BPEL Designer Runtime Extension Point Requirements and Use Cases

2006-05-05

Purpose

This document presents a set of requirements and use cases describing the functionality provided by the Runtime Extension Point in the Eclipse BPEL Designer. This extension point will allow third-party providers to integrate various BPEL engines into the editor.

Extension Point refers to the runtime extension point; extensions are those plug-ins consuming the extension point and integrating a BPEL engine.

High-Level Overview of Interaction Between User, Runtime Extension Point and Extension

1. User selects process(es) to be deployed.
2. REP displays first page of deployment wizard with list of available engines (installed extensions).
3. User selects particular engine.
4. REP displays second page of deployment wizard allowing overriding of global configuration settings for selected engine.
5. REP displays third (and final page) of deployment wizard allowing user to save local copy of deployment archive(s), if supported by extension.
6. REP (providing access to process model, bpel files, configuration paramters) hands responsibility over to extension.
7. Extension responsible for carrying out processing required to deploy selected process(es), such as generating deployment descriptor, storing required files in archive, etc.
8. Extension carries out (pre deployment) engine-specific validation and informs REP/user of any problems via problems view.
9. Extension deploys deployment unit on engine using its transfer mechanism and informs REP/user of outcome.

Requirements

1. **Access to Process Representation & Configuration Parameters**
    a. Extension can obtain .bpel file of the process to be deployed.
    b. Extension can obtain instance of editor's model representing process.

c. Extension can obtain information about WSDL files involved in defining the process (e.g., namespaces and locations).
d. Extension can obtain relevant configuration parameters (e.g., deployment location, transfer mechanism, etc.).
e. Extension can save copy of deployment archive in project workspace.

2. **Validation**
   a. Extension point checks validation status of process selected for deployment prior to deployment and issues warning in case there are outstanding issues.
   b. Extension can add engine-specific validation errors and warnings to the problems view.

Comment [BW1]: Should it prevent deployment from proceeding in case it finds errors? Won't deploy successfully anyway.

3. **Deployment Wizard**
   a. Displays list of installed engines (deployed extensions) and their versions.
   b. Extension can add itself to list of installed engines (deployed extensions).
   c. Provides configuration page allowing user to manipulate engine-specific configuration parameters.
   d. Extension can provide such a configuration page in wizard to allow user to override global settings for selected engine.
   e. Allows user to save a local copy of deployment archive, if extension indicates file extension for deployment archives.

4. **Configuration**
   a. Extension can add preference page to editor's pages in order to allow setting of global engine-specific parameters (deployment location, transfer mechanism, host URL).

5. **Real-Time, In-Process Monitoring?**

Investigate WST Server Framework

High-Level Use Cases

Actors

- Runtime Extension Point (REP) – The extension point in question.
- Editor – BPEL Designer as a homogenous component.
- Extension – Any third-party runtime extension.
- Process Modeller (PM) – User modelling BPEL process in BPEL Designer.

Use Cases

Pre-Deployment Checks

Prerequisites:

- PM has editor with workflow open and selected Deploy menu item *OR*
- PM right-clicked on .bpel file in package explorer and selected Deploy from the context menu.

Successful Case:

1. REP checks if selected resource/active editor is dirty and finds it to be clean.
2. REP checks validation status of selected resource/active editor and finds no problems (yeah, right).

Post-conditions:

- Upon successful pre-deployment checks, deployment can continue.

Alternative 1:

1a REP finds selected resource/active editor to be dirty.

1a1 REP informs PM about this in dialogue asking to confirm save operation or cancel deployment.

1a2 PM confirms save operation.

1a3 REP saves resource (which triggers validation by BPEL Designer).

Post-conditions:

- Pre-deployment checks continue as in successful case (with validation status check).

Alternative 2:

1a2b PM chooses to cancel deployment (doesn't remember having changed process).

Post-conditions:

Post-conditions:

- Pre-deployment checks aborted and PM returned to BPEL Designer.

Alternative 3:

2a REP finds errors in validation status of selected resource/active editor.

2a1 REP informs PM that recommended to fix problems before deployment and gives option to proceed regardless or to abort.

2a2 PM chooses to abort deployment.

Post-conditions:

- Pre-deployment checks aborted and PM returned to BPEL Designer.

