Process Contexts

This feature is implemented in Apache ODE 2.0 only.

Abstract

To allow the circulation of transport metadata from messages to processes and then from processes to messages, we want to add support for different levels of contextual data to our process engine. The main use cases are:

- Conversation ids, such as those of the Stateful Exchange Protocol and WS-Context
- Tracing ids, mostly with ServiceMix.
- Credentials (such as tokens) for authentication and authorization
- Eventually some WS-RM related context.

Architecture

The aim of this proposal is to make transport metadata accessible within BPEL processes and allow modelers and deployers to define which metadata should be propagated between process and partners. Therefore, we model such metadata in terms of contexts. Contexts are modeled as a bag of metadata information represented as key-value pairs and are attached to partner links. A partner link may maintain multiple contexts; each of them is identifiable by a name attribute to distinguish context used for different purposes (e.g. security context vs. tracing context).

Whenever a message arrives through ODE’s integration layer, the message is passed through a defined set of context interceptors. These interceptors are in charge to extract relevant transport metadata from messages, prepare the data and store it in terms of key-value pairs into a named context associated to the partner link from which the message came from. The context data is persistently stored in the runtime database.

For outgoing messages, <invoke> activities can be configured to take context data from a (different) partner link, pass it again through a context interceptor which translates the data back to transport-specific metadata. Such propagation configuration can be defined per invoke activity in the process model or on a global level in the deployment descriptor.

Context data can be assigned to variables. Therefore, the context data is translated into XML data, following a predefined schema. It is possible to define which contexts of a partner link are copied to a variable. Similarly, context data can be copied from a variable back into the context of a certain partner link. Contexts can be used to classify or tag process instances and it is useful to make this information accessible for process management tools. Hence, this specification extends the process management API to make context data queryable and accessible.

Contexts

Contexts are modeled as a bag of metadata information represented as key-value pairs (string, string) and are attached to partner links. Contexts have a name attribute. Contexts are persisted in ODE’s runtime database. Long values are stored as a CLOB whereas shorter values (< 250 characters) are automatically stored in an indexed VARCHAR field.

Since contexts should also be accessible in BPEL, there is also an XML-based representation. It is defined as follows, the according XMLSchema can be found here

```
<?xml version="1.0" encoding="UTF-8"?>
  <ctx:context name="foo">
    <ctx:value key="bar">baz</ctx:value>
  </ctx:context>
  <ctx:context name="tracing">
    <ctx:value key="id">0815</ctx:value>
  </ctx:context>
</ctx:contexts>
```

Context Interceptors

At the transport level, message headers can be translated to context data and contexts can be translated to message headers. Context interceptors define how the headers are set to/from contexts and which contexts are affected. Interceptors are Java classes that implement the following interface. They can be registered either globally in ode’s property configuration file or per process in the deploy.xml.
public interface ContextInterceptor {

    /**
     * Configures the interceptor. This method will be called immediately after
     * instantiation of the implementing class. The passed element will contain the
     * configuration elements given in the deploy.xml. In case of a declaration in
     * ode-xxx.properties, the method won't be called.
     *
     * @param configuration
     */
    void configure(Element configuration);

    /**
     * Translates the data stored within the context object into SOAP headers or
     * vice versa.
     *
     * If direction is OUTBOUND, context data must be converted into message headers
     * if direction is INBOUND, context data must be extracted from the message.
     */
    void process(ContextData ctx, MessageExchangeDAO mexdao, IOContext.Direction dir) throws ContextException;
}

There is also a more high-level interface defined in terms of an abstract class:

public abstract class AbstractContextInterceptor implements ContextInterceptor {

    public abstract void configure(Element configuration);

    /**
     * @see org.apache.ode.bpel.context.ContextInterceptor#process(org.apache.ode.bpel.rapi.ContextData, org.
     *
     * public void process(ContextData ctx, MessageExchangeDAO mexdao, IOContext.Direction dir) throws
     * ContextException {
     *     // dispatch
     * }
     *
     */

    abstract void onPartnerInvoke(ContextData ctx, Message msg);
    abstract void onPartnerReply(ContextData ctx, Message msg);
    abstract void onProcessInvoke(ContextData ctx, Message msg);
    abstract void onProcessReply(ContextData ctx, Message msg);
}

**Working with Contexts in BPEL**

**Declaring Context Interceptors**

In order to use context interceptors, their implementation must be registered with ODE. There are two options to do this:

- Engine-wide registration - the context interceptor will be available for all processes that are deployed to this ODE instance.
- Per-process registration - the context interceptor will only be available for a particular process model.

