An Interface from YAWL to OpenERP

Joerg Evermann

Faculty of Business Administration, Memorial University of Newfoundland, Canada
jevermann@mun.ca

Abstract. The paper describes an interface from the YAWL workflow management system to the OpenERP enterprise system. The interface is implemented as a codelet, and provides access to the full range of OpenERP information and functions. The paper provides an overview of the design of the codelet, the data types for its use, and an example application.

Keywords. YAWL, OpenERP, workflow management, enterprise system

1 Introduction

The YAWL (Yet Another Workflow Language) workflow management system, through its modular architecture that is based on web-services and other standards, provides multiple ways to extend the system for use with both SOA (service-oriented architecture) and legacy applications. One important interface requirement in many usage scenarios is to an ERP (enterprise resource planning) system. While many ERP systems provide their own best-practice processes and workflow engines, these engines and their workflow description languages are rarely complete with respect to workflow patterns, based on a formal foundation like the YAWL language (which provides design-time analysis capabilities), or as easy to configure and use as the YAWL language and system.

The OpenERP system is an open-source ERP system that provides core modules such as sales, purchasing, accounting, production management, as well as extensions for point-of-sales, project management, etc. Among open-source ERP systems, it is one of the most mature and feature complete systems. OpenERP provides its own process model and workflow engine. However, the configuration language is XML based and there is no recognizable formal underpinning for the workflow description language.

This suggested the development of an interface from YAWL to OpenERP, so that OpenERP functionality can be used in a YAWL workflow. The remainder of the paper describes the implementation of this interface as a YAWL codelet. The next section provides an introduction to the OpenERP system, followed by design choices for the interface codelet. This is followed by a description of the codelet parameters and data types, and an example workflow.

The codelet, associated XML Schema data type definitions, and YAWL example workflow definitions are available under the GPL v2 license.

2 The OpenERP System from the YAWL perspective

OpenERP is developed using the Python language and provides a business object model that abstracts, through an object-relational mapping layer, from the underlying physical data structures and functions. On top of this object model, OpenERP defines a process model and workflow mechanism for many of the business objects. OpenERP provides an XML-RPC based web-services interface to both its business objects and its workflow mechanism. This interface provides the following generic operations on all business objects:

- Create (returns the new business object ID)
- Search (returns a set of business object IDs that match a query)
- Read (returns a set of attribute values for a given list of business object IDs and a given list of attribute names)
- Write (updates the provided attributes of business objects with a given list of IDs with the provided new value)
- Delete ("Unlink")

OpenERP also provides a means to call methods defined on the business objects. However, calling the methods directly (outside the built-in, intended workflow) may lead to issues such as the (implicit) pre-conditions (as per the built-in workflow) not being met, or the consequent actions (as per the built-in workflow) not being executed. Thus, calling the business object methods directly is not recommended. Instead, OpenERP provides a mechanism to send "signals" to its workflows. These signals can be used to advance the built-in workflow for a business object. For example, a signal may be sent to confirm a draft sales quotation and transform it to a sales order. The OpenERP workflow mechanism then calls the appropriate methods on the business object. While this is a "safe" mechanism to interact with the OpenERP system, it also entails the following constraints:

- The externally defined and externally controlled workflow must be essentially isomorphic to the internally configured workflow
- Developing an external workflow requires a thorough understanding of the built-in workflow, the states of the business objects, and the business object methods that are called for state transitions.

The OpenERP workflow model is based on object states and transitions between them. Each object *state* is associated with a method, whereas transitions are either triggered by signals, or triggered by changes in attribute values. The workflow model is described in XML language but can be depicted graphically, as in Figure 1 for the sales workflow. Transitions are annotated by pre-conditions on top of the horizontal bar, and signals below. Transitions with pre-conditions only are data triggered, whereas transitions with signals are triggered when OpenERP receives that signal.

Fig. 1. OpenERP sales workflow (adapted from http://doc.openerp.com/)

3 Design

YAWL provides different ways to integrate external systems, the three most prominent being the web-services invoker service, the codelet mechanism, and the external data gateway. Any of these can in principle be used to develop the interface.