Alternative 4:

2a2b PM chooses to continue regardless

Post-conditions:

- Who knows…

[Note: Does it really make sense to provide the option to continue in light of validation errors (might make sense for warnings though)? Deployment is likely to fail anyway, the engine might reject deployment or an extension runs into trouble trying to get problematic process into deployable format.**]**

Parsing Extensions

Prerequisites:

- PM has editor with workflow open and selected Deploy menu item *OR*
- PM right-clicked on .bpel file in package explorer and selected Deploy from the context menu.
- RPE has successfully completed Pre-Deployment Checks

Successful Case:

1. REP parses manifests of its extensions.
2. REP stores list of relevant information, such as names of engines and their versions.

Post-Conditions:

- RPE has a list of extensions to be used for populating first page of Deployment Wizard (with list of available engines).

Alternatives:

1a REP does not find any relevant extensions

Post-Conditions:

- RPE has an empty list for populating the first page of Deployment Wizard (list will display no engine for selection).

Deployment Wizard

Prerequisites:

- PM has editor with workflow open and selected Deploy menu item *OR*

- PM right-clicked on .bpel file in package explorer and selected Deploy from the context menu.
- RPE has successfully completed Pre-Deployment Checks

Successful Case:

1. REP displays first page of wizard showing list of engines available for selection.
2. PM selects engine and selects Next.
3. REP obtains engine-specific configuration page (2nd page) from relevant extension.
4. REP populates configuration page with global (or project-wide) settings from preference store.
5. PM selects Next.
6. REP obtains file extension for deployment archive/file from relevant extension.
7. By default, RPE populates third page with path to current project + process name/project name + file extension for archive.
8. PM chooses to accept settings and clicks Finish.

Post-conditions:

- Extensions now do their magic of getting the process(es) deployed.

Alternative 1:

1a List of available engines empty.

1a1 PM clicks cancel.

Post-conditions:

- Deployment is aborted and PM returned to BPEL Designer.

Alternative 2:

3a Extension does not provide engine-specific configuration page.

3a1 REP obtains archive file extension from extension.

Post-conditions:

- Continues as in successful case.

Alternative 3:

5a PM modifies settings.

5a1 PM clicks Next.

Post-conditions:

- Continues as in successful case.

Alternative 4:

6a Extension does not provide archive file extension.

6a1 REP displays third page of wizard w/o populating with default.

Post-conditions:

- Continues as in successful case.

Alternative 5:

8a PM modifies location.

8a1 PM clicks Finish.

Post-conditions:

- Continues as in successful case.

[Note: This whole configuration page business looks messy. Should enable extension to run the whole configuration business. Investigate WST Server framework for some potential insights into how this might be done.**]**

Deploy Project with Multiple BPEL Processes

The only differences to the case where just one process has been selected for deployment:

- PM right-clicks project and selects Deploy Process from context menu OR
- PM selects several .bpel files (across projects?) OR
- PM selects project and selects Deploy Process from menu.
- REP generates list of .bpel files to be deployed.
- REP uses Pre-Deployment Checks on all selected processes.

Update Problems View

Prerequisites:

- PM has successfully completed Deployment Wizard and responsibility passed to relevant extension.
- Extension has carried out engine-specific validation and wishes to inform PM of problems.

Successful Case:

1. Extension notifies REP of problems during engine-specific validation and passes list of errors/warnings to be displayed in Problems View.
2. REP notifies PM of problems during deployment.
3. REP displays list of issues in Problems View.

Post-conditions:

- PM bangs her head on the table.

[Note: Is it really necessary to let REP take care of that? Instead, an extension should take care of this itself. Or maybe not?**]**

Data Request by Extension

A list of data items that an extension can obtain from the REP:

- .bpel files selected for deployment.
- Instance (read-only?) of editor's process model.
- Configuration settings PM has committed to in Deployment Wizard (i.e. deployment location, transfer mechanism, location for storing local copy of deployment archive).
- The list of bpws:imports per process to be deployed.

Comment [BW2]: Alternatively, extensions could be left to get model instance themselves.

Comment [BW3]: Again, does the REP need to provide this or can we just leave an extension figure this out, if needed? Nice to have utility method, but not necessary.

Extension Responsibilities

Expectations of extensions consuming this REP. Shown what extension can expect from REP; now clarify what functionality extension needs to implement.

- Provide a configuration page for use in the Deployment Wizard and the preference pages of BPEL Designer. This configuration page should ask the PM to supply any information needed by the extension to deploy a process.
- Provide the file extension of used by the extension to store deployment archives.
- Extension should carry out engine-specific validation (in order to avoid PM having to check some log or web-based portal for details about deployment outcome) and notify REP of any errors and warnings encountered. Similarly, an extension should notify REP of successful deployment.
- Extensions are responsible for processing the data provided into a format suitable for deployment on their respective runtime environment and for carrying out the actual deployment.