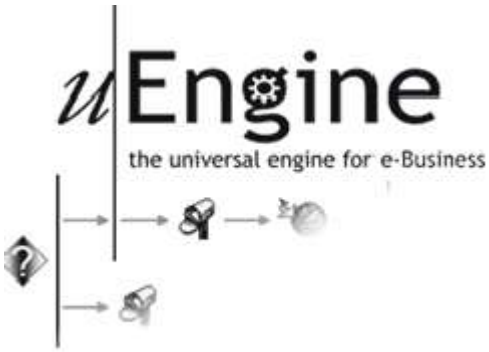


Jinyoung Jang, uEngine Administrator
uEngine Open Source Project 2004-07-01
Translated by Xinlei Zhao

uEngine 开发手册
ver 1.0

1. 简介

现今的电子商务环境基于象Web Services的网络应用交互过程。为了支持这样的环境，既需要整合不同的商业资源而且要与技术的灵活保持同步。uEngine 是基于web services的工作流管理系统，支持以上的适应性和灵活性。uEngine 的主要应用范围是商业流程管理、B－B 整合与应用整合。而且，uEngine 是一个开放代码的软件，以便广泛的开发者参与和合作。

[提示] 这个指南将描述uEngine version v1.0 RC-0 standalone 需要以下几项技术基础:

- 工作流管理系统 (WfMS) or 商业流程管理系统 (BPMS)
- Web Services (SOAP/WSDL/UDDI)
- J2EE
- 面向对象应用结构
- Javabeans 结构和映射
- 设计形式
- XML and XML Binding Framework (JAXB)
- Swing 结构
- JSP and Portlets

2. uEngine的特点

2.1. 现今工作流管理系统的热点问题

基于工作流的开发使开发者从繁重的过程管理实现中例如：涉及、运行和管理中解脱出来。但是，以下障碍仍然阻碍它大范围的在开发过程中应用：：

- 能不能在一个工作流built－in包里来实现所有的功能？（用户化和适应性）

即使一个工作流系统包含很多功能，它还是需要根据商业和技术要求添加新的活动类。增加一个新的结构化功能需要很多系统系数和功能的改变。“能不能在一个时间和预算内把所有的东西都用一个工作流包来实现”，这个问题经常出现在项目管理者的面前。

- 在其他环境中能否应用？（移植性和开放性）

这个问题是与上面提到基于包应用发展相同的问题。这些功能在一个特定的环境下工作良好，但是在另一个环境下就无法工作。这个问题会影响到这个包究竟会不会被采用的决定上。

- 能否与其他工作流引擎交互? (交互性)

为了在交互－工作流概念下实现B2B整合，需要不同的工作流相互访问。但是，尽管有很多

标准化的努力，现在仍然很难实现在不同的环境或者语言下不同工作流引擎的交互性。

2.2. uEngine 设计原理

uEngine 反映 1) 控件结构(component frameworks), 2) Web Services, 和 3) 开放的设计原理架构来解决上面提到的问题。

*u*Engine =

A Workflow Management System
+
A CBD Tool
+
A Web Services Coordination Tool

2.2.1. 基于控件的适应的革新的引擎

uEngine 是一个建立在控件结构上的引擎，它实现逻辑和技术的分开使开发者能够很容易的添加新的功能而不用考虑实现一个工作流的产品，例如：XML，Web Services, Swing, Messagine Queuing 甚至是EJBs。并且，不仅新的功能甚至新的流程都可以轻松的添加到独特的结构组织中去。这是uEngine与其他工作流引擎相比另一个显著的特性。

2.2.2. 基于 Web Services 架构

在uEngine下设计和执行商业流程可以通过互联网实现Web Services。这可以用WSDL-SOAP动态地整合面向服务的架构（SOA），从而使这些商业流程在互联网上自由的相互访问。为了达到这个目的，uEngine使用了这些设计思想：基于XML流程变量来实现广泛的数据交互，动态Web Services 活动象发现、激活和运载一个服务，与广泛的资源管理，它包含象通过Web Services 的接口软件和人力资源的各种商业资源。

2.2.3. 简单和开放的架构

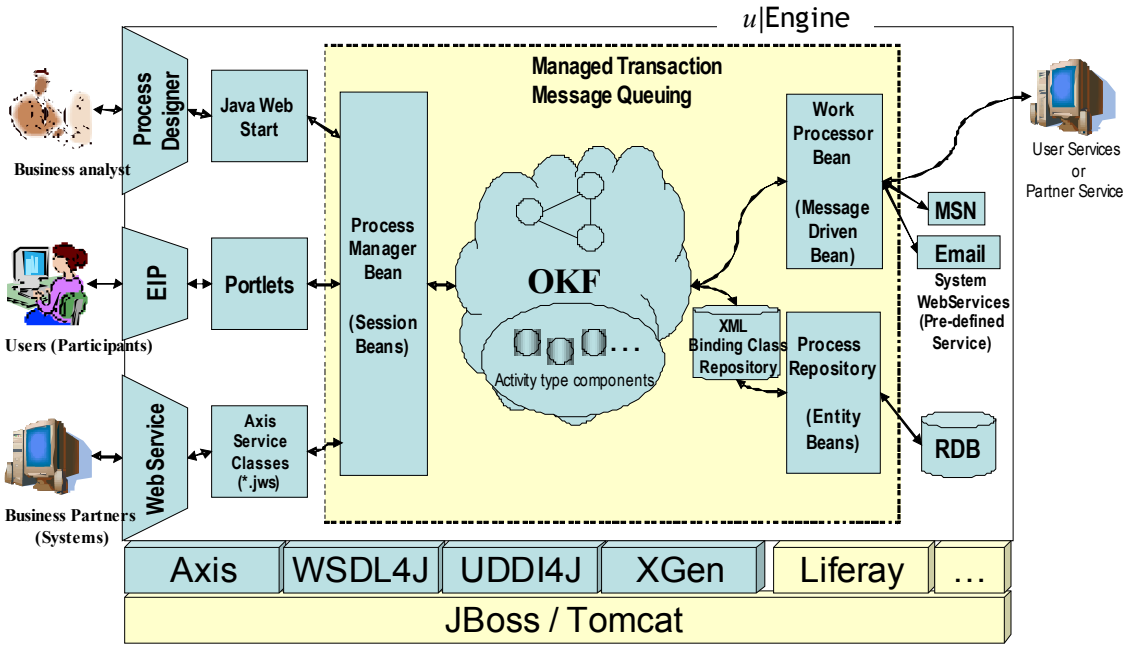
一个好的用户化是一个工作流引擎的重要条件。一些软件使用比实际需要多很多很复杂的技术，迫使开发者不得不全部理解他们，这样的软件在灵活的e-Business 不会是一个成功的环境。uEngine 有两个技术应用策略。一个是“尽可能少地使用技术”，另一个是“尽量选择比较熟悉的架构”。这些策略最大限度地实现用户化，而且使更多的开发者理解和最终选择uEngine。