As the YAWL web-service invoker service requires a valid WSDL file and uses SOAP rather than XML-RPC, this would have required a translation server that accepts SOAP requests and issues XML-RPC requests in turn. Alternatively, a new XML-RPC web-service invoker service could have been built as an alternative to the

existing SOAP based one. Either of these alternatives was considered to be too technically complex for the limited time-frame of the project.

An external data gateway could be constructed to access either the business object information in OpenERP, or directly access the underlying relational data. Technically, one could also imagine that this might be used to access methods or send workflow signals, but this would not be conceptually sensible, as the data gateway is intended primarily for data access.

Instead, a simple codelet was developed that accepts input and provides output using pre-specified data types. Two options were investigated:

- Offer access to specific OpenERP business objects, their data, methods, and workflow signals. In this scenario, XML data types would need to be developed that reflect the OpenERP business object model, e.g. a data type for the "sales order" object, a data type for "sales order line" object, etc. This would remove the burden of data type development from the YAWL process designer, but would at the same time limit the flexibility of the codelet to a fixed set of business objects, their data and methods as determined by the codelet design.
- Offer access to generic OpenERP operations (see Section 2) using general-purpose data types. This requires the codelet user, i.e. the YAWL process designer, to develop appropriate business object data types and deal with the specifics of data transformation on the YAWL side, e.g. as part of the input and output mappings for tasks. The benefit is that the codelet does not prescribe specific data types, and it can access any OpenERP business object or workflow. The codelet was developed based on this, second model.

The codelet itself is stateless and establishes a new connection to the OpenERP system for every call, thus requiring the OpenERP connection information with every call. While this may not be as efficient as returning a connection handle to the YAWL workflow, the fact that the YAWL workflow may be long-running means that a connection handle in the YAWL workflow data might expire. Further, as tasks in the YAWL workflow may be assigned to different (human) resources, maintaining a quasi-persistent connection handle would also force the same OpenERP user account for the entire workflow, which may not be desirable in practice.

4 Parameters and Data Types

Table 1 below lists the input parameters for the codelet for every OpenERP call. The codelet returns a result named `Result` of type `ResultType`, described below.

Parameter	Type	Description
`URL`	`xsd:String`	Hostname for OpenERP
`Port`	`xsd:Integer`	Network port for OpenERP
`Database`	`xsd:String`	OpenERP database to select
`Username`	`xsd:String`	Username for OpenERP
`Password`	`xsd:String`	Password for OpenERP

`Object`	`xsd:String`	Type of OpenERP business object on which method or action is to be called
`Method`	`xsd:String`	OpenERP method name
`Parameters`	`ParameterType`	Parameters appropriate for the called method

Table 1. OpenERP codelet input parameters

The content of the `method` parameter is limited to the five generic methods for OpenERP business objects: `search`, `write`, `read`, `delete`, `create`, and the additional `action`, which is used for sending signals to OpenERP workflows. The ParameterType data type for the input `parameters` is defined as follows:

```
<xsd:complexType name="ParameterType">
 <xsd:choice>
  <xsd:element name="SearchParameters" type="SearchParamsType"/>
  <xsd:element name="CreateParameters" type="CreateParamsType"/>
  <xsd:element name="ReadParameters" type="ReadParamsType"/>
  <xsd:element name="WriteParameters" type="WriteParamsType"/>
  <xsd:element name="DeleteParameters" type="DeleteParamsType"/>
  <xsd:element name="ActionParameters" type="ActionParamsType"/>
 </xsd:choice>
</xsd:complexType>
```

The `ActionParamsType` allows the codelet user to send a signal to an OpenERP business object with a certain ID:

```
<xsd:complexType name="ActionParamsType">
 <xsd:sequence>
  <xsd:element name="Action" type="xsd:string"/>
  <xsd:element name="ID" type="xsd:integer"/>
 </xsd:sequence>
</xsd:complexType>
```

Another example, the `WriteParamsType`, is defined in Listing 1 in the appendix and allows the codelet user to write a set of values to a set of fields for business objects whose ID is specified in the provided list.

