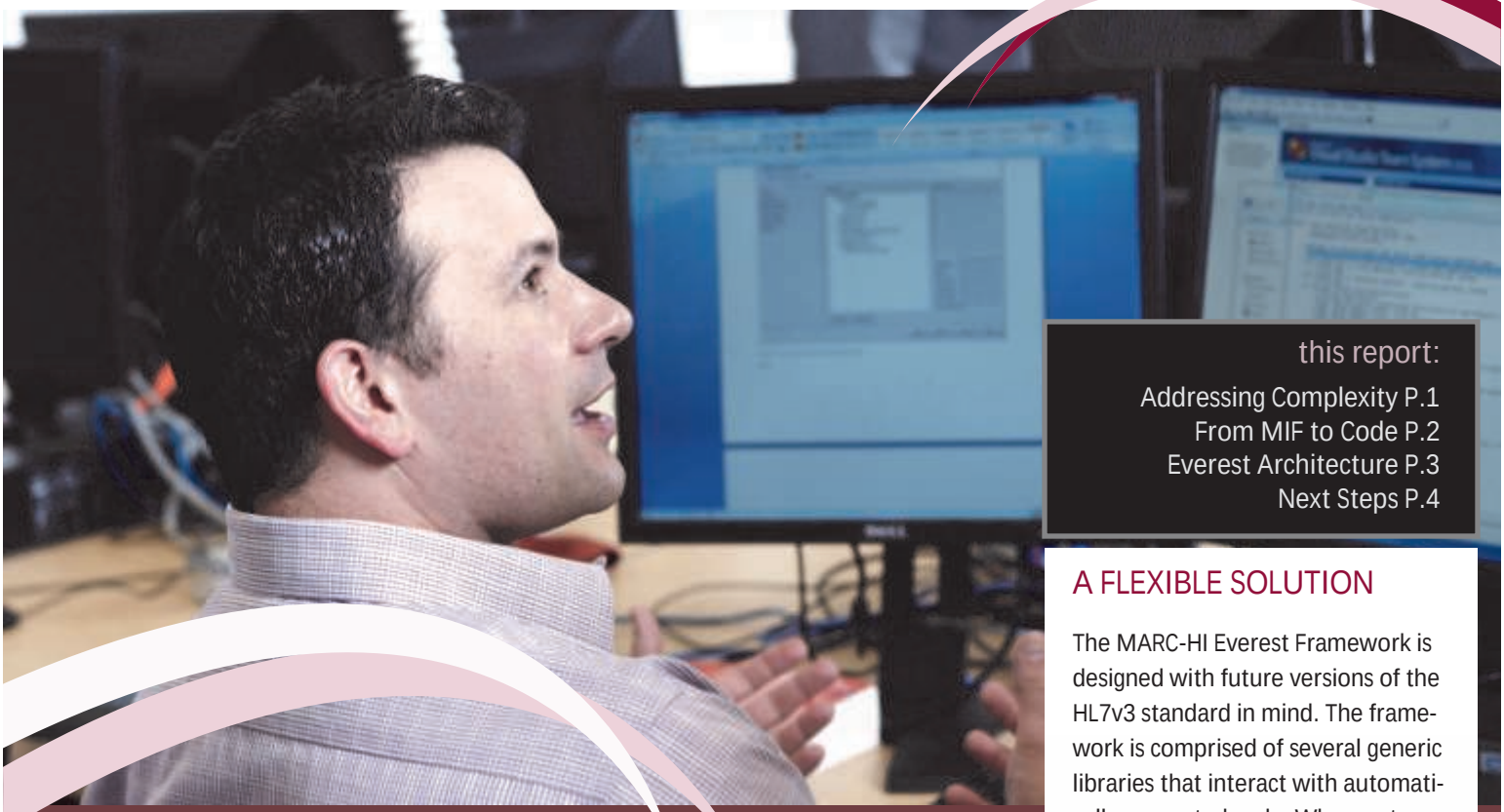
The Everest Framework

A flexible Application Programming Interface for HL7v3 messaging
Technical Report, April 2011



Addressing the complexity of HL7v3 Standards

Standards are the basis for interoperability in any technology stack. As an industry, standards provide the basis for many of the successful technologies that are in use today. Users can view web pages, check e-mail and can share files between many different systems thanks to well defined data exchange formats.

Very few developers that use these standards based interfaces actually deal with the raw interfaces themselves. Developers rely on Application Programming Interfaces (API) to abstract the details away from the developer.

In the healthcare industry, Health Level 7 (HL7) provides a series of messaging standards known as "Version 3" (HL7v3). These standards provide a rich information model intended to guarantee the semantic interoperability of health systems. Although HL7 standards are robust, and well defined, very few APIs exist. Developers are stuck learning the complex nuances of the raw HL7v3 standard.

In 2009, a team of students, faculty, and lab staff developed and released the MARC-HI Everest Framework. The Everest Framework provides a consistent, flexible and well documented API that eases the burden of learning the raw HL7v3 standard.