3. uEngine 架构的理解

这部分会描述整体的架构和基本的模型，这会有助于深入理解源代码。

3.1. uEngine 体系架构

uEngine 是一个建立在最熟悉的架构基础上，这些架构在设计原理中已经提过。



3.1.1. J2EE 作为系统机构:

J2EE 提供一个稳定的易于管理的环境，这个环境支持在业务处理、消息传递和持久性工作流逻辑。一个静态bean（过程管理bean）作为一个服务协议来接受对过程管理的请求和Web Service 激活，一组消息beans（工作流程Bean，错误过程Bean）是过程流程控制的可靠过程而三个实体bean保存过程信息。所有这些商业逻辑都不是用EJBs[1]来实现的。但是，大部分请求都实际上由 OKF 运行，OKF是 uEngine 核结构，而且J2EE提供了一个服务协议和一个环境实现方法。

3.1.2. Web Service 接口连接工作流之间的控件

uEngine's 工作流控件，执行引擎，工作任务分配，过程设计，运行和管理网络协议等等通过 Web Services 接口（SOAP）来整合。这样更容易更替部分控件和提高他们各自的系统使之能成为再利用的系统。例如：uEngine's 自定义分布与Liferay's 协议系统通过SOAP整合，后者是EIP 解决方案的开放代码。开发者很容易通过WSDL定义替换工作表系统。

3.1.3. 基于stack的源代码:

uEngine 仅采用了开放的源代码作为自己的stack。 Apache's Axis 用来 Web Service stub generation 和显示过程。 WSDL4J 提供绑定信息而且 UDDI4J 与 UDDIbrowser (在 sf.net) 提供 web service discovery 功能.

3.1.4. 广泛的 persistence 机制 – XML 序列化与实体beans

uEngine 在XML和对象下执行一个广泛的数据类型。为了避免一个复杂的表示结构，uEngine 用 XGen序列化所有对象，XGen是一个开放代码的JAXB 工具包，并且可以存储实体beans。这是一

[1] 这是一个折中的方法，避免了过度使用EJB。而且，它允许加入新的uEngine控件模型。

个简单但是实用的不通过复杂的数据设计来表现多样数据的解决方案。

3.1.5. Portlets 作为信息提供者:

uEngine 提供一套标准化的portlets来进行过程管理、用户管理和统计。这样允许为每个个体展现免费的个性化的信息。

3.1.6. Java Web Start实现简单访问:

uEngine's 过程设计和一些内嵌工作项目管理例如文本开发可以使用JWS技术通过网络浏览器访问到。

[提示] uEngine 使用一个GUI 应用整合结构 "Metaworks"来实现过程设计，工作项目管理，管理协议等等。

3.2. uEngine 服务协议 -"过程管理" 接口

"org.uengine.processmanager.ProcessManagerBean" 是一个静态，来接受外部要求。以下是一个coarse-grained 接口定义[2]

```
public String initializeProcess(String processDefinition, String instanceId) throws RemoteException;
public String initializeProcess(String processDefinition) throws RemoteException;
public void executeProcess(String instanceId) throws RemoteException;
public void setProcessVariable(String instanceId, String scope, String varKey, Serializable val) throws RemoteException;
public Serializable getProcessVariable(String instanceId, String scope, String varKey) throws RemoteException;
public String getProcessVariableInXML(String instanceId, String scope, String varKey) throws RemoteException;
public Hashtable listProcessVariableValues(String instanceId) throws RemoteException;
public void putRoleMapping(String instanceId, String roleName, String endpoint) throws RemoteException;
public String getRoleMapping(String instanceId, String roleName) throws RemoteException;
public Serializable sendMessage(String instanceId, String message, Serializable payload) throws RemoteException;
public Serializable sendMessageXML(String instanceId, String message, String payload) throws RemoteException;
public String[] listProcessDefinitionNames() throws RemoteException;
public String[] listProcessInstanceIds() throws RemoteException;
public String[] listProcessInstanceIds(String definitionName) throws RemoteException;
public ProcessDefinitionRemote getProcessDefinitionRemote(String processDefinition) throws RemoteException;
public ProcessDefinitionRemote getProcessDefinitionRemoteWithInstanceId(String instanceId) throws RemoteException;
public void addProcessDefinition(String definitionName, String definition) throws RemoteException;
public String viewProcessDefinitionFlowChart(String processDefinition, Map options) throws RemoteException;
public String viewProcessInstanceFlowChart(String instanceId, Map options) throws RemoteException;
```

[2] 这种设计减少了使用的复杂性，避免了激活网络包和存储静态对象的内存的浪费。

```
public Hashtable getActivityDetails(String instanceId, String tracingTag) throws RemoteException;
public String getActivityStatus(String instanceId, String tracingTag) throws RemoteException;
public String getProcessDefinition(String processDefinition, String encodingStyle) throws RemoteException;
public void flowControl(String command, String instanceId, String tracingTag) throws RemoteException;
public void removeProcessInstance(String instanceId) throws RemoteException;
public void removeProcessDefinition(String definitionName) throws RemoteException;
```

[提示] 若寻找每一个操作代表的意义，请参阅"was/server/default/deploy/uengine-web" 目录下JSP 文件或者 javascripts provided in the script console.

