Brief guide for XML, XML Schema, XQuery for YAWL data perspective*

Carmen Bratosin

March 16, 2009

1 Data perspective in YAWL

YAWL engine files are XML based. Therefore, YAWL uses XML for data perspective also. For a quick guide about XML a good starting point is: `http://www.w3schools.com/xml/default.asp`.

Basic data types are already supported by YAWL:

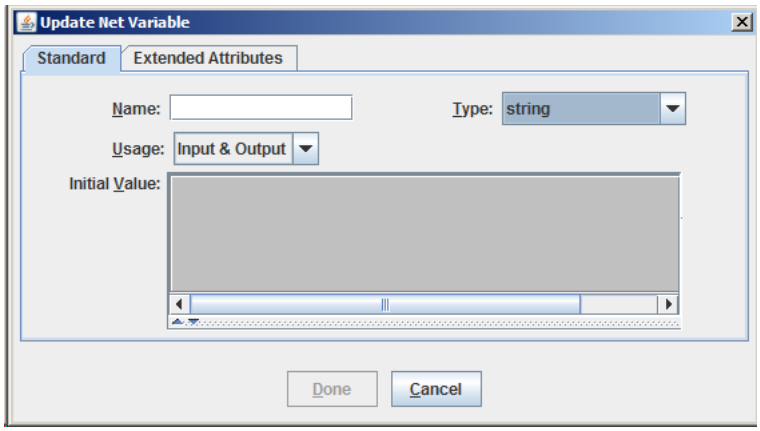

Figure 1: YAWL Data Types

2 Creating your own data types

In order to create new data types, a XML Schema (`http://www.w3schools.com/schema/default.asp`) has to be defined.

*This document is intended to be a complement of the YAWL Editor Manual.

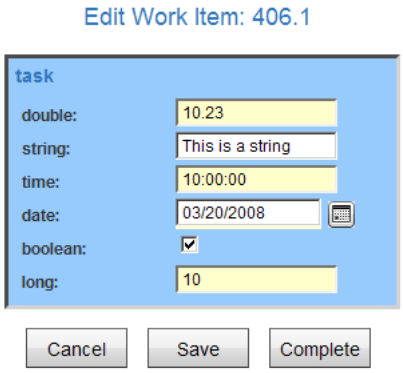

Figure 2: Examples of values for basic types

2.1 Complex type

A complex type is a sequence of different tags. For example, a *client* data type contains the tags: *ID, Name, Address and Phone*.

```
<xs:complexType name="client">
    <xs:sequence>
     <xs:element maxOccurs="1" minOccurs="1" name="ID" type="xs:long"/>
     <xs:element maxOccurs="1" minOccurs="1" name="Name" type="xs:string"/>
     <xs:element maxOccurs="1" minOccurs="1" name="Address" type="xs:string"/>
     <xs:element maxOccurs="1" minOccurs="1" name="Phone" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
```

The display of this data type by the YAWL engine is presented in Figure 3.

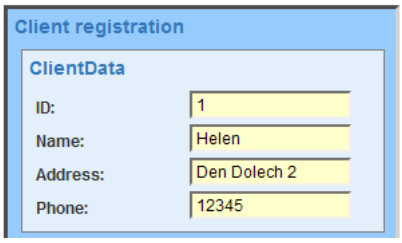

Figure 3: YAWL form for complex type "client"