One of the challenges in the codelet design is the fact that fields in OpenERP business objects that refer to other business objects may be of cardinality one-to-one, many-to-one, or many-to-many. This is reflected in the `typeEnumerationType` in Listing 1 and the fact that the cardinality of the `<value>` element of the `fieldValuePairType` (Listing 1) is unbounded. The types `many2one` and `one2many` are only supported for reading of values, not writing or creating. When the codelet reads fields of this type, OpenERP returns a serialization of a Java array that contains the IDs and names of the referred to business object. The codelet decodes this information and encodes the list of IDs and names in the appropriate XML datatypes. When used for writing or creating business objects, only the first `<value>` element is read, and assumed to be of type string.

The results that the codelet returns depend on the invoked method. Alternatively, the codelet returns an error, either passed back from OpenERP, or an exception in the codelet, or an error encountered by the codelet, e.g. when the input paramters do not match the invoked method type.

```
<xsd:complexType name="ResultType">
 <xsd:choice>
  <xsd:element name="Error" type="xsd:string"/>
  <xsd:choice>
   <xsd:element name="SearchResults" type="SearchResultsType" />
   <xsd:element name="CreateResults" type="CreateResultsType" />
   <xsd:element name="ReadResults" type="ReadResultsType" />
   <xsd:element name="WriteResults" type="xsd:boolean" />
   <xsd:element name="DeleteResults" type="xsd:boolean" />
   <xsd:element name="ActionResults" type="xsd:boolean" />
  </xsd:choice>
 </xsd:choice>
</xsd:complexType>
```

As an example, the results of reading a set of fields for a set of business objects are encoded in the following `ReadResultsType`:

```
<xsd:complexType name="ReadResultsType">
 <xsd:sequence>
  <xsd:element name="FieldValuePairList"
               type="FieldValuePairListType"
               maxOccurs="unbounded" />
 </xsd:sequence>
</xsd:complexType>
```

Listing 2 in the appendix shows the complete XML that is sent from YAWL as input to the codelet (for a work item of task „ReadSalesOrders“). It reads details, specified in `<FieldList>`, from the sales orders (`<Object>`) whose ID is given in `<IDList>`. The (abbreviated) response document that is passed back from the codelet is shown in Listing 3 in the appendix. It contains a `<FieldValuePair-List>` for each sales order. Within this list are the `<FieldValuePair>` elements that indicate the field name, value and field type.

5 Use and Examples

This section illustrates the use of the codelet for managing sales orders in Open-ERP. Figure 1 above shows the sales order workflow from the OpenERP perspective, Figure 2 below shows the YAWL workflow for creating and processing sales order.