3.3. uEngine Persistence 结构

uEngine 从内核分离出 Persistence 逻辑。uEngine 从 "org.uengine.kernel.ProcessDefinitionFactory" 提取过程定义并且从 "org.uengine.kernel.ActivityInstance" 提取实例数据，它还授权给自定义实现需求－EJBActivityInstance， 使用 JAXB 和实体beans。开发者可以根据他们自己的需要来编写他们自己的用户化的Persistence 结构。

自定义 persistence 机制

ProcessDefinitionFactory 和 EJBActivityInstance 使用间接的方案尽可能少限制来保持对象的多样。这个方案是通过用XGen序列化所有对象，XGen是一个开放代码的JAXB 工具包，并且可以存储实体beans。这是一个简单但是实用的不通过复杂的数据设计来表现多样数据的解决方案。但是，在后期查询和分析已经存储的数据时不是一个好方案。自定义 persistence 机制与以下三个实体beans和 XML 序列化一起使用：

1. org.uengine.persistence.processdefinition.ProcessDefinitionRepositoryBean
- Comment: 在XML保存过程定义
- Fields: ID –过程定义的名字, Description – 过程定义的描述, Definition – 过程定义内容

[提示] 如果你需要在相关数据库查询基础上分析过程定义，你可以设计和书写grained table 和实体beans。

2. org.uengine.persistence.processinstance.ProcessInstanceRepositoryBean
- Comment: 存储过程实例基本信息
- Fields: ID – 过程实例 ID, Definition – 实例本身过程定义名字, StartedDate – 开始时间

3. org.uengine.persistence.processdefinition.ProcessVariableRepositoryBean

- Comment: 存储每个过程变量值和与过程实例有关的数据，例如：过程步骤、职务mapping图等等。它没采用复杂的数据库设计，但是不方便查询与分析。
- Fields: ID – 每一项特定的ID, InstanceId – 自身所属的过程实例, KeyString – 指定值的关键字文本, Value – 值本身

[提示] 当你需要分析或者查询过程数据，我们推荐使用SQLActivity。也就是说：不是通过直接使用uEngine 的数据，而是产生audit数据并在以后的应用阶段通过一个好的形式被处理。这种方式可以提供一个良好的执行并降低复杂性。

3.4. uEngine 的活动控件模型－紧凑形式和JavaBeans

表 1 显示出uEngine 使用的活动控件的对象模型。这个模型是著名'Composite Pattern[3]' 模型的拓展形式。

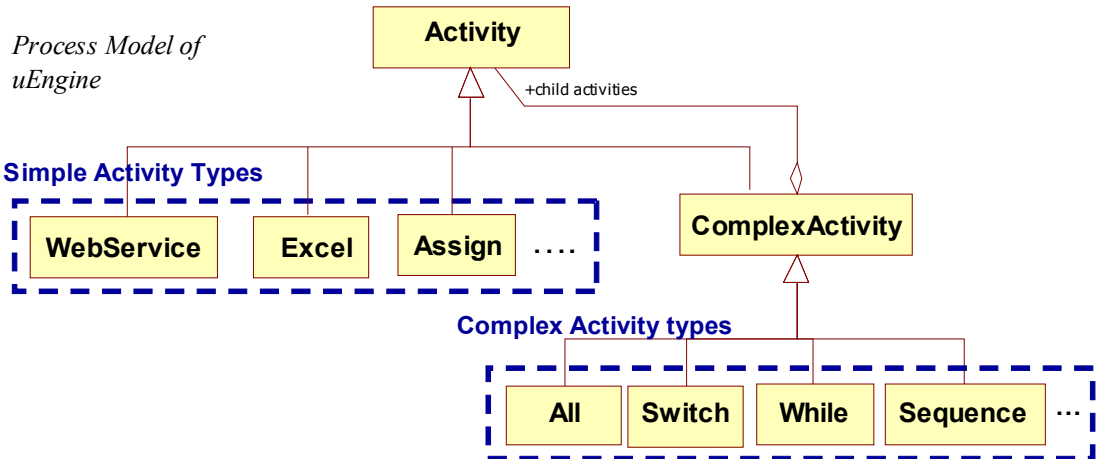


Figure 1. uEngine 的活动控件模型

这个自我封装的对象模型在容纳模块流程模型上是很好的。而且它不仅可以容纳简单的活动，即只包含任务逻辑的活动，还可以包含复杂的活动，即包含它子活动流程控制逻辑的活动。比如：Web Service 激活，发送E-mail，人的任务。这些是简单的活动，顺序、All, Switch 和 Try~Catch 是复杂的活动类型，它可以包含简单活动及包含子活动的复杂活动。

除了meta 模型外，uEngine 活动控件模型遵守Javabeans 的setter/getter 程序可以允许由映射支持的系统设计。uEngine 过程设计作为一个活动类型的Bean 盒子并产生输入形式和相关的GUI 通过得到活动类型对象模型来编辑活动类型。这可以使开发者不用担心象是使用AWT还是

[3] 请参考gang of four's的书 – "design patterns" 。

Swing这样的GUI 的实现来添加新的活动类型。

3.4.1. 主要的活动类型 – 活动和复杂活动

为了理解uEngine 的活动控件模型在实际中的应用，有必要来看一下源代码中“Activity.java” 和“ComplexActivity.java”，他们是主类。

[ToDo: 一个 collarboration 图表显示设计者，执行服务器，和管理、监控工具与这两个类的关系。

#Activity.java

Activity.java 包含主要的方法来表示一个活动的行为，属性和反馈的信息。

1. Properties (Fields)

```
public abstract class Activity{

        Activity parentActivity ;   // its parent activity (an ComplexActivity)
        String name;
        String tracingTag;          // it can be a unique identifier for the activity in a process definition tree
        int retryLimit;             // count for retry
        int retryDelay;             // time interval for retry (in seconds)
        Role[] roles;                           // role declarations
        ProcessVariable[] processVariableDescriptors;    // process variable declarations
```

2. Constructor

```
        public Activity(){}         // there sould be an empty constructor for reflection
```

3. Logics called by enactment engine

These methods would be called by the enactment engine at runtime. The usual invocation sequence is ‘createInstance → beforeExecute → executeActivity → onEvent (if done or failed) → afterExecute’.

```
        protected ActivityInstance createInstance(ActivityInstance instanceInCreating) throws Exception{
                return instanceInCreating;
        }
        protected void beforeExecute(ActivityInstance instance) throws Exception{}
        abstract protected void executeActivity(ActivityInstance instance) throws Exception;
```

```
protected void afterExecute(ActivityInstance instance) throws Exception{}
protected void onEvent(String command, ActivityInstance instance, Object payload) throws Exception{}
```

4. Logics called by monitoring/admin tools

These methods stand for monitoring and admin time and provide some information or controls on the running processes. They may be called by any reasons caused in the execution time as well.

```
protected synchronized String getProcessStatus(ActivityInstance inst, String roleName){} // returns the status

public void stop(ActivityInstance instance) throws Exception{}                    // replies to a stop request
public void suspend(ActivityInstance instance) throws Exception{}        // replies to a suspend request
public void resume(ActivityInstance instance) throws Exception{}         // replies to a resume request
public void skip(ActivityInstance instance) throws Exception{}                    // replies to a skip request
public void compensate(ActivityInstance instance) throws Exception{}     // replies to a undo request

public ActivityDesigner createDesigner(){}          // provides a GUI for editing this activity

public Map getActivityDetails(ActivityInstance inst){// provides activity details with keyed-values
        return details;
}

public ValidationContext validate(){}               // returns validation result with the current configuration of the activity.
                                                    // This method may usually be invoked at the design time.
```

