Project Report

Group 014
COMP2021 Object-Oriented Programming (Fall 2023)
Author: KONG Zirui
Other group members:
LIN Zhanzhi
Raynell Lu Soon Hong
ZHEN ZHANG Alex

1 Introduction

This document describes Group 014's design and implementation of a command-line-based task management system. The project is part of the course COMP2021 Object-Oriented Programming at PolyU.

2 My Contribution and Role Distinction

I contributed to the GUI design programming and our group's template structure of the report, user manual and presentation slides. Compared to other group members, I programmed for TMS Application and implemented basic TMS tests.

3 A Command-Line Task Management System

In this section, we describe the system's overall design and implementation details.

3.1 Design

Our System is composed of 10 classes, which have 1 *public* class (*TMS*) and 9 *private* classes (*Task*, *TaskSet*, *Node*, *Criterion*, *BasicCriterion*, *BinaryCriterion*, *NegatedCriterion*, *CriterionSet* and *FinishTime*).

TMS class is the core class in TMS. It contains 10 functions including 2 *boolean* functions, 6 *void* functions and 2 *String* functions. This class is to receive messages from the GUI input area *CommandField*, and pre-handle and hand out messages to other corresponding functions.

Task class is to create *primitiveTask* and *compositeTask* with strict name & description & duration & prerequisite inspection mechanism.

TaskSet class is to store all the *primitiveTask* and *compositeTask*, and modify data with different instructions. This class can also return the basic information of task data like *printTask* and *reportDuartion*.

Node class is necessary for our task store operation. It can help the system to handle order problems between different tasks.

Criterion class is the original class of *BasicCriterion*, *BinaryCriterion* and *NegatedCriterion* class. This class can set a criterion so that we can search for tasks we need.

BasicCriterion class is to create the basic criterion. It can be stored in *CriterionSet* class.

BinaryCriterion class is to combine two different basic criteria into one binary criterion.

NegatedCriterion class is to negate an existing basic criterion.

CriterionSet class is to store criterion and return the information of it.

FinishTime class is to calculate the final finish time of composite tasks.

That's all our TMS design, and our TMS main structure and Application structure diagrams are attached at the end of this report.

3.2 Requirements

[REQ1]
1) The requirement is implemented.
2) First, the arguments are validated. Then they are processed to be passed to the Task constructor, creating a new task. After that, the new task is registered in the taskSet.
3) If the provided name, description, or duration is invalid, the process stops and reports an error. If the given name is already in use, the process stops and reports an error. If not all the specified prerequisites are defined, the process stops and reports an error.

[REQ2]
1) The requirement is implemented.
2) First, the arguments are validated. Then they are processed to be passed to the Task constructor, creating a new task. After that, the new task is registered in the taskSet.
3) If the provided name or description is invalid, the process stops and reports an error. If the given name is already in use, the process stops and reports an error. If not all the specified prerequisites are defined, the process stops and reports an error.

[REQ3]
1) The requirement is implemented.
2) First, the arguments are validated. Then the delectability of the task is checked. If the task is delectable, it is deleted along with its relevant tasks.
3) If the given name is invalid, the process stops and reports an error. If the given name does not correspond to any task, the process stops and reports an error. If the given task is not deletable, the process stops and reports an error. Deleting a primitive task results in its erasure. Deleting a composite task results in its erasure along with all its direct subtasks. A task is delectable only if deleting it, according to the specified rules, does not result in any remaining tasks that cannot be completed. A task is considered incompletable if it has a non-existent prerequisite or subtask, or if it has itself as a direct or indirect prerequisite or subtask.

[REQ4]
1) The requirement is implemented.
2) First, the arguments are validated. Then the changeability of the new value is checked. If the new value is changeable, the relevant updates are applied.
   - For a new name, it must not be already in use. All other tasks that have the current task as a direct prerequisite or subtask need to be updated with the new name. Finally, the task's name is changed in the taskSet registry.
   - For a new description or duration, only the validity of the new value is checked. The update involves simply changing the corresponding field in the taskSet registry.
   - For new prerequisites or subtasks, the changeability is determined by whether it would result in any incompletable tasks. This is detected by assuming the new prerequisites or subtasks and checking if the task becomes a direct or indirect prerequisite or subtask of itself

using Depth First Search. The update involves changing the corresponding field in the taskSet registry.
3) If the new value is invalid, the process stops and reports an error. If the new name is already in use, the process stops and reports an error. If the new prerequisites or subtasks would cause incompletable tasks, the process stops and reports an error.

