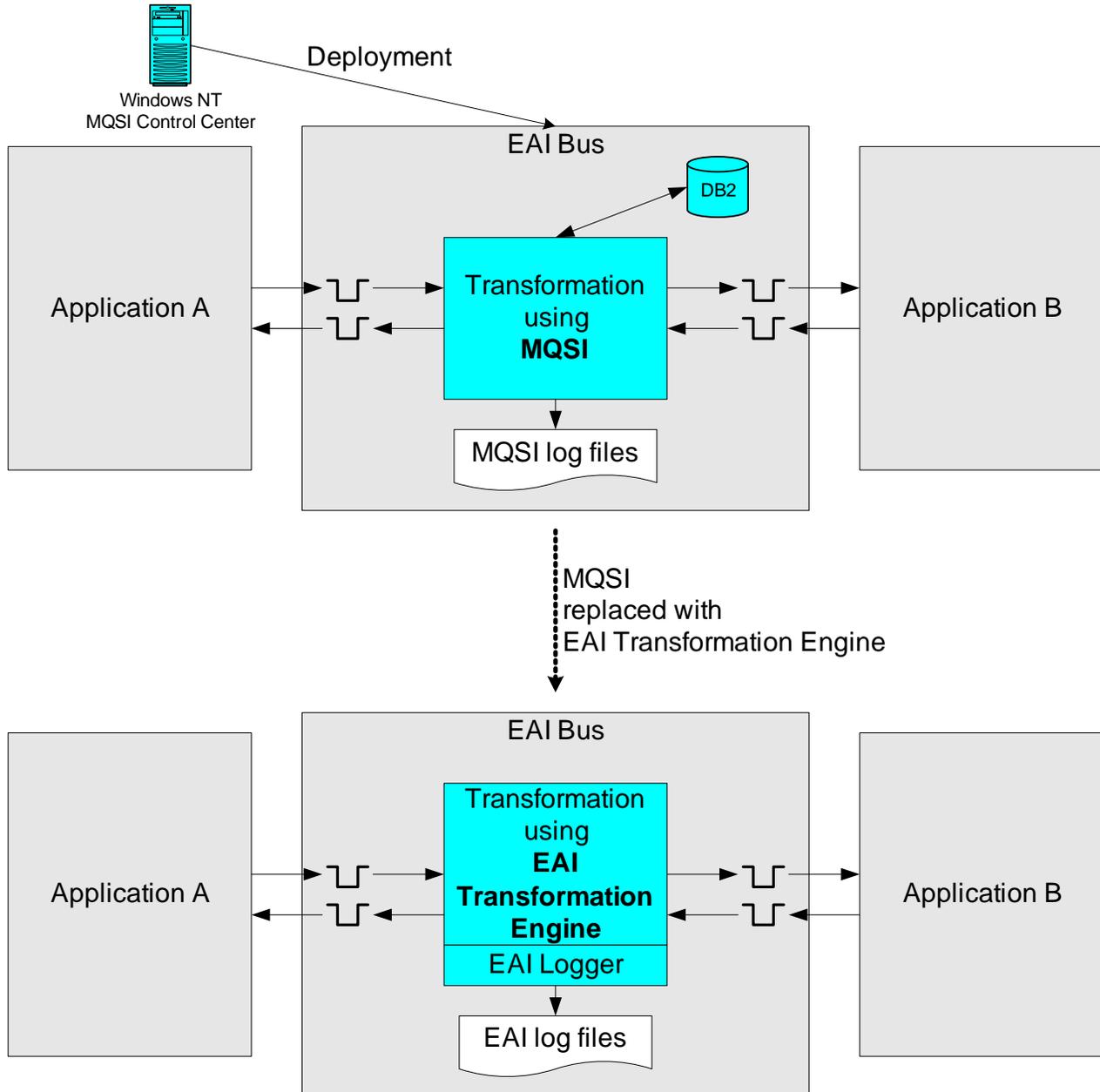


custom solution is deployed so that if there are any problems with it, all transformation can be switch back to MQSI in a few minutes.