5. Utility methods

```
public Activity getRootActivity(){ return root; }      // returns the most parent activity
public ProcessDefinition getProcessDefinition(){  // returns the ProcessDefinition this activity belongs
        return (ProcessDefinition)getRootActivity();
}

public void fireComplete(ActivityInstance instance) throws Exception{
                                    // developer use this method to demarcate where the activity succeed
        onEvent(ACTIVITY_DONE, instance, null);
}
```

```
        public void fireFault(ActivityInstance instance, Exception fault) throws Exception{ //use when the activity has failed
                onEvent(ACTIVITY_FAULT, instance, fault);
        }
}
```

#ComplexActivity.java

ComplexActivity.java 是Activity.java 的一个子类，它的延伸主要是管理子活动。

```
public class ComplexActivity extends Activity {
```

1. Properties (Fields)

```
        private Vector childActivities;          // a vector holds its child activities
```

2. Constructor

```
        public ComplexActivity(){}
```

3. [Concrete method] The overriden 'createInstance' method – initializes its own properties as well as the childs'.

```
        public ActivityInstance createInstance(ActivityInstance instanceInCreating) throws Exception{

                super.createInstance(instanceInCreating);

                //Sets current running state
                instanceInCreating.set(getTracingTag(), "currentStep", "0");

                //Lets each child activity initialize process instance
                Vector childActivities = getChildActivities();
                for(Enumeration enum = childActivities.elements(); enum.hasMoreElements(); ){
                        Activity child = (Activity)enum.nextElement();
                        child.createInstance(instanceInCreating);
                }

                return instanceInCreating;
        }
```

4. The overridden 'executeActivity' method – will contains its flow control logic. Following is the example of the sequence.

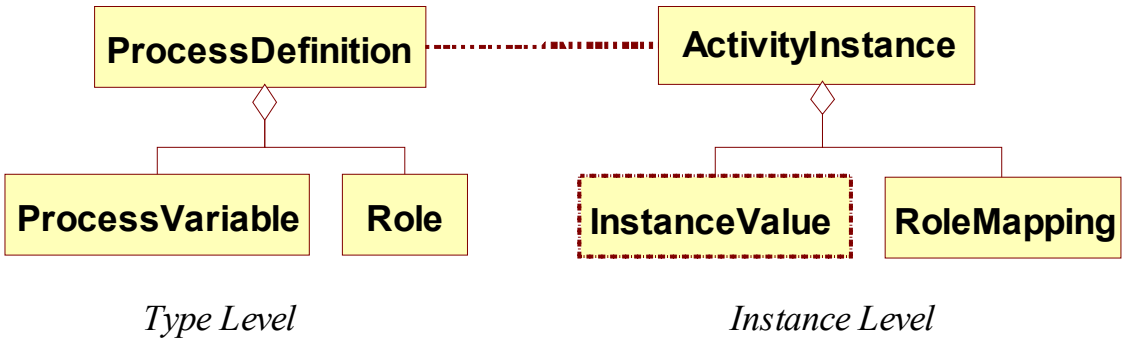
```
protected void executeActivity(ActivityInstance instance) throws Exception{
        int currStep = getCurrentStep(instance);
        if(currStep >= childActivities.size()){
                instance.fireDone();

                fireComplete(instance);

                return;
        }

        Activity childActivity = (Activity)getChildActivities().elementAt(currStep);
        childActivity.executeActivity(instance); // The actual code uses JMS
}
```

5. The overridden 'onEvent' method – This method processes not only its own messages but also childs' messages so that it can listen to the status of its childs.

```
protected void onEvent(String command, ActivityInstance instance, Object payload) throws Exception{
        //dispatching leaf childs
        if(command.equals(CHILD_DONE)){
                int currStep = getCurrentStep(instance);
                currStep++;
                setCurrentStep(instance, currStep);

                executeActivity(instance);              //execute next
        }else
                super.onEvent(command, instance, null);
}

....
}
```

[提示] 为了观察活动类型中现实的例子，请参看org.uengine.kernel 包中活动类型，他们的名字以'~Activity'结尾。

3.4.2. 最重要的活动类 – 过程定义



类 ProcessDefinition 是一个最高层的复杂类，它包含一个过程定义树。既然这个类继承 SequenceActivity，它的子类都是顺序排列的。这个类还包含一些显示过程定义的一些额外方法。

3.4.3. 类的任务和类的任务图

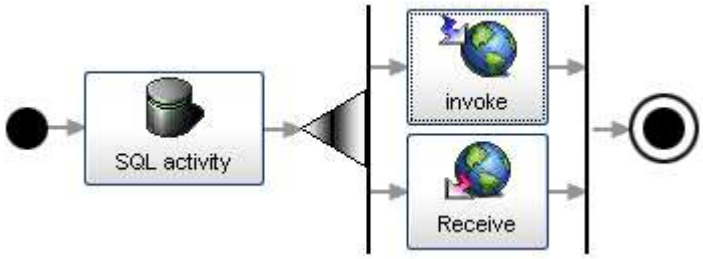
类的任务代表一个商业过程定义的任务，类的任务图是任务分派的一个实例。类的任务包含任务定义的识别，类的任务图存储了实际工作者的email地址或者系统终端URL地址。

3.4.4. Class ProcessVariable

ProcessVariable은 프로세스 변수 선언 클래스로서, 변수명, 타입, QName(XML일 경우)등을 담도록 합니다. 실제값은 ActivityInstance의 set메서드로 XML로 변환되어 저장됩니다.

