

Gastkolumnist Joe Pluta

EGL und System i

Ich beschäftige mich jetzt schon seit einer ganzen Weile mit EGL, und je länger ich mit dieser Sprache arbeite, umso mehr sehe ich, dass sie nahezu perfekt für System i geeignet ist - besser jedenfalls, als jede andere verfügbare Lösung.



Es gibt auch einiges, was mich an EGL stört. Das hat jedoch größtenteils mit der Preisgestaltung, der Lizenzierung und anderen unerfreulichen Dingen zu tun, die in den Abgründen großer Unternehmen vor sich gehen. Als ermutigend empfinde ich jedoch Gespräche, die ich mit Mitarbeitern von IBM hatte. Dort glaubt man an EGL und will, dass die Sprache verwendet wird. Ich gehe also davon aus, dass IBM schon herausfinden wird, was man tun muss, damit EGL auch in die Hände derjenigen gelangt, die diese Sprache brauchen.

Womit wir beim Kernpunkt dieses Artikels wären: Wer in der System i-Community braucht EGL, und, was noch wichtiger ist: Warum?

Braucht wirklich jeder EGL?

Im Ernst: "Jeder" kommt der richtigen Antwort schon recht nahe. Es wäre leichter, diejenigen zu benennen, die EGL nicht brauchen, als alle die aufzuzählen, die diese Sprache benötigen. Aber zunächst zu den Gründen:

Der erste ist vielleicht, dass EGL die über lange Zeit ungeklärte Frage nach einer GUI-Schnittstelle für System i beantwortet. Die Forderung, dass IBM eine Ersatz-GUI für den 5250 zur Verfügung stellen soll, ist immer wieder laut geworden. Tatsächlich hat IBM das schon vor Jahren getan: Mit WebSphere, JSP und dem unglaublichen Java Toolkit. Das Problem ist allerdings, dass hierfür ein paar Java-Kenntnisse erforderlich sind. Nicht gerade besonders umfangreiche; aber aus irgend einem Grunde braucht man Java nur zu erwähnen, und schon stürzen manche Programming Manager schreiend aus dem Haus.

(Ich freilich nicht. Ich wünschte, ich hätte Java schon gehabt, als wir noch komplexe PC-Anwendungen in C und C++ schreiben mussten. Als Plattform ist Java den meisten anderen Programmierungsumgebungen mit Ausnahme meines geliebten i5/OS architektonisch überlegen.)

Mit EGL können Sie dagegen komplette webbasierte Anwendungen erstellen, ohne eine einzige Zeile Java lernen zu müssen. EGL besitzt eine einfache, deklarative Syntax und ist eine ausgesprochene High-Level-Sprache. Schon allein mit der Phrase "get customers" lässt sich ein vollständiger Applikationsprozess bauen: Daten aus einer SQL-Tabelle laden, in ein dynamisches Array einfügen und dieses dem Benutzer dann in einer Tabelle präsentieren. Einfache Abfrageanwendungen benötigen etwa ein Dutzend Codezeilen. So kann sich der Programmierer auf die tatsächlichen Arbeitskomponenten des Programms konzentrieren, zum Beispiel auf die SQL-Anweisung zum Abrufen der Daten.

Was wären die Alternativen?

Doch wie steht es mit den Alternativen zu EGL? Natürlich gibt es für jede davon Anwendungsmöglichkeiten, aber in den meisten Fällen sind diese recht begrenzt. Mit dem Folgenden werde ich es mir vielleicht mit einigen Leuten verderben, aber wenn Sie meine Ausführungen objektiv betrachten, werden Sie mir sicher zustimmen:

Ich beginne mit RPG. Ganz einfach gesagt: RPG als CGI-Sprache zu verwenden, ist nur dann gerechtfertigt, wenn Sie keine Zeit oder keine Lust haben, noch eine weitere Sprache zu lernen, wenn Sie dazu noch jede Menge Rechenleistung auf Ihrem System i übrig haben und wenn es Ihnen nichts ausmacht, den Server für das Internet zu öffnen. Jedes dieser Kriterien verringert die Anzahl der in Frage kommenden Geschäfte so weit, bis RPG-CGI nur noch eine kleine Nischenlösung ist -- sicherlich nicht nutzlos, aber keine Option, wenn Sie nicht haargenau diesem Profil entsprechen.

