Grenzüberschreitung

Polyglotte Workflows mit Activiti und Silverlight

Wolfgang Pleus

Grafische Oberflächen mit Silverlight setzen Maßstäbe im Bereich der Benutzerfreundlichkeit und Produktivität. Gleichzeitig entwickelt sich Activiti zu einer ernst zu nehmenden Workflow-Engine. Der Artikel zeigt, wie sich beide Technologien zu einer leichtgewichtigen und agilen Workflowlösung kombinieren lassen.

Polyglotte Entwicklung – ist das sinnvoll?

Von polyglotter Softwareentwicklung wird gesprochen, wenn für die Realisierung eines Softwareproduktes unterschiedliche Programmiersprachen nebeneinander eingesetzt werden. Im JEE-Bereich war Java lange Zeit die einzige Sprache. In letzter Zeit kamen Sprachen wie Groovy, Scala oder Clojure dazu. Polyglotte Entwicklung ist vor allem dann sinnvoll, wenn sie zu einem Produktivitätszuwachs führt.

Beispielsweise eignet sich Groovy hervorragend für die Realisierung von domänenspezifischen Sprachen und kann zu diesem Zweck neben Java verwendet werden. .NET verfügt seit der ersten Stunde über zahlreiche Programmiersprachen. Dazu gehören beispielsweise C#, VB.NET, Delphi, Boo oder C++. Auch jenseits der eigenen VM kann eine gemischttechnologische Lösung sinnvoll sein, wenn sie in Bezug auf Nutzen und Produktivität Vorteile generiert.

Die Wahrnehmung der jeweils anderen Plattform ist im .NET- und JEE-Lager selten objektiv, sondern oft von Vorbehalten und Vorurteilen geprägt. Diese Haltung führt nicht selten dazu, dass die für eine Problemstellung adäquaten Technologien nicht zum Einsatz kommen. Eine unvoreingenommene Haltung kann jedoch sehr zum Vorteil für die zu realisierende Lösung sein. Schließlich gilt es als Erfolgsrezept, jedes Jahr eine neue Programmiersprache zu lernen (siehe [HuTh99]). Warum also nicht mal C# oder Java?

Während JEE seine Stärken vor allem im Server-Bereich zeigt, verfügt .NET im Bereich grafischer Oberflächen über hervorragende Technologien. Gerade eine Kombination beider Plattformen kann manchmal einen Produktivitätsgewinn generieren.

Aufgrund der syntaktischen und technologischen Nähe gestaltet sich die gemischte Entwicklung mit C# und Java relativ einfach. Auf sprachlicher Ebene finden sich sowohl Java- als auch .NET-Entwickler in der jeweils anderen Plattform schnell zurecht. Dies liegt einerseits an der syntaktischen Ähnlichkeit von C# und Java. C# ist Java wesentlich ähnlicher als beispielsweise Clojure oder Scala. Andererseits kommen die gleichen Entwurfs- und Architekturmuster, wie beispielsweise das Model-View-Controller-Muster, zum Einsatz. Selbst im Bereich der eingesetzten Bibliotheken gibt es eine große Überschneidung, da viele wichtige Frameworks, wie beispielsweise Hibernate, Lucene, Log4j oder JUnit, für beide Plattformen zur Verfügung stehen und so die Wiederverwendung des Wissens gegeben ist.

Sowohl Activiti als auch Silverlight lässt sich lizenzkostenfrei entwickeln und betreiben. Gerade in einem agilen Entwicklungskontext haben sich beide Technologien sehr bewährt. Silverlight ermöglicht es mit SketchFlow, Fachanwender kontinuierlich in den Designprozess einzubeziehen. Activiti ermöglicht durch BPMN die gemeinsame Prozessmodellierung durch Fachabteilung und IT.

