**Initial Lessons Learned from VistA Novo**

January 15, 2014
Andy Gregorowicz, Dave Hill, Andrew Hubley, Jon Palmer

This paper is a brief summary of the findings that the MITRE VistA Novo team gathered over the project's initial implementation. The purpose of the paper is to share our experiences with our government sponsors and the open source software community.

Overall, our decisions to target FHIR as an interface standard and use JavaScript as a development language remain sound choices. However, we wanted to share some of the lessons learned during the effort.

## FHIR

### Lesson 1: FHIR is Simple and Easy on the Client side

FHIR is a great specification for writing a software client, such as a mobile application or web site to consume clinical information. Prior standards have often been complex and verbose, which can be challenging for many clients operating with relatively low processing power and bandwidth. FHIR uses JSON, which is less verbose than XML and natively supported in JavaScript, used in every browser and currently the most popular programming language. The blood pressure demo application using the RESTful FHIR interfaces in VistA Novo was written in just a few hours and most of that time was spent on user interface issues.

### Lesson 2: Simple does not necessarily equal Easy on the Server side

Creating a server side implementation of FHIR can be a larger effort that what appears on the surface. The FHIR server is composed of several simple features. Each of these features is straightforward and can be understood quickly. The sum of these features remains straightforward, but if they are all to be utilized by clients, they must be implemented on the server. When you consider the basic CRUD actions, search, version history, handling of different content representations (mime-types) and bundles, the resulting implementation of this simple specification can be large. This general notion is captured in the talk Simple Made Easy. In other words, FHIR is simple but it is not necessarily easy to implement on the server side.

### Lesson 3: Use the FHIR Tool Chain

Due to the broad nature of the FHIR specification, the VistA Novo team would strongly advise any team implementing a FHIR server to do so with the assistance of the HL7 FHIR tool chain. Using this tool chain allows developers to automatically generate much of the software that they need to host FHIR resources. Additionally, since the FHIR specification is still under active development, using the tool chain provides a way to stay up to date with changes in the specification with minimal manual intervention. The HL7 FHIR code generator delegates the tedious task of

supporting dozens, or perhaps hundreds, of FHIR models to those closest and most passionate about the standard – the FHIR development team itself. The reference FHIR code produced by the HL7 FHIR code generator promotes faster adoption of the latest FHIR standard as well as data interoperability between FHIR systems since the reference code in each system interprets FHIR data in the same way.

During the course of this effort, the VistA Novo team benefited from using the HL7 FHIR tool chain. Changes were made in the FHIR specification as a result of the Fall HL7 Work Group Meetings. These changes were reflected in the tool chain so VistA Novo was able to keep up to date with the changes made in the specification by simply regenerating our implementation.

**JavaScript**

**Lesson 4: A Synchronous JavaScript Interface is Needed**

VistA Novo was implemented using the JavaScript programming language. The software runs in a server environment called node.js. The dominant programming model used in many node.js software libraries is asynchronous.

Using this asynchronous interface added complexity. For example, when retrieving a document from the MongoDB database, the query function provided by the database access library does not return the document directly. Instead, the function accepts a callback. A callback is another function that contains the logic to be used to operate on the document from the database.

The node.js software libraries are structured in this way to better suit its architecture. Unfortunately, our team found this model more difficult to work with than a traditional synchronous approach. Over time, we improved our abilities to develop and troubleshoot code using this methodology, but doubt we could work as proficiently using an asynchronous paradigm. We would strongly encourage future JavaScript library developers to provide synchronous interfaces to their functionality wherever possible.

**Lesson 5: The node.js environment was fast and full featured**

Positively, the VistA Novo team found the node.js environment to exhibit great performance. Start up and response times allowed for rapid software testing. Additionally, the JavaScript and node.js ecosystems are quite rich. When seeking libraries for capabilities such as testing, making HTTP requests, or generating XML, we had several choices available.