3.4.5. 过程定义树

uEngine 的过程定义实例会形成一个树结构的基于活动控件模型上的。下面是一个例子。




```
public class SimpleProcess extends ProcessDefinition{
    public SimpleProcess(){
        setProcessVariables(new ProcessVariable[]{
```

```
    new ProcessVariable(
        "name", "testvar",
        "type", String.class
    }),
    new ProcessVariable(new Object[]{
        "name", "purchaseOrder",
        "type", String.class
    })
});

Activity act = new SQLActivity();
addChildActivity(act);

ComplexActivity comp = new AllActivity();

WebServiceActivity wact = new WebServiceActivity();
wact.setRole(Role.forName("worker"));
wact.setPortType("WQSService");
wact.setOperationName("orderWork");
wact.setParameters(new Object[] {
    "writePurchaseOrder",
    "http://localhost:8085/axis/services/TroubleTicket",
    null,
    "parameter"
});
wact.setOutput(ProcessVariable.forName("result"));
comp.addChildActivity(wact);

AssignActivity assignAct = new AssignActivity ();
assignAct.setVariable(ProcessVariable.forName("testvar"));
assignAct.setVal("content");
comp.addChildActivity(assignAct);

addChildActivity(comp);

ReceiveActivity receive = new ReceiveActivity();
```

```
        receive.setOutput(ProcessVariable.forName("purchaseOrder"));
        receive.setMessage("PurchaseOrderArrived");
        addChildActivity(receive);

    }
}
```

3.5. uEngine 的控件结构－开放的内核结构

uEngine 是一个基于控件结构不仅经常提高它的功能还保持它的质量和原先的设计。特别的，uEngine 的基于控件按照工作流发展的模式提供了一个系统分析和发展方法。也就是说，这个方法可以帮助开发者在面向过程商业应用中产生更多的控件。

开放内核结构 (OKF) 是uEngine 控件结构的名字。表 2 显示用UML表示的OKF的类型操作过程。

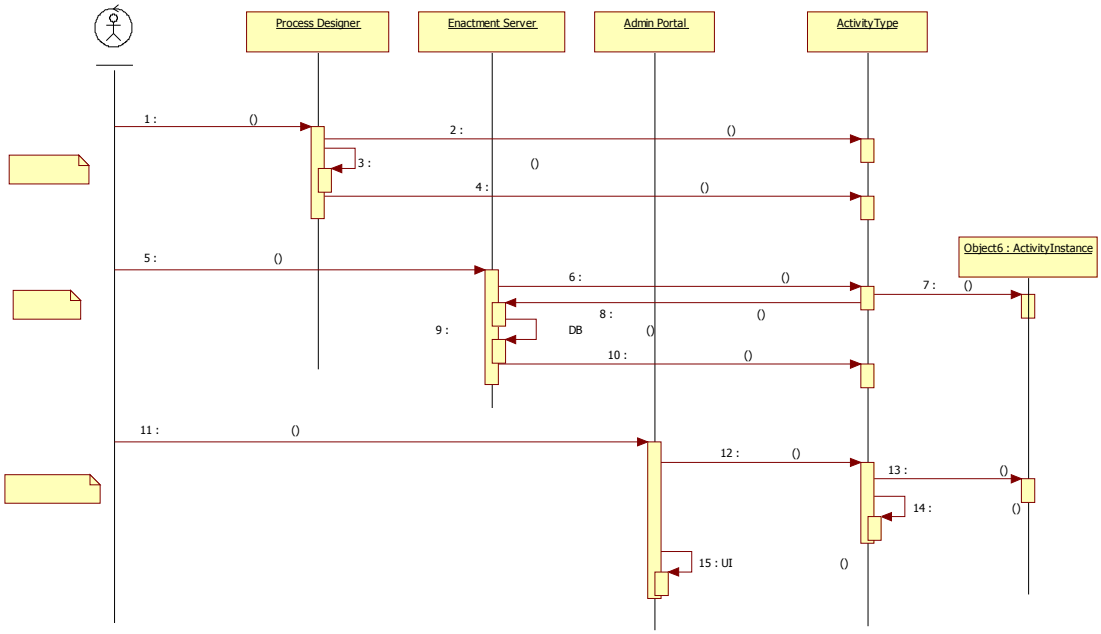


Figure 2. 'Inversion of Control' of Open Kernel Framework

OKF的内核管理所有的应用控制和在过程实例生命周期内：设计、证明有效性、管理、观察、Persistence和执行的过程中必要时激活活动控件的方法。这种“控制的倒置”提供了一种简便、稳定的管理方法而不必担心许多的技术问题。

3.5.1. 一个活动类实现的例子

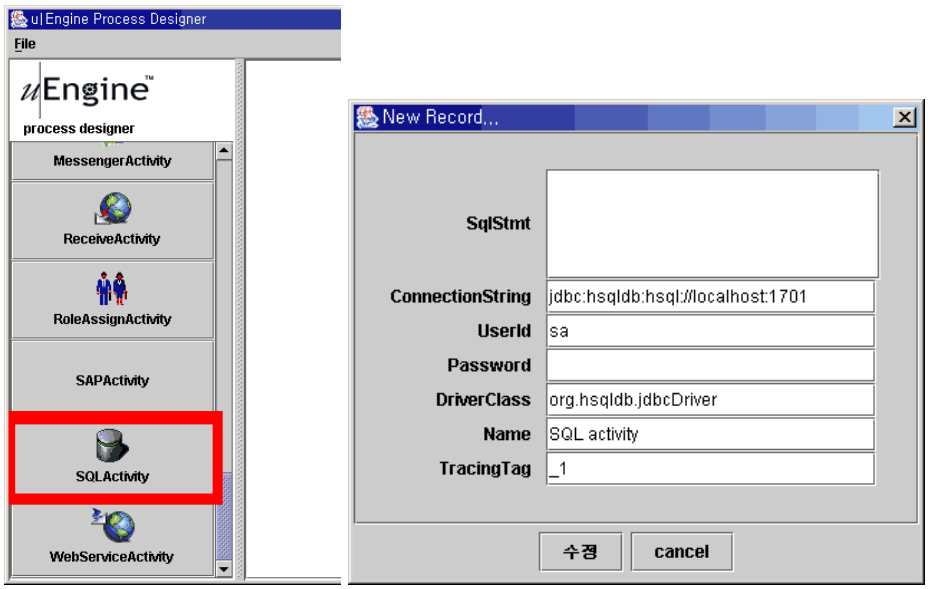
下面的源代码是一个活动类实现的例子，它描述了 uEngine 和活动类型控件相互之间的关系。

```
public class SQLActivity extends Activity{

    //Activity property declarations ////////////////////////////
    //-----------------------------------------------------------
    // Each property should have its setter/getter following the javabeans programming convention
    // so that uEngine's framework can access the values by reflection APIs.

    TextContext sqlStmt;
        public TextContext getSqlStmt(){
            return sqlStmt;
        }
        public void setSqlStmt(TextContext value){
            sqlStmt = value;
        }

    String connectionString;
        public String getConnectionString(){
            return connectionString;
        }
        public void setConnectionString(String value){
            connectionString = value;
        }

    String userId;
        public String getUserId(){
            return userId;
        }
        public void setUserId(String value){
            userId = value;
        }

    String password;
        public String getPassword(){
```

```
                return password;
        }
        public void setPassword(String value){
                password = value;
        }

String driverClass;
        public String getDriverClass(){
                return driverClass;
        }
        public void setDriverClass(String value){
                driverClass = value;
        }

public SQLActivity(){}

public ActivityInstance createInstance(ActivityInstance instanceInCreating) throws Exception{
        super.createInstance(instanceInCreating);

        //Make a space to store the instance property for result log
        instanceInCreating.set(getTracingTag(), "resultLog", "");

        return instanceInCreating;
}

public void executeActivity(ActivityInstance instance) throws Exception{
        Class.forName(getDriverClass());
        Connection con = DriverManager.getConnection(getConnectionString(), getUserId(), getPassword());

        PreparedStatement prepStmt = con.prepareStatement(getSqlStmt().parse(this, instance));
        ResultSet rs = prepStmt.executeQuery();

        if(rs.next()){
                ByteArrayOutputStream resultLog = null;

                try{
```

```
                    ResultSetMetaData rmeta=rs.getMetaData();
                    for(int i=1; i<=rmeta.getColumnCount(); i++){
                        String fieldName = new String(rmeta.getColumnName(i));

                        instance.set("", fieldName, (Serializable)rs.getObject(fieldName));
                    }
                }catch(Exception e){
                    ...
                    e.printStackTrace(new PrintStream(resultLog));
                }

                if(resultLog!=null)
                    instance.set(getTracingTag(), "resultLog", resultLog.toString());
            }

            prepStmt.close();

            fireComplete(instance);
        }

        public Map getActivityDetails(ActivityInstance inst){
            Hashtable details = (Hashtable)super.getActivityDetails(inst);
            details.put("actual executed SQL", );

            return details;
        }

}
```