Die Wiederverwendbarkeit des plattformspezifischen Codes stellt eine Einschränkung des gemischttechnologischen Ansatzes dar. Angesichts der Vorteile ist dies in der Regel tolerierbar, insbesondere da client- und serverseitiger Code meist andere funktionale Schwerpunkte aufweist. Portable Datentypen sollten mithilfe von XML Schema beschrieben werden. Aus dieser technologieneutralen Beschreibung lässt sich der plattformspezifische Code generieren. JEE und .NET bieten dafür eine Vielzahl von Werkzeugen wie beispielsweise `wsdl2java, xjc` oder `svcutil` an.

Die Beispiele in diesem Artikel wurden mit den folgenden Technologien entwickelt und getestet:

- Java SDK 1.6.0_24
- Apache Ant 1.7.1
- Activiti 5.4 [Act]
- Microsoft Visual Web Developer Express2010 [Web]

Kurzer Prozess

Um die wesentlichen Aspekte zu verdeutlichen, wird ein einfacher Prozess verwendet. Abbildung 1 beschreibt einen Bestellprozess für ein Mittagessen, der vom Benutzer *Gonzo* in der Rolle *Catering* gestartet wird. Der Benutzer *Kermit* aus dem *Management* bestimmt in dem interaktiven Benutzerschritt *Place order*, welche Speisen gewünscht sind. Kermit und Gonzo sind Standardbenutzer der Activiti-Installation. Nach der Bestätigung durch Gonzo wird die Bestellung im Schritt *Order food* verarbeitet. Die Aktivitäten des Prozesses sind in Listing 1 zu sehen.

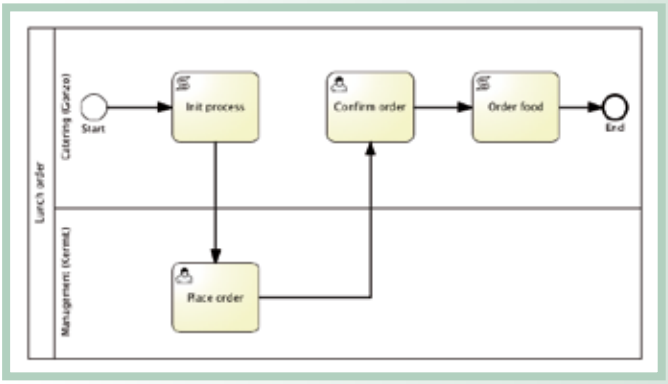

Abb. 1: Bestellprozess

```
<scriptTask id="init" name="Init" scriptFormat="groovy">
  <script>
    out:print "Initializing ordering prozess\n";
    def theorder =
      new net.pleus.order.v1.Order(
        day:new Date().plus(1), quantity:2,
        meal:net.pleus.order.v1.Meal.fromValue("Sushi"))
    lunch = new net.pleus.order.v1.Lunch(order:theorder)
  </script>
</scriptTask>

<userTask id="placeorder" name="Place order"
          activiti:formKey="order.form">
  <documentation>What would you like to eat?</documentation>
  <extensionElements>
    <activiti:formProperty id="theorder"
```

```
      expression="#{lunch.order}" type="order"/>
    </extensionElements>
    <humanPerformer>
      <resourceAssignmentExpression>
        <formalExpression>kermit</formalExpression>
      </resourceAssignmentExpression>
    </humanPerformer>
  </userTask>

  <userTask id="confirm" name="Confirm order">
    <documentation>Order now?</documentation>
    <humanPerformer>
      <resourceAssignmentExpression>
        <formalExpression>gonzo</formalExpression>
      </resourceAssignmentExpression>
    </humanPerformer>
  </userTask>

  <scriptTask id="orderfood" name="Order food" scriptFormat="groovy">
    <script>
      out:print "Ordering $lunch.order.quantity
        $lunch.order.meal.value on lunch.order.day.dateString\n"
    </script>
  </scriptTask>
```