A FLEXIBLE SOLUTION

The MARC-HI Everest Framework is designed with future versions of the HL7v3 standard in mind. The framework is comprised of several generic libraries that interact with automatically generated code. When a standards development organization (SDO) releases a new version of their standard, developers can create new versions of Everest on their own.

THE NEED FOR AN API

Utilizing XML Schema (XSD) files that are released by HL7 and Canada Health Infoway (CHI) is tempting. However this approach has several limitations.

- Important information related to the message structures is lost within the XSDs.
- Common classes are not reused across interactions as XSD has no facility for template or generic classes.
- Generating from an XSD tightly couples your application to one version of the ITS.

From MIF to C# and beyond

Model Interchange Format is published by SDOs and defines HL7v3 models. Everest utilizes these files to generate a variety of artifacts.

HL7v3 standards are actively being improved and developed by a dedicated team of volunteer clinicians , modelers, and software implementers. The complexity of v3 comes from the fact that the implementable artifacts (the ITS) are not normative.

Rather, in HL7v3, the models are normative. Using a form of restrictive inheritance, modelers derive a series of Domain Information Models (DMIM) from the Reference Information Model (RIM). Domain models describe which components of the RIM are applicable to a particular healthcare domain.

From a DMIM, modelers use Microsoft Visio to diagram a Refined Message Information Model (or RMIM). RMIMs describe the necessary data to complete an event within the healthcare system (for example, discharging a patient).

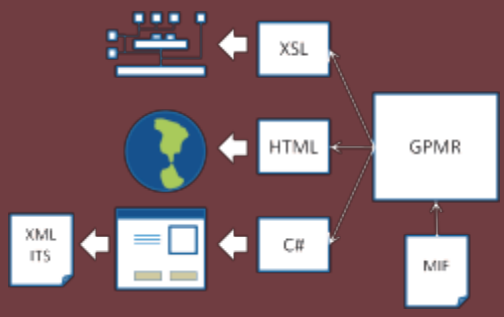
After generating the RMIM, modelers run a utility known as the v3Generator to create a series of machine consumable XML files known as Model Interchange Format (MIF) files.

These MIF files can't be used directly by programmers as they merely describe the structure of each message component.

Making the MIF files Usable

To make MIF files usable, they must first be converted to an implementable technology. HL7 defines an XML based Implementable Technology Specification (ITS) which uses XML Schema (XSD) to describe classes.

Everest uses a slightly different approach. Using the General Purpose MIF Renderer (GPMR) utility, developers consume a series of MIF files and generate one of a variety of usable outputs. These include C#, HTML (for documentation), XSD and XSL outputs.



Each of these outputs can be used in a variety of manners. The MARC-HI group uses the HTML output of GPMR on our Wiki, the XSL output in our HIAL and the C# output in a variety of reference implementation projects.

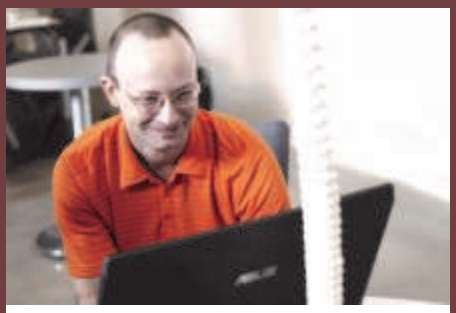
Generating the API

When GPMR generates C# code, it appends useful functions and hides certain complexities of the v3 standard.

For example, if a message element has a fixed value, GPMR will ensure that developers don't see the option to set a particular field.

GPMR can also optimize the v3 structures that are defined in the MIF. For example, if two or more modelers use slightly different "Author" structures, GPMR can combine the two different structures into one.

In addition to optimizing message structures, GPMR does not assume that messages will be sent using XML. GPMR appends metadata attributes to each generated property and class that describe its logical structure name (rather than physical XML name). This means that developers can choose, at runtime, which output format they would like to communicate.



Principles of Everest

Students are the foundation of what the MARC-HI and iDeaWorks does. It became apparent very early on that students on a 4 month co-op term were having difficulty learning the necessary HL7v3 and XML skills necessary to become productive.

Developer productivity became the guiding principle of the Everest Framework. Everest code was designed to follow best practices for .NET applications and follow many of the naming conventions.

This makes it easy for skilled .NET developers to leverage their existing experience to start solving problems and stop worrying about the standards.

Everest is designed to integrate with existing .NET facilities such as the Windows Communication Foundation (WCF) and follows standard .NET configuration practices.

Why not Java?

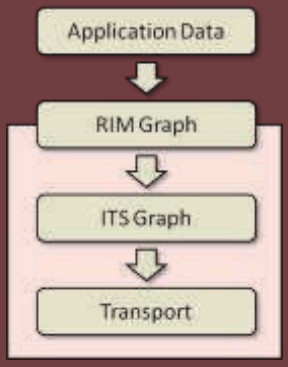
Many times we've been asked why there is no Java version of the Everest Framework.