[REQ5]
1) The requirement is implemented.
2) First, the argument is validated. Then the information of the task is printed, with each attribute on a separate line.
3) If the given argument is invalid, the process stops and reports an error. If the given name does not correspond to any task, the process stops and reports an error.

[REQ6]
1) The requirement is implemented.
2) The information of all registered tasks is printed. The number of existing tasks is displayed. Then each task's information is printed.
3) There are no error cases.

[REQ7]
1) The requirement is implemented.
2) First, the argument is validated. Then the duration of the task is calculated. For a primitive task, the duration is stored during its definition. For a composite task, the duration is calculated by constructing a graph of the task and its direct and indirect subtasks. The simulation process determines when tasks can start based on their prerequisites being finished.
3) If the given name is invalid, the process stops and reports an error. If the given name does not correspond to any task, the process stops and reports an error.

[REQ8]
1) The requirement is implemented.
2) First, the argument is validated. Then the earliest finish time of the task is calculated. For a composite task, the earliest finish time is the same as its duration. For a primitive task, the calculation involves simulating the multitasking process to determine the minimum number of hours required to finish the task.
3) If the given name is invalid, the process stops and reports an error. If the given name does not correspond to any task, the process stops and reports an error.

[REQ9]
1) The requirement is implemented.
2) First, the argument is validated. Then the argument is processed to identify the critical path of the project. The critical path is determined by calculating the latest start and finish times for each task in the project. The critical path is the sequence of tasks with the longest total duration, meaning any delay in these tasks will directly impact the project's overall duration.
3) If the given argument is invalid, the process stops and reports an error. If the given name does not correspond to any task, the process stops and reports an error.

[REQ10]
1) The requirement is implemented.

2) When initializing the Task Management System (TMS) Controller component, you can create the IsPrimitive criterion, which is implemented as a basic criterion. This criterion is defined with a condition that checks if the duration of a task is greater than 0.
3) No error cases.

[REQ11]
1) The requirement is implemented.
2) First, the arguments are validated and then used to create a new corresponding criterion, which is then registered.
3) If the given name is invalid, the process stops and reports an error. If the given name does not correspond to any task, the process stops and reports an error.

[REQ12]
1) The requirement is implemented.
2) Collect the defined criteria from the registry, criterionSet, and output their information one by one. For criteria that are composed of other criteria, the printing is recursive until it resolves to the information of a basic criterion.
3) No error cases

[REQ13]
1) The requirement has been implemented.
2) All registered tasks are tested against the specified condition, and when they satisfy the condition, they are displayed.
3) If the search criterion is not defined, the process is halted, and an error message is reported.