Listing 1: Bestellprozess für ein Mittagessen

Prozessdaten

Der Prozess verwendet Daten vom Typ **`Order`**. Diese werden im Benutzerschritt **`Place order`** gefüllt und im Schritt **`Order food`** weiter verabeitet. Um eine portable und technologieneutrale Typdefinition zu erhalten, empfiehlt sich die Definition in XML Schema (s. Listing 2).

```
<xs:schema id="LunchOrder"
           targetNamespace="net/pleus/order/v1"
           elementFormDefault="qualified"
           xmlns="net/pleus/order/v1"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:simpleType name="Meal">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Pizza" />
      <xs:enumeration value="Sushi" />
      <xs:enumeration value="Burger" />
      <xs:enumeration value="Salad" />
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="Lunch">
    <xs:sequence>
      <xs:element name="order" type="Order"
                  minOccurs="1" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="Order">
    <xs:sequence>
      <xs:element name="meal" type="Meal"
                  minOccurs="1" maxOccurs="1" />
      <xs:element name="day" type="xs:dateTime"
                  minOccurs="1" maxOccurs="1" />
      <xs:element name="quantity" type="xs:int"
                  minOccurs="1" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Listing 2: Order.xsd-Schema

Activiti verwendet Java-Klassen für die Abbildung der Prozessdaten. Diese können beispielsweise mithilfe des im Java SDK enthaltenen Tools **`xjc`** aus dem XML-Schema generiert werden. Da Activiti in der verwendeten Version noch keine Variableninitialisierung innerhalb von BPMN unterstützt, wird dies mithilfe eines Groovy-Skripts im Prozessschritt **`init`** erledigt (s. Listing 1). Damit Activiti die generierten Typen verwenden kann, müssen sie als jar-Datei in das Verzeichnis **`TOMCAT_HOME\webapps\activiti-rest\WEB-INF\lib\`** kopiert werden. Der Prozess wird als zip-Datei gepackt und mithilfe der Activiti-Probe-Webanwendung deployt.

Benutzerformulare

Für eine Workflowlösung ist die Integration von Benutzern essenziell. Activiti unterstützt standardmäßig ein Postkorbmodell, mit dem Benutzern und Benutzergruppen Aufgaben zugewiesen werden können. Activiti verfügt dazu über Erweiterungen, die es erlauben, HTML-basierte Eingabeformulare für BPMN-Benutzeraufgaben festzulegen. Daten des Prozesses werden über JUEL-Ausdrücke [Juel] in die Formulare eingebettet. Diese Formulare werden dem Benutzer im Activiti-Explorer angezeigt, sobald der Prozess die Benutzeraufgabe erreicht und eine Benutzerinteraktion erforderlich wird.

Während dieser Ansatz für einfache Anwendungen ausreichend sein kann, erfordern gerade Geschäftsanwendungen meist eine nahtlose Integration in die Clientlandschaft des Unternehmens. Zudem werden oft zusätzliche Informationen benötigt, um dem Anwender qualifizierte Entscheidungen zu ermöglichen.

Activiti erlaubt den Zugriff auf Benutzeraufgaben und Formularparameter über ein natives Java-API. Teilweise ist dieses API über eine REST-Schnittstelle ansprechbar. Diese REST-Schnittstelle ist in der hier eingesetzten Version noch im Experimentierstadium und kann beispielsweise keine Daten schreiben. Daher eignet sie sich noch nicht vollständig für den produktiven Einsatz.

Um beliebigen Clienttechnologien Zugriff auf Activiti zu ermöglichen, kann das native Activiti-API mit einer Webservice-Schnittstelle versehen werden. Um das Prinzip zu verdeutlichen, implementieren wir exemplarisch die Methoden **`getTaskFormData`** und **`submitTaskFormData`** des Activiti-FormService, mit denen sich Formulardaten lesen und schreiben lassen. Diese werden mit Apache CXF [Cxf] entwickelt und als Java-Webanwendung (Activiti-API) parallel zur Activiti-Installation auf einem Tomcat-Server deployt. Der FormService stellt einen einfachen Wrapper zur Verfügung, der die Methoden des Activiti-FormService über SOAP zugänglich macht. Da der Wrapper unabhängig von den konkreten BPMN-Prozessen ist, kann er, einmal entwickelt, für beliebige Prozesse verwendet werden. Die Implementierung findet sich unter [SIGS].

Formulardaten

Die Teile des Prozesszustandes, die als Formulardaten verfügbar sein sollen, werden als Teil des **`userTask`**-Elements **`placeorder`** innerhalb des BPMN-Prozesses mit dem Attribut **`formProperty`** deklariert (s. Listing 1). Standardmäßig unterstützt Activiti die Datentypen **`long`**, **`string`**, **`date`** und **`enum`**. Für anspruchsvolle Szenarien ist die Unterstützung komplexer Domänentypen essenziell. Um das zu ermöglichen, kann Activiti durch eigene Formulartypen erweitert werden. Ein neuer Formulartyp wird durch eine Ableitung der Klasse **`AbstractFormType`** realisiert. Listing 3 zeigt die Implementierung des **`OrderFormType`** zur Serialisierung einer Bestellung aus dem obigen Beispiel.