New created data types can be used as part of new defined data types, as in the below example:

```
 <xs:complexType name="flight">
  <xs:sequence>
   <xs:element maxOccurs="1" minOccurs="1" name="Company" type="xs:string"/>
   <xs:element maxOccurs="1" minOccurs="1" name="StartHour" type="xs:time"/>
   <xs:element maxOccurs="1" minOccurs="1" name="EndHour" type="xs:time"/>
   <xs:element maxOccurs="1" minOccurs="1" name="Price" type="xs:decimal"/>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="travelPlan">
   <xs:sequence>
    <xs:element maxOccurs="1" minOccurs="1" name="Onward" type="flight"/>
    <xs:element maxOccurs="1" minOccurs="1" name="Return" type="flight"/>
   </xs:sequence>
 </xs:complexType>

<xs:complexType name="bill">
    <xs:sequence>
     <xs:element maxOccurs="1" minOccurs="1" name="Client" type="client"/>
     <xs:element maxOccurs="1" minOccurs="1" name="TravelPlan" type="travelPlan"/>
     <xs:element maxOccurs="1" minOccurs="1" name="TotalPrice" type="xs:decimal"/>
    </xs:sequence>
</xs:complexType>
```

2.2 Enumeration type

If we want to use a variable that restricts the number of possible values to a predefined set, the enumeration type is used. For example the following type restricts the value of data type *clientAnswer* to the values *Cancel, Refine and Accept*:

```
 <xs:simpleType name="clientAnswer">
  <xs:restriction base="xs:string">
  <xs:enumeration value="Cancel"/>
  <xs:enumeration value="Refine"/>
  <xs:enumeration value="Accept"/>
  </xs:restriction>
 </xs:simpleType>
```

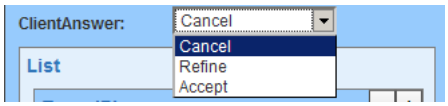

Figure 4: YAWL form for an enumeration type

2.3 List type

For defining a list type, we first have to define the element of the list:

```
<xs:complexType name="flight">
    <xs:sequence>
     <xs:element maxOccurs="1" minOccurs="1" name="Company" type="xs:string"/>
     <xs:element maxOccurs="1" minOccurs="1" name="StartHour" type="xs:time"/>
     <xs:element maxOccurs="1" minOccurs="1" name="EndHour" type="xs:time"/>
     <xs:element maxOccurs="1" minOccurs="1" name="Price" type="xs:decimal"/>
    </xs:sequence>
</xs:complexType>
```

Then we define the list as a sequence of list elements. The attributes *maxOccurs* and *minOccurs* defines the maximum (minimum) size of a list. For an unbounded list *maxOccurs="unbounded"*.

```
<xs:complexType name="flightList">
   <xs:sequence>
          <xs:element maxOccurs="5" minOccurs="1" name="Flight" type="flight"/>
   </xs:sequence>
</xs:complexType>
```

3 XQuery for updating variables and/or flow details

The structure of an XML document is like a directory structure. Therefore in order to obtain the value from a node, we have to give the full path. It is possible to query an entire tag or just the value in that tag.

A start point: `http://www.w3schools.com/xpath/default.asp` and `http://www.w3schools.com/xquery/default.asp`

3.1 Obtaining the value of a node from an XML document

If the variable to be updated is a simple variable, by using the button *add XQuery for element's content* we create a query as e.g. in Figure 6.

If the variable is a complex type, by performing the same action as above, a query is created as in e.g. Figure 7.

A third option is to create a complex type from other variables. In Figure 8, we construct the value of variable *Bill* such that:

- the tag *Client* has the value of net variable *ClientData*.
- the tag *TravelPlan* has the value of the first element from the list *TravelPlan*.
- the tag *Total Price* is equal to 0.