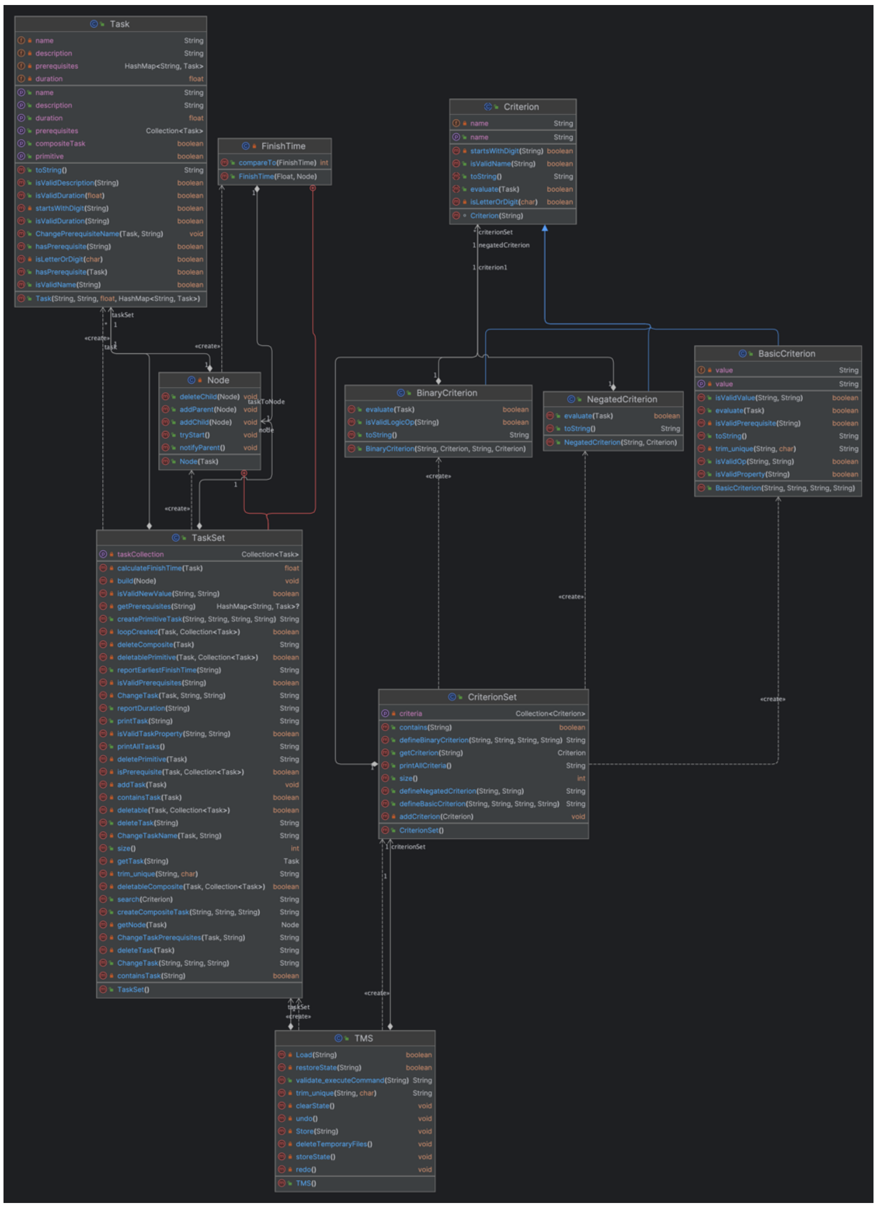
[REQ14]
1) The requirement is implemented.
2) It ensures the existence of the specified file at the given path and then saves the current program state, taskSet, and criterionSet to the file.
3) If the given path is invalid, the process is halted by throwing an exception, and an error message is reported.

[REQ15]
1) The requirement is implemented.
2) It ensures the existence of the specified file at the given path and then attempts to read the current program state, taskSet, and criterionSet from the file.
3) If the given path is invalid, the process is halted by throwing an exception, and an error message is reported. If the given file is incorrectly organized, the process is halted, and an error message is reported.

[BON1]
1) The requirement is implemented.
2) At the beginning of the program's execution, a frame containing three components is created. The first component is a chat area that displays input commands and output messages. The second component is a command area where users can enter commands. The third component is a button that triggers communication with the TMS using the entered command or clicking the button.

3) To handle messages exceeding the result area, the scroll bar is kept at the bottom, showing the latest output. When the user adjusts the dimensions of the frame, the layout of the three components is adjusted simultaneously.

[BON2]
1) The requirement is implemented.
2) Program states are stored as temporary .data files after successful modifications of the program state. Two stacks are used to track the last state and next state by storing the paths of the corresponding .data files. When performing an undo, the current state (top of the undo stack) is moved to the top of the redo stack, and then the state stored at the top of the undo stack is restored. When performing a redo, the state at the top of the redo stack is restored, becoming the top of the undo stack. When a modification of the program state is triggered, the redo stack is cleared since the program state develops in another branch.
3) If the undo stack contains only the current state, the process halts the undo operation and reports it. If the redo stack is empty, the process halts the redo operation and reports it.