Dann gibt es die verschiedensten Skriptsprachen. Jede davon hat ihren eigenen Zweck, und größtenteils sind sie ganz gut darin, schnell eine vorhandene Anwendung zu installieren. Plone ist zum Beispiel durch Python ein wirklich leistungsstarkes Content-Management-System. Ebenso existieren Dutzende ausgewachsener professioneller Anwendungen für PHP. Und wenn Sie eine CRM-Anwendung auf Ihrem System i einsetzen möchten, ist dies eine relativ unkomplizierte Option.

Leider gibt es bei Skriptsprachen zwei große Probleme. Zum einen umgehen sie per Definition das Model-View-Controller-Konzept und verführen dazu, die Geschäftslogik mit der Darstellung zu vermischen. Damit es jetzt Massendemonstrationen unter der Losung "Aber Yahoo verwendet auch PHP!" vor meinem Haus gibt, möchte ich eines klarstellen: Es geht mir nicht darum, dass man mit PHP keine soliden MVC-Anwendungen schreiben könnte, sondern darum, dass man es nicht muss und nicht dazu angehalten wird.

Fairerweise will ich nicht verschweigen, dass eine Reihe von Templating Engines existiert, mit denen sich Geschäftslogik und Benutzeroberfläche schon irgendwie voneinander trennen lassen. Die Entscheidung für eine dieser Templating Engines bedeutet jedoch auch, dass man sich auf einen bestimmten Dialekt festlegen muss. So gibt es, um nur ein Beispiel zu nennen, einen himmelweiten Unterschied zwischen Smarty und PHPTAL. Das ist kaum ein Problem, wenn Sie nur ein eigenständiges Wiki zum internen Gebrauch installieren wollen. Es macht aber eine Menge aus, wenn Sie das nächste strategische Entwicklungstool für die IT-Architektur Ihres Unternehmens finden müssen.

Das zweite Problem besteht darin, dass Skriptsprachen (mit Ausnahme von PHP) für Verbindungen zu System i-Anwendungen weder vorgesehen noch wirklich geeignet sind, außer möglicherweise durch den doch recht beschränkten Umweg über gespeicherte Prozeduren. Und wenn Zend auch etwas anbietet, das wie eine System i-spezifische Schnittstelle aussieht, stellt sich doch bei näherer Betrachtung heraus, dass diese Lösung kaum mehr ist als ein Wrapper über dem Aufruf einer gespeicherten Prozedur. Sicher, es gibt ein "PHP Toolkit" und die "i5program template"; und ich hoffe, dass ich in den nächsten Wochen dazu kommen werde, diese näher unter die Lupe zu nehmen. Aber selbst, wenn das eine komplette Kopie der Java Toolbox ist, bleibt es doch eine Kopie.

Noch einmal: Wenn es um ein eigenständiges Utility geht, spielt dies alles keine Rolle. Wie es bei einer einzelnen Anwendung unter der Haube aussieht, ist nicht so bedeutend. Etwas ganz anderes ist es jedoch, wenn Sie das Werkzeug auswählen sollen, mit dem Sie die Suite Ihrer Kernanwendungen pflegen und erweitern wollen.

EGL ist für die Kommunikation geschaffen

Im Gegensatz zu den erwähnten Lösungen wurde EGL dafür entwickelt, mit anderen Sprachen und Plattformen, und zwar ganz besonders mit System i, zu kommunizieren. Mit Hilfe der Java Toolbox kann EGL RPG-Programme direkt über die i5/OS-Hostserver aufrufen. Im Moment sind bei EGL noch einige Aspekte des Konnektivitätsdesigns zu klären. Sobald EGL jedoch in der Lage sein wird, eine persistente Verbindung zum Host zu verwenden, werden die Aufrufe so schnell sein, dass sie eine praktisch nahtlose Verbindung zwischen Benutzeroberfläche und Geschäftslogik ermöglichen - auch, wenn sich diese auf unterschiedlichen Rechnern befinden. Stellen Sie sich vor, Sie rufen ein RPG-Programm auf und sehen nur Millisekunden nach dem Tastendruck das Ergebnis in einem Browser. Ein derartiges Maß an Interoperabilität bietet EGL - und das ohne eine einzige Zeile Java-Code.

Diese Interoperabilität stammt zu einem großen Teil aus dem zentralen Baustein der EGL-Programmierung, dem "Record". Mit dem Begriff "Record" sind in unserer Branche im Laufe der Jahre eine Vielzahl von Dingen bezeichnet worden. Das grundlegende Konzept einer Gruppe aufeinander bezogener Felder ist jedoch eines, das in anderen Sprachen häufig nicht vorhanden, oder wenn, dann nur schwach umgesetzt ist. C und die direkten C-Abkömmlinge unterstützen Strukturen. Viele Sprachen unterstützen Arrays, aber oft bedeutet das nicht viel mehr, als dass sich eine Menge von Objekten gleichzeitig handhaben lässt.