```
public class OrderFormType extends AbstractFormType {
  public String getName() {
    return "order";
  }

  @Override
  public Object convertFormValueToModelValue(String json) {
    return new Gson().fromJson(json,Order.class);
  }

  @Override
  public String convertModelValueToFormValue(Object obj) {
    return new Gson().toJson(obj);
  }
}
```

Listing 3: OrderFormType.java

Die Serialisierung erfolgt hier exemplarisch mit GSON [Gson], womit ein Transfer komplexer Datentypen im JSON-Format ermöglicht wird. Um eigene Datentypen zu verwenden, werden sie in der Activiti-Konfiguration registriert (s. Listing 4). Dadurch wird es möglich, dem Benutzer den Prozesszustand ganz oder teilweise über Formulare bereitzustellen.

```
<beans>
  <bean id="orderFormType" class="net.pleus.activiti.OrderFormType"/>
  <bean id="processEngineConfiguration"
        class="org.activity.engine.iml.cfg.
               StandaloneProcessEngineConfiguraton">
    <property name="customFormTypes">
      <list>
        <ref bean="orderFormType"/>
      </list>
    </property>
    ...
  </bean>
</beans>
```

Listing 4: Registrierung OrderFormType in activity.cfg.xml

Der Client

Nach diesen Vorbereitungen ist es grundsätzlich möglich, mit jeder REST- und SOAP-fähigen Technologie Benutzeroberflächen für Activiti zu entwickeln. Im hier gezeigten Beispiel fällt die Wahl auf Silverlight. Silverlight ist durch die Unterstützung wichtiger Web-Standards in hohem Maße interoperabel und lässt sich somit sehr gut mit Activiti verbinden. Zudem können mit Silverlight Benutzeroberflächen erstellt werden, die in Bezug auf Interaktivität, Performance und Usability Maßstäbe setzen und sich somit insbesondere für hoch interaktive Branchenanwendungen eignen. Silverlight-Anwendungen können mit Visual Studio Express [Web] lizenzkostenfrei entwickelt und nahtlos in einen Maven- oder Ant-basierten Build-Prozess integriert werden [Pleus10].

Um von einer Silverlight-Anwendung auf serverseitige Services zugreifen zu können, muss eine Datei mit Namen **`clientaccesspolicy.xml`** (s. Listing 5) in das Root-Verzeichnis des Webservers kopiert werden. In diesem Beispiel ist das der Tomcat-Server der Activiti-Installation. Diese Datei definiert Regeln für den Zugriff auf HTTP-Ressourcen.

