Emergency Data Interoperability Demonstration

October 27, 2004

Summary Observations

Introduction

The DM Initiative (DM), the Emergency Interoperability Consortium (EIC), and ComCARE are promoting the Emergency Data Exchange Language (EDXL) header as a proposed Extensible Markup Language standard for the emergency management and response community.

On October 27th, an interoperability demonstration was put together by several groups in the EIC to showcase the potential for sharing alerts and information across a geographic region, regardless of a user's technology and applications.

Both during and after this effort, several implementation challenges were identified as developers attempted to translate the working group EDXL standard into integrated applications. Overall, while the data exchanged using EDXL (the "what") added value to the effort there were many areas left to interpretation on the ways that each application supporting EDXL could pass data back and forth (the "how"). The varying interpretations increased the development time required to connect applications together and reduced any investments in code reuse. This document provides additional details regarding the issues encountered while trying to prepare for the demonstration.

Issues Identified

During the course of preparing and creating interfaces for the Oct 27th EDXL demo, the following issues were identified. These issues increased the development effort involved in linking EPAD Connect to other applications. From our discussions with other vendors, it is clear the same problems affected everyone other than Battelle.

Issue #1: No Standard Interfaces Defined

The latest EDXL standards document includes information regarding the data to pass between two applications, but it does not address the data flow process and functions used to exchange EDXL data. Each group participating in the demo was required to develop their own unique interfaces for exchanging EDXL data. Writing to these unique interfaces added to the development and debugging time.

There are two basic standard languages in this IP computer world: Java, and .Net. And standardized ways of communicating between them have been developed. EPAD Connect is written in .Net. In the case of DMIS, DMIS provided EPAD Connect with a URL and example code (in Java only) demonstrating how to send and receive EDXL messages into / from the DMIS demo environment. The interfaces (functions) provided by DMIS included parameters specific to the DMIS architecture (such as COG ID). When EPAD Connect tried to create an interface to the DMIS web service in Microsoft .NET, the DMIS Web service could not be accessed using standard coding practices. Sending authentication information (login/password) was not possible from a .NET client to the DMIS web service and the implementation for

passing EDXL header information did not match what was documented in the latest draft standard.

In the end, EPAD Connect was required to enhance the Java sample code provided by DMIS and force our .NET applications to execute the customized Java sample code.

We saw this again with the Maryland GIS application EMMA. While EPAD Connect was not successful in integrating with the EMMA application prior to the demonstration, the preliminary review of the EMMA interfaces identified some non-standard integration challenges. In particular, the EMMA interface was not based on SOAP and a standard SOAP interface (ex: a WSDL file) was not provided by EMMA. Had EMMA based their EDXL interface on SOAP and provided a WSDL file, EPAD Connect could have used Microsoft .NET to automatically generate the code necessary to interface to EMMA. Without standard SOAP interfaces, EPAD Connect was forced to write custom code for each application.

It would save a great deal of programming time if we were to define a standard way of Java and .Net communicating using EDXL.

Issue #2: No EDXL Schema Defined

The EDXL message format currently lacks a XML schema which each application could use to validate a message before it is sent. Almost every application looking to support EDXL during the demonstration had issues because they were not providing required fields and/or they included incorrect field values. Standardizing the development of EDXL messages against a common schema would reduce integration and debugging time.

During efforts to interface the EPAD Connect application to other vendors involved in the demonstration, it became evident that each group defined their own implementation of the fields and values in the EDXL header. Fields which were listed as required in the draft standard specification were not being provided by every vendor and each vendor participating in the demonstration had issues providing valid EDXL data values. This led to multiple conference calls and meetings prior to the demonstration and required some real-time data transformation during the demonstration. It is recommended that a standard EDXL schema is developed and that its use is required by all participating applications.

Issue #3: Using the SOAP Header added complexity

Making EDXL data part of the SOAP header added additional complexity and development time. Most of the newer development environments available today (Microsoft Visual Studio, Together Studio) include “wizard” like capabilities to add a web service to your application. These “wizard” capabilities typically do not include reading / writing to the SOAP header.

Additionally, when the exact interfaces between each vendor were defined for the demonstration, a serious limitation was uncovered due to the fact that you could only place one set of EDXL data into the SOAP header. In practice, it was uncovered that

there was a need to send multiple alert messages in a single SOAP message. Not having the ability to place EDXL data values for each message into the SOAP header forced vendors such as DMIS to “get creative” and work around the draft standards. DMIS provided a mechanism for sending multiple EDXL data values in a single SOAP message. These workarounds were reasonable for the demo, but these variations from the standard forced additional work to customize interfacing applications.