3.1 EAI Transformation Engine

Retaining the current hub and spoke architecture, the EAI team developed and implemented standardized transformation and routing through the use of Java tools. The EAI Transformation Engine is a Java program (technical documentation in [Appendix A](#)) that can simultaneously process application messages from multiple sources and dispatch specialized transformation and response modules. Using an [interface](#) to the Transformation Engine, EAI developers create transformation modules to handle customized transformation within the Transformation Engine for each interface. The format of the transformation modules is designed to be easy to learn and use. No knowledge of the WebSphere MQ communications infrastructure is required. Testing, debugging, and maintenance are all much easier to do using the EAI Transformation Engine with the EAI Logger than with MQSI and Control Center.

Below is a simple diagram illustrating the EAI architecture components involved in the MQSI Retirement:



The EAI Transformation Engine replaces MQSI, but it alone does not transform messages. Like MQSI uses custom message flows to transform messages, the EAI Transformation Engine uses



transformation modules to transform messages. The EAI team developed the following transformation modules:

- Financial (COD to FMS)
- Confirmation/Error Response (FMS to COD)
- Financial (FMS to COD)
- Confirmation/Error Response (COD to FMS)
- Institution Data (COD to FMS)
- Unpaid Teacher Cancellation Liabilities (eCB to FMS)

Development and unit testing of the EAI Transformation Engine began on October 14, 2002 and ended on June 16, 2003 (date when last changes were completed). Development and unit testing of the transformation modules began on March 18, 2003. The EAI Transformation Engine with transformation modules for each of the interfaces was deployed to the IST environment on June 2003 and was tested as part of the COD R2.1 IST testing effort. COD R2.1 IST testing for these interfaces was completed on July 16, 003. The EAI Transformation Engine with transformation modules was deployed to the Production environment on August 8, 2003.

This development effort replaced the following MQSI interfaces in Production:

- Financial between COD and FMS
- Institution Data between COD and FMS
- Unpaid Teacher Cancellation Liabilities between eCB and FMS

For documentation of test results, see deliverable number 117.1.1c EAI Operations Services Performance Report III (correspondence # 03EDU0509).



4 Appendix A - Technical Documentation

Included in this deliverable is documentation generated by [Javadoc](#), which is the tool from Sun Microsystems for generating API documentation in HTML format from doc comments in source code.

For technical documentation of the **Generic Web Interface with Connection Pooling**, see: [02_GenericWebInterface.html](#)

It contains links to these html files:

- [IEAI.html](#) - the public interface for the Generic Web Interface
- [EAIFactory.html](#) - a simple class that has a single method that returns an EAI object.
- [EAIMQQueueManager.html](#) - an extension of com.ibm.mq.MQQueueManager that works with connection pooling
- [EAIMQQueueManagerConnections.html](#) - manages a list of pooled connections to an MQSeries queue manager
- [EAIMQQueueManagerFactory.html](#) - handles pooled connections to multiple MQSeries queue managers
- [EAIMQQueueManagerHolder.html](#) - holds an EAIMQQueueManager object and its associated in-use boolean value
- [EAIpool.html](#) - this class acts as a front-end for sending/receiving to/from MQSeries
- [TestRequestReceive.html](#) - this class is used for testing the Generic Web Interface
- [EAIException.html](#) - EAIException is thrown when something within the EAI classes reports an error

For technical documentation of the **Transformation Engine** (for MQSI Replacement), see: [03_TransformationEngine.html](#)

It contains links to these html files:

- [EAITransformEngine.html](#) - the transform engine processes a single message
- [EAIProcessCommandMessages.html](#) - processes the "stop" and "refresh" engine commands
- [EAIProcessInitQ.html](#) - reads a message from a queue, invokes a method to process the message into an ArrayList of strings, then writes the ArrayList to another queue as a series of messages
- [EAIProcessPerf.html](#) - placeholder class that does nothing
- [EAIProcessQ.html](#) - base class for handling an input queue
- [EAIProcessQFactory.html](#) - generates an instance of an EAIProcessQ implementation for a given EAI engine name
- [EAIProcessQPerf.html](#) - processes performance events
- [EAIProcessQStandard.html](#) - extends the abstract class EAIProcessQ, which in turn extends Thread
- [EAITransformEngineTest.html](#) - this class is used for testing the EAITransformEngine



[EAIRollbackException.html](#) – EAIRollbackException is thrown by the transformMessage method of the TransformMQ class or its extensions when something in the transformation reports an error