```
<access-policy>
  <cross-domain-access>
    <policy>
      <allow-from http-request-headers="*">
        <domain uri="http://*"/>
      </allow-from>
      <grant-to>
        <resource include-subpaths="true" path="/"/>
      </grant-to>
    </policy>
  </cross-domain-access>
</access-policy>
```

Listing 5: clientaccesspolicy.xml

Bei dem Beispiel handelt es sich um eine einfache Silverlight-Anwendung. Da der Fokus auf der Activiti-Integration liegt, wird auf grafische Effekte weitgehend verzichtet (s. Abb. 2). Die Benutzeroberfläche wird in XAML definiert (s. Listing 6).

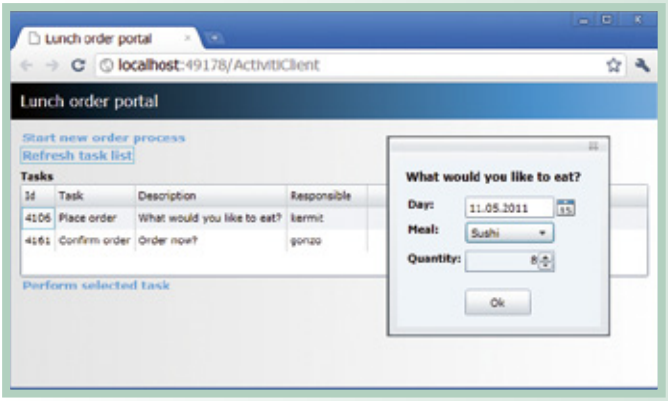

Abb. 2: Das Bestellportal

```
<StackPanel>
  <HyperlinkButton x:Name="Start" Content="Start new order process"
                   Click="Start_Click"/>
  <HyperlinkButton x:Name="Show" Content="Refresh task list"
                   Click="Show_Click"/>
    <TextBlock Text="Tasks" FontWeight="Bold" Margin="0,5,0,0"/>
    <my:DataGrid x:Name="TaskList" AutoGenerateColumns="false">
      <my:DataGrid.Columns>
        <my:DataGridTextColumn Header="Id" Binding="{Binding id}"
          IsReadOnly="True"/>
        <my:DataGridTextColumn Header="Task" Binding="{Binding name}"
          IsReadOnly="True"/>
        <my:DataGridTextColumn Header="Description"
          Binding="{Binding description}" IsReadOnly="True"/>
        <my:DataGridTextColumn Header="Responsible"
          Binding="{Binding assignee}" IsReadOnly="True"/>
      </my:DataGrid.Columns>
    </my:DataGrid>
  <HyperlinkButton x:Name="Perform" Content="Perform selected task"
                   Click="Perform_Click"/>
</StackPanel>
```

Listing 6: XAML-Auszug des Hauptdialoges

Silverlight verfügt über leistungsfähige Datenbindung. Daher reicht es aus, das **`DataGrid`** durch Setzen der Eigenschaft **`ItemsSource`** zu füllen (s. Listing 8).

Prozess starten

Der Start des Bestellprozesses erfolgt über die Standard-Activiti-REST-Schnittstelle **`process-instance`**. Diese erwartet die Prozess-Id im JSON-Format. Eine einfache Möglichkeit, diese HTTP-POST-Anfrage abzusetzen, besteht in der Verwendung der Silverlight-Klasse **`WebClient`** (s. Listing 7).

```
private void Start_Click(object sender, RoutedEventArgs e) {
  var body = @"{""processDefinitionId"":""lunchorder:1:6751""}";
  WebClient client = CreatePOSTClient(body);
  client.UploadStringCompleted += (s, a) => {
    var ex = new Regex("\"id\":.{0,1}\"([0-9]*)\"");
    var id = ex.Match(a.Result).Groups[1].Value;
    if(!String.IsNullOrEmpty(id)) {
      HtmlPage.Window.Alert(
        "Process [" + id + "] started successfully.");
    }
  };