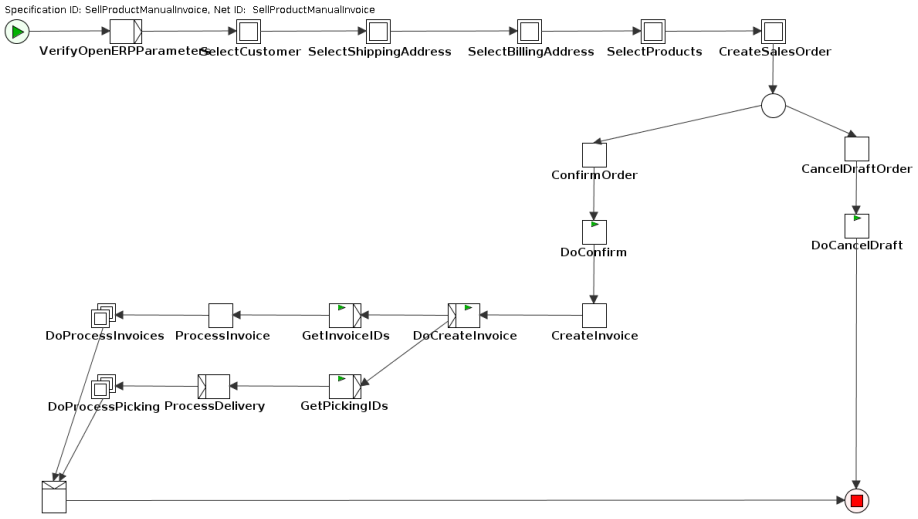

Fig. 2. Sales order process

The codelet design decisions have an impact on the usage of the codelet in two important ways. First, the direct representation of the basic OpenERP method calls (search, read, etc.) leads to typical combinations of search-read sequences as two automated tasks in the YAWL workflow. Second, the relatively low degree of abstraction of the codelet parameters suggests that the data transformations in the YAWL task input-/output-mappings are not trivial. For example, a data type for a sales order needs to be transformed to and from a `FieldValuePairList` when invoking the codelet. Thus, the use of the codelet requires considerable XQuery expertise.

The following is an example XQuery used to convert the results of a read operation for OpenERP "shops" to a data structure that represents a shop in YAWL and is used for presentation to the user. Note the selection of the second `Value` element in the field value pair list. This results from a one-to-many cardinality between shops and warehouses, pricelists, and companies in OpenERP. As explained above, in these cases, OpenERP returns an array of values, the first of which is the ID, which is not informative for the user, while the second is the name of the business object.

```
{ for $x in Create-
SalesOrder/Result/ReadResults/FieldValuePairList
  return
   <Shop>
    <ID>{$x/FieldValuePair[Field='id']/Value/text()}</ID>
    <Name>{$x/FieldValuePair[Field='name']/Value/text()}</Name>
    <PaymentDefaultID>
{$x/FieldValuePair[Field='payment_default_id']/Value[2]/text()}
    </PaymentDefaultID>
```

```
    <WarehouseID>
{$x/FieldValuePair[Field='warehouse_id']/Value[2]/text()}
    </WarehouseID>
    <PricelistID>
{$x/FieldValuePair[Field='pricelist_id']/Value[2]/text()}
    </PricelistID>
    <CompanyID>
{$x/FieldValuePair[Field='company_id']/Value[2]/text()}
    </CompanyID>
   </Shop> }
```

6 Conclusion

This paper presented a YAWL codelet to access the OpenERP system. The codelet exposes low-level access functionality, rather than business-level objects or methods. This low level of abstraction requires the codelet user to have a thorough understanding of the OpenERP data model, methods and workflows. At the same time, this design makes the codelet useful for the widest range of applications. Codelet users and workflow designers may also use YAWL features to build additional layers of abstraction on top of this foundation. For example, YAWL worklets could be defined that aggregate some of the codelet functions, e.g. the search-read combinations, into assemblies that are meaningful at the business level.

7 Bibliography

Instead of a formal bibliography, this section provides an annotated list of web addresses for relevant information on OpenERP.

http://doc.openerp.com/v6.0/developer/6_22_XML-RPC_web_services/index.html

This page provides information on the XML-RPCS architecture of the OpenERP web-services and includes sample code to access these services from various languages.

http://doc.openerp.com/v6.0/developer/6_21_web_services/index.html

This page provides information on the basic methods exposed by OpenERP business objects, and the parameters they require.

http://doc.openerp.com/v6.0/developer/2_5_Objects_Fields_Methods/index.html

This page provides an overview over the business object model in OpenERP, the different types of fields and the basic methods for all OpenERP business objects.

http://doc.openerp.com/v6.0/developer/3_9_Workflow_Business_Process/index.html

This page provides information on the workflow concept used by OpenERP, how workflows are defined in OpenERP and their connection with business object states and methods.

Information on specific business objects and workflows must be discerned from the OpenERP source code (written in Python). In an OpenERP installation, these are found in the `/server/openerp/addons` directory and subdirectories of those.

