This page last changed on Jun 17, 2010 by volodymyrk.

Project Phase Template 1.2 Requirements Specification

Scope

Overview

A project is usually executed in a predefined set of phases for a particular customer. Requiring the user to manually define the phase hierarchy is laborious and unnecessary. The component provides a template mechanism to handle this scenario. Template storage is pluggable and can be added without code changes. An XML storage is provided with this release.
Version 1.1 adds the following:

- database persistence
- a category classification for templates
- extra fields to templates
- a new way of applying the templates.
- a migration utility to translate XML templates to DB templates.

This new version must be backwards compatible, i.e. if this version is used instead of version 1.0 in an existing application, it must read the old XML template file and the application code shouldn't need to be changed in order to continue working.
Version 1.2 adds the following:

- flexible selection of phases (some can be stated to be ignored)
- upgrade of documentation to current TC standards, including porting the documentation to TCUML

Logic Requirements

Phase Template

Phase template is a set of predefined project phases and their dependencies. The phases will not have start time defined.

Template Storage

A set of project phases are stored as a template and will be assigned a template name. The template storage needs to be pluggable. The API should have a way to return all the configured template names. For this release an XML based persistence implementation should be provided. XML schema should be designed to store a single template.

Phase Generation

Given a template name and an optional project start date, the project phases can be generated based on the template.

Phase Generation setting the date for an intermediate phase

This way of generating phases enables the user to additionally specify when a phase (other than the first) starts. It is used, for example, to specify both registration and review start dates, and let the submission length be automatically adjusted.

The following fields are required for generating the phases:

- Template name: the template used to generate the phases.
- Project start date: (optional) indicates when the first phase of the project starts. If omitted, the start date generator will be used to calculate that date.
- Variable length phase id: it's the id of the phase whose length must be adjusted to meet required start dates. In the above example, it would be the id for the submission phase.
- Fixed phase id: it's the id of the phase (other than the first one) whose date will be specified. In the above example, it would be the id for the review phase.
- Fixed phase start date: it's the start date for the fixed phase.

The variable phase length must be previous to the fixed time phase, and the start date for the project must be previous to the start date of the fixed phase.

Start Date Strategy

If project start date is not provided, it will be generated from a pluggable strategy.
The default implementation should generate a relative time in a week. It could be configured to return 9:00 am next Thursday.

Database Template Storage

Database persistence must be implemented. The designer must define the table structure (including primary keys, foreign keys, indexes, constraints) for this purpose, providing a schema in FabForce DBDesigner as well as the DDL for creating them.

Project Categories

In each template, a project category can be specified. The project category is an integer value, representing categories like design, development, architecture, specification and so on, but this component doesn't care about the actual meaning of this value.

Get the template for a category

This must retrieve all the templates belonging to the specified project category. Templates that didn't specify a project category won't be retrieved using this operation.

Get default template

A category can define a default template. This operation must retrieve the default template for the specified category. If the category didn't define a default template, either return null or throw an exception (designer choice).

Template Additional data

Templates can store a description and a creation date. Those fields are optional.

Migration utility

Some XML templates will be migrated to DB. In order to do that, it must be provided a standalone utility that reads an XML file and generate SQL insert commands.
Those commands will be sent to standard output so that the user can copy-paste and run them.

Flexible phase selection

The current version of this component takes a template name and a requested start date and generates a Project shell with the phases as defined in that template. The upgrade will provide the facility to select which phases to ignore.
For example, the XML template would contain the following elements that define a phase in it:
<PhaseType id="registration" typeId="1" typeName="Registration"/>

<Phase id="registrationPhase" length="172800000" type="registration"/>
The first element defines a phase type "registration", and the second element defines a phase "registrationPhase". If we wanted to have the generation ignore this phase, we would pass "registrationPhase" to it.

If a phase is dependent on another phase, and that other phase is removed, it is needed to ascertain if that removed phase has any dependencies, and assign them to the first phase:
Exmaple 1: Suppose we have phases A, B, C. C depends on B, and B depends on A. We wish to omit phase B. That would result in C being dependent on A.
Exmaple 2: Suppose we have phases A, B. B depends on A. We wish to omit phase A. Then B has no dependecies.