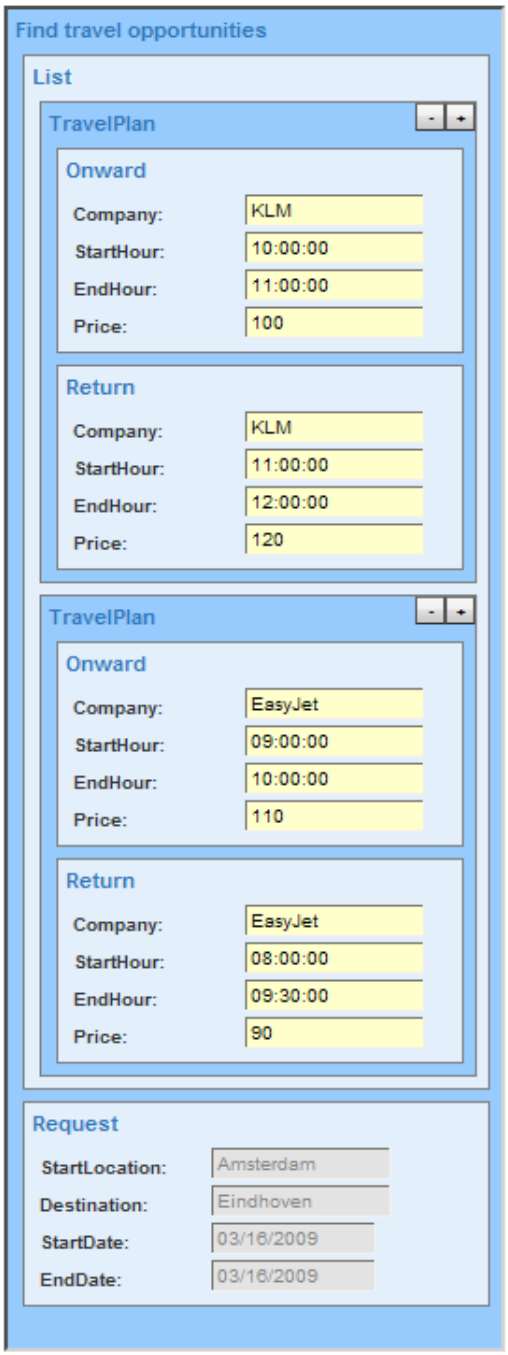

Figure 5: YAWL form for list type

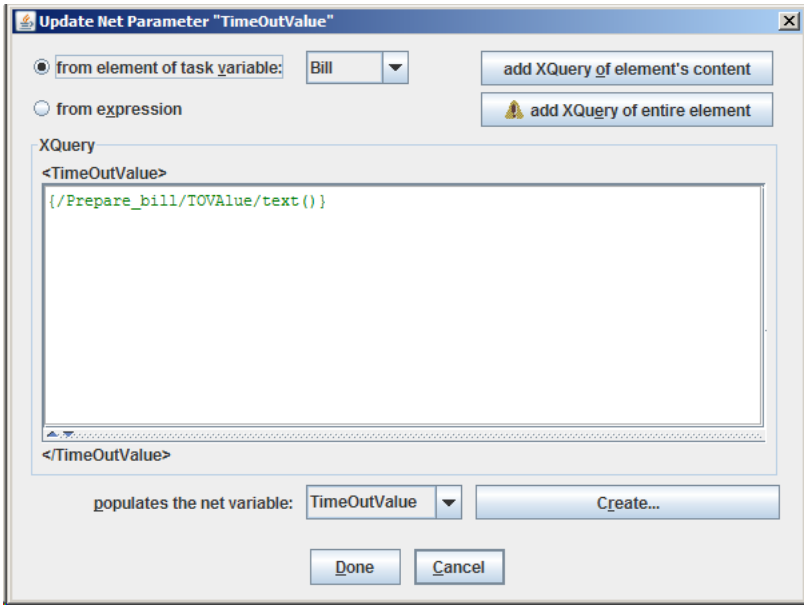

Figure 6: XQuery for a simple type

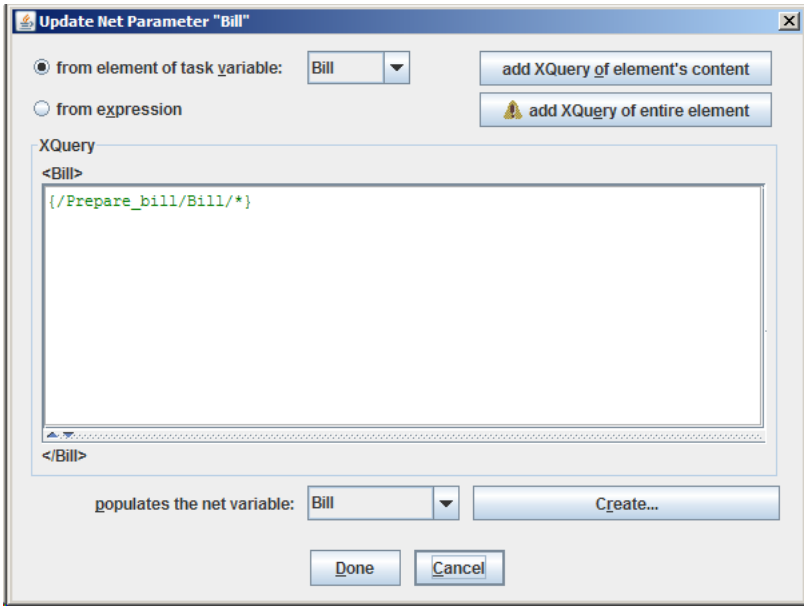

Figure 7: XQuery for an entire complex data type

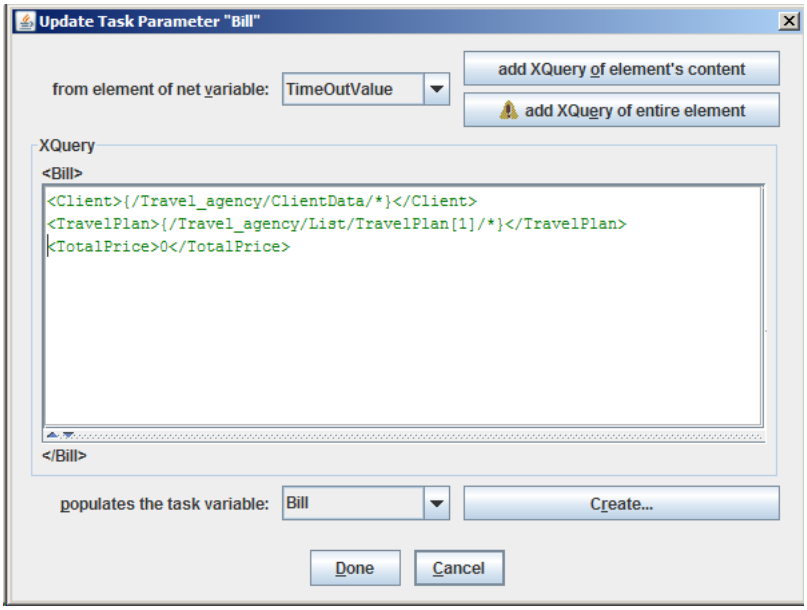

Figure 8: Creating the tags of a complex type from different variables

3.2 Flow details

For the XOR split tasks, we have to set up also the flow details (see Figure 9). Note that if that is not the case the engine will choose as the next step always same task (the default one). The choice can be made based on data variables.

3.2.1 Boolean values

For a boolean variable the query should be similar as in Figure 10.

3.2.2 Number values

In Figure 11, the choice is determine by the value of a number variable. As in other programming languages the following boolean operators can be used: $=$, $<$, $>$, $!=$, $<=$, $>=$.

Note: The YAWL Editor has a typo, instead of *number(* is *(*number). Therefore you have to manually replace the curly bracket with a parenthesis.

3.2.3 Enumeration values

If the choice is determine by an enumeration variable (see *ClientAnswer* example from section 2.2), the queries should be similar to the one in Figure 12 where the values should correspond to the set of allowed values.

