Prototype of the Web-Based AOP Data Processor

Stanford Hooker
NASA/CVO
Baltimore, Maryland

David Stroud
UMBC/CVO
Baltimore, Maryland

Description of the Development Environment

The prototype application was developed using Adobe Flex and is based on the CVO processor in its most basic form. The reasons for choosing Flex are as follows:

- Flex is an open source framework for developing and maintaining a Rich Internet Application (RIA). It is designed to consistently deploy with all the major browsers, desktops, and operating systems.
- Flex is very similar to “traditional” software development environments, like RealBasic, XCode, Metroworks, etc., in that you write code, debug it, and deploy the application. These traditional packages, however, only leverage a single technology, are mostly proprietary, and require multiple compiles to make the application available to the various OSs, and usually require recompiling for OS updates.
- Flex on the other hand combines multiple proven technologies (such as, AS 3.0, XML, and CSS), and leverages HTML, Flash Player, and AIR as the runtime environment. Use of these runtime environments means the OSs can be different, and change over time, but if the end-user has the latest version of a plug-in, they will always be able to utilize the software, without the developer having to rewrite and recompile on a regular basis.

Definitions

- Actionscript (AS) 3.0 is an object-oriented programming language and follows ECMA-262 specifications. It is very similar to Javascript, in fact you could use Javascript in a Flex application with no issues. AS was used to convert the Basic code used for CVO processor visualizations. AS is a sophisticated language with the needed properties, methods, and functions to do high-order math. AS was also used to build the user interface (UI) navigation logic.
- Extensible Markup Language (XML) is a general purpose specification. It's major purpose is to aid in sharing commonly understood structured data within multiple information systems. It is recommended by W3C, and is an open standard that specifies lexical grammar and parsing requirements. XML (actually MXML) was used in the layout of all the UI components. The MXML components communicate directly with AS. In fact, the MXML components get turned into AS during compilation.
- Cascading Style Sheets (CSS) is a style-sheet language that uses XML to describe how to present a document or, in this case, the UI. It is typically used to set global and sometimes local style rules (color, font, and size). In the case of Flex and this project, it was used to set the graphical icons for the buttons. CSS is a standard that is understood across all the major browsers.

Definitions (*cont.*)

- Adobe Integrated Runtime (AIR) is a free downloadable plug-in. The plug-in, available for all the major OSs, provides the runtime environment for AIR applications. AIR applications can be developed using Flex, Ajax, and Flash and then deployed directly to the desktop instead of having to use an HTML page and a browser for display. So, what used to be considered a web application, can now be deployed as a stand-alone application that uses web technologies, but more importantly, with access to a local drive. Such access has not been possible with prior web applications because of security issues.
- Flex programs that run in the Flash runtime environment have the capability to connect to servers and databases, but not the local machine. Flex programs that use AIR can connect to the local drives and make calls to remote servers. Flex communicates with "back-end" technologies using any number of "middleware" languages (e.g., ASP, PHP, and Coldfusion). This project uses PERL in the form of common gateway interface (CGI) scripts during the two steps of pre-processing, the loading and sending of all data (both in the plot and in populating all the UI components), and in the processing phase. CGI will also be used to communicate between the Flex application and the database residing on the CVO servers that will archive all of the uploaded data and results from the user community.

Description of the User Environment

There is no need for a browser to use the processor. The only time a browser is required is to register as a user, download the software, and access updates. The entire server–client environment can be hosted on a single machine, which, of course, is how the application is built and tested in the first place (note that this will also allow the entire environment to be taken into the field). The usual architecture, however, will be a user located at a site far removed from the server. In this scenario, the user environment is as follows:

- Once approved, the applicant will receive an email with a pass key that will permit access to a secure page on the NASA/CVO web site containing instructions for the download and installation of both the AIR plug-in and the PROSIT application.
- Logging in to PROSIT will start a session, meaning that the application on the client machine will be in communication with the CVO server. This is not a streaming connection, but is done intermittently on an as-needed basis. The important point is that in order for users to make use of the software, they have to be connected to the internet, but they do not have to be using a browser (i.e., the situation will be very similar to using a mail client, like Eudora).

Description of the User Environment (*cont.*)

- All transactions during a work session follow the same basic structure: front-end (client), middleware (server), back-end (server), and front-end (client). As an example: The user either requests or submits data via the UI. These data are sent to the appropriate CGI executable, via the internet connection, that resides on the server. The CGI script runs and processes the request or submission. This may return a value to PROSIT immediately or trigger one or more additional scripts in a sequence that do other processing, which then either return data, or store data for later use on the server.
- For example: The user clicks the Next button on the Ingest page, which triggers the client to select the Pre-processing tab and a request a CGI script on the server to send a list of all available projects to be pre-processed. This information is displayed, the user selects a project, and clicks on the Pre-process button, which sends the name back to a CGI script on the server. The CGI script finds the project and runs a correction program and then a conversion program. The pre-processed data are stored on the server, to be loaded during a future step. Once these steps are complete, the CGI script informs the client whether this is has been completed successfully or not. If successful, the UI enables both the Next button in the Pre-processing Tab and the Depth Interval Tab, so the user can go to the next step.