Hug a Cactus Inc.

FreeTime
Software Architecture Document

Version 2.1

Revision History

Date	Version	Description	Author
7/March/2006	1.0	SAD Part One	Brian Johnson
15/March/2006	1.1	SAD Complete Document	Brian Johnson
25/April/2006	2.0	2nd Iteration Partial SAD	Brian Johnson
2/May/2006	2.1	2nd Iteration SAD	Brian Johnson

Table of Contents

1. Introduction 4

2. Architectural Goals and Constraints 4

3. Logical Architectural Representation 4

4. Domain Model 6

5. Design [Class] Model 7

6. Use-Case Views 8
   - 6.1 System Sequence Diagrams 8
     - 6.1.1 Add Item 8
     - 6.1.2 Manage Item 8
     - 6.1.3 Make Project 9
     - 6.1.4 Contextualize Item 9
     - 6.1.5 Search Item 10
     - 6.1.6 Display Calendar 10
   - 6.2 Use-Case Realizations 11
     - 6.2.1 Add Item 11
     - 6.2.2 Manage Item 11
     - 6.2.3 Make Project 12
     - 6.2.4 Contextualize Item 13
     - 6.2.5 Search Item 13
     - 6.2.6 Display Calendar 13

Software Architecture Document

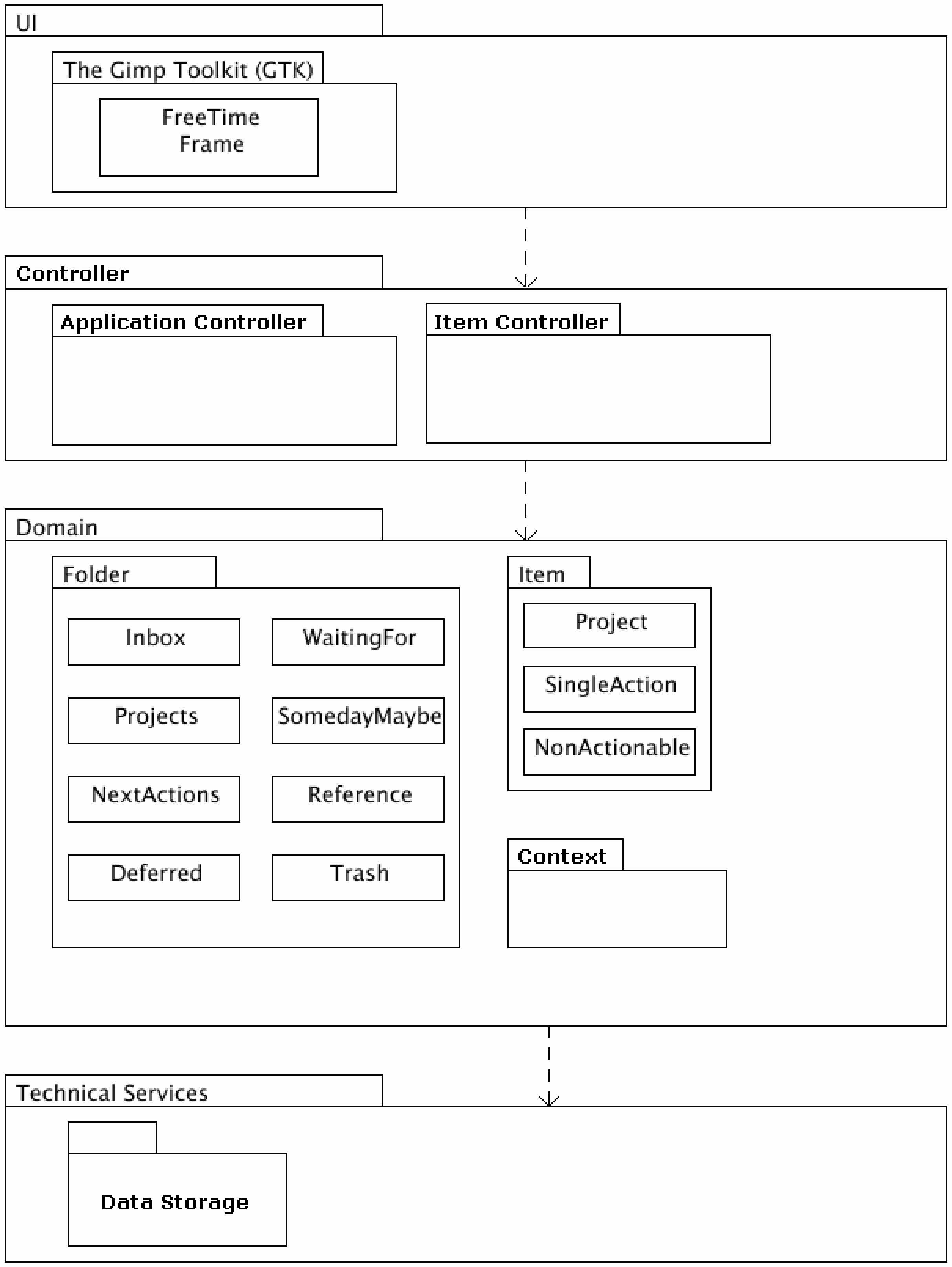
1. Introduction

The purpose of the Software Architecture Document is to give an overview of the FreeTime time management application. The scope of the Software Architecture Document is the FreeTime software. See the glossary of the FreeTime Vision document for definitions, acronyms, and abbreviations. The rest of this document discusses architectural aspects of the FreeTime software: the architectural goals and constraints, the logical representation, the domain model, the design class model, and use case views.

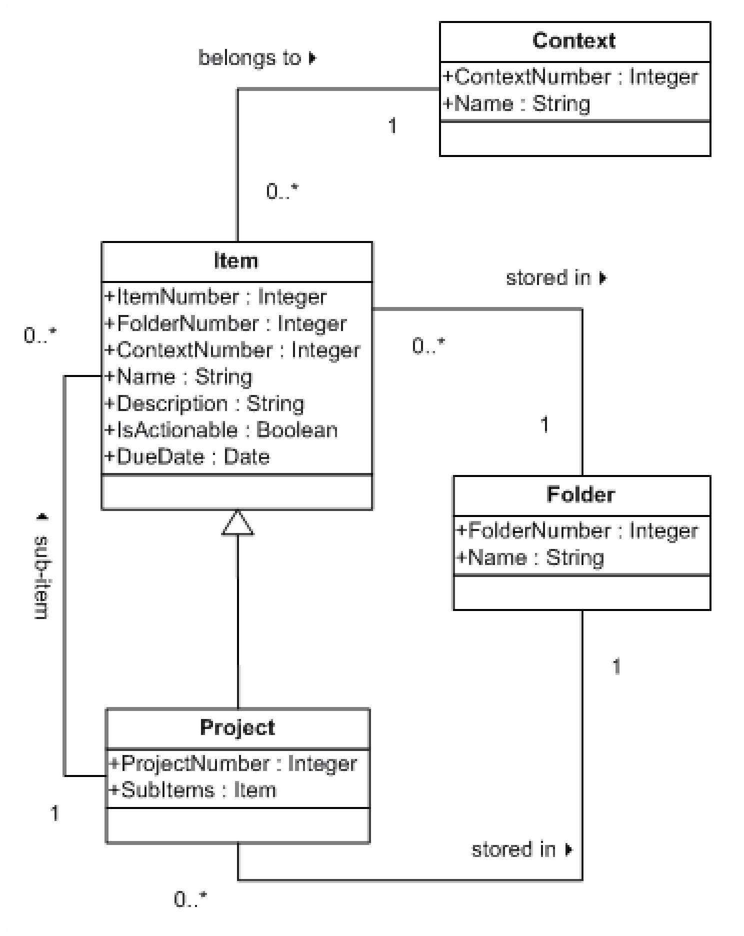
2. Architectural Goals and Constraints

FreeTime will be highly accessible and customizable. To these ