Use of BPEL in LEAD

Introduction

Business Process Execution Language (BPEL) is industry standard, ratified by OASIS, that describes how Web Services can be orchestrated to provide a new service. This capability is used in LEAD to create workflows that orchestrate LEAD services to run users’ experiments.

There are multiple ways in which LEAD interacts with BPEL. We describe here how BPEL is currently used with Grid Process Execution Language (GPEL) Engine and what would it take to interchange it with different workflow engine (for example Active BPEL: http://www.active-endpoints.com/active-bpel-engine-overview.htm ). During implementation of GPEL and integration of it with LEAD we paid great attention to use standards (we are very close to BPEL 2.0) and to not introduce non-standard approaches if a standard approach would work.

Composition of BPEL workflows

In LEAD workflows are composed using XBaya Workflow Composer. The composer provides convenient GUI and a simple graph representation that allows users to create workflows by selecting lead services, drag-and-drop them on a canvas and connect them together by simply drawing connecting lines. XBaya then takes care to generate a BPEL work (use from menu File/Generate BPEL Script to see it). The generated workflow has not only generated the BPEL document but as well a WSDL document that describes Web Service interface of the generated workflow as every workflow is also a service and can be used in other workflows enabling hierarchical composition and workflows reuse.

Required changes to use other BPEL engine: verify and correct if necessary any inconsistencies with BPEL 2.0 – our generated BPEL is pre-2.0 and some small changes are necessary.

Creation of BPEL workflows

BPEL standard describes semantics of workflow execution but leaves it up to implementations to how actually deploy BPEL workflows in a workflow engine. In GPEL we use simple REST-based interface. Workflow deployment is set of POST operations that transmit documents (BPEL workflow and its associated WSDL files for services it uses and some additional metadata documents such as workflow priority) to GPEL Engine. Each document has unique id, in particular there is one id used to identify
a deployed workflow that we call workflow template id. When using GPEL we have a separate stage for actual workflow instance creation. This allows each instance to use a unique set of services (different WSDL files for each instance) and create per-instance execution parameters such as what resources should be used.

Most of BPEL engines do not support such two-stage process. Instead a workflow instance is automatically created when a new input message arrives and that instance uses whatever WSDL files and parameters were set when BPEL workflow was initially deployed. As a consequence it may be required that in LEAD we will deploy a new workflow for each user experiment. Some BPEL workflow engines support versioning of workflows but use it to run multiple different instances of slightly different workflows may be abuse of that capability (versioning is meant to help when transitioning between old and new workflow not to allow many multiple versions that are equal in importance). Moreover if versioning is use one would need to be extremely careful that in time between deployment of a new version of workflow and actual sending of input message (see below) no other version is created because then wrong version may be used (this is not a bug but feature – in most cases users want to run the latest version and not concurrently multiple versions of workflows).

**Required changes to use other BPEL engine:** as mentioned there is no standard of how BPEL workflows (BPEL script and associated files such as WSDL and other metadata) but most of BPEL engines package multiple documents (typically compressed ZIP file) and send it to BPEL engine for deployment using some proprietary API. This process would need to be encapsulated into a library that could be used by portal during experiment creation: portal would find out what type of BPEL engine is used (or what user selected) and use correct deployment procedure from the library. This process would need to be done for every experiment as every experiment may use different service instances (or QoS in future) which we encapsulate in WSDL files and associated metadata.


Additional change would be required that use a workflow engine API to retrieve an id for the workflow created so it can be tracked in events and used to retrieve workflow status by portal. It looks that in ActiveBPEL there is some functionality to add such extension but it requires further investigation (starting with looking how ActiveBPEL management console is accessing workflow engine to get list of deployed processes, their status etc. – this is popular topic on ActiveBPEL forums [http://forums.active-endpoints.com/](http://forums.active-endpoints.com/) )

**Workflow Instance Execution: Input**
When workflow is deployed the portal sends also initial parameters as a message that initiates a workflow execution. This message has format described in the workflow WSDL (that was generated by XBaya) and parameter values are obtained from user in experiment builder wizard process. This is a standard way to invoke a BPEL workflow and should be supported by all BPEL workflow engines.

However there is a set of additional, context, parameters that are not described as WSDL but are common for all LEAD experiments. Those parameters are sent as SOAP header and include such items like experiment id, where to send events, location of permanent services to use etc. This header is then added by GPEL engine to every input message sent to any other service used in a workflow. There is currently no standard way to designate a SOAP header that should be automatically forwarded to all services, a way to make a header ‘sticky’.

*Required changes to use other BPEL engine:* an extension of a workflow engine would need to be deployed that would extract the input header and then add it to all outgoing messages. As this is fairly low level manipulation of SOAP messages and SOAP headers are not available in BPEL workflows it is not possible to implement it by modifying to generated BPEL workflow unless workflow engine has non-standard capabilities available from inside workflow to access a whole SOAP message content including headers.

It seems there is some functionality to add such extension in ActiveBPEL or if not directly supported source code of ActiveBPEL can be directly modified (standard version is open source).

### Workflow Instance Execution: Events

During workflow execution GPEL engine generates event to notify that workflow was started etc.

*Required changes to use other BPEL engine:* this not a standard feature and again an extension of BPEL workflow would need to be added to send such events (the same way as in adding support for LEAD context header).

Alternatively generated BPEL workflow could be modified to have `<invoke>` to send events (before or after actual activity).

### Conclusion

Due to use of standard, BPEL, in LEAD it is possible to replace GPEL with a commercial BPEL workflow engine, or even have multiple BPEL workflow engine used in LEAD. The amount of work is not large and mostly deals with non-standard parts of using BPEL workflows such as deployment and runtime execution requirements that are specific to LEAD (events and passing context header).

Ability to choose which workflow engine to use for each experiment execution may be very useful capability for users.