Added API

The current PhaseTemplate interface has two applyTemplate methods. These will be left unchanged. A third one will be added to support an extra parameter:
+applyTemplate(templateName:String,startDate:Date,leftOutPhases:String[]):Project
The leftOutPhases parameter (designer can provide better name) will take one or more phase names that will be left out. If any of the phases mentioned does not exist in the template, an exception should be thrown.
This method is meant to suggest an approach. The designer may provide a better approach.
This will require changes to the persistence interface and the existing implementations. The details are left to the designer, and the above approach may be used as well. Since the persistence is not exposed directly to external users, the existing methods may be modified to accommodate the upgrade as long as the current functionality is not broken.

Project documentation update

The existing project documentation must be updated to TopCoder standards:

- The project must be ported to TCUML. It currently exists in ZUML
    - The documentation nodes in TCUML must be cleaned up of any ZUML artifacts, including artifact HTML tags to make the notes readable.
- The component documentation must incorporate any changes that exist in the source. For example, the TCUML does not incorporate a DB-based persistence

Required Algorithms

XML Schema should be provided.

Example of the Software Usage

Upon creation of a project, user will provide a start date for the project and pick a template to use. The project's phases are generated and presented to the user. User can further make adjustments to the timeline and save the phases.

Future Component Direction

User interface can be designed to edit the phase templates.

Interface Requirements

Graphical User Interface Requirements

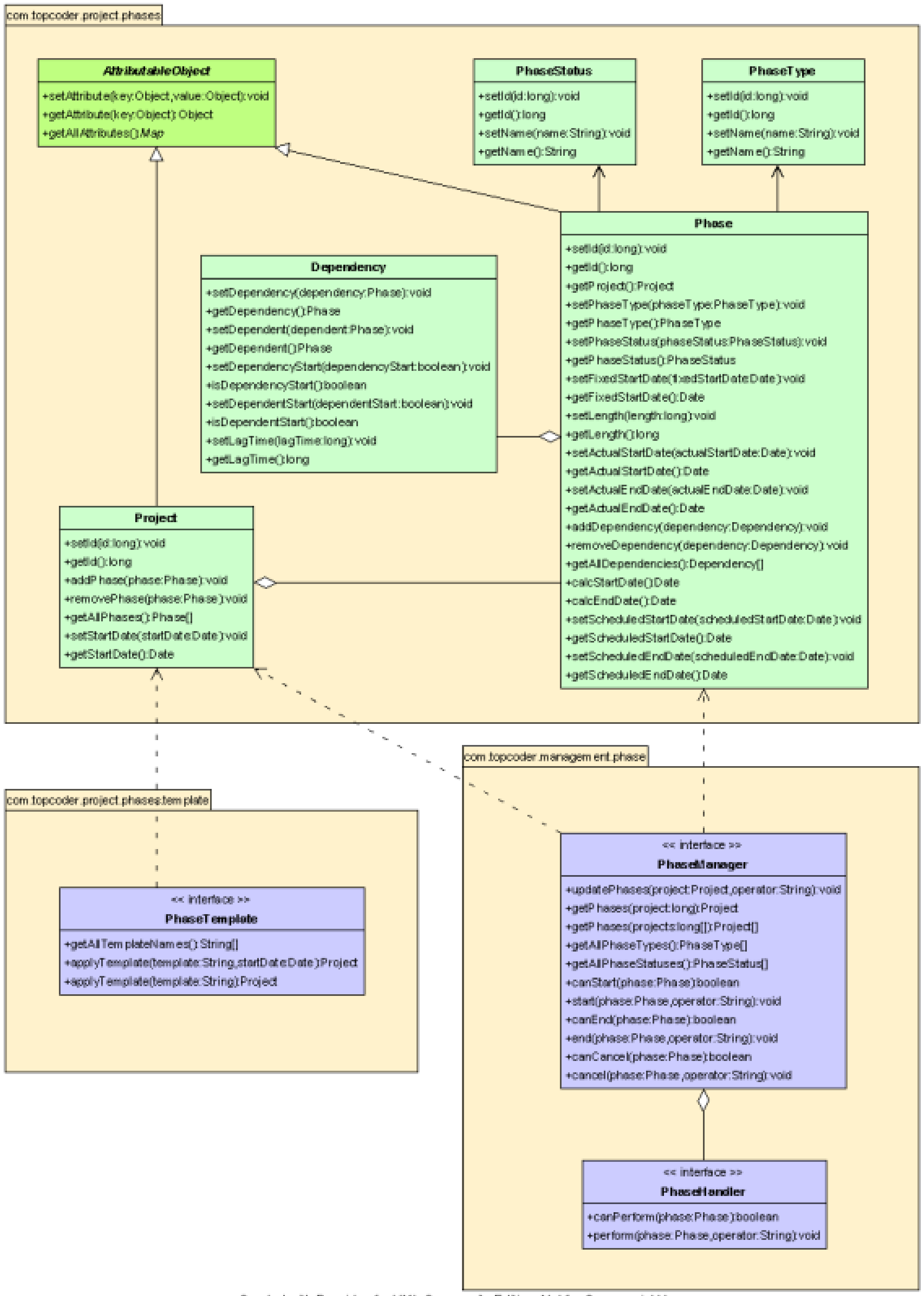
None.

