Composable Software, Collaborative Development, and the CareWeb Framework

Douglas K. Martin, MD. Professor of Clinical Medicine, Indiana University School of Medicine; Research Scientist, Regenstrief Institute. Indianapolis, IN.

Abstract

The CareWeb Framework (CWF) enables the software developer to build complex, richly interactive, web-based applications in a modular fashion. In addition to offering an extensible component model for application development, the CWF provides support for component registration and discovery, event management for local or global event delivery, context management for collaborative sharing of common contexts, and layout management for the composition, persistence, exchange, and reconstitution of user interface configurations. The CWF has been used as the basis for a complete EHR and CPOE system and has been ported to several open source EHRs, including OpenMRS, VistA, and RPMS. The CWF is open source software (http://www.carewebframework.org) built upon open source software.

Purpose

Traditional software design approaches result in monolithic software applications that are difficult to configure, adapt and evolve. As the basis for our next generation of software offerings, we have created an application framework that facilitates a modular approach to application development. Advantages over the traditional approach include:

- Emphasizes re-usable modules
- Is easily extended
- Can dynamically adapt to environmental factors
- Facilitates rapid prototyping of new functions
- Promotes collaborative development

Underlying Technologies

While we examined many candidates as foundational technologies for our framework, a few stood above the rest; among them are:

- **Spring Framework** – Created a decade ago, the Spring Framework is in its fourth major iteration. It provides key core functionality including object lifecycle management and automatic object configuration and discovery.
- **ZK Framework** – An open source, AJAX based web UI framework, the ZK Framework provides a richly interactive web experience with a programming paradigm that more closely approximates that of thick client development.
- **JQuery** – A popular JavaScript library and the foundation on which the ZK Framework is built, JQuery simplifies JavaScript development.

- **Apache ActiveMQ Server** – While not a required technology for the CWF, Apache's Java Messaging Service compliant server provides powerful message subscription and delivery services that are leveraged by the CWF for global event delivery.
- **Apache Maven** – Maven greatly simplifies the complexities of versioning and dependency management in any architecture with many moving parts by automatically including correct versions of required components during application build and deployment.

Core Services

The CWF provides several core services that are critical to the provision of a seamless user experience that belies the compositional nature of the user interface. Key among these are:

- **Context Management** – Heavily based on the HL7 CCOW specification, the CWF provides core support for the management of shared context (think user, patient, encounter, etc.). Plugins may respond to or initiate context changes. Context changes follow a democratic process in that subscribers have an opportunity to vote on whether a requested context change may proceed. The context management subsystem handles these interactions automatically, greatly simplifying the task of creating shared contexts.
- **Event Management** – CWF's event management subsystem uses a subscribe/publish model for event delivery. Events may be delivered locally within a specific application instance, or globally to subscribers that may reside virtually anywhere. Events are extremely flexible and can be used for a variety of purposes, from simple inter-component communication to a complete secure chat framework.
- **Plugin Registration and Discovery** – By registering themselves with the CWF, plugins allow themselves to be discovered by other plugins. One of the benefits of this is that it permits plugins to be automatically wired to services they may wish to consume. For example, context change subscribers simply declare their interest in a specific context change by implementing the appropriate interface. The context manager monitors plugins as they are registered and detects this interface, automatically subscribing it to the correct context change event producer.
- **Layout Design and Management** – Key to the composable nature of the CWF user interface, layouts provide a means to persist user interface configurations and reconstitute them at a later time. An integrated layout designer allows privileged users to create layouts on the fly. Layouts may be exported, imported, and even embedded in other layouts.

Interoperability

The CWF proper is domain agnostic. That is, it doesn't care whether we are building an EMR or a banking application. This separation of framework from domain is by design and is important to maximize the flexibility of the CWF. It isn't until we start providing plugin services and widgets supporting clinical functions that it becomes bound to the healthcare domain. In doing so, one can choose to use a proprietary domain model specific to a given underlying implementation, thereby limiting the usability of components outside that environment, or one can adopt a common domain model to maximize interoperability across varied backend implementations. The CWF does not dictate a particular approach,

nor does it preclude mixing approaches. We have designed components that are specific to our operating environment that have limited utility elsewhere. However, the recent rise of the HL7 FHIR specification has provided the opportunity to create truly interoperable components. The Clinical Abstraction Layer (CAL) is a CWF add-on that provides a platform for developing FHIR-aware plugins that can run against FHIR-compliant systems. We have adapted several of our implementation-specific components to run within this platform. For example, we have FHIR-based versions of our patient selector and document viewer.

Current Ports

We have successfully grafted the CWF onto several clinical systems besides our own. These include OpenMRS, VistA, and RPMS. The VistA port leverages Medsphere's RPC broker for authentication, data access, and event management. We have developed a data serialization framework for VistA that enables the materialization of domain objects using standard formats such as JSON and XML. We have expanded this framework to include support for the FHIR specification. Plugins can choose to use standard remote procedure calls for data access, or use the provided FHIR client to perform RESTful calls (which are transparently redirected through the existing broker connection).

Available Resources

The CWF source code, documentation, and sample applications are offered under the Mozilla Public License 2.0 and may be obtained from http://www.carewebframework.org. Hosted on GitHub, several projects are available for download:

- **carewebframework-core** – This is the core framework providing the basic services outlined above.
- **carewebframework-icons** – These are open source icon collections packaged for easy reference within the CWF.
- **carewebframework-ohj** – Allows inclusion of online help content that uses the Oracle Help for Java (OHJ) format. The CWF also provides support for the Java Help format out of the box.
- **carewebframework-highcharts** – This is a ZK component wrapper for the popular Highcharts JavaScript graphing library.
- **carewebframework-smart** – This adapter allows SMART-compliant plugins to be used within the CWF as if they were native plugins.
- **carewebframework-fhir** – This is an adaptation of Graham Grieve's FHIR Java libraries.
- **carewebframework-cal** – This is the Clinical Abstraction Layer platform as described above.
- **carewebframework-openmrs** – This is the OpenMRS adapter for the CWF.
- **carewebframework-vista** – This is the VistA adapter for the CWF.
- **carewebframework-rpms** – This is the RPMS adapter for the CWF.

Current Deployments

As of this writing, CWF-based applications are used throughout the state of Indiana in support of the Indiana Health Information Exchange. A comprehensive CWF-based EMR

and CPOE system serves the inpatient, outpatient, and emergency departments of our longtime healthcare partner, Eskenazi Health. Both of these deployments serve thousands of concurrent users.

Conclusions and Future Directions

The CWF has enabled us to develop applications in a more agile and collaborative fashion than was previously possible. In our environment, our suite of plugins now numbers in the hundreds. Developers are able to work on functionality independently and we have much more freedom in how and when to deploy it. The experience of adapting the CWF to run on top of environments other than our own has likewise been extremely valuable, allowing us to refine the framework and make it more generalizable.

The composable nature of the CWF permits innovation at another level – that of the end user. Previous experience has shown that users will innovate in unexpected ways when given the tools to do so. The ability to export and share such innovations is also of critical importance, promoting the establishment of communities of common interest around these artifacts.

We are encouraged by our initial experience with the FHIR specification. While still only a draft standard, we believe that it will be key in developing truly interoperable clinical systems and provide opportunities for collaborative development not heretofore achievable. Its value is not only as a data exchange standard, but also as a generalizable data model.

In the universe that is open source, collaboration is essential. Monolithic and closed application architectures make this difficult to achieve. We believe that the open and modular design of the CWF is much better aligned with the spirit of open source software development. We invite fellow collaborators to give it a try.