编译和登陆完新的活动控件，uEngine会提供象你们下面看到的一个设计时间支持系统。：



<ToDo: plus monitoring time image>

3.5.2. 其他控件接口

还有一些其他与活动控件相关的控件接口，主要是用来支持控制工作流管理周期的拓展功能。

以下是与一个活动控件有关的控件接口：

- Process Viewers (org.uengine.kernel.viewer.*)
  - Comment: Viewer components that is for generates HTMLs that shows a flow chart of a process definition or a running process instance.

Interface:

```
public interface ActivityViewer{
    // Reads given acitivity's properties and render its flow chart in HTML. And returns it.
        StringBuffer render(Activity act, ActivityInstance inst, Map options);
}
```

Naming rule: [ActivityName]Viewer ex) SQLActivityViewer

Time to be used: Monitoring

Caller classes: org.uengine.processmanager.ProcessManagerBean

- Activity Descriptors (org.uengine.kernel.descriptor.*)

Comment: Description classes that represent additional attributes on acitivty types. They are commonly used in the design time and provides metadata for GUI generation. The 'Metaworks' framework is related to this component interface.

Interface: (A subclass of class 'Table', which is a component interface of metaworks framework. See metaworks developer guide for details)

Naming rule: [ActivityName]Descriptor ex) SQLActivityDescriptor

Time to be used: Design, Monitoring

Caller classes:

- Activity Designers (org.uengine.kernel.designer.*)

Comment: GUI components that can present given activity definition.

Interface:

```
public interface ActivityDesigner{
        public Activity getActivity();          //returns editing activity
        public void setActivity(Activity value);
        public void onDropped(java.util.Vector components);//will be called when an activity is dropped on this activity
        public ComplexActivityDesigner getParentDesigner();     //returns the parent activity's editor
        public void openDialog();           //open an editing form window for editing this activity
        public java.awt.Component getComponent();   //returns an actual awt(or swing) control
}
```

Naming rule: [ActivityName]Designer ex)SQLActivityDesigner

Time to be used: Only Design

Caller classes: will be generated by the org.uengine.kernel.Activity's method 'getActivityDesigner' and will be used by org.uengine.processdesigner.ProcessDesigner.

- Activity Image (org.uengine.kernel.images.*)

Comment: Icon image files that represents each activity type

Naming rule: [AcitivityName].gif ex)SQLActivity.gif

Time to be used: Design, Monitoring

以下控件接口经常在uEngine 系统中使用

- Unified Serializers (org.uengine.components.serializers.*)

Comment: This interface provides a standard protocol to access variety kinds of serialization mechanisms.

Interface:

```
public interface Serializer{
        public boolean isSerializable(Class cls);  //returns whether this
        public void serialize(Object sourceObj, OutputStream os, Hashtable extendedContext) throws Exception;
```

```
        public Object deserialize(InputStream is, Hashtable extendedContext) throws Exception;
}
```

Name rule: [NameOfMechanism]Serializer ex)BPELSerializer, BeanSerializer, AxisSerializer 등

Time to be used: All the time

Caller classes: uengine.kernel.GlobalContext

- Process Model Adapters (org.uengine.processpublisher.*)

Comment: Adapter interface for importation or exportation of uEngine process model into various process languages or formats. When user requests another expression of uEngine process definition, uEngine first converts the source model into the destination model by using these adaptors and then the corresponding serializers would emits proper XML with these converted objects into the destination language.

Interface:

```
public interface Adapter{
        public Object convert(Object src, Hashtable keyedContext) throws Exception;
}
```

Naming Rule:

1. Importer: [LanguageName]/importer/[SourceLanguageActivityName]Adapter
2. Exporter: [LanguageName]/exporter/[DestinationActivityName]Adapter
   ex) WebServiceActivityAdapter

Time to be used: All the time

Caller classes: The serializers for each language. ex) BPELSerializer.java of BPEL

- Relation Strategies (org.uengine.components.relations.*)

Comment: This interface component-ize relations between human resources or resolution policies to resolve a eligible actor for a role within organization or directory. A Strategy pattern. An example usage of relation strategies is dynamic staff resolution.

