# COMP4001 Object-Oriented Software Development

## Preliminary Design

Groupmembers: Richard MANTIK (2287103)
Marvyn Jiexiang MEI (2244246)
Agus SALIM (2251992)
Hendri KURNIAWAN (3104443)

## Contents

# 1 Introduction

## 1.1 Overview

The aim of this project is to develop a project planner that provides a convenient way of planning and managing a project. The project planner should be able to perform tasks that are typical to other project planners, such as Microsoft Project. The project shall be developed using Object Oriented methods, and will be written in C++.

## 1.2 Objectives

The ultimate aim of this project is to develop a high quality project planner. The scope of the project will include:

1. The functional and formal specifications of this system. This report will address the functional aspects of the requirements definition.

2. A series of working prototype that would allow the client to evaluate the progress of this project.

3. The final software that would be demonstrated and evaluated at the end of this development.

## 1.3 Scope of Work

This report will include class diagrams, sequence diagrams, and CRC cards, as part of the Object Oriented methodology, to specify the system. These will help in the development of the entire system.

# 2 Project Requirements

## 2.1 Minimum Requirements

The project planner should contain the following features as part of the minimum requirements specified by the client.

1. Gantt charts.

2. Save / Load project files

3. Merge two or more saved project files

4. Edit a project: adjust times of the segments, add/remove resources.

5. Display milestones, in addition to tasks

6. Allow priorities to be assigned to each task

## 2.2 Additional Requirements

The project planner, in addition to the minimum requirements, will contain the following requirements.

1. A GUI for the user to interact with the project planner

2. A graphical view to display Gantt Chart, in addition of the text based Gantt Chart

3. Check for over-utilisation of a resource when assigning resources to tasks, and when merging two projects containing the same resource.

# 3 Design Patterns

## 3.1 Model View Controller

There are 3 components in the application that are frequently changed: the data, the application logic and the user interface. It will be very difficult to maintain an application that has interdependencies between these components, because changes anywhere in these three components propagate the modification thorough the entire application. In addition, high coupling between components makes them difficult to be reused because one class depends on the other classes. To solve this problem, Model View Controller can be used to decouple the data access, the application logic, and the interface.

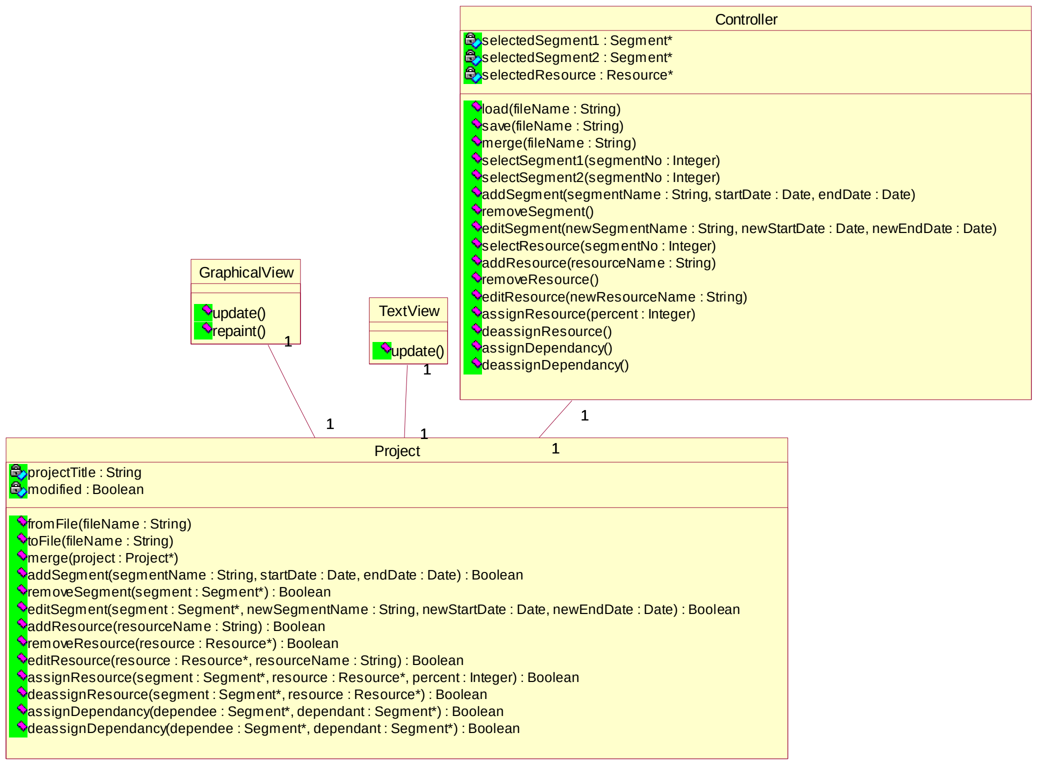

Controller
selectedSegment1 : Segment*
selectedSegment2 : Segment*
selectedResource : Resource*
load(fileName : String)
save(fileName : String)
merge(fileName : String)
selectSegment1(segmentNo : Integer)
selectSegment2(segmentNo : Integer)
addSegment(segmentName : String, startDate : Date, endDate : Date)
removeSegment()
editSegment(newSegmentName : String, newStartDate : Date, newEndDate : Date)
selectResource(segmentNo : Integer)
addResource(resourceName : String)
removeResource()
editResource(newResourceName : String)
assignResource(percent : Integer)
deassignResource()
assignDependancy()
deassignDependancy()
GraphicalView
update()
repaint()
TextView
update()
1
1
1
1
1
1
Project
projectTitle : String
modified : Boolean
fromFile(fileName : String)
toFile(fileName : String)
merge(project : Project*)
addSegment(segmentName : String, startDate : Date, endDate : Date) : Boolean
removeSegment(segment : Segment*) : Boolean
editSegment(segment : Segment*, newSegmentName : String, newStartDate : Date, newEndDate : Date) : Boolean
addResource(resourceName : String) : Boolean
removeResource(resource : Resource*) : Boolean
editResource(resource : Resource*, resourceName : String) : Boolean
assignResource(segment : Segment*, resource : Resource*, percent : Integer) : Boolean
deassignResource(segment : Segment*, resource : Resource*) : Boolean
assignDependancy(dependee : Segment*, dependant : Segment*) : Boolean
deassignDependancy(dependee : Segment*, dependant : Segment*) : Boolean

Table of icons for attributes and operations

**Attributes**

| | |
|---|---|
| Public | 🔷 |
| Private | 🔒🔷 |

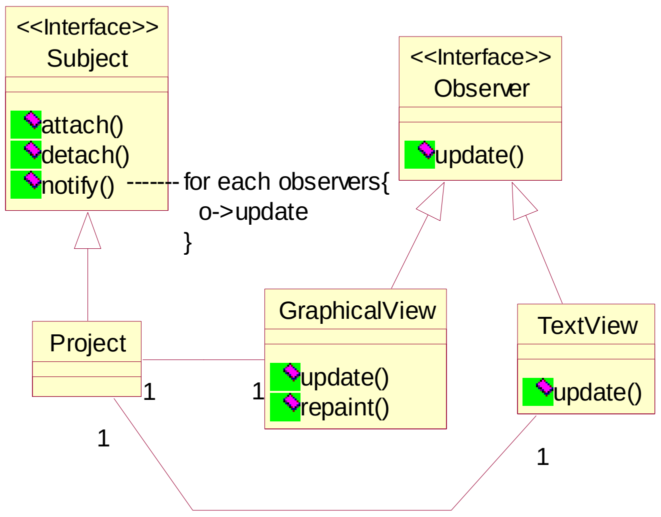
**Operation**

| | |
|---|---|
| Public | ◆ |
| Private | 🔒◆ |

The Controller acts as Controller, Project acts as Model, and both GraphicalView and TextView acts as Views. The Controller accepts inputs from the user and interprets this input as commands that are sent to the Project object. The Project manages its state and handles the queries that are sent from the Controller. Both of the GraphicalView and TextView are responsible to present the state of the project object to the user. Any component can change its behaviour without adverse effects to the other components.