  client.UploadStringAsync(new Uri(_serverUrl + "/process-instance",
    UriKind.Absolute), body);
}
```

Listing 7: Starten des Prozesses

Die Methode **CreatePOSTRequest** übernimmt die Authentifizierung und Konfiguration des **WebClient**s. Der JSON-String wird mit der Prozess-Id zusammengesetzt. Diese wird beim Deployment des Prozesses von Activiti automatisch vergeben und kann über Activiti-Probe ermittelt werden. Die Antwort erfolgt wie jede Serverinteraktion asynchron und wird mittels einer Lambda-Expression (das Äquivalent zum Closure in Groovy) über das Ereignis **UploadStringCompleted** verarbeitet. Die neue Prozessinstanz kann im Activiti-Explorer angezeigt werden.

Aufgaben anzeigen

Bei der Abfrage der zu erledigenden Aufgaben kann ähnlich vorgegangen werden. Diesmal wird die Anfrage per HTTP-GET gestellt. Um komplexere JSON-Datenstrukturen zu verarbeiten, kann der **DataContractJsonSerializer** eingesetzt werden. Dafür wird zunächst die JSON-Strukur als C#-Klasse **GetTaskResponse** implementiert. Der Serializer übernimmt die automatische Konvertierung zwischen JSON und C# (s. Listing 8). Eine manuelle Bearbeitung der JSON-Daten ist nicht erforderlich.

```
private void Show_Click(object sender, RoutedEventArgs e) {
  WebClient client = CreateGETClient();
  client.OpenReadCompleted += (s, ea) => {
    DataContractJsonSerializer ser = new DataContractJsonSerializer(
     (typeof(GetTasksResponse)));
    var res = (GetTasksResponse)ser.ReadObject(ea.Result);
    TaskList.ItemsSource = res.data;
  };

  client.OpenReadAsync(new Uri(_serverUrl +
    "/tasks?assignee=kermit"));
}
```

Listing 8: Anzeigen der Aufgabenliste

Das Ergebnis der Abfrage wird an das DataGrid-Steuerelement (s. Listing 7) der Oberfläche gebunden. So wird die Aufgabenliste automatisch visualisiert.

Aufgabe ausführen

Die Aufgabenliste enthält zusätzlich zu den angezeigten Informationen eine Task-Id und den Formularnamen **order.form**, der im BPMN-Prozess in der Aktivität **placeorder** definiert wurde. Mit der Task-Id lassen sich die Formulardaten über den Webservice-Aufruf **getTaskFormData** von Activiti anfordern. Um das zu erreichen, wird der Silverlight-Anwendung eine Servicereferenz hinzugefügt. Dabei werden der **FormServiceClient** sowie die Nachrichtenklassen aus der WSDL-Datei vollständig generiert. Das von GSON serialisierte **Order**-Objekt wird über die Hilfsmethoden **JSON2NET** und **NET2JSON** mit dem **DataContractJsonSerializer** zwischen .NET-Objekten und JSON konvertiert. Das **Order**-Objekt wird an das Silverlight-Formular **OrderForm** gebunden. Die Bestelldaten werden so über die Silverlight-Datenbindung automatisch im Modell **Order** aktualisiert. Nach dem Schließen des Dialogs werden die Änderungen über **submitTaskFormData** an Activiti übermittelt und der Prozess somit fortgesetzt (s. Listing 9).

JSON-Date-Portabilität

Die JSON-Repräsentation von Datumswerten ist nicht standardisiert. Daher kann es bei einer Cross-Plattform-Entwicklung dazu kommen, dass Datumswerte nicht korrekt ausgetauscht werden. Die von GSON verwendete Repräsentation wird als „day“:“*May 15, 2010 00:00:00 AM*“ serialisiert, während .NET Datumsfelder als „day“:“*\/Date(1273874400000+0200)\/*“ serialisiert. Die Nummer steht für Millisekunden seit dem 1. Januar 1970 UTC. In GSON lässt sich das Datumsformat über einen Aufruf von **GsonBuilder.setDateFormat** anpassen. In .NET lässt es sich mittels regulärer Ausdrücke vor der Serialisierung anpassen. Dieser Ansatz wird im Beispiel verfolgt.

