The CareWeb Framework Innovation Project

Objectives

The objectives of this Innovation Project were as follows:

1) Demonstrate the capabilities of the CareWeb Framework (CWF), a Java-based, server-centric, open-source, Ajax web framework, running atop a VistA instance.
2) Demonstrate the ease with which legacy VistA/RPMS code can be ported to the CWF.
3) Port selected plugin components from the RPMS-EHR, prioritized according to their contribution to MU certification.
4) Demonstrate an evolutionary approach to a next generation EMR, embracing old and new technologies side-by-side.

Background

The CareWeb Framework (CWF) was developed by Doug Martin and colleagues at the Center for Biomedical Informatics (CBMI) at the Regenstrief Institute in Indianapolis as a common platform for the collaborative development of complex clinical applications in a web environment. It has been used as the foundation for a full-featured electronic medical record and order entry system (Gopher) running at our partner hospital (Eskenazi Hospital), and a result viewing/search application for the statewide health information exchange (Indiana Health Information Exchange). Built upon a stack of open source Java and JavaScript technologies and modeled after the highly successful VueCentric Framework (the underpinnings of the RPMS-EHR), the CWF permits the construction of complex applications in a modular and collaborative fashion using a programming paradigm that is already familiar to VistA/RPMS developers. The CWF provides a number of capabilities that facilitate the seamless integration of discrete plugin components into a cohesive application. Foremost among these are:

- Management of shared context such as user, patient, encounter. Drawing from the CCOW context model, this service coordinates changes to shared context across subscriber components within an application instance.
- A powerful pub-sub event model that supports the propagation of events within an application instance and across application boundaries.   Coupled with server push capability, this enables the creation of highly responsive applications that can react to a variety of internal and external events.
- Component registration and discovery services use a service bus approach to permit building highly modular, extensible, dynamic applications.
- Created using a visual designer tool, layouts are snapshots of the UI that may be persisted and later reconstituted. An important application of layouts is to create

alternate views of an application that are tailored to environmental factors such as user preference, role, physical location, specialty, etc.

- A help content manager supports the integration of on-line help content created using standard help authoring tools. Help content is fully searchable and context sensitive.
- Security services are based on open source technologies and can be adapted to multiple authentication schemes.

Plugin Component Selection

A committee comprised of both clinical and non-clinical representatives selected five RPMS-EHR plugin components as candidates for this project:

- Chief Complaint
- Immunizations
- Family History
- Patient Goals
- Clinical Information Reconciliation (CIR) Tool

Because anticipated development resources could not be secured due to contractual barriers, and because of the unexpected complexity of the plugin, the CIR tool was eventually removed from the list of candidates. The remaining four plugins were successfully and faithfully ported to the CWF. In each case, both client-side code (2 written in Visual Basic, 2 in C#) and server-side M code and files had to be ported. For server-side components, the extent of changes required was directly related to the degree of overlap and/or conflict with existing VistA code and/or files. This varied widely by plugin. For client-side code, porting was relatively straightforward because the programming model of the ZK Framework (the web UI framework used by the CWF) is very similar to that of traditional thick client programming. For some plugins, we attempted to faithfully reproduce the look and feel of the original. For others, we took some liberties with the UI design to improve usability over the original.

Infrastructure Development

Much of the initial effort was devoted to exposing and integrating various VistA services. From the outset, we wanted to demonstrate that old technologies (e.g., RPC broker) and new technologies (e.g., web services) could be used side-by-side. This allows for a more evolutionary approach to infrastructure modernization, permitting one to leverage the substantial existing inventory of VistA services while gradually migrating to more contemporary technologies. This effort included the following tasks:

- Create a Java-based RPC broker client. Because the Medsphere RPC broker supports essential capabilities not offered by the VistA RPC broker (specifically, asynchronous calls and event propagation), this was chosen as a starting point. Because changes to the server-side M code were required to support web-based authentication, the server code was moved from the CIA namespace.