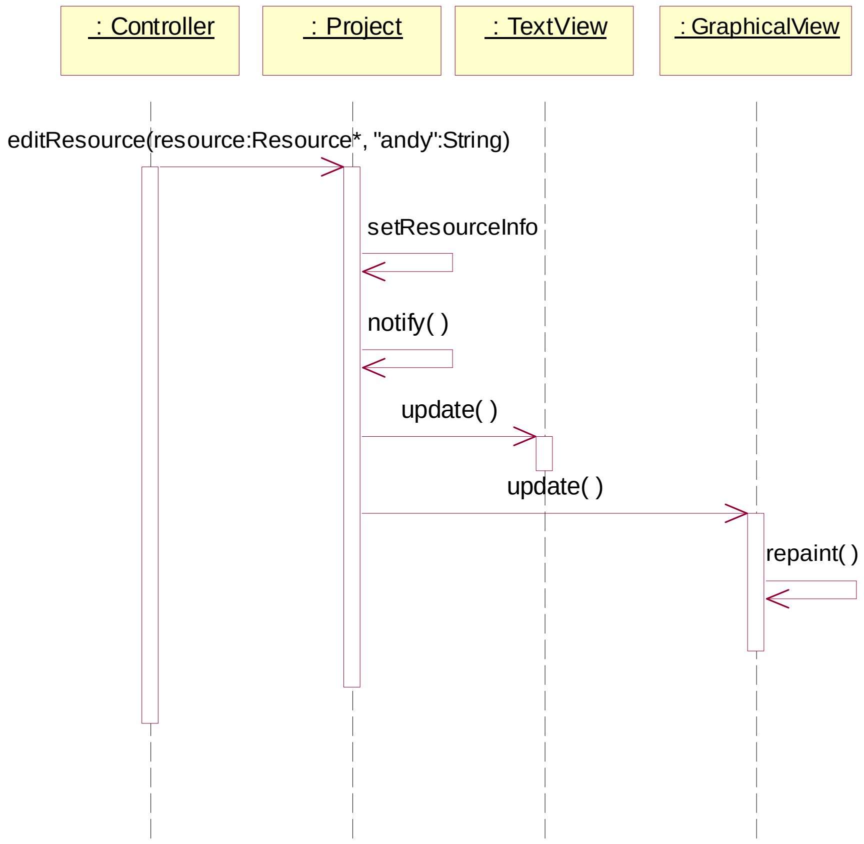
## 3.2 Observer

As a consequence of partitioning the components into model and views, the need to maintain the consistency between each state of the components arises. Both of the GraphicalView and TextView represents the same Project's at one time using different format. Interdependencies between them are undesirable, because it will reduce the reusability of the components. In this case, Observer pattern can be used to establish the relationships between the project's data and two of the representations. Here, the project object acts as a subject that will notify two of the observers, namely GraphicalView and TextView whenever there is a change of the project's state.



The Observer pattern makes the coupling between the data and the views minimal. As a result, the data and views can be varied and reused independently. Since the observers have no knowledge of each other, a new type of view can just be added to the subject easily at any time without modifying other observers.

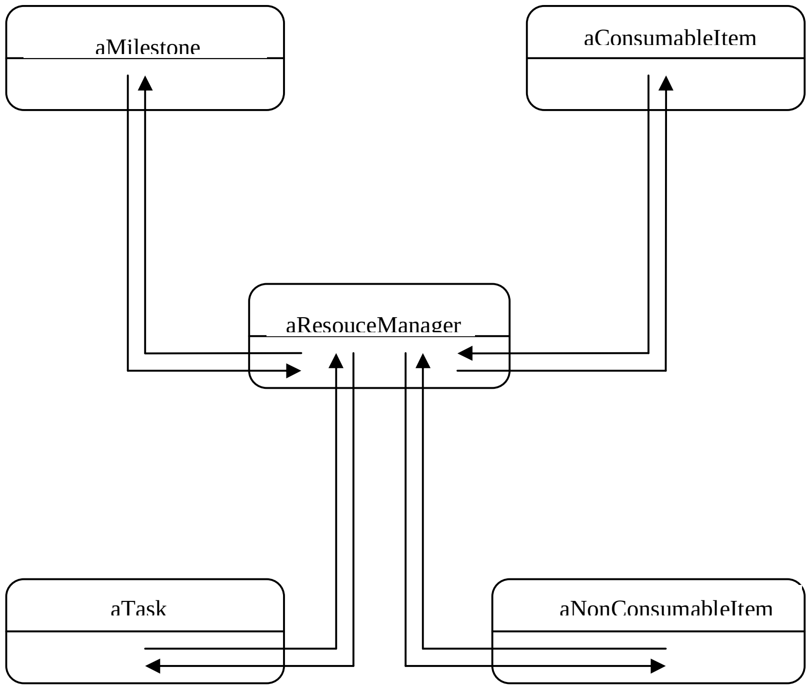
The following sequence diagram illustrates the interaction between a project and its views when an event "edit resource" occurs. Upon receiving edit calls from the Controller, the project object update its internal state and call its method "notify()" that will send the message to all of its views to synchronize the state with the project's state.


: Controller
: Project
: TextView
: GraphicalView
editResource(resource:Resource*, "andy":String)
setResourceInfo
notify( )
update( )
update( )
repaint( )

## 3.3 Mediator

There are dependencies between the segment and resource in a project. For example, when the user assign a particular resource to a particular task, the system will check the availability of the resource before the assignment is allowed, i.e. depending of the segments' priority, those with higher priority get assigned to the resource more likely. Another case is when a user wants to update the start date or end date of a task that has already assigned to a particular resource, the system needs to check whether there is a clash of that resource usage at that particular date range. If this behaviour is localized in either resource or segment, when the change of the system behaviour is desirable, we can't simply reuse those classes. Instead, they have to be customized to reflect the change of dependencies.

The problem can be solved by introducing the ResourceManager that will serve as the mediator and keep the segment and resource object from referring to each other explicitly. Consequently, ResourceManager promotes loose coupling between segments and resources. Segment and resource objects communicate with each other indirectly without knowing each other, through the ResourceManager. As a result, the ResourceManager encapsulates the collective behaviour among segments and resources. Changing the behaviour requires extending or replacing the ResourceManager only, thus enhances reusability of colleague classes.

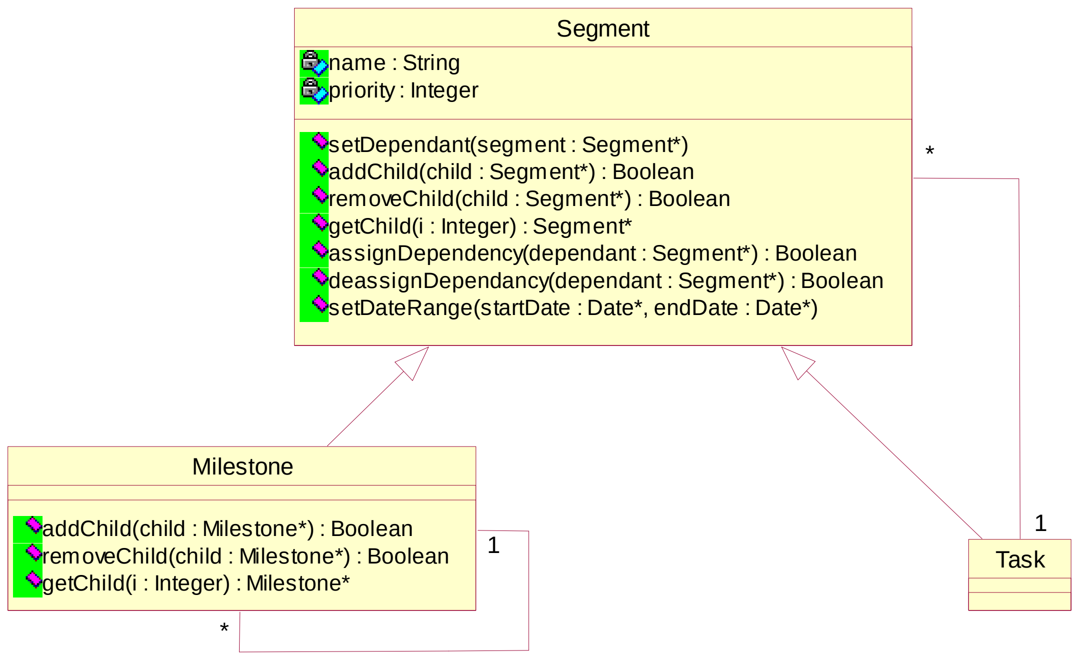


## 3.4 Composite

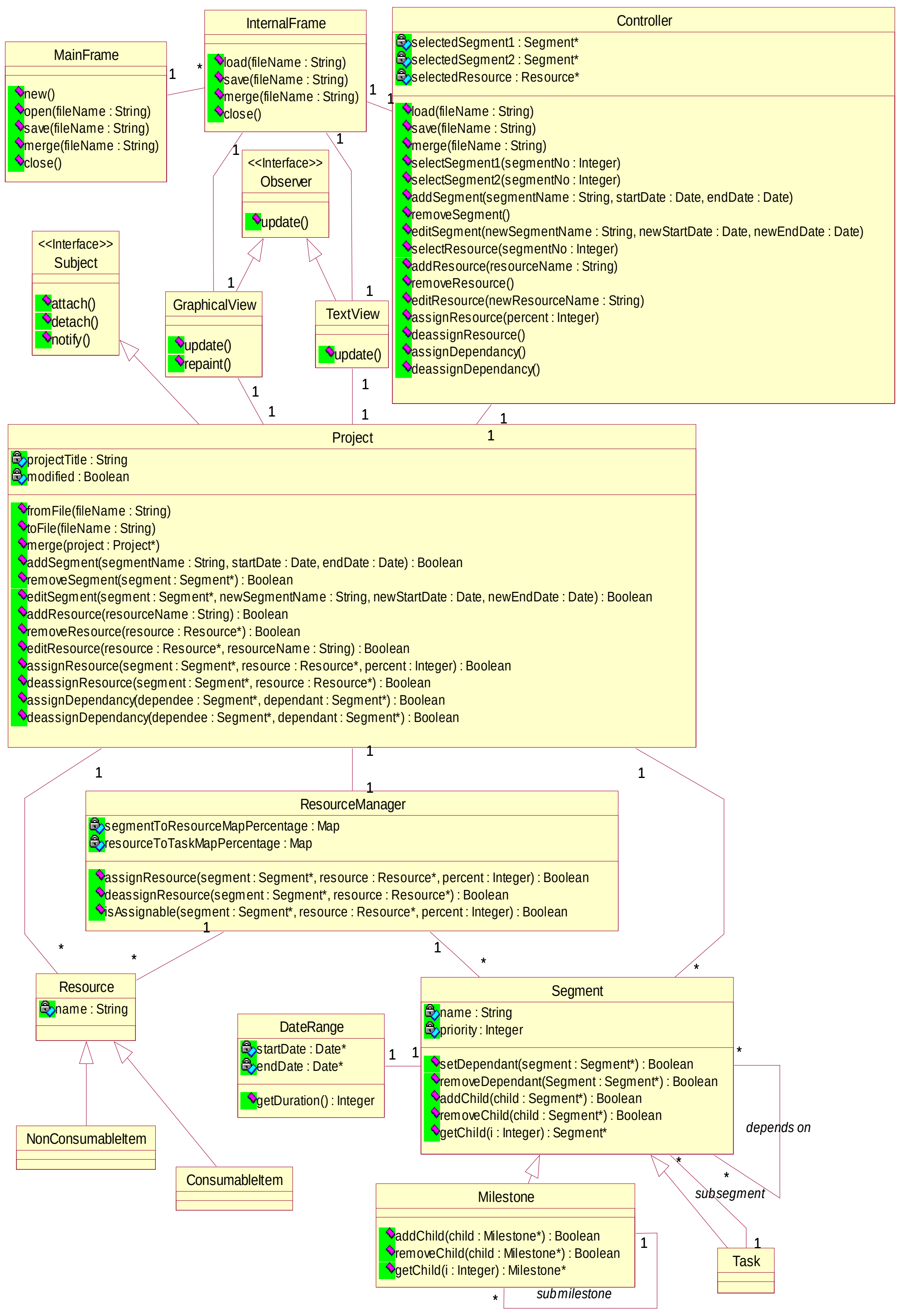
Tasks can be divided into subtasks and milestones, and subtasks can in turn be divided still into smaller subtasks and milestones. In contrast, milestones can only be composed of smaller milestones. A simple solution will treat the task and milestone objects differently. This forces the application to distinguish these objects and makes client's code more complex.