The answer is simple, Everest is designed with the developer in mind. Doing a Java version of Everest would require a rewrite as Java developers follow different patterns than .NET developers.

Also, we feel that focusing on one technology stack allows us to make a better framework as developers can become experts rather than a "jack of all trades".

Flexible runtime architecture

HL7v3 standards are in a constantly evolving state. That combined with the fact that Everest may be utilized by any number of different software vendors made it necessary to make the Everest flexible.



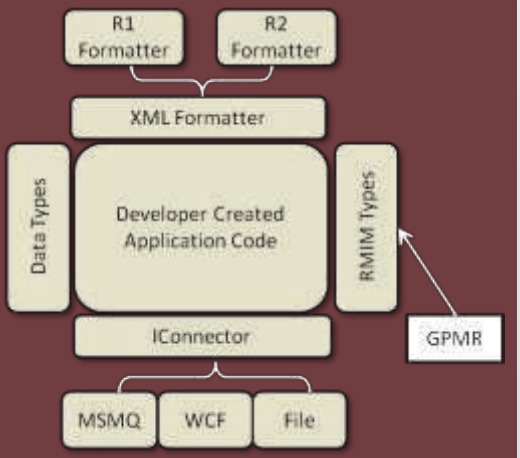
Using the Everest Framework, developers write a RIM graph that transforms their application data into the RMIM classes generated by GPMR. Developers use a Formatter to graph their RMIM structures into an ITS (a process called "graphing").

Once graphed, an RMIM class can be sent to a remote system using a Connector which ensures that the message instance is received by the remote party.

Formatters and Connectors can be swapped out at runtime, making it possible to write code to send/receive messages using files. Using the same program, a user could configure sending messages to/from a WCF endpoint.

Using GraphAides and the data-types library, it is possible for one application using one standards release to support datatypes R1 and R2 (ISO harmonized).

Each of the components of Everest can be swapped at runtime, and with the embedded Meta-Data on RMIM and DataTypes, further flexibility can be gained through clever reflection algorithms.



HISTORY
The Everest Timeline

Nov 2008 - Proof of Concept API
Jan 2009 - GPMR Started
Jul 2009 - GPMR Deki Renderer
Aug 2009 - Demo for Canada Health Infoway
Sept 2009 - First public release
Nov 2009 - Canada Health Infoway Sponsored Release
Jan 2010 - Structure Optimizer for GPMR
Jul 2010 - Multithreaded Formatters
Nov 2010 - Universal Messages Added
Jan 2011 - Release Candidate 1

UNIVERSAL
Support for HL7v3 NE2008

Everest Release Candidate 1 (released in January 2011) includes experimental support for HL7v3 universal messages. This means that Everest is able to communicate with systems developed for markets outside of Canada.

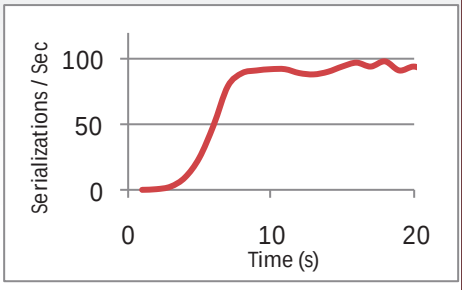
It is also possible to communicate with IHE PIX/PDQv3 registries using the Everest Framework. Samples are available in the Release Candidate 1 release of the MARC-HI Everest Framework

Providing faster serialization via CodeDom

Q: Why does the first serialization of a message take so long?

The reason for this behavior is the way that Everest serializes messages. When a formatter first encounters a message type, it dynamically generates the necessary code to serialize the message structure. Whenever the formatter is asked to serialize a message with the same structure it reruns the code it generated before.

The graph to the right illustrates the serializations per second from a sample of 10 random message types serialized on 10 threads. The performance of the formatter increases the more the formatter encounters similarly structured messages.



The Everest Team

Faculty: Duane Bender, Brian Minaji, Mark Yendt

Lab Staff: Justin Fyfe, Jaspinder Singh, Trevor Davis

Students: Andria Chiravalle, Jose Aleman, Terence Cook, Brian VanArragon, Stuart Philp, Craig Clark, Matthew Ibbotson, Corey Gravelle

Next Steps and Upcoming Features

Custom Everest Workflow Foundation Activities

The Everest team is currently working on a series of Workflow Foundation (WF) activities. These activities will allow developers to drag-and-drop Everest messages onto their workflows.

Testing and integration of CeRX 4.3, NE2009 and NE2010

Everest supports pan-Canadian Standards (pCS) R02.04.01, R02.04.02, R02.04.03 and Universal NE2008 messages. Work is being done to support CeRX4.3 and NE2009/2010 messages.

Release of Everest 1.0

Everest documentation and code continues to be developed and tested. Everest 1.0 (with GPMR 1.0) will be ready early 2012.

Download:
everest.marc-hi.ca

iDeaWorks Mohawk College

135 Fennell Ave West
Hamilton, Ontario Canada
L9C 1E9

905.575.1212 x4738

mohawkcollege.ca/ideaworks