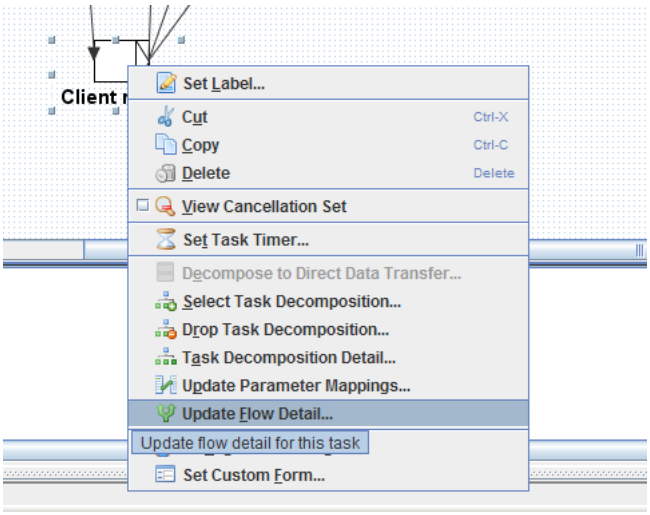

Figure 9: Flow detail option

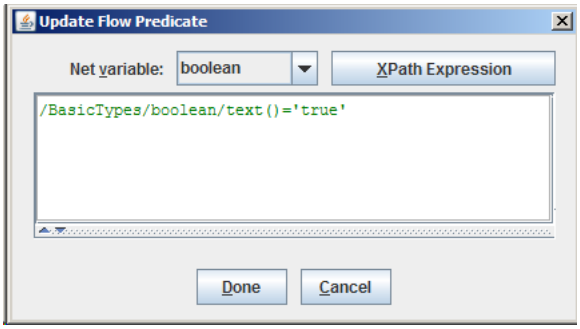

Figure 10: Query a boolean variable

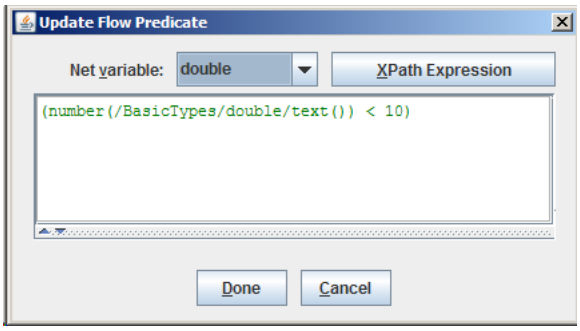

Figure 11: Query a double variable

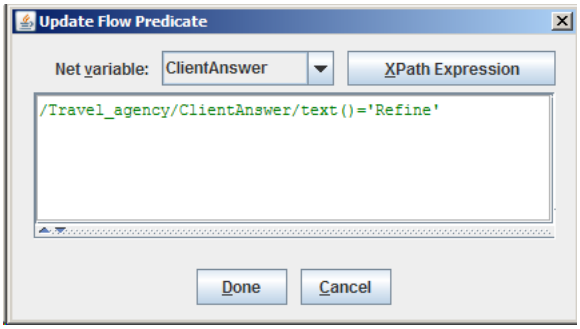

Figure 12: Query an enumeration variable

3.3 Complex queries

More complex expressions can be used in order to determine the next task enabled. For example, Figures 13 and 14 present a small workflow containing an OR split and the expressions used to enable the two tasks. Even if neither one of the expressions are evaluated as *true*, the engine will choose the most bottom one as enabled by default. Figures 15 and 16 presents two runs of the workflow. In the first run both conditions are satisfied, but in the second one only one of them.