An alternative solution will make use of the Composite design pattern that model the task's composition recursively and allows the client's code to treat task and milestone uniformly, i.e. whenever a method addChild on task is called, it can also take a Segment. Client doesn't have to know whether they are dealing with task or segment in this case. Additionally, this will make things easier when new kinds of segment are introduced to the system. Newly defined subclasses will always work with the existing structures.

Note: A milestone can't be divided into subtasks. This will require the client code to use run time check to enforce this restriction so that only addChild() that takes Milestone as an argument is called. This kind of check should also be applied when a new subclass is added with a certain restriction.


Segment
name : String
priority : Integer
setDependant(segment : Segment*)
addChild(child : Segment*) : Boolean
removeChild(child : Segment*) : Boolean
getChild(i : Integer) : Segment*
assignDependency(dependant : Segment*) : Boolean
deassignDependancy(dependant : Segment*) : Boolean
setDateRange(startDate : Date*, endDate : Date*)
*
Milestone
addChild(child : Milestone*) : Boolean
removeChild(child : Milestone*) : Boolean
getChild(i : Integer) : Milestone*
1
*
Task
1

## 4 Class Diagram


MainFrame
new()
open(fileName : String)
save(fileName : String)
merge(fileName : String)
close()
InternalFrame
load(fileName : String)
save(fileName : String)
merge(fileName : String)
close()
Controller
selectedSegment1 : Segment*
selectedSegment2 : Segment*
selectedResource : Resource*
load(fileName : String)
save(fileName : String)
merge(fileName : String)
selectSegment1(segmentNo : Integer)
selectSegment2(segmentNo : Integer)
addSegment(segmentName : String, startDate : Date, endDate : Date)
removeSegment()
editSegment(newSegmentName : String, newStartDate : Date, newEndDate : Date)
selectResource(segmentNo : Integer)
addResource(resourceName : String)
removeResource()
editResource(newResourceName : String)
assignResource(percent : Integer)
deassignResource()
assignDependancy()
deassignDependancy()
<<Interface>>
Observer
update()
<<Interface>>
Subject
attach()
detach()
notify()
GraphicalView
update()
repaint()
TextView
update()
Project
projectTitle : String
modified : Boolean
fromFile(fileName : String)
toFile(fileName : String)
merge(project : Project*)
addSegment(segmentName : String, startDate : Date, endDate : Date) : Boolean
removeSegment(segment : Segment*) : Boolean
editSegment(segment : Segment*, newSegmentName : String, newStartDate : Date, newEndDate : Date) : Boolean
addResource(resourceName : String) : Boolean
removeResource(resource : Resource*) : Boolean
editResource(resource : Resource*, resourceName : String) : Boolean
assignResource(segment : Segment*, resource : Resource*, percent : Integer) : Boolean
deassignResource(segment : Segment*, resource : Resource*) : Boolean
assignDependancy(dependee : Segment*, dependant : Segment*) : Boolean
deassignDependancy(dependee : Segment*, dependant : Segment*) : Boolean
ResourceManager
segmentToResourceMapPercentage : Map
resourceToTaskMapPercentage : Map
assignResource(segment : Segment*, resource : Resource*, percent : Integer) : Boolean
deassignResource(segment : Segment*, resource : Resource*) : Boolean
isAssignable(segment : Segment*, resource : Resource*, percent : Integer) : Boolean
Resource
name : String
NonConsumableItem
ConsumableItem
DateRange
startDate : Date*
endDate : Date*
getDuration() : Integer
Segment
name : String
priority : Integer
setDependant(segment : Segment*) : Boolean
removeDependant(Segment : Segment*) : Boolean
addChild(child : Segment*) : Boolean
removeChild(child : Segment*) : Boolean
getChild(i : Integer) : Segment*
depends on
subsegment
Milestone
addChild(child : Milestone*) : Boolean
removeChild(child : Milestone*) : Boolean
getChild(i : Integer) : Milestone*
submilestone
Task
1
*

**Description**

**- MainFrame and InternalFrame**: The application has a main frame that holds the menu items such as: new, open, save, merge and close document. The internal frames that represent documents reside at the main frame's desktop, for example when you open a document, a new internal frame is created and being added to the main frame's desktop.

**- Controller, GraphicalView and TextView**: The InternalFrame has panels that hold the controller object that is used by user to manipulate the state of the project and the views that presents it to the user. These components are part of the main GUI that interacts with user.

**- Project**: This class encapsulates all information about the project document, namely the name of the project document, the modified flags and the references to the segments and resources that is contained in a project document.

**- ResourceManager:** ResourceManager class handles the assignment of a resource to a segment, and keep the mapping between them in a map. For efficient computation, the ResourceManager keeps both the map from segment to resource with corresponding percentage and the map from resource to segment with the same percentage. It also determines whether the modification of the date period of a task doesn't clash with other task that uses the same resource at that period.

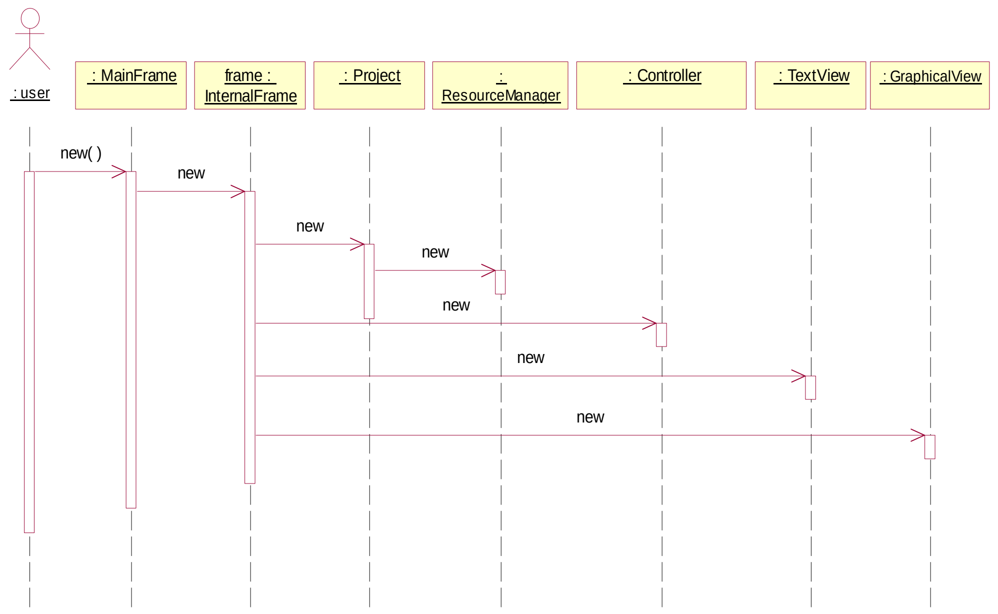
**- Segments, Tasks, and Milestones**: Segments can be either tasks or milestones. While tasks can be made of segments, milestones can only be made of milestones. And tasks may depend on other segment; that is a task may not start before other tasks or milestone which it is depended on.

**- Resource, NonComsumableItem and ConsumableItem:** Tasks or Milestones may require resource in their progresses. There are 2 types of resource: consumable resources, i.e. the ones that can be depleted in its use, and non consumable resources, i.e. the ones that can't be depleted.

