Project Phase Template Requirements Specification

1. Scope

1.1 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.

1.2 Logic Requirements

1.2.1 Phase Template

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

1.2.2 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.

1.2.3 Phase Generation

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

1.2.4 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.

1.3 Required Algorithms

XML Schema should be provided.

1.4 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.

1.5 Future Component Direction

User interface can be designed to edit the phase templates.

2. Interface Requirements

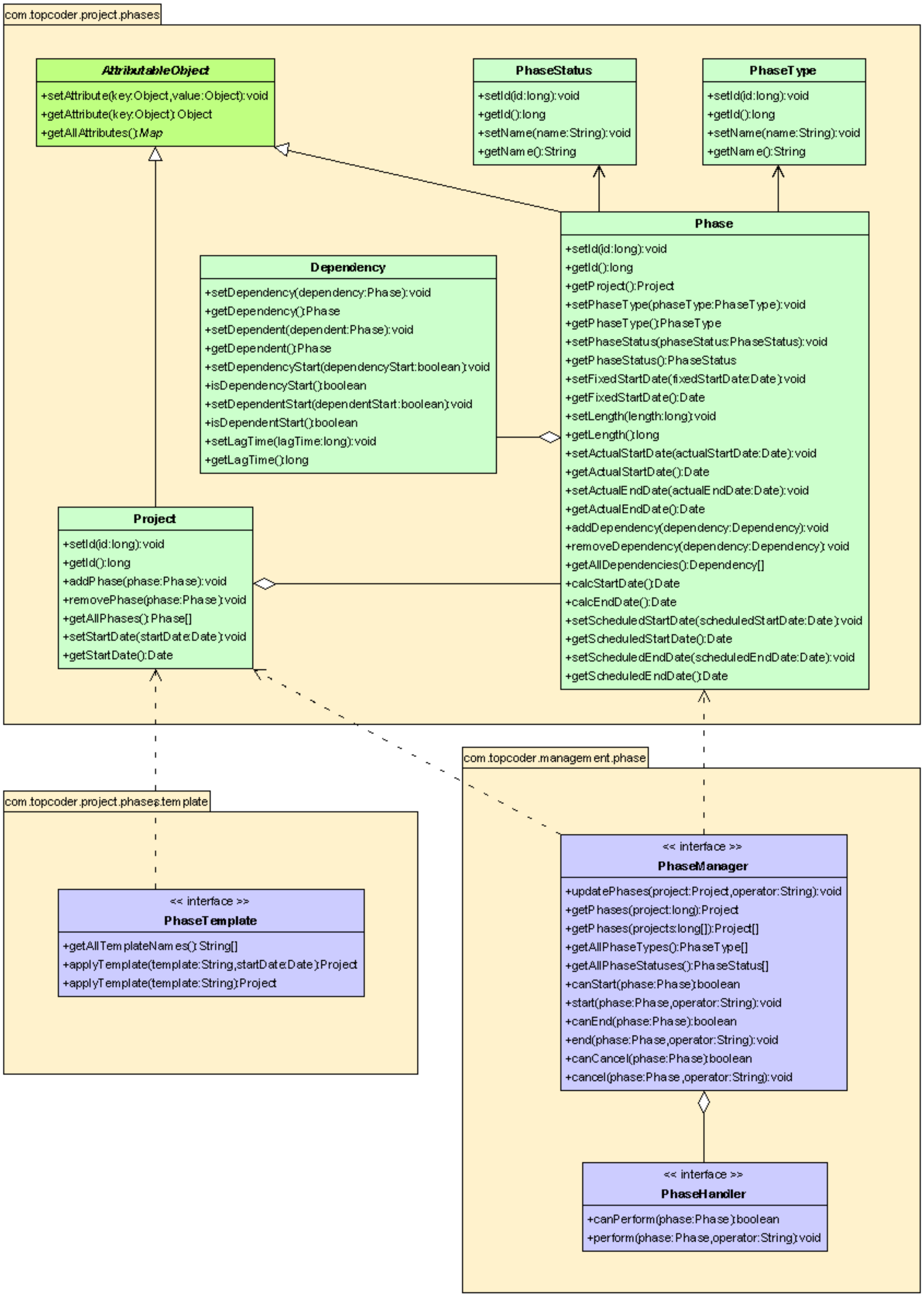
2.1.1 Graphical User Interface Requirements

None.

2.1.2 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.

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.

2.1.3 Environment Requirements

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

2.1.4 Package Structure

com.topcoder.project.phases.template

3. Software Requirements

3.1 Administration Requirements

3.1.1 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

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

XML

3.2.2 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.

3.2.3 Third Party Component, Library, or Product Dependencies:

None.

3.2.4 QA Environment:

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

3.3 Design Constraints

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

3.4 Required Documentation

3.4.1 Design Documentation

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

3.4.2 Help / User Documentation

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