In der tatsächlichen Welt der Programmierung besitzt ein Record Informationen auf Feld-Ebene und auf Record-Eben. Hiermit sollte es möglich sein, die Anwendung zu definieren, und im besten Fall lässt sich mit diesen Metadaten ein Großteil des Codes generieren. Das Konzept von Ruby on Rails, das die Programmierkonventionen über die Anwendungsconfiguration stellt, ist in gewisser Weise eine Erweiterung des Metadaten-Konzepts mit überschreibbaren Standardattributen.

EGL geht hier noch einen Schritt weiter, indem es ermöglicht, alle Komponenten eines Record zu definieren, um dann mit einer einzigen Phrase (zum Beispiel dem zuvor erwähnten "get customers") den jeweiligen Code zu erzeugen. EGL versteht von sich aus SQL und MQSeries, so dass Sie mit sehr wenig Aufwand Bibliotheksfunktionen für die Verbindung zu i5/OS-Programmen erstellen können. Außerdem können Sie mit EGL die zum Code dazugehörigen Metadaten definieren, von Spaltenüberschriften bis zu Validierungsroutinen. Sie können die Eigenschaften eines Feldes einmal definieren, dann immer wieder in den verschiedensten Datensätzen verwenden und schließlich diese Datensätze auf verschiedenste Seiten anwenden - schon ist die gesamte Routinearbeit erledigt.

Ein WYSIWYG-Entwicklungstool für JavaServer-Faces

Dass der 5250 so lange so gut funktioniert hat, hatte seinen Grund: Er war das beste Tool zum Erstellen der Anwendungen, die Unternehmen brauchten. Die Unternehmen brauchten keine Tastendruck-für-Tastendruck-Kommunikation, sie waren darauf angewiesen, dass Benutzer schnell und effizient große Blöcke von Daten eingeben können. Die Schnittstelle war unkompliziert, und das Programm ließ sich direkt in einen einzelnen Bildschirm einbinden. Daher war die Blockmodus-Schnittstelle des 5250 die beste Lösung.

Heutzutage ist die Anwendungslogik erheblich komplexer - dies erklärt den Boom von Konzepten wie MVC. JavaServer Faces (JSF) ist die Benutzeroberfläche, mit der sich dieses Konzept am besten durchsetzen lässt. Während Skriptsprachen manchmal die Trennlinie zwischen HTML und Programmlogik verwischen, ist JSF eine Obermenge von JSP Model 2, der Java-Variante von MVC. Die JSF-Seite ist eine ganz andere Entität als der Servlet-Code, der sie verarbeitet. Da EGL von JSF abhängt, zwingt es den Entwickler dazu, MVC-gemäß zu denken. Und das Record-Konzept von EGL ist fest in den Editor integriert: Der Editor erkennt nicht nur die Datenkomponenten der Records, sondern auch die Metadaten, und kann alle diese Attribute automatisch formatieren.

Und damit bin ich beim Glanzpunkt von EGL angelangt, nämlich der WYSIWYG-Unterstützung der Sprache. Diese basiert auf der grundlegenden JSF-Unterstützung der Rational-Produkte. Um jedoch das Anfügen von Code an UI-Komponenten wirklich zu erleichtern, bringt EGL noch die Möglichkeit zur Point-and-Click-Programmierung mit. Wenn Sie schon einmal in Visual Studio ein Formular gezeichnet und dann den Widgets des Formulars Code hinzugefügt haben, können Sie in EGL sofort mit dem WYSIWYG-Seitendesigner loslegen. Sie können Variablen aus Ihrem Programm mit der Maus auf die Seite ziehen und dann deren Attribute festlegen, oder Sie können ein Widget (zum Beispiel eine Schaltfläche) aus einer Palette auf die Seite ziehen und mit einer Funktion Ihres Programms verbinden.

Noch wichtiger ist die Flexibilität

Damit haben wir also eine eindrucksvolle Liste der Gründe, die EGL so perfekt für das System i und für die Anwendungsentwicklung im Allgemeinen machen. Die Syntax ist einfach. Die Sprache baut auf dem Konzept "Record" auf, das die Erstellung eines Vorrats von Informationen auf Datenebene ermöglicht. EGL unterstützt dazu Bibliotheksfunktionen zum Einkapseln Ihrer Unternehmensregeln. Sie können diese Records verwenden, um schnell mit Hilfe eines WYSIWYG-Editors Webseiten zu erstellen, der wiederum sämtliche Funktionen von Rational nutzen kann. Sie können eine vollständige, mehrstufige Anwendung erstellen und komplett debuggen, ohne die Rational-IDE zu verlassen.