Interface:

```
public interface Relation {
        public RoleMapping findRelative(RoleMapping roleMap);
}
```

Naming Rule: [RelationName]Relation

example: An relation strategy for resolving headquarter's endpoint.

```
public class HeadQuarterRelation implements org.uengine.kernel.Relation{
        public RoleMapping findRelative(RoleMapping roleMap) {
            //Extracts the headquarter's endpoint from database and binds it to 'headquarter'
            return headquarter;
        }
    }
```

3.5.3. 活动控件目录结构

这部分会解释如何制作一个uEngine控件包和如何把它们插入到uEngine内核中去。uEngine 活动控件是一个jar文件，支持目录结构和下面描述的metadata形式。:

目录结构:

```
Mycom.jar
    + META-INF
      + MANIFEST.MF
      + activitytypes.xml
    + com
      + [companyname]
        + [projectname]
          + [ActivityTypeName].class
            + images
              + [ActivityTypeName].gif
            + viewer
              + [ActivityTypeName]Viewer.class
            + descriptor
              + [ActivityTypeName]Descriptor.class
            + designer
              + [ActivityTypeName]Designer.class
```

Metadata文件实例:

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.4.0_01" class="java.beans.XMLDecoder">
 <object class="java.util.ArrayList">
  <void method="add">
   <string>com.mycom.PDFGenerationActivity</string>
  </void>
 </object>
</java>
```

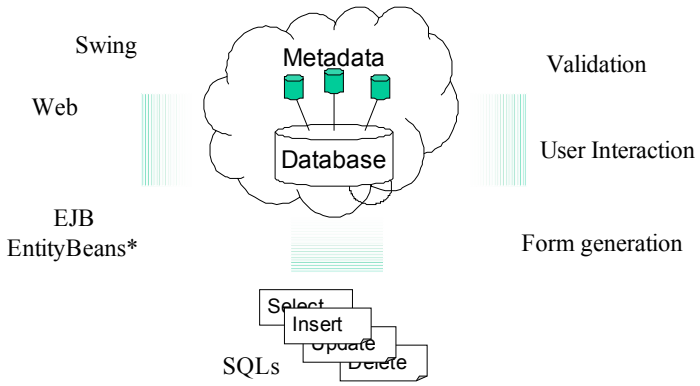
做完以上jar文件并运行一个any 目标来注册这个控件：

```
Ant DeployActivityTarget -Dparam.component.dir=<location of the jar file> -D param.component.file=<name of the jar file>
```

[提示] 你会看到 'was/server/default/deploy/mycom.jar' 作为索引.

3.6. Metaworks 结构

Metaworks 是“面向metadata 应用整合结构”的缩写。它可以通过索引metadata的应用对象来实现象SQL、UI、Web、EJBs等的软件控件。这对集中管理和metadata改变有好处，而且可以产生一个公开的程序。比如：仅仅改变 metadata XML就可以自动地更改所有地技术实现。不仅如此，metawork的象“从初始化的用户化”和“JIT UI 个性化”发展方法可以极大地减少面向市场的时间，降低费用。



Metaworks 用在uEngine过程设计和工作项目管理的实现上。

[提示] Metaworks 是一个 uEngine's 姐妹项目并可以单独运行。更多参考，请看

http://metaworks.sourceforge.net.

4. uEngine 开发者安装指南

4.1. 安装和运行 uEngine standalone RC-0

Directory Name	Comment and key files in
Demo	**Demo processes and docuements**
Doc	**Installation guide, developer guide, apidocs**
Lib	**Libarary jars**
Metaworks	**Metaworks project directory**
Src	**uengine project directory**
/bin	**Compilation directory**
/build	**Directory for building 'uengine.jar'**
/exposedprocesses	프로세스 정의를 웹서비스로 공개했을 때 생기는 스켈레톤 클래스들이 여기에 임시로 생성됨. 생성후에 **deploy/axis.war/WEB-INF/classes**에 복사됨
/resources	**EJB** 설정파일들 **(**디플로이 데스크립터 등**)**
/src	**Source files**
/scripts	**Scripts for scripting console**
/stubs	**Temporal directory for generating Web Service stubs. Process designer will generates them with a ant call. The generated stubs will be copied to 'was/server/default/deploy/stubs.jar'.**
build.xml	ant툴로 사용할 수 있는 여러가지 명령들을 제공
Scriptconsole.bat	스크립팅 콘솔을 실행하는 배치파일
Was	기본 WAS로 제공된 JBoss 디렉토리
/bin/run.bat or run.sh:	JBoss실행을 위한 스크립트
/server/default/deploy /uengine.jar	uEngine 엔진 바이너리
/server/default/deploy /uengine-settings.jar/org/uengine/uengine.properties	uEngine 설정파일. 이 파일을 열어서 몇 가지 설정을 맞추어 주어야 합니다.
/server/default/deploy	uEngine 웹어플리케이션들(JSP)

首先，下载在 www.sf.net/projects/uengine uEngine standalone v1.0 RC-0 并运行 'was/bin' 的 run.bat 文件。

[提示] uEngine 在JRE v1.4测试。 如果你的服务器正常启动，在Run.bat 的首栏加入如下的一段文字：

```
Set JAVA_HOME=<Directory for JRE1.4>
```

[提示] 有些时候, JBoss 可能不能运行数据库文件，在Run.bat 加入如下一行：

```
Cd <The absolute path of JBoss_HOME\ bin>
```

[提示] 既然ExcelActivity 用FTP作为文档服务器，你需要提供一个FTP服务器实例和相关的信息。你可以提供信息给uengine.properties.

```
excelsheetactivity.uploadftpaddress=localhost
excelsheetactivity.uploadftpdirectory=uengine_upload
excelsheetactivity.ftpid=administrator
excelsheetactivity.ftppw=18925
workitemhandler.address=localhost:8082
```

4.2. CVS 应用更新

uengine 应用更改，你可以从模块“uEngine” 和“uEngine-web”到“$UENGINE_ROOT$/src” 和“$UENGINE_ROOT$/was/server/default/deploy/uengine-web”更新。

[提示] 模块 ‘src’ 与 ‘ejbjar’ 不再使用.

4.3. 用 Ant 编辑

首先，你在运行ant需要如下设置：

```
C:\>set JAVA_HOME=c:\j2sdk1.4.0_01
C:\>set JBOSS_HOME=c:\uengine\standalone2\was
```

在“$UENGINE_ROOT$/src” 目录下，你可以呼叫ant来编辑所有源代码，并发布给WAS。

```
C:\>cd c:\uengine

C:\uengine>ant

Buildfile: build.xml


prepare:


compile:

    [delete] Deleting 36 files from C:\uengine\standalone2\src\bin


ejbjar:


makeJar:

        [jar] Building jar: C:\uengine\standalone2\src\build\uengine.jar

deploy2jboss:

       [copy] Copying 1 file to C:\uengine\standalone2\was\server\default\deploy

       [copy] Copying 2 files to C:\uengine\standalone2\was\server\default\deploy\ext.ear\portal-ejb.jar\org\uengine\admin\portlet\processmanager


applySettings:

        [jar] Building jar: C:\uengine\standalone2\settings\bin\uengine_settings.jar


signjar:

       [echo] jarfile=C:\uengine\standalone2\src/../settings/bin/uengine_settings.jar


generateProcessDesignerJWS:


signjar:

