Template Selector 1.0 Component Specification

1. Design

This component provides a default implementation of "template selection algorithm" as outlined by the Requirements Specification document.

The single method of TemplateSelectionAlgorithm interface provided with this component accepts the references to ComponentVersion and TemplateHierarchy which are developed as the separate projects. The purpose of this component is to locate the templates corresponding to specified component version using the specified template hierarchy as a start point.

This component implements the Strategy design pattern which allows the clients to switch to implementations of template selection algorithm other than default provided with this component without affecting the code using the template selection algorithm to locate the desired templates.

1.1 Design Patterns

None.

1.2 Industry Standards

None.

1.3 Required Algorithms

TemplateSelector.selectTemplates(ComponentVersion, TemplateHierarchy):

1. Locate the TemplateHierarchy matching the specified component version calling the findMatchingTemplateHierarchy method supplying the provided arguments.
2. If such a hierarchy is found (not null) then return the templates provided by that hierarchy.
3. Otherwise return an empty array of templates.

TemplateSelector.findMatchingTemplateHierarchy(ComponentVersion, TemplateHierarchy):

1. Create an empty Set which will hold the template hierarchies already visited by this method. This set is used to prevent the method from falling into eternal loop if the specified template hierarchy contains the loops.
2. Declare a single variable of TemplateHierarchy type and initialize it with reference to the provided TemplateHierarchy. This variable must be interpreted as "current" hierarchy which is being analyzed.
3. Start the loop which must exit when "current" hierarchy becomes null.
4. Within the loop do the following:
    1. Get child nodes provided by the specified "current" hierarchy instance.
    2. Get the attribute names provided by the specified ComponentVersion instance.
    3. Iterate sequentially over the child nodes and find a first one whose name matches a name or value of any of attributes (ignore the nodes which are found in set of visited hierarchies).

4. If no matching child node has been found at previous step then obtain the list of technology types provided by the specified `ComponentVersion` instance and repeat the iteration as specified by previous step but comparing the child node names against the names of technology types (ignore the nodes which are found in set of visited hierarchies).
5. If no matching child node has been found after execution of previous two steps then return `"current"` hierarchy.
6. Otherwise add `"current"` hierarchy to set of visited hierarchies. Initialize the `"current"` hierarchy with reference to a child hierarchy found at any steps 3 or 4.

5. When the loop exits return the `"current"` hierarchy.

1.4 Component Class Overview

TemplateSelectionAlgorithm

This is an interface specifying the contract the template selection algorithm. The main purpose of this interface is to provide the clients of Template Selector component with a freedom to switch to different template selection algorithms without affecting the code depending on the algorithm. So whenever a new implementation of template selection algorithm is put to production only a logic instantiating the algorithm implementation may need to be updated. All other code which makes use of template selection algorithm remains unaffected.

This interface defines a single method which is used to locate all the templates corresponding to specified component version using the supplied template hierarchy as a start point.

TemplateSelector

This class is a default implementation of `TemplateSelectionAlgorithm` interface which implements a template selection algorithm as specified by the `Requirements Specification` document.

This class also serves as an extension point for possible future implementations of `TemplateSelectionAlgorithm` interface. It provides a protected `findMatchingTemplateHierarchy(ComponentVersion, TemplateHierarchy)` method which could be overridden by the subclasses in accordance with a specific logic used to locate the template hierarchy matching the specified component version. So this class may be interpreted as implementation of `Template Method` design pattern. The difference is that the template method is not abstract and this class provides a default implementation of template method.

1.5 Component Exception Definitions

This component does not declare any custom exceptions. The following standard exceptions are declared to be thrown by the classes/interfaces declared by this component:

java.lang.NullPointerException

This exception is thrown from the methods of `TemplateSelector` class if any of specified `ComponentVersion` or `TemplateHierarchy` arguments is `NULL`.

1.6 Thread Safety

The single `TemplateSelector` class defined by this component is thread safe. It does not maintain any private state.

2. Environment Requirements

2.1 Environment

No special environment settings are required to use this component.

2.2 TopCoder Software Components

- None.

2.3 Third Party Components

- None

3. Installation and Configuration

3.1 Package Name

`com.topcoder.buildutility`

3.2 Configuration Parameters

None.

Parameter	Description	Values

3.3 Dependencies Configuration

None.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute ‘ant test’ within the directory that the distribution was extracted to.

4.2 Required steps to use the component

In order to use this component the instance of class implementing `TemplateSelectionAlgorithm` interface must be obtained (for example, `TemplateSelector`).

The `ComponentVersion` instance providing the details for the requested component version and a `TemplateHierarchy` representing the `“root”` node of template hierarchy must be obtained (such details are out of scope of this document).

The `TemplateSelectionAlgorithm.selectTemplates` method must be called to obtain the templates matching the specified component version and provided by the specified template hierarchy.

4.3 Demo

The code below demonstrates the steps described above. The code makes use of `Template Loader 1.0` component to load the requested top-level template hierarchy.

```
// Consider the following hierarchy of templates

                        Start
                         / \
                      Java .NET
                       /  \
             WebServer    Weblogic
                / \
          Debug   WebServer


// Obtain necessary data

// Build the component version

ComponentVersion componentVersion = new ComponentVersion(1);
componentVersion.setAttribute(“WebServer”, “Debug”);
componentVersion.setTechnologyType(“Java”);

// Load the top-level template hierarchy

TemplateLoader loader = new TemplateLoader(“configuration namespace”);
TemplateHierarchy templateHierarchy
      = loader.loadTemplateHierarchy(“Start”);


// Obtain an implementation of template selection algorithm

TemplateSelectionAlgorithm templateSelector = new TemplateSelector();


// Get the templates matching the component version

Template[] templates
    = templateSelector.selectTemplates(componentVersion,
                                       templateHierarchy);
```

```
// Check if any matching template has been found

if (templates.length == 0) {
      // Notify user
} else {
      // Do something useful with found templates
      //
      // The ‘templates’ array will contain a templates provided by the
      // template hierarchy referenced by following path :
      // “Java” – “WebServer” – “Debug”, where “Java” matches the
      // technology type, “WebServer” macthes the attribute name and
      // “Debug” matches the attribute value.
}

// Update the “WebServer” attribute and select the templates again
componentVersion.setAttribute(“WebServer”, “Production”);
templates = templateSelector.selectTemplates(componentVersion,
                                             templateHierarchy);


// Check if any matching template has been found
if (templates.length == 0) {
      // Notify user
} else {
      // Do something useful with found templates
      //
      // Now the ‘templates’ array will contain a templates provided by
      // the template hierarchy referenced by following path :
      // “Java” – “WebServer” – “WebServer”, where “Java” matches the
      // technology type, “WebServer” macthes the attribute name and
      // “WebServer” matches the attribute name too.
}
```

5. Future Enhancements

None.