External Interfaces

Design must adhere to the interface diagram definition. Designer can choose to add more methods to the classes/interfaces, but must keep the ones defined on the diagram as a minimum. Source files can be found in the distribution.
Notice that the following interface diagram is for version 1.0. The designer must update the diagram to meet the current requirements.

[TOPCODER]

Phase Management Interface Diagram


com.topcoder.project.phases
AttributableObject
+setAttribute(key:Object,value:Object):void
+getAttribute(key:Object):Object
+getAllAttributes():Map
PhaseStatus
+setId(id:long):void
+getId():long
+setName(name:String):void
+getName():String
PhaseType
+setId(id:long):void
+getId():long
+setName(name:String):void
+getName():String
Phase
+setId(id:long):void
+getId():long
+getProject():Project
+setPhaseType(phaseType:PhaseType):void
+getPhaseType():PhaseType
+setPhaseStatus(phaseStatus:PhaseStatus):void
+getPhaseStatus():PhaseStatus
+setFixedStartDate(fixedStartDate:Date):void
+getFixedStartDate():Date
+setLength(length:long):void
+getLength():long
+setActualStartDate(actualStartDate:Date):void
+getActualStartDate():Date
+setActualEndDate(actualEndDate:Date):void
+getActualEndDate():Date
+addDependency(dependency:Dependency):void
+removeDependency(dependency:Dependency):void
+getAllDependencies():Dependency[]
+calcStartDate():Date
+calcEndDate():Date
+setScheduledStartDate(scheduledStartDate:Date):void
+getScheduledStartDate():Date
+setScheduledEndDate(scheduledEndDate:Date):void
+getScheduledEndDate():Date
Dependency
+setDependency(dependency:Phase):void
+getDependency():Phase
+setDependent(dependent:Phase):void
+getDependent():Phase
+setDependencyStart(dependencyStart:boolean):void
+isDependencyStart():boolean
+setDependentStart(dependentStart:boolean):void
+isDependentStart():boolean
+setLagTime(lagTime:long):void
+getLagTime():long
Project
+setId(id:long):void
+getId():long
+addPhase(phase:Phase):void
+removePhase(phase:Phase):void
+getAllPhases():Phase[]
+setStartDate(startDate:Date):void
+getStartDate():Date
com.topcoder.project.phases.template
<< interface >>
PhaseTemplate
+getAllTemplateNames():String[]
+applyTemplate(template:String,startDate:Date):Project
+applyTemplate(template:String):Project
com.topcoder.management.phase
<< interface >>
PhaseManager
+updatePhases(project:Project,operator:String):void
+getPhases(project:long):Project
+getPhases(projects:long[]):Project[]
+getAllPhaseTypes():PhaseType[]
+getAllPhaseStatuses():PhaseStatus[]
+canStart(phase:Phase):boolean
+start(phase:Phase,operator:String):void
+canEnd(phase:Phase):boolean
+end(phase:Phase,operator:String):void
+canCancel(phase:Phase):boolean
+cancel(phase:Phase,operator:String):void
<< interface >>
PhaseHandler
+canPerform(phase:Phase):boolean
+perform(phase:Phase,operator:String):void

Created with Poseidon for UML Community Edition. Not for Commercial Use.

Environment Requirements

- Development language: Java1.4
- Compile target: Java1.4

Package Structure

com.topcoder.project.phases.template

Software Requirements

Administration Requirements

What elements of the application need to be configurable?

- The persistence storage to use
- The strategy to generate project start date
- The fixed time in a week in the default start date strategy

Technical Constraints

Are there particular frameworks or standards that are required?

XML

TopCoder Software Component Dependencies:

- Project Phases
- Configuration Manager

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

Third Party Component, Library, or Product Dependencies:

None.

QA Environment:

- Solaris 7
- RedHat Linux 7.1
- Windows 2000
- Informix 10
- Windows 2003

Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines.

Required Documentation

Design Documentation

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

Help / User Documentation

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