**- Date Range:** A class that encapsulate the information about the date.

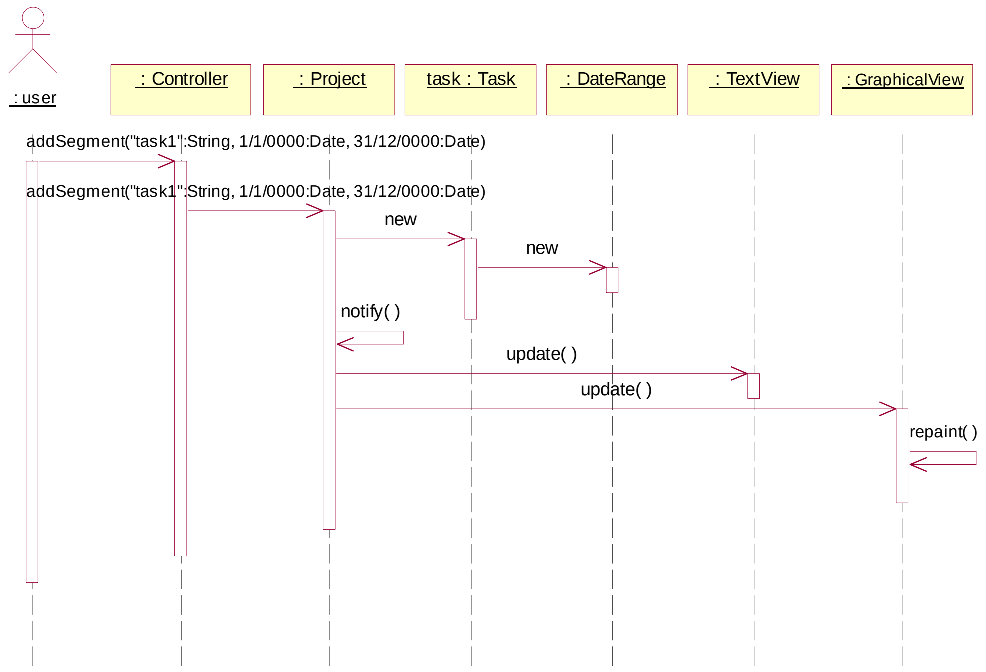
## 5 Sequence Diagrams

### 5.1 Create New Project



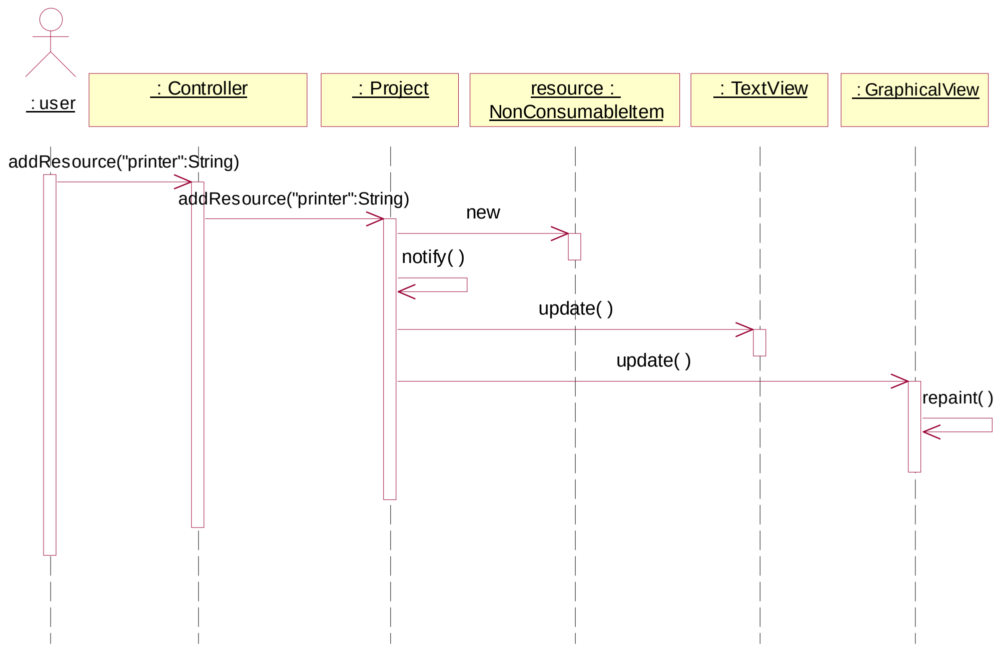
| Typical Flow of Events | Actor Action | System Response |
|---|---|---|
| | 1. Click "New" menu item on the main frame. | 2. Create a new internal frame. |
| | | 3. Create a new project and resource manager associated with it. |
| | | 4. Create GUI components, i.e. Controller and Views |
| **Precondition** | System needs sufficient memory to open up a new project. | |
| **Postcondition** | A new project internal frame is opened, containing a new project. | |

## 5.2 Add Segment



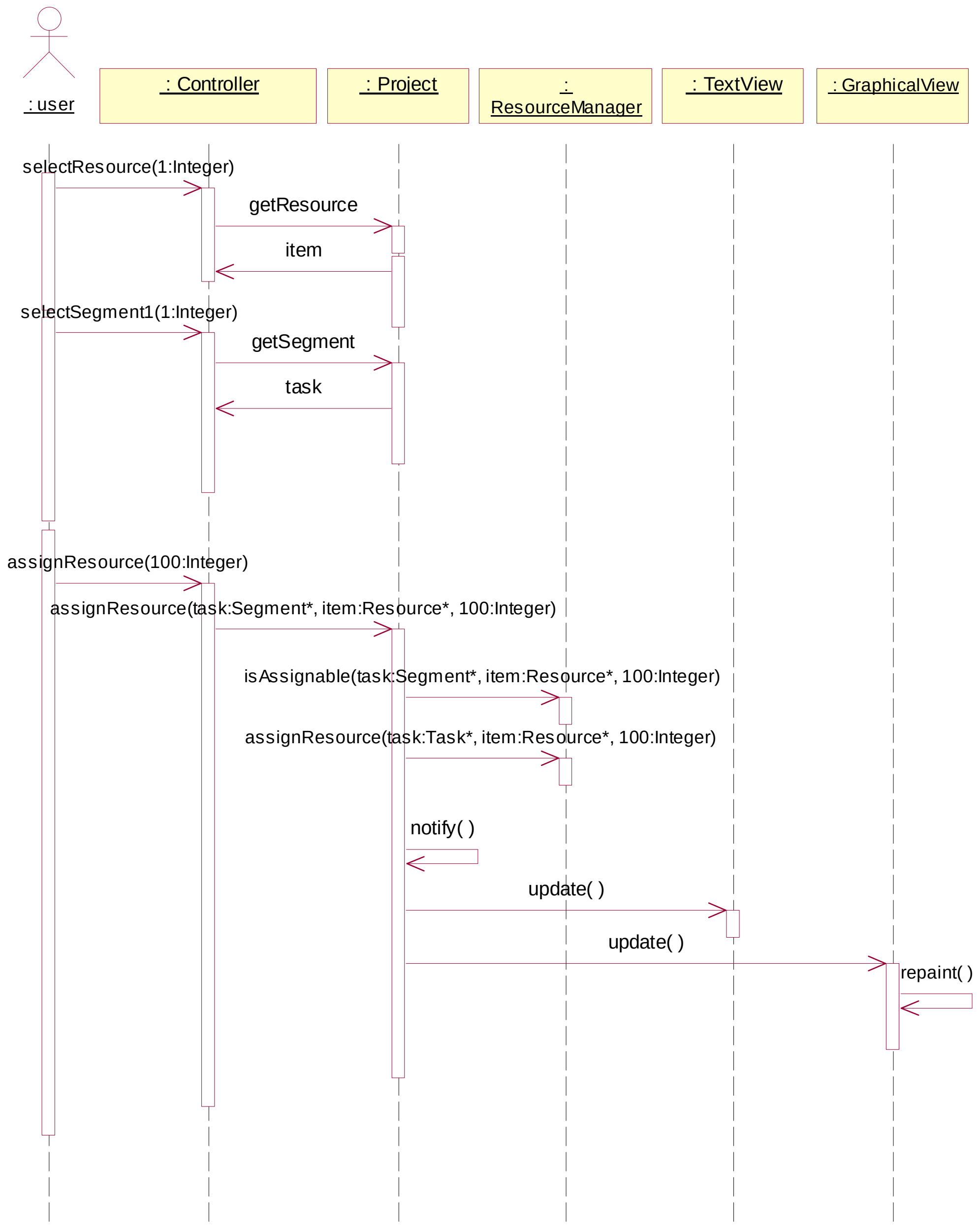
| Typical Flow of Events | Actor Action | System Response |
|---|---|---|
| | 1. Enter new task information field in the textfield that is contained in the controller and click “Add Segment” button. | 2. Create a new segment and its date range. Associate this segment to the project. |
| | | 3. Update views. |
| **Precondition** | System needs sufficient memory to store the new segment and its associated data. | |
| **Postcondition** | The new segment and its associated data are correctly stored in the project. | |