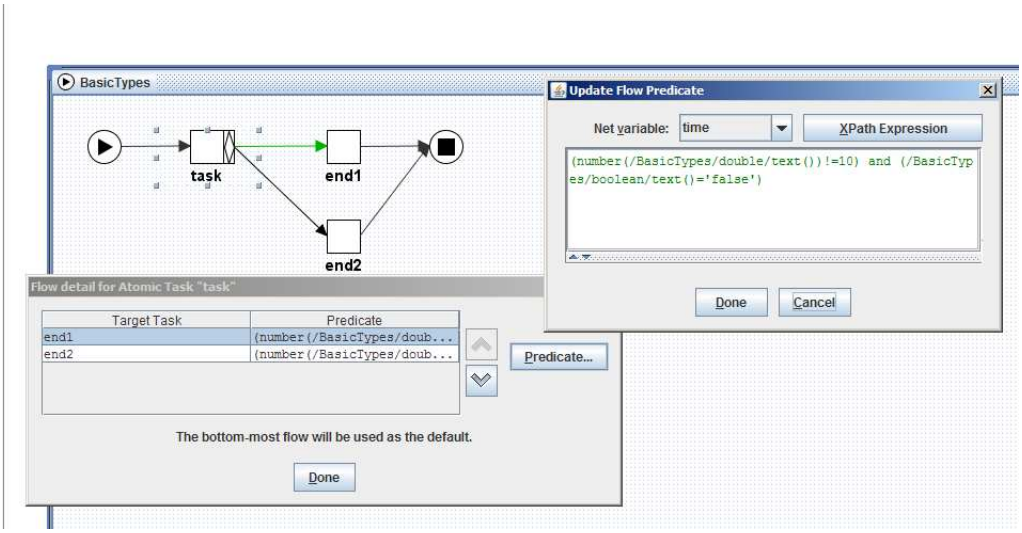

Figure 13: Complex queries: condition for enabling task *end1*

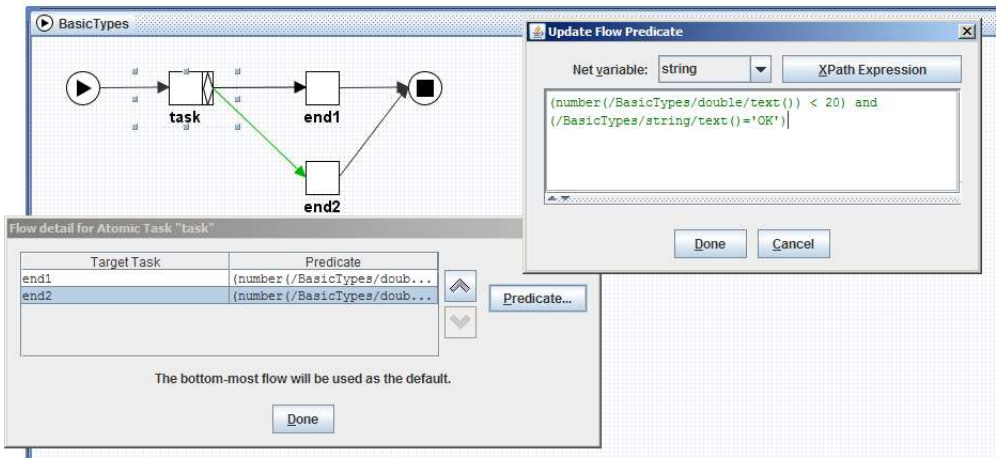

Figure 14: Complex queries: condition for enabling task *end2*

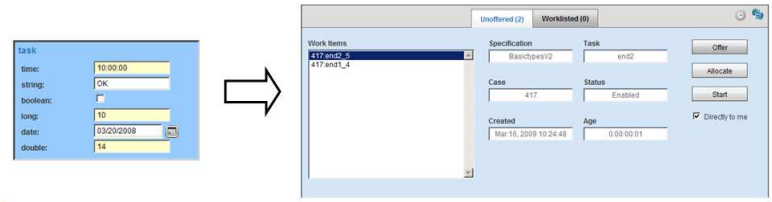

Figure 15: Scenario 1

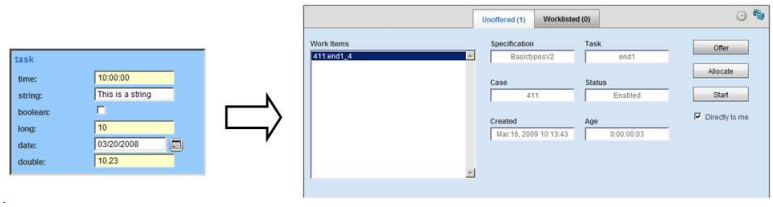

Figure 16: Scenario 2