Und wenn Sie das alles noch nicht überzeugt, dieses Argument ist unwiderstehlich: Wenn Sie ein Scripting-Tool für Web-Applikationen (zum Beispiel RPG-CGI oder PHP) verwenden, benutzen Sie wahrscheinlich HTML als Schnittstelle. Die neuesten Versionen dieser Tools unterstützen ja gerade mal AJAX. Und wenn Sie damit auch recht schnell eine Anwendung zusammensetzen können, haben Sie danach bestimmt eine Menge UI-spezifischer Logik in Ihrem Code - sofern Sie nicht höchst vorsichtig mit Ihrer Architektur umgehen. Und das bedeutet, dass Sie beim Umstieg auf die nächste Benutzeroberfläche eine Menge Arbeit haben werden, nur um die Codebasis zu übertragen.

Bei EGL wird der Code dagegen aus sehr viel hochrangigeren Spezifikationen generiert. Und wenn die nächste EGL-Release zum Beispiel eine echte Rich-Client-Benutzeroberfläche unterstützt, können Sie sehr wahrscheinlich den größten Teil Ihrer Anwendung mit äußerst wenig Aufwand auf die neue Schnittstelle übertragen. Und statt erst alle Templates und dergleichen nachzurüsten, können Sie einfach beginnen, die neuen Funktionen sofort zu nutzen. Darüber könnte man doch mal nachdenken, oder?





Über den Autor

Joe Pluta ist Präsident von Pluta Brothers Design, Inc. und System i-Evangelist für moderne Technologiestrategien wie EGL. Seine Karriere umfasst Jahrzehnte in der Anwendungsentwicklung auf IBM-Midrange-Systemen, wobei er verschiedene Positionen in Entwicklung und Management inne hatte. Unter anderem war er Manager of Architecture für den weltgrößten Hersteller von AS/400-Software, der die ersten kommerziellen Client/Server-Anwendungen für IBM-Midrange-Systeme produzierte.

Joe Pluta hat umfangreiche Veröffentlichungen zu WebSphere Development Studio Client for iSeries (WDSC) und Eclipse vorgelegt. Darüber hinaus hat er Beiträge zum Eclipse-Projekt und zur IBM Java Toolbox für System i geleistet. Im Rahmen seiner langfristige Beteiligung an der Entwicklung dieser Tools ist er erstmals 2005 mit EGL in Kontakt gekommen. Nach eingehender Untersuchung ist Joe Pluta überzeugt, dass EGL die Technologie ist, mit der System i seine Rolle als weltweit wichtigste Entwicklungsplattform für Geschäftsanwendungen wieder geltend machen kann. Er hat eine Reihe von Machbarkeitsstudien für EGL entwickelt, darunter eine AJAX-Schnittstelle, die RPG als Geschäftslogik verwendet.

Joe Pluta hat zahlreiche Artikel über EGL verfasst und hält amerikaweit Vorträge zum Thema, zum Beispiel für Benutzergruppen und auf der bevorstehenden iSeries DevCon-Konferenz in Las Vegas. Er ist der einzige System i-Branchenexperte, der zur Rational Business Developer's Conference von IBM eingeladen wurde, um dort über EGL zu referieren. Sein Anliegen ist, die Grenzen des Anwendungsdesigns durch Verwendung der besten verfügbaren Technologien immer weiter auszudehnen.

Veröffentlichungen

- IBM Systems Mag
[HUhttp://www.ibmssystemsmag.com/i5/april07/developer/12353p1.aspx](http://www.ibmssystemsmag.com/i5/april07/developer/12353p1.aspx)
[HUhttp://www.ibmssystemsmag.com/i5/july07/developer/16318p1.aspx](http://www.ibmssystemsmag.com/i5/july07/developer/16318p1.aspx)
- MC Press
[HUhttp://www.mcpressonline.com/mc?.6b36d2b1](http://www.mcpressonline.com/mc?.6b36d2b1)
[HUhttp://www.mcpressonline.com/mc?.6b3a121c](http://www.mcpressonline.com/mc?.6b3a121c)
[HUhttp://www.mcpressonline.com/mc?.6b3a3717](http://www.mcpressonline.com/mc?.6b3a3717)
[HUhttp://www.mcpressonline.com/mc?.6b3cd66b](http://www.mcpressonline.com/mc?.6b3cd66b)