## 5.3 Add Resource



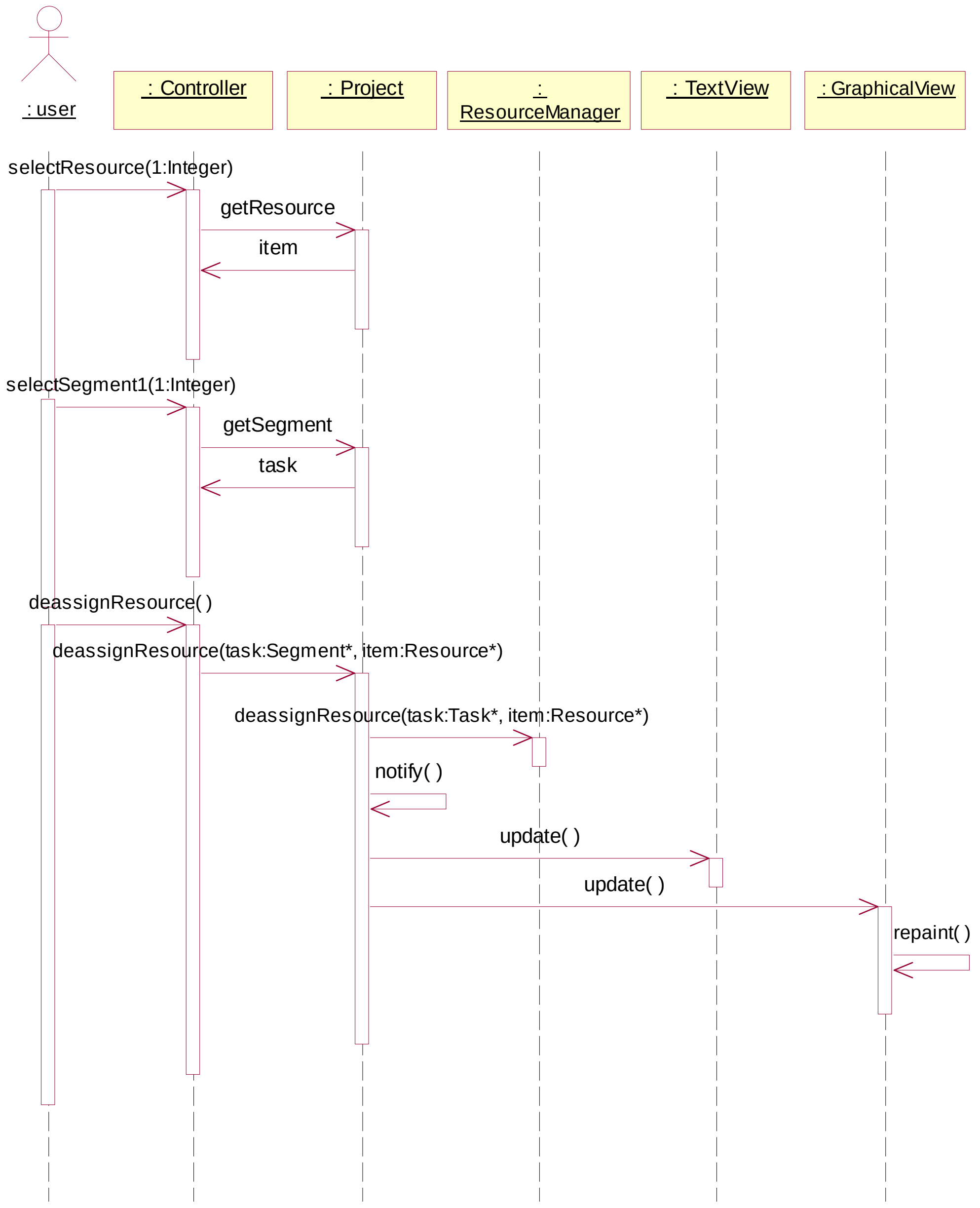
| **Typical Flow of Events** | **Actor Action** | **System Response** |
|---|---|---|
| | 1. Enter resource information in the text field and click “Add Resource” button. | 2. Create new resource and associate it to the project. |
| | | 3. Update views. |
| **Precondition** | System needs sufficient memory to store the new resource. | |
| **Postcondition** | A new resource is added to the project. | |

## 5.4 Assign Resource to Task


: user
: Controller
: Project
: ResourceManager
: TextView
: GraphicalView
selectResource(1:Integer)
getResource
item
selectSegment1(1:Integer)
getSegment
task
assignResource(100:Integer)
assignResource(task:Segment*, item:Resource*, 100:Integer)
isAssignable(task:Segment*, item:Resource*, 100:Integer)
assignResource(task:Task*, item:Resource*, 100:Integer)
notify( )
update( )
update( )
repaint( )

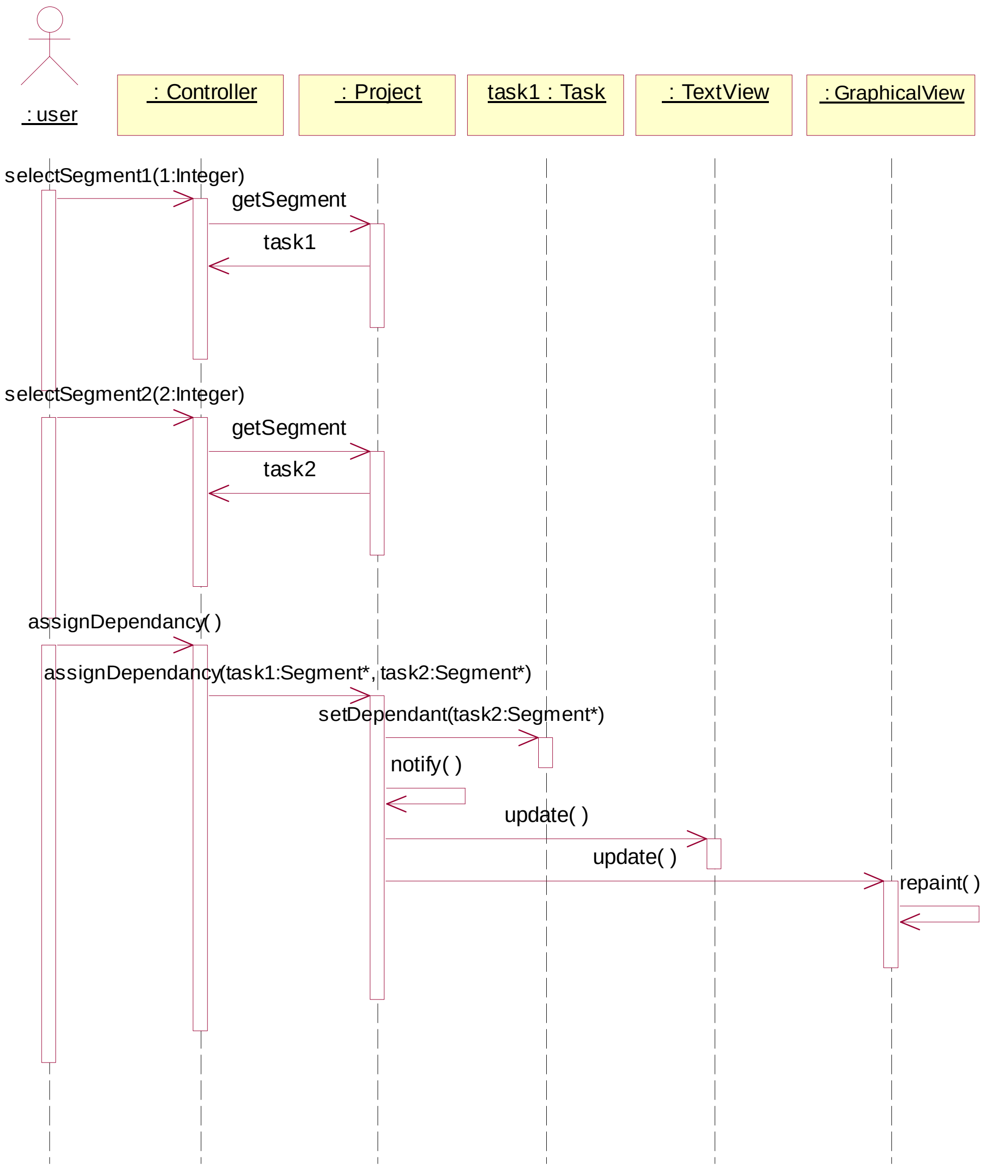
| **Typical Flow of Events** | **Actor Action** | **System Response** |
|---|---|---|
| | 1. Enter the number associated with the resource to be assigned, then click “Select Resource” button. | 2. Retrieve the reference to the corresponding resource, if any. |
| | 3. Enter segment’s number to which the resource is to be assigned to, then click “Select Segment“ button. | 4. Retrieve the reference to the corresponding segment, if any. |
| | 5. Click “Assign Resource” button. | 6. Check whether the resource is available. If it is, assign the resource to the segment with the corresponding percentage. |
| | | 7. Update all views. |
| **Precondition** | 1 Both of the resource and the segment exist.<br>2. The resource utilisation must not exceed 100% at any specific time. | |
| **Postcondition** | Resource is properly assigned to task, with no over-utilisation. | |

