1. Requirements Specification

2. Scope

2.1 Overview

Provided the name of a top-level node, the Template Loader then creates, caches, and returns an object representation of the template hierarchy.

2.2 Logic Requirements

2.2.1 Read the template hierarchy out of the TopCoder component database. The database schema in Figure 1.0 needs to be created (does not currently exist) for this component. Each template node (*temp_hier*) can have zero, one, or many template associations. The *temp_hier_maping* table represents these associations. The parent level template node's must have unique names (all other level names uniqueness does not matter). There can be multiple top-level nodes within the same schema. Additionally, the parent identification (*parent_temp_hier_id*) and identification (*temp_hier_id*) are the same for top-level nodes. For all other nodes the parent identification points to that node's parent node (they are not the same). Figure 1.0 outlines the template hierarchy database schema used by this component.

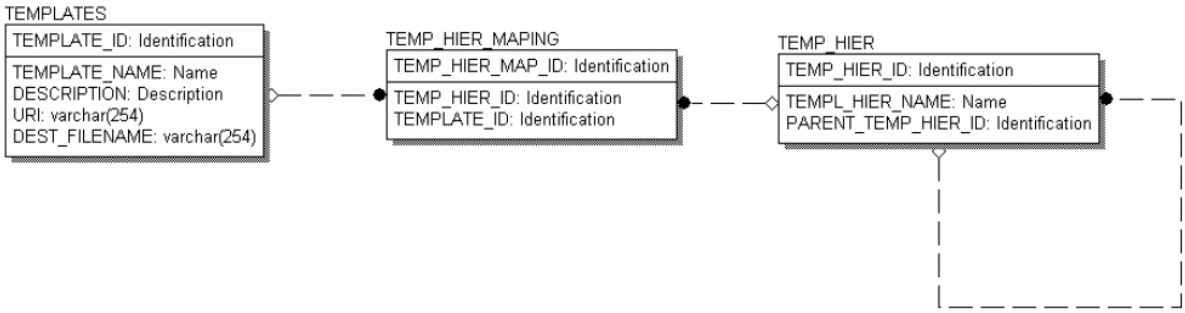

Figure 1.0 Template schema

2.2.2 Return the template hierarchy object outlined below in Figure 2.0. The is component is responsible for the design and development of the template hierarchy classes. Part of these classes public methods have been outlined to support external dependencies. These public methods must be implemented in the design and development of these classes. Each node in the template hierarchy can have zero or more template associations.

Dave Messinger, does TopCoder have a file server that we use to place files on accessed by the application or do we store the files directly as blobs within the database? If file server then can I get the uri? (make file server uri configurable, the uri within the database should be relative to a point that we define in the configuration settings)

```
package com.topcoder.buildutility.template;

/**
 * <p>This class diagram for the template hierarchy class is by no means fully completed.
It is merely a stubbed
 * representation of the functionality such that other dependent components can know what
to expect.
 * The underlying implementation is left to the designer responsible for this class.</p>
 */
```

```
public class TemplateHierarchy {

    /**
     * <p>Returns the template hierarchy node's name.  The name of the node in the
    template hierarchy
     * is used as part of the template selection process.  The name corresponds to either
    technology types
     * or component attributes.</p>
     *
     *
     * @return The name of the node in the template hierarchy is used as part of the
    template selection process.  The name corresponds to either technology types or
    component attributes.
     */
        public String getName() {…}

    /**
     * <p>Each node in the template hierarchy can have zero to many template associations.
    An empty array must be returned, a null return value is not allowed..</p>
     *
     * @return Template[] Each node in the template hierarchy can have zero to many
    template associations. An empty array must be returned, a null return value is not
    allowed.
     */
        public com.topcoder.buildutility.template.Template[] getTemplates() {…}

    /**
     * <p>Returns all of the template hierarchy nodes directly below the current node in
    the hierarchy.</p>
     *
     * @return Returns all of the template hierarchy nodes directly below the current node
    in the hierarchy. An empty array must be returned if there are no children, a null
    value must not be returned.
     */
        public com.topcoder.buildutility.template.TemplateHierarchy[] getChildren() {…}
 }



/**
 * <p>This class diagram for the template class is by no means fully completed.  It is
merely a stubbed
 * representation of the functionality such that other dependent components can know what
to expect.
 * The underlying implementation is left to the designer responsible for this class.</p>
 */
public final class Template {

    /**
     * <p>Returns the name of the template. Each template name must uniquely identify that
    template (i.e. no two templates can have the same name).</p>
     *
     * @return Returns the name of the template.
```

```
     */
        public String getName() {…}

    /**
     * <p>Returns the description of this template. Descriptions are optionaly but they
    will help a great deal when developers are trying to determine the purpose of the
    template.</p>
     *
     * @return String Description of the temlpate
     */
        public String getDescription() {…}

    /**
     * <p>Returns the intended or suggested file name corresponding to the template.  This
    file name is more than likely not the
     * name used to store the content on the server side.  This file name was designed to
    be the suggested name when
     * the transformation occurs from the template to the actual file and the file is
    written in destination location.
     * The file name is required and cannot be null.</p>
     *
     * @return Suggested file name, cannot be null.
     */
        public String getFileName() {…}

    /**
     * <p>Returns the tempalte as a data stream. A template must contain data.</p>
     *
     * @return Returns the tempalte as a data stream.
     */
        public java.io.InputStream getData() {…}
}
```