Engine-wide context interceptors are registered in ODE's properties file (either `ode-axis2.properties` or `ode-jbi.properties`) using the properties key `context.interceptors`. 

```
ode-axis2.properties

... 
ode-axis2.context.interceptors = my.full.qualified.context.interceptor.class.name 
...

ode-jbi.properties

... 
ode-jbi.context.interceptors = my.full.qualified.context.interceptor.class.name 
...

Per-process context interceptors are declared in ODE's deployment descriptor deploy.xml:

deploy.xml

```xml
    xmlns:pns="http://ode/bpel/unit-test"
    xmlns:wns="http://ode/bpel/unit-test.wsdl">
    <process name="pns:HelloWorld2">
        <active>true</active>
        <provide partnerLink="helloPartnerLink">
            <service name="wns:HelloService" port="HelloPort"/>
        </provide>
        <context>
            <propagate from="helloPartnerLink" to="helloPartnerLink2" context="*"/>
            <interceptor>
                <class-name>org.apache.ode.bpel.context.TestInterceptor</class-name>
                <config>
                    <myparam1>x</myparam1>
                    <myparam2>y</myparam2>
                </config>
            </interceptor>
        </context>
    </process>
</deploy>
```

The context-specific configuration takes place in the `<context>` element. The `<interceptor>` element is used to declare and register context interceptor implementations. The `<class-name>` element takes the full qualified class name of the context interceptor implementation (which must implement the ContextInterceptor interface). The content of the `<config>` element will be passed to configure method of the interceptor, immediately after instantiation.

Static propagation rules can be configured directly in the deploy.xml. Using the `<propagate>` element, ODE can be configured to propagate all contexts named in the `context` attribute (space-separated list) from a certain partner link to another partner link. To propagate all contexts the `context` attribute should be set to `*`. These rules are automatically evaluated when an invoke activity is being executed. Before a BPEL process instance invokes an operation of a certain partner link, the propagation logic performs a look-up for propagation rules that have that partner link referenced by their `to` attribute. Subsequently the specified contexts are retrieved from the partner link specified by the `from` attribute of the rule, passed to the context interceptor that translates the context data into transport meta data, which is then added to the outgoing message.

Customize Context Propagation in the Process Model

In addition to static propagation rules, context propagation can be explicitly configured for outbound interaction activities (i.e. `<invoke>` and `<reply>`). Therefore, we introduce a new BPEL extension element `<ctx:propagate>` in the `http://ode.apache.org/contextPropagation` namespace (to which the `ctx`-prefix is bound to), following the syntax as described below:

```xml
<bpel:invoke ...>
    <ctx:propagate ctx:fromPartnerLink="testPartnerLink" ctx:context="*"/>
</bpel:invoke>
```
<b:reply ...>
  <ctx:propagate ctx:fromPartnerLink="testPartnerLink" ctx:context="*"/>
</b:reply>

- ctx:fromPartnerLink identifies the partner link the context data should be retrieved from.
- ctx:context identifies which contexts should be propagates. This is a space-separated list of context names. * serves as a wild card to copy all contexts.

The context data identified by this propagation rule will be passed to all registered context interceptors which enhance the outgoing message accordingly.

All partner link references are resolved by the interaction activity, i.e. only partner links can be referenced that are visible to the respective activity.

**Accessing and Assigning Context Data**

Context data should be accessible within BPEL processes and from the outside, e.g. for management purposes. For the former, the `<assign>` activity has been extended to support copying data from and to contexts of a partner link. The according `from/to specs` are defined as follows:

### "from/to specs for the work with context data"

```xml
<assign>
  <!-- copy 'tracing' and 'security' contexts to a variable. -->
  <copy>
    <from partnerLink="pl1" context="tracing security"/>
    <to>$var</to>
  </copy>

  <!-- copy context data from $var to pl1. Only 'tracing' and 'security' contexts will be written, other contexts remain unchanged. -->
  <copy>
    <from>$var</from>
    <to partnerLink="pl1" context="tracing security"/>
  </copy>

  <!-- copy 'tracing' and 'security' contexts from pl1 to pl2. Pre-existing contexts in pl2 will be purged (*). -->
  <copy>
    <from partnerLink="pl1" context="tracing security"/>
    <to partnerLink="pl2" context="*"></to>
  </copy>

  <!-- copy 'tracing' and 'security' contexts from pl1 to pl2. Pre-existing contexts in pl2 will be merged (+). -->
  <copy>
    <from partnerLink="pl1" context="tracing security"/>
    <to partnerLink="pl2" context="+"/>
  </copy>

  <!-- copy all contexts from pl1 to pl2. Pre-existing contexts in pl2 will be merged (+). -->
  <copy>
    <from partnerLink="pl1" context="*"/>
    <to partnerLink="pl2" context="+"/>
  </copy>
</assign>
```

- `context="*"` in `from-specs` selects all contexts.
- `context="*"` in `to-specs` replaces existing contexts with context data selected by the `from-spec`.
- `context="+"` in `to-specs` merges existing contexts with context data selected by the `from-spec`.

When assigning context data to variables, the variable should be declared to be an instance of the context XSD (see Contexts section). It is also possible to copy only fragments of a context data set by applying an XPath query to the `from-spec`.
In order to make context data accessible for external management tools, the ProcessManagement-API has been extended. TODO: to be completed.