```
private void Perform_Click(object sender, RoutedEventArgs e) {
  var client = new FormServiceClient();
  var task = TaskList.SelectedItem as GetTasksResponseValue;

  if (task.formResourceKey == "order.form") {
    // Callback Formulardaten lesen
    client.getTaskFormDataCompleted += (s, a) => {
      // Order von JSON nach .NET umwandeln
      var tfd = a.Result as taskFormData;
      Order order = JSON2NET(tfd);

      // OrderForm als Popup erzeugen
      var dlg = new ChildWindow();
      dlg.Content = new OrderForm(order, dlg);
      dlg.Closed += (s1, a1) => {
        // Order von .NET nach JSON umwandeln
        var fps = new formProperty[1] {
          new formProperty {name = tfd.formProperties[0],id,
                            value = NET2JSON(order)}};

        // Formulardaten schreiben
        client.submitTaskFormDataAsync(task.id, fps);
      };

      // OrderForm anzeigen
      dlg.Show();
    };

    // Formulardaten lesen
    client.getTaskFormDataAsync(task.id);
  }
}
```

Listing 9: Aufgabe durchführen

Für Java-Entwickler dürfte die asynchrone Ausführung mittels Lambda-Expressions zunächst etwas ungewohnt sein. Durch die Asynchronität wird ein „Einfrieren“ der Benutzeroberfläche während der Webservice-Aufrufe effektiv verhindert. Der Implementierungsaufwand für hochwertige Benutzeroberflächen mit Silverlight ist gering. Eine Generierung

der Oberflächen zur Laufzeit ist ebenfalls denkbar, um den Aufwand noch weiter zu verringern. Im Beispiel ist das Formular `OrderForm` Bestandteil des Silverlight-Clients. Durch das Managed Extensibility Framework [Mef] können Formulare zur Laufzeit nachgeladen werden, um so ein modulares Deployment zu erreichen.

Fazit

Dieser Artikel zeigt, wie sich eine lizenzkostenfreie BPMN-basierte Workflowlösung aufsetzen lässt, die sich nahtlos in bestehende Frontends integriert. Die Eigenentwicklung beschränkt sich dabei auf ein Minimum. Sowohl Activiti als auch Silverlight bestechen durch ihre hohe Produktivität und Nähe zum Endbenutzer. Durch die Modularität und Standardunterstützung beider Technologien ist eine Integration leicht möglich. Der vollständige Quellcode ist unter [SIGS] zu finden.

Literatur und Links

[Act] Activiti, `www.activiti.org`
[Cxf] Apache CXF, `cxf.apache.org`
[Gson] google-gson, A Java library to convert JSON to Java objects and vice-versa, `code.google.com/p/google-gson`
[HuTh99] A. Hunt, D. Thomas, The pragmatic programmer, Addison-Wesley, 1999
[Juel] Java Unified Expression Language, `juel.sourceforge.net`
[Mef] Managed Extensibility Framework, `mef.codeplex.com`
[Pleus10] W. Pleus, Silverlight für Java Enterprise !?, in: JavaMagazin, 05/2010, `entwickler.de/zonen/portale/psecom,id,101,online,3292.html`
[SIGS] Quellcode des vorgestellten FormService `http://www.sigs-datacom.de/nc/fachzeitschriften/javaspektrum/archiv.html`
[Web] Visual Web Developer Express, `www.microsoft.com/express/Web`

Wolfgang Pleus arbeitet als Technologieberater, Autor und Trainer im Bereich moderner Softwarearchitekturen. Seit über fünfzehn Jahren unterstützt er internationale Unternehmen bei der Realisierung komplexer Geschäftslösungen auf der Basis von Java EE und .NET. Er ist Mitglied im Accelsis SOA/BPM Competence Team.
E-Mail: wolfgang.pleus@pleus.net