Figure 2.0 Template class hierarchy

2.2.3 A configuration setting drives the ability of this component to cache the returned hierarchies as they are created by the top-level node's name. The top-level node's name must be unique from all other top-level nodes. The configuration turns on or off the caching option. When off, the data representation of the template hierarchies are not cached for future use. When on, the data representation of the template hierarchies are cached for future use; thus, increasing performance by avoiding subsequent database calls to build the data objects. The caching occurs on demand as the objects are requested. The first time a user requests a template hierarchy (assuming caching Is turned on) the object is created and then added to the cache. Instead of loading all of the objects into the cache when the application starts.

2.3 Required Algorithms

None

2.4 Example of the Software Usage

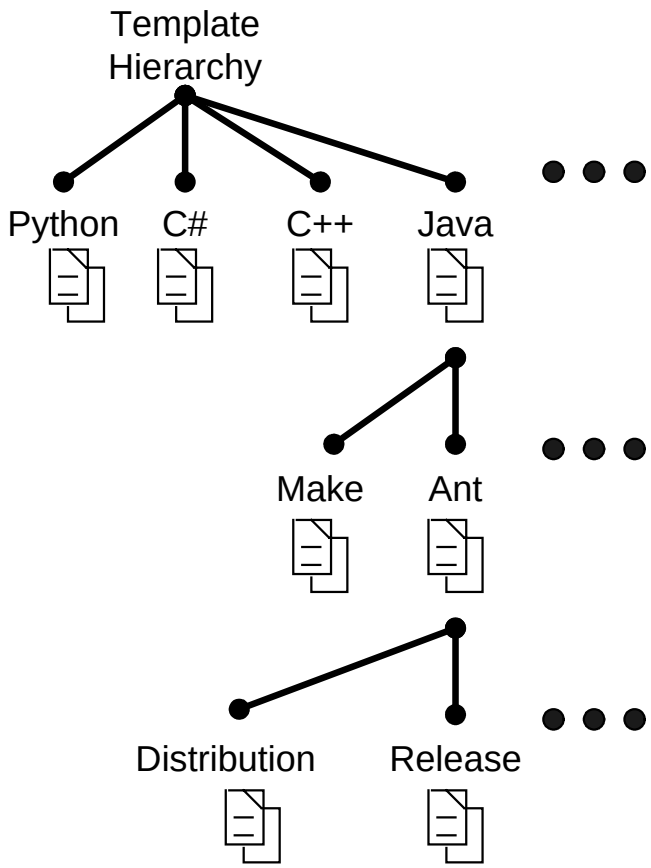

Figure 3.0 Template hierarchy sample

A template hierarchy must be populated by the end-user of the build script generation tool as part of the deployment and setup process. This hierarchy allows the tool to automatically choose a template based on the technology types and attributes associated with each component-version. Figure 3.0 outlines an example of this hierarchy.

2.5 Future Component Direction

This custom component is being developed for use with a build script generation application. The application uses the template hierarchy to select one or more templates.

3. Interface Requirements

3.1.1 Graphical User Interface Requirements

None

3.1.2 External Interfaces

None

3.1.3 Environment Requirements

- Development language: Java 1.4
- Compile target: Java 1.3, Java 1.4
- Database: Informix 9.21

3.1.4 Package Structure

com.topcoder.buildscript.template

4. Software Requirements

4.1 Administration Requirements

4.1.1 *What elements of the application need to be configurable?*

4.1.1.1 Database connection information.

4.1.1.2 Caching on/off (see 2.2.3)

4.2 Technical Constraints

4.2.1 Are there particular frameworks or standards that are required?

None

4.2.2 TopCoder Software Component Dependencies:

XML 2 SQL
BaseException
Simple Cache
DB Connection Factory
Logging Wrapper

**Please review the TopCoder Software component catalog for existing components that can be used in the design.

4.2.3 Third Party Component, Library, or Product Dependencies:

None

4.2.4 QA Environment:

- Windows 2000
- Windows 2003
- RedHat Linux 7.1
- Solaris 7

4.3 Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines. Modifications to these guidelines for this component should be detailed below.

4.4 Required Documentation

4.4.1 Design Documentation

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification
- XML Schema

4.4.2 Help / User Documentation

- Design documents must clearly define intended component usage in the 'Documentation' tab of Poseidon