- Create a web services infrastructure. The VPR web server was initially chosen. However, because of anomalies with the GT.M version of the code base and the inability to fully customize the response payload, a different approach was eventually selected (see next).
- Given that server-side changes to both the RPC broker and the web server code base were required, both were redesigned by building them atop a common TCP connection manager. Called NETSERV, this package centralizes the configuration of TCP-related services and obviates the need for every package that requires TCP connectivity to "roll their own" as is the current approach. This codebase is being contributed for possible future incorporation into the VistA kernel.
- Create a property persistence adaptor for CWF. Since CWF provides only a rudimentary mechanism for persisting property values (under the assumption that most host systems provide this capability natively), an adaptor is required to make use of the VistA mechanism for persisting property values (as XPAR parameters).
- Create a FHIR service endpoint. A number of FHIR-based plugins exist for the CWF (for example, a patient selection plugin). To leverage these, a framework for serializing VistA domain objects was developed (the VistA Serialization Framework). This framework uses a table-driven approach to serialize domain objects to one of a number of possible formats. When coupled with the NETSERV web server, a RESTful FHIR service endpoint can be created. The framework can support multiple FHIR DSTU versions in both JSON and XML formats as well as custom serialization formats. A number of domain objects have been implemented (patient, encounter, organization, observation, condition, location, medication, etc.). Currently only read and query operations are supported (i.e., write-back operations are not supported).
- Build out support for creating and managing encounters. Several of the plugins to be ported from RPMS require the ability to create or select an encounter. We modeled encounter support after RPMS-EHR. All query operations were implemented using RESTful FHIR service calls. All write operations used existing RPC's (since FHIR-based write-back services have not yet been developed).
- Add HTML Help (CHM) format support to the CWF help system. This is the format employed by the RPMS-EHR and by CPRS. Doing this allowed us to integrate existing help content into the application.

Plugin Component Development

Chief Complaint: This was the simplest of the ported plugins. Because the M code and files had no overlap or conflict with existing VistA code, porting of the server-side components was straightforward. Like the original, this plugin leverages CWF event propagation to signal changes to the chief complaint to other instances of the application that may be viewing the same patient.

Immunizations: This was the most complicated of the ported plugins, both the client-side code (because of the number of function points) and the server-side code (because of significant overlap and conflicts with the VistA Immunization package). For the latter, we leveraged work being done under the VIMM effort to reconcile differences between the VA and IHS immunization packages and that of Sam Habiel to port the full functionality of the IHS Immunization package to VistA. Further complicating matters is that the RPMS-EHR immunization plugin has additional supporting code that required modification. In the end, the result was less than satisfactory. While we were able to implement all capabilities (though there remains some anomalous behavior), the VA and IHS immunization packages have not been fully reconciled as some data elements are fundamentally different and require special handling in the UI. We believe that much work remains to truly converge the two implementations, that such a goal is achievable and desirable, and that the current effort will not achieve this to a satisfactory degree.

Family History: This plugin was of intermediate complexity, primarily because of the need to port the IHS Standard Terminology package for the lookup of SNOMED terms. While IHS uses a centralized terminology service for lookups, they also provide an offline mode. We did not attempt to set up a terminology server instance but rather relied on the offline mode for this demonstration. We also identified and fixed some previously unrecognized bugs in the M code that were uncovered during testing. We will share these changes with IHS for possible incorporation into their codebase.

Patient Goals: We took significant liberties with what we considered to be a rather confusing UI design to create a more intuitive user experience (UX). We also made some backward-compatible changes to the server-side M code to improve the efficiency of data retrieval. We will share these changes with IHS for possible incorporation into their codebase.

Conclusions

We were successful in meeting all of the objectives of the project. The majority of effort was expended in building out the necessary infrastructure to expose VistA services rather than the porting of the plugin components themselves. In every case, we were able to replicate the richness of the thick client user experience – a testament to the power of the ZK Framework. Because the programming model employed by ZK more closely approximates that of thick client development than more traditional web development frameworks, the work of porting legacy code is greatly simplified. ZK's visual component model is very similar to those of Delphi, Visual Basic, and C#. Similarly, ZK provides the ability to create visual layouts declaratively as well as programmatically. The result is a development experience that will be very familiar to developers of thick client applications.

While we ended up not leveraging any work from the eHMP initiative, several artifacts of this project should be of interest to that group. Foremost among these are the NETSERV and VistA Serialization Framework packages. As eHMP web service offerings mature, these could be easily consumed by CWF components.

All of the code generated from this effort may be downloaded from either the CWF Github site (for infrastructure code) or the author's personal Github site (for plugin code):

- https://github.com/carewebframework
- https://github.com/mdgeek

Acknowledgements

I would like to thank Medsphere Systems, Inc. for contributing effort toward this project. Specifically, Phillip Salmon ported the Chief Complaint and Immunization plugin components. Phillip was able to quickly achieve a high level of proficiency despite having limited Java and web development experience.

I would also like to thank Kristopher "Kit" Teague for his superlative management of this project and for running interference that allowed us to focus on development tasks. Without his efforts, this project would not have achieved its objectives.