Task
name String
description String
prerequisites HashMap<String, Task>
duration float
name String
description String
duration float
prerequisites Collection<Task>
compositeTask boolean
primitive boolean
toString() String
isValidDescription(String) boolean
isValidDuration(float) boolean
startsWithDigit(String) boolean
isValidDuration(String) boolean
ChangePrerequisiteName(Task, String) void
hasPrerequisite(String) boolean
isLetterOrDigit(char) boolean
hasPrerequisite(Task) boolean
isValidName(String) boolean
Task(String, String, float, HashMap<String, Task>)
FinishTime
compareTo(FinishTime) int
FinishTime(Float, Node)
Criterion
name String
name String
startsWithDigit(String) boolean
isValidName(String) boolean
toString() String
evaluate(Task) boolean
isLetterOrDigit(char) boolean
Criterion(String)
criterionSet
negatedCriterion
criterion1
Node
deleteChild(Node) void
addParent(Node) void
addChild(Node) void
tryStart() void
notifyParent() void
Node(Task)
taskToNode
node
taskSet
task
«create»
BinaryCriterion
evaluate(Task) boolean
isValidLogicOp(String) boolean
toString() String
BinaryCriterion(String, Criterion, String, Criterion)
NegatedCriterion
evaluate(Task) boolean
toString() String
NegatedCriterion(String, Criterion)
BasicCriterion
value String
value String
isValidValue(String, String) boolean
evaluate(Task) boolean
isValidPrerequisite(String) boolean
toString() String
trim_unique(String, char) String
isValidOp(String, String) boolean
isValidProperty(String) boolean
BasicCriterion(String, String, String, String)
TaskSet
taskCollection Collection<Task>
calculateFinishTime(Task) float
build(Node) void
isValidNewValue(String, String) boolean
getPrerequisites(String) HashMap<String, Task>?
createPrimitiveTask(String, String, String, String) String
loopCreated(Task, Collection<Task>) boolean
deleteComposite(Task) String
deletablePrimitive(Task, Collection<Task>) boolean
reportEarliestFinishTime(String) String
isValidPrerequisites(String) boolean
ChangeTask(Task, String, String) String
reportDuration(String) String
printTask(String) String
isValidTaskProperty(String, String) boolean
printAllTasks() String
deletePrimitive(Task) String
isPrerequisite(Task, Collection<Task>) boolean
addTask(Task) void
containsTask(Task) boolean
deletable(Task, Collection<Task>) boolean
deleteTask(String) String
ChangeTaskName(Task, String) String
size() int
getTask(String) Task
trim_unique(String, char) String
deletableComposite(Task, Collection<Task>) boolean
search(Criterion) String
createCompositeTask(String, String, String) String
getNode(Task) Node
ChangeTaskPrerequisites(Task, String) String
deleteTask(Task) String
ChangeTask(String, String, String) String
containsTask(String) boolean
TaskSet()
CriterionSet
criteria Collection<Criterion>
contains(String) boolean
defineBinaryCriterion(String, String, String, String) String
getCriterion(String) Criterion
printAllCriteria() String
size() int
defineNegatedCriterion(String, String) String
defineBasicCriterion(String, String, String, String) String
addCriterion(Criterion) void
CriterionSet()
TMS
Load(String) boolean
restoreState(String) boolean
validate_executeCommand(String) String
trim_unique(String, char) String
clearState() void
undo() void
Store(String) void
deleteTemporaryFiles() void
storeState() void
redo() void
TMS()
Diagram 1 TMS Main Structure Design

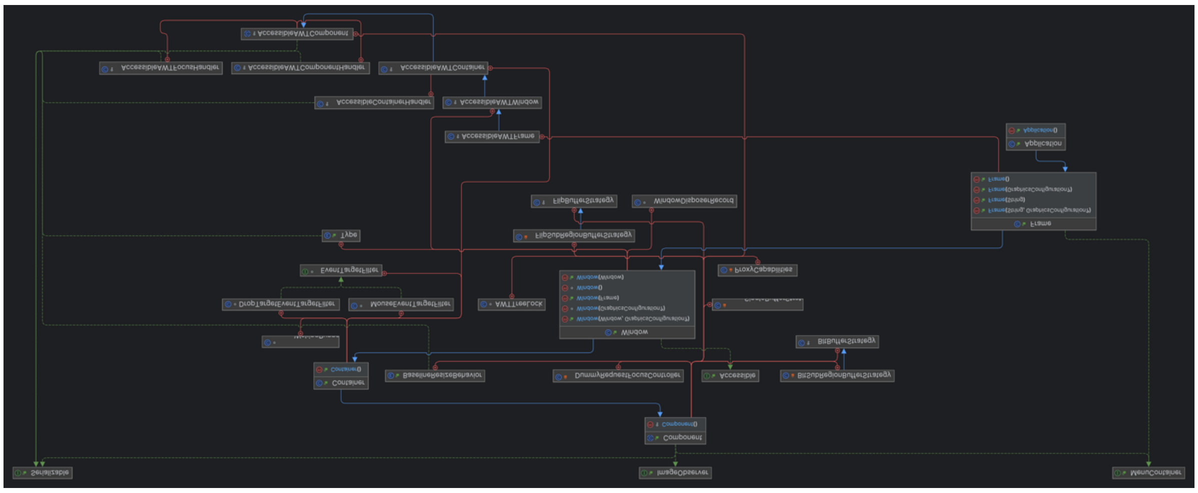


Diagram 2 TMS Application Structure Design