## 5.5 De-assign Resource



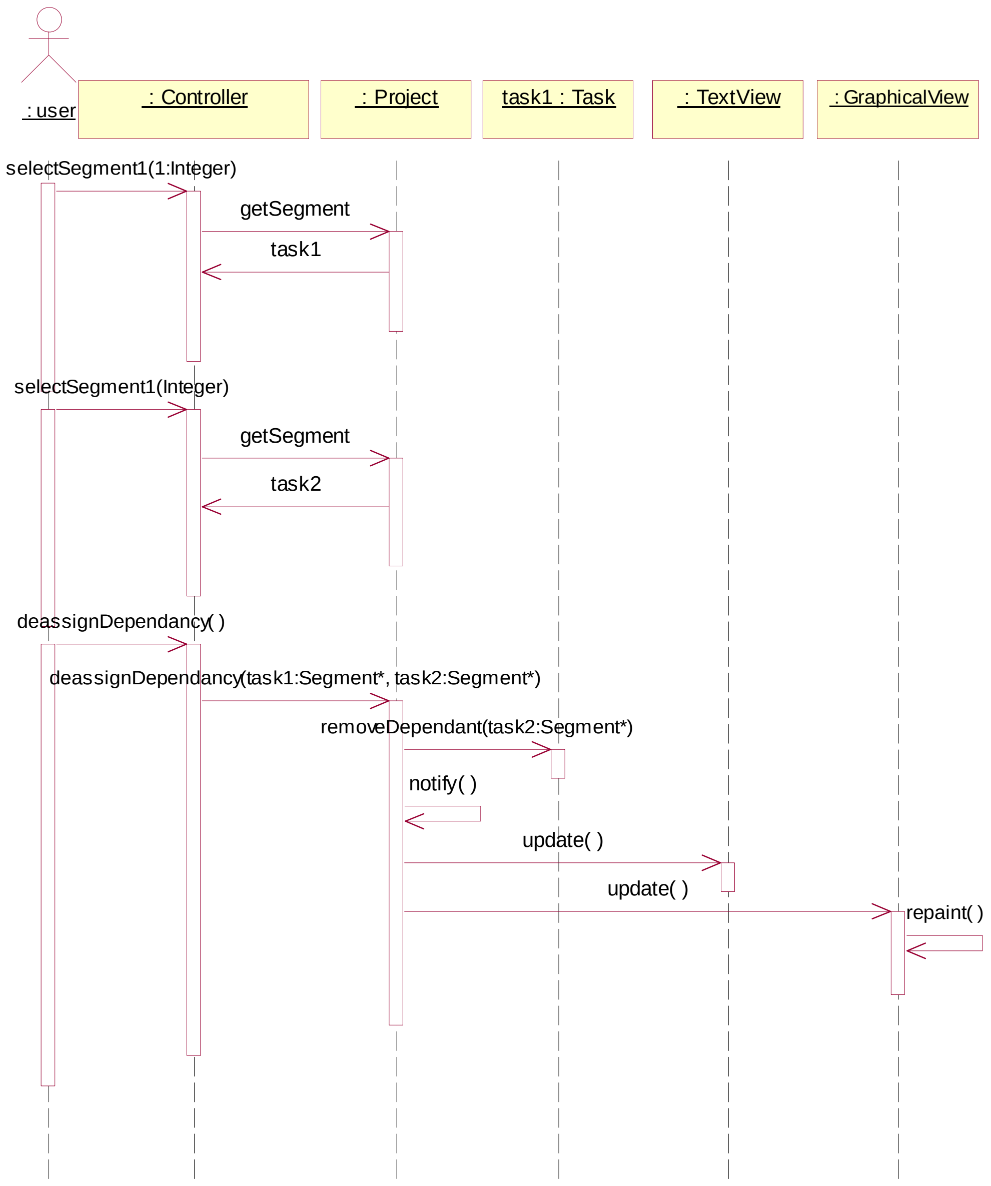
| **Typical Flow of Events** | **Actor Action** | **System Response** |
|---|---|---|
| | 1. Enter the number associated with the resource, then click “Select Resource” button. | 2. Retrieve the reference to the corresponding resource, if any. |
| | 3. Enter segment’s number to which the resource was assigned to, then click “Select Segment“ button. | 4. Retrieve the reference to the corresponding segment, if any. |
| | 5. Click “Deassign Resource” button. | 6. Break the assignment between resource and segment, if any. |
| | | 7. Update all views. |
| **Precondition** | 1. Both of the resource and segment exist<br>2. The task-resource relation must exist. | |
| **Postcondition** | Resource is properly de-assigned from task. | |

### 5.6 Assign Segment's Dependency



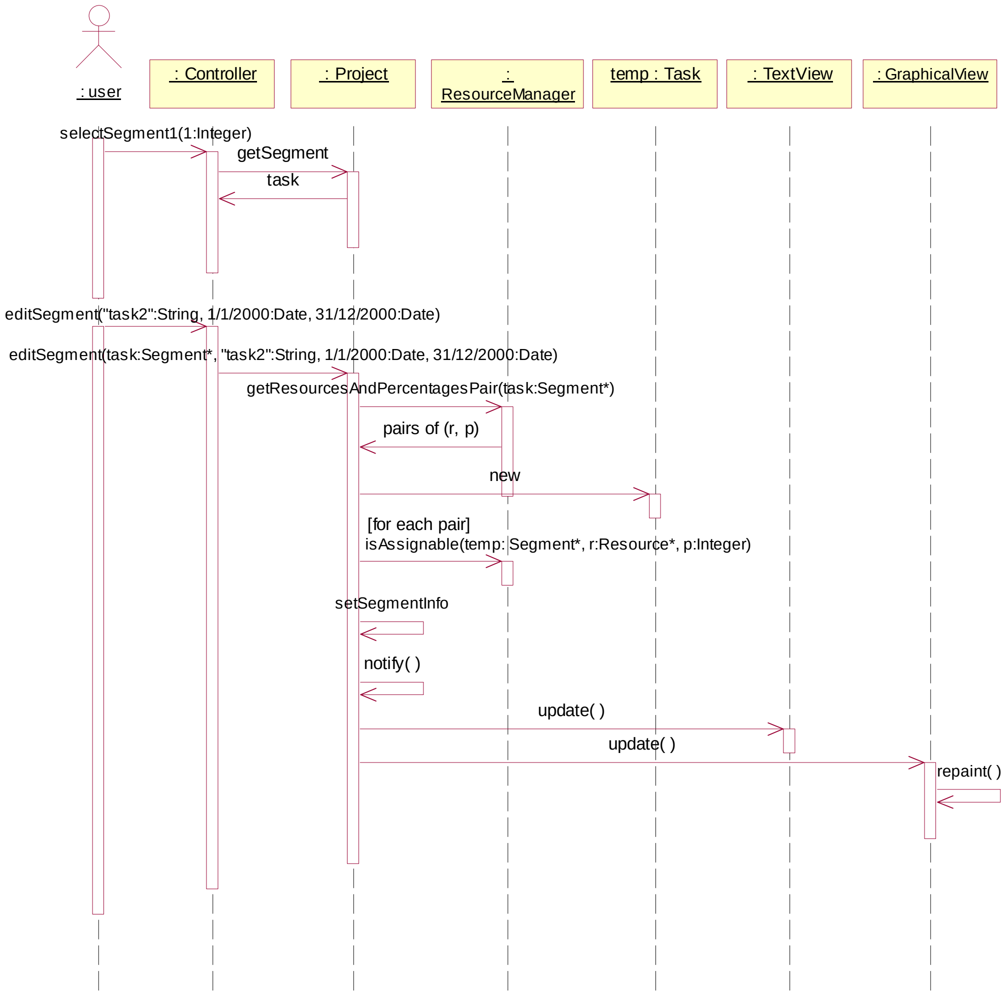
| **Typical Flow of Events** | **Actor Action** | **System Response** |
|---|---|---|
| | 1. Enter the first segment’s number which is to be dependee, then click select segment. | 2. Retrieve the reference to the corresponding segment, if any. |
| | 3. Enter the second segment’s number which is to be dependant, then click select segment. | 4. Retrieve the reference to the corresponding segment, if any. |
| | 5. Click “Assign Dependancy” button. | 6. Link the dependency between the 2 segments. |
| | | 7. Update all views. |
| **Precondition** | 1. Both of the segments exist.<br>2. Both segments must not overlap in time. | |
| **Postcondition** | A dependency has been assigned between two segments. | |