8 Appendix

Listing 1.

```
<xsd:complexType name="WriteParamsType">
 <xsd:sequence>
  <xsd:element name="IDList" type="IDListType"/>
  <xsd:element name="FieldValuePairList"
               type="FieldValuePairListType"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="FieldValuePairListType">
 <xsd:sequence>
  <xsd:element name="FieldValuePair" type="FieldValuePairType"
               minOccurs="1" maxOccurs="unbounded" />
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="FieldValuePairType">
 <xsd:sequence>
  <xsd:element name="Field" type="xsd:string" />
  <xsd:element name="Value" type="xsd:string"
               maxOccurs="unbounded"/>
  <xsd:element name="Type" type="typeEnumerationType"
               minOccurs="0"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="typeEnumerationType">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="string"/>
  <xsd:enumeration value="integer"/>
  <xsd:enumeration value="long"/>
  <xsd:enumeration value="boolean"/>
  <xsd:enumeration value="float"/>
  <xsd:enumeration value="double"/>
```

```
  <xsd:enumeration value="many2one"/>
  <xsd:enumeration value="one2many"/>
 </xsd:restriction>
</xsd:simpleType>
```

Listing 2

```
<ReadSalesOrders>
  <URL>ec2-175-139-127-88.compute-1.amazonaws.com</URL>
  <Port>8069</Port>
  <Database>BicycleShop</Database>
  <Username>admin</Username>
  <Password>password</Password>
  <Object>sale.order</Object>
  <Method>read</Method>
  <Parameters>
    <ReadParameters>
      <IDList>
        <ID>21</ID>
        <ID>20</ID>
        <ID>19</ID>
        <ID>27</ID>
        <ID>25</ID>
        <ID>24</ID>
      </IDList>
      <FieldList>
        <Field>id</Field>
        <Field>state</Field>
        <Field>name</Field>
        <Field>client_order_ref</Field>
        <Field>order_policy</Field>
        <Field>picking_policy</Field>
        <Field>shipped</Field>
        <Field>invoiced</Field>
        <Field>amount_total</Field>
        <Field>picking_ids</Field>
        <Field>invoice_ids</Field>
        <Field>order_line</Field>
      </FieldList>
    </ReadParameters>
  </Parameters>
</ReadSalesOrders>
```

Listing 3.

```
<codelet_output>
  <Result>
    <ReadResults>
      <FieldValuePairList>
        <FieldValuePair>
          <Field>id</Field>
          <Value>21</Value>
          <Type>integer</Type>
        </FieldValuePair>
        <FieldValuePair>
          <Field>order_policy</Field>
          <Value>manual</Value>
          <Type>string</Type>
        </FieldValuePair>
        <FieldValuePair>
          <Field>picking_policy</Field>
          <Value>direct</Value>
          <Type>string</Type>
        </FieldValuePair>
        <FieldValuePair>
          <Field>shipped</Field>
          <Value>true</Value>
          <Type>boolean</Type>
        </FieldValuePair>
        <FieldValuePair>
          <Field>amount_total</Field>
          <Value>565.0</Value>
          <Type>double</Type>
        </FieldValuePair>
        <FieldValuePair>
          <Field>name</Field>
          <Value>SO012</Value>
          <Type>string</Type>
        </FieldValuePair>
        <FieldValuePair>
          <Field>state</Field>
          <Value>progress</Value>
          <Type>string</Type>
        </FieldValuePair>
        <FieldValuePair>
          <Field>order_line</Field>
          <Value>13</Value>
          <Type>one2many</Type>
        </FieldValuePair>
        <FieldValuePair>
```

```
          <Field>invoice_ids</Field>
          <Value>2</Value>
          <Type>one2many</Type>
        </FieldValuePair>
        <FieldValuePair>
          <Field>client_order_ref</Field>
          <Value>false</Value>
          <Type>boolean</Type>
        </FieldValuePair>
        <FieldValuePair>
          <Field>picking_ids</Field>
          <Value>3</Value>
          <Type>one2many</Type>
        </FieldValuePair>
        <FieldValuePair>
          <Field>invoiced</Field>
          <Value>false</Value>
          <Type>boolean</Type>
        </FieldValuePair>
      </FieldValuePairList>
<!-- more FieldValuePairList elements go in here ... -->
    </ReadResults>
  </Result>
</codelet_output>
```