       [echo] jarfile=C:\uengine\standalone2\src/build/uengine.jar


BUILD SUCCESSFUL

Total time: 1 minute 30 seconds
```

4.4. 用Eclipse 编辑

<TODO>

以下配置不是必须的，但他们对于使用附加功能来说是必要的。

4.5. 支持Web Services 配置

4.5.1. 对过程设计的配置

uEngine 在设计阶段呼叫 Ant 目标来产生 Web Services 节点，显示它的过程并管理WSDLs。因此，uEngine 过程设计需要一些Ant工具和Jboss 目录的相关信息。通过点击菜单栏下的'Edit > Settings'，你可以使用配置窗口。以下是一个配置的例子。

[提示] 空着'uEngine.axis.directory'. uEngine 会从JBoss root 路径来判断相应取值。仅仅当你需要一个特殊配置的时候在改变它。

4.5.2. 自定义 Web Services

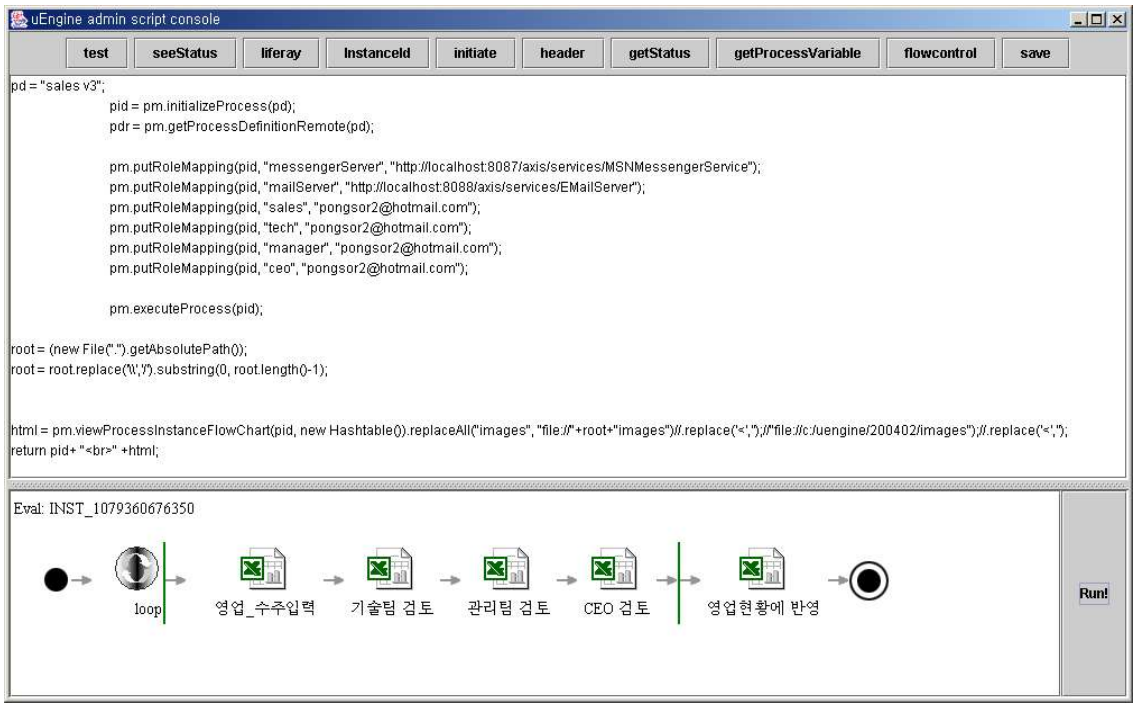
uEngine 为工作项目管理、通信、和信息传递[4]提供一些定义好的Web Services。他们提供一个 loosely coupled 接口和如何实现uEngine 兼容的系统资源。他们由Axis 应用目录下自定义产生。既然这些Web Services 是完全独立的，你可以更改源代码来改变配置。以下是一个介绍如何配置MSNMessengerService的例子。

[4] 他们是 WebServiceActivity的子类，是已经配置好的为每个服务web service 活动。

```
package org.uengine.webservices.msnmessenger.impl;

public class MSNMessengerServiceImpl implements
org.uengine.webservices.msnmessenger.MSNMessengerService{
        final static String msnUserId = "uengine_82@hotmail.com";
        final static String password = "pongsor2";
        ....
}
```

4.6. 应用Script 控制台

这个scripting 工具是为 uEngine 的adhoc 接口服务的。复杂的用户或者开发者可以使用javascript 通过uEngine 实例来获得或者检测一些信息。你可以想象这类似与一个Oracle下的一个SQL*Plus。



你可以运行$UENGINE_HOME$/src/scriptconsole.bat 来启动script 控制台。用法是：首先选择一个script 点击它。然后改变一些参数或者添加一些参数然后点击右下脚的“Run”按钮。运行结构会显示在下面的面板里。这个script 文件存在$UENGINE_HOME$/src/scripts。

5. uEngine 源代码概述

5.1. 包概述

Package name	Comment and key components inside
org.uengine.admin	ScriptConsole.java: The script console
org.uengine.admin.portlet.processmanager	Portlets for process management.
org.uengine.components.resources	(Not used)
org.uengine.components.resources.database	(Not used)
org.uengine.components.serializers	Unified serializers i.e.) BPELSerializer, AxisSerializer, BeanSerializer (see 3.5.2)
org.uengine.contexts	Context classes represent a type of object. This is related to the metaworks and declarative programming.
org.uengine.example	Example processes in form of source code
org.uengine.kernel	The core package that contains core activity types and classes participating in the process model. Especially, GlobalContext.java is a singleton utility class, which provides centeric functions and is referenced and used by a lot of classes.
org.uengine.persistence.processdefinition	The EntityBean stores process definitions
org.uengine.persistence.processinstance	The EntityBean stores process instances
org.uengine.persistence.processvariable	The EntityBean stores process variables

org.uengine.processdesigner	The package of Process designer
org.uengine.processdesigner.inputters	Input policies for each types of classes.
org.uengine.processmanager	A Stateless Bean for process management service.
org.uengine.processpublisher	Adapters for supporting multiple process languages (see 3.5.2).
org.uengine.processpublisher.bpel.export	The processpublisher for BPEL4WS import/export
org.uengine.queue.faultqueue	A Message Driven Bean for fault signaling. (Queue Name: uEngine-FaultQueue)
org.uengine.queue.workqueue	A Message Driven Bean for work queueing. (Queue Name: uEngine-WorkQueue)
org.uengine.security.authority	(Alpha)
org.uengine.util	Utility classes
org.uengine.util.ftp	
org.uengine.util.mailing	
org.uengine.webservice	Default Web Services

http://kldp.net/projects/uengine