## 5.7 De-assign Dependency



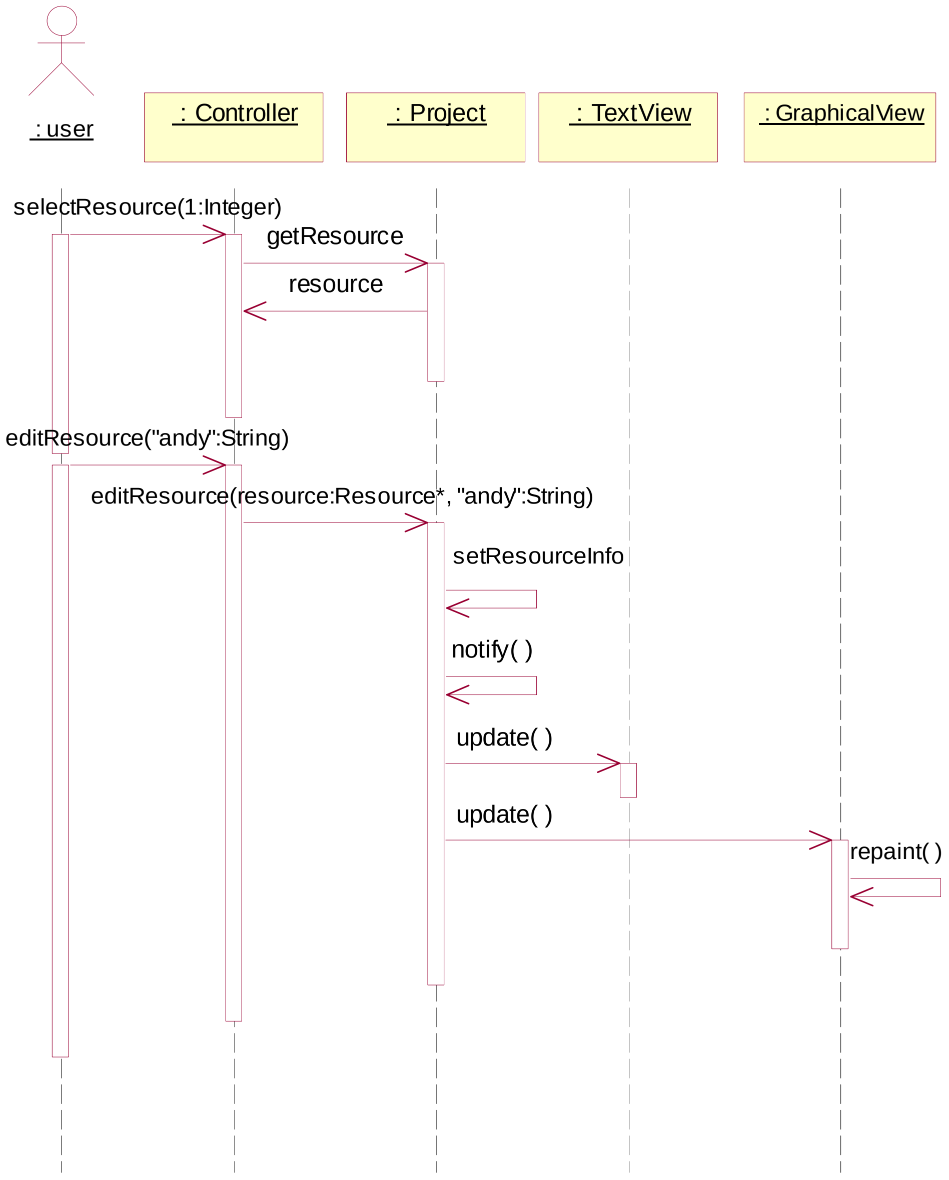
| **Typical Flow of Events** | **Actor Action** | **System Response** |
|---|---|---|
| | 1. Enter the dependee’s number, then click select segment. | 2. Retrieve the reference to the corresponding segment, if any. |
| | 3. Enter the dependant’s number, then click select segment. | 4. Retrieve the reference to the corresponding segment, if any. |
| | 5. Click the “Deassign Dependency” button. | 6. Unlink the dependancy between 2 segments, if any. |
| | | 7. Update all views. |
| **Precondition** | 1. Both of the segments exist<br>2. Two segments must have a dependency between them. | |
| **Postcondition** | A dependency has been de-assigned between two dependent segments. | |

## 5.8 Edit Segment


: user
: Controller
: Project
: ResourceManager
temp : Task
: TextView
: GraphicalView
selectSegment1(1:Integer)
getSegment
task
editSegment("task2":String, 1/1/2000:Date, 31/12/2000:Date)
editSegment(task:Segment*, "task2":String, 1/1/2000:Date, 31/12/2000:Date)
getResourcesAndPercentagesPair(task:Segment*)
pairs of (r, p)
new
[for each pair]
isAssignable(temp: Segment*, r:Resource*, p:Integer)
setSegmentInfo
notify( )
update( )
update( )
repaint( )

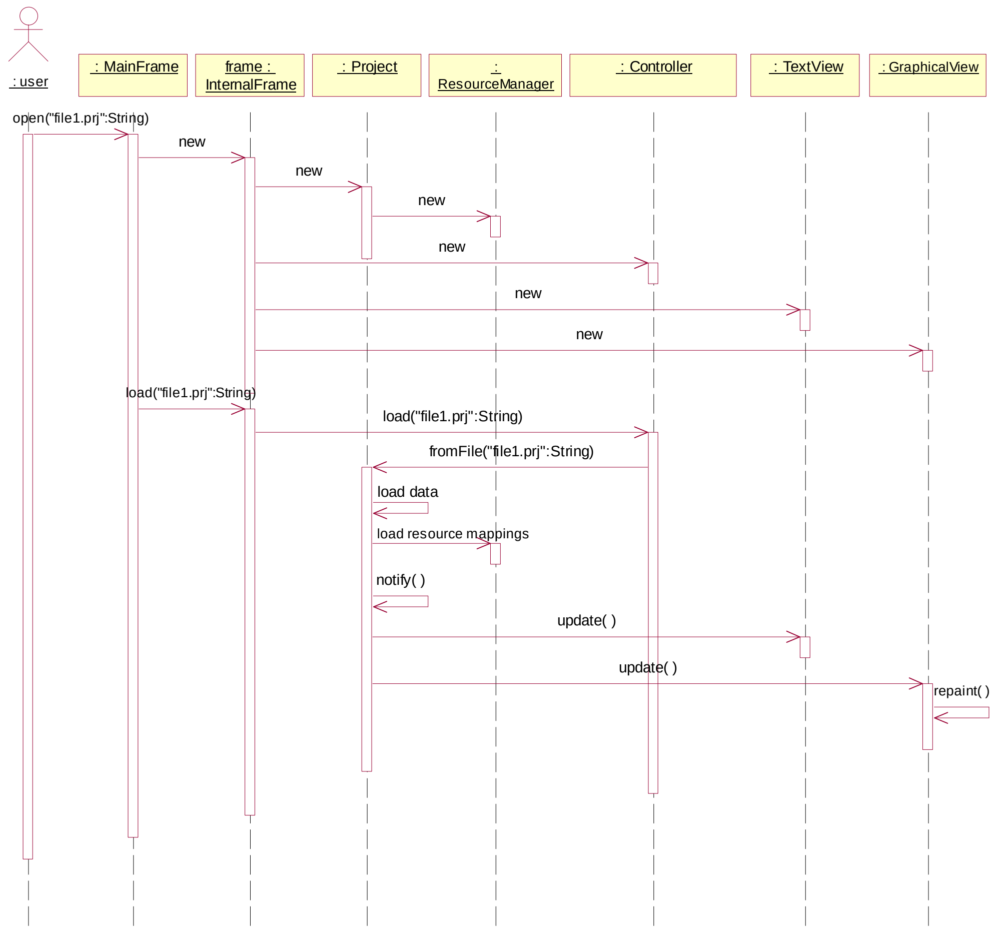
| **Typical Flow of Events** | **Actor Action** | **System Response** |
|---|---|---|
| | 1. Enter the segment number to be edited, then click select segment. | 2. Retrieve the reference to the corresponding segment, if any. |
| | 3. Enter new information about the segment, then click “Update Segment” button. | 4. For each resource associated with the segment, check whether the resource is not overused with the new date period. If ok, then update the segment. |
| | | 5. Update all views. |
| **Precondition** | 1. The segment exists.<br>2. Resources associated with the segment are not overused even with the new date period. | |
| **Postcondition** | The segment is updated with the new information. | |

## 5.9 Edit Resource



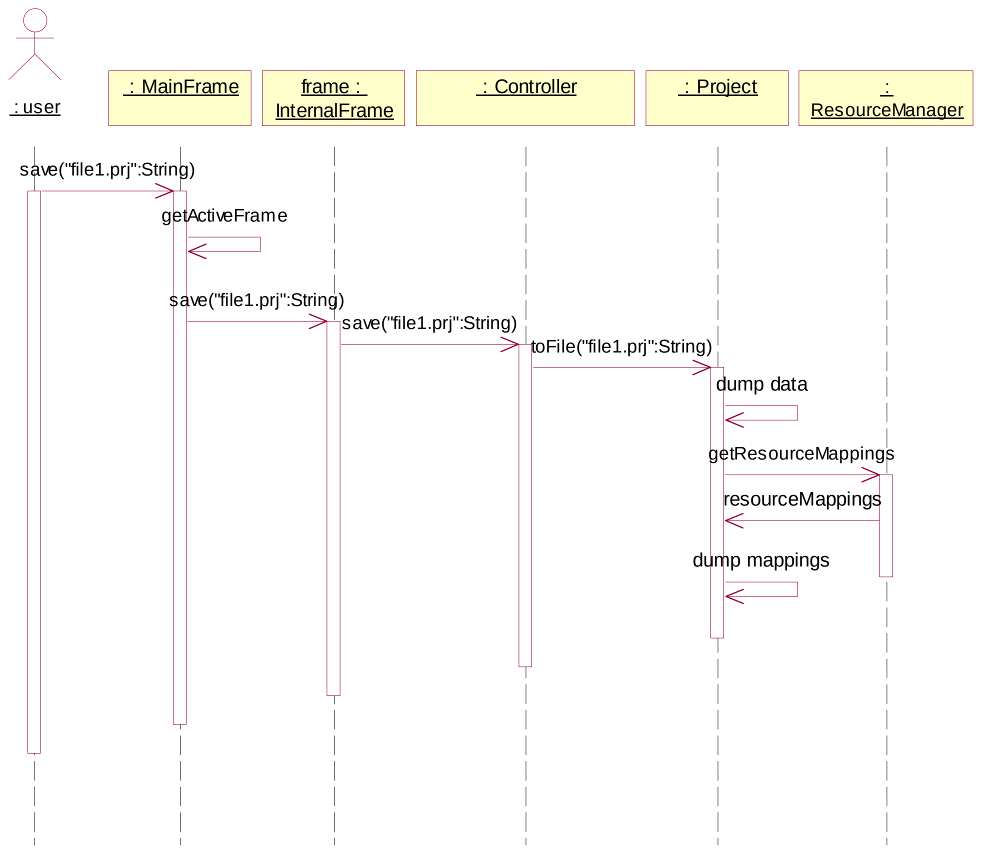
| Typical Flow of Events | Actor Action | System Response |
|---|---|---|
| | 1. Enter the resource number to be edited, and then click select resource. | 2. Retrieve the reference to the corresponding resource, if any. |
| | 3. Enter new information about the resource, then click “Update Resource” button. | 4. Update the resource information. |
| | | 5. Update all views. |
| **Precondition** | The resource exists. | |
| **Postcondition** | The resource is updated with new information | |

## 5.10 Open Project



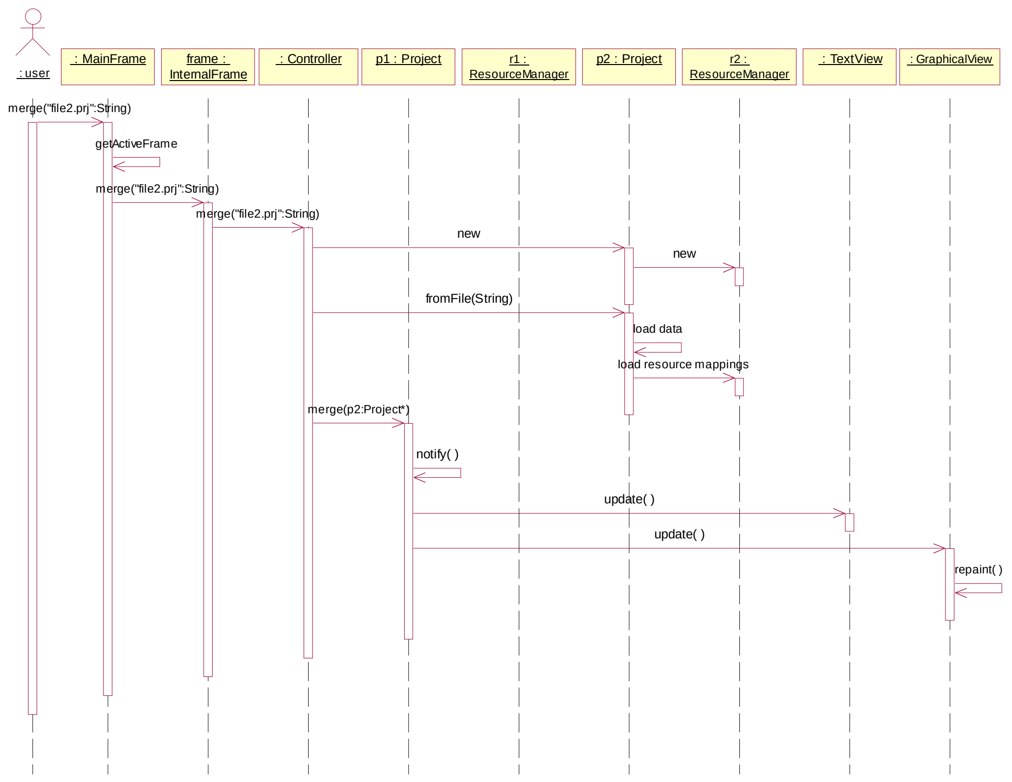
| Typical Flow of Events | Actor Action | System Response |
|---|---|---|
| | 1. Click on the "Open" menu item in the main frame. | 2. Display the FileOpenDialog object. |
| | 3. Enter the filename and click "Open" button. | 4. Create required objects as in the "Create New Project" scenario |
| | | 5. Load the data in the file. |
| | | 6. Update all views. |
| **Precondition** | 1. Specified file must exist on system.<br>2. System needs sufficient memory to load the project.<br>3. File must be in the correct format. | |
| **Postcondition** | A new project internal frame (with its internal frame controller) is opened, containing the loaded project. | |

### 5.11 Save Project



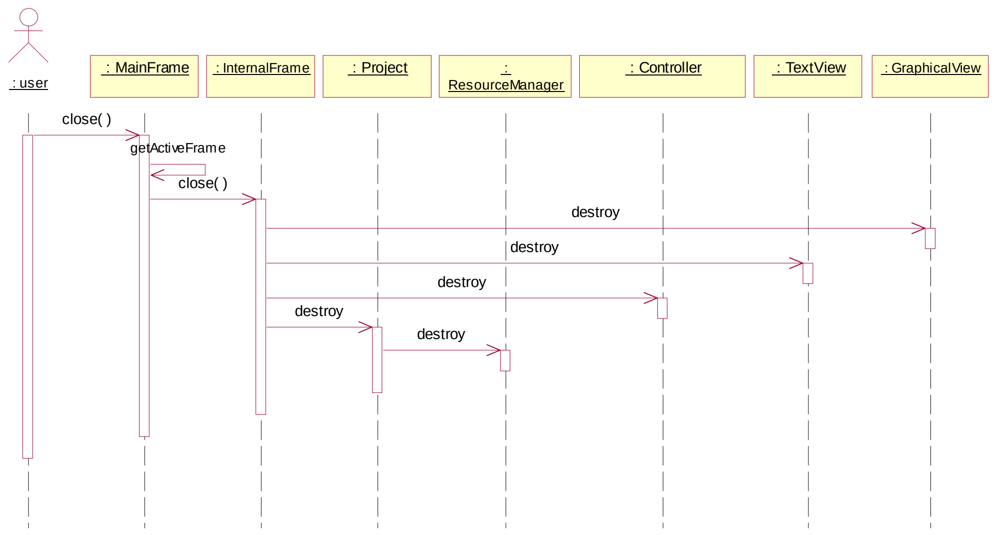
| **Typical Flow of Events** | **Actor Action** | **System Response** |
|---|---|---|
| | 1. Click on the “Save As” menu item on the mainframe. | 2. Display the FileSaveDialog object |
| | 3. Specify the filename and click “Save” button. | 4. Dump all the data in the specified file |
| **Precondition** | 1. There must be an open project.<br>2. System needs sufficient disk space to save the file.<br>3. File must not locked by the other application. | |
| **Postcondition** | Data is saved in the application specific format into file. | |

## 5.12 Merge Project



| **Typical Flow of Events** | Actor Action | System Response |
|---|---|---|
| | 1. Click “Merge” on menu item on the main frame. | 2. Display FileOpenDialog object. |
| | 3. User selects file to merge with, and click “Open” button. | 4. Load the data associated with the opened (merged) file into temporary project object. |
| | | 5. Merge it with the opened project. |
| | | 6. Update all views. |
| **Precondition** | 1. There must be an opened project.<br>2. Specified file must exist on system.<br>3. System needs sufficient memory to load the second project. | |
| **Postcondition** | Selected file is merged into currently opened file. | |

## 5.13 Close Project



| Typical Flow of Events | Actor Action | System Response |
|---|---|---|
| | 1. Click “Close Project” on menu item. | 2. Close the active frame, which in turn destroy all components that contained in it, including the Project and Resource Manager object. |
| **Precondition** | There is at least one active frame in the main frame’s desktop. | |
| **Postcondition** | The project internal frame and project document are properly closed. | |

## 6 CRC Cards

| **Main Frame** | |
|---|---|
| Handles the document specific tasks, such as new, open, save and close. | InternalFrame |
| Container for InternalFrame. | |

| **InternalFrame** | |
|---|---|
| Container for controller and views. | Main Frame |
| | Controller |
| | TextView |
| | GraphicalView |

| **TextView** | |
|---|---|
| Display the state of the project document in text based format. | InternalFrame |
| | Project |

| **GraphicalView** | |
|---|---|
| Display the state of the project document in graphical format. | InternalFrame |
| | Project |

| **Controller** | |
|---|---|
| Handles user inputs that in turn modify the state of project document. | Internal Frame |
| | Project |

| **Project** | |
|---|---|
| Encapsulates the document information including segments and resources. | Controller |
| Response to the query that is sent by the controller. | TextView |
| Notify changes of the internal state to the views. | GraphicalView |
| | Segment |
| | Resource |
| | ResourceManager |

| **ResourceManager** | |
|---|---|
| Keep the information about the assignment of each resource to the corresponding segments. | Project |
| Check the availability of the resource. | Segment |
| | Resource |

| **Resource** | |
|---|---|
| Holds information about the resource. | Project |
| | ResourceManager |
| | NonConsumableItem |
| | ConsumableItem |

| **NonConsumableItem** | |
|---|---|
| Represents the Resource that can’t be depleted. | Resource |
| | Project |
| | ResourceManager |
| | NonConsumableItem |
| | ConsumableItem |

| **ConsumableItem** | |
|---|---|
| Represents the Resource that can’t be depleted | Resource |
| | Project |
| | ResourceManager |
| | NonConsumableItem |
| | ConsumableItem |

| **Segment** | |
|---|---|
| Holds the information about the segment | Segment |
| | Project |
| | ResourceManager |
| | DateRange |
| | Milestone |
| | Task |

| **Task** | |
|---|---|
| Represents a task | Segment |
| | Project |
| | ResourceManager |
| | DateRange |
| | Milestone |
| | Task |

| **Milestone** | |
|---|---|
| Represents a milestone | Segment |
| | Project |
| | ResourceManager |
| | DateRange |
| | Milestone |

| | |
|---|---|
| | Task |

| DateRange | |
|---|---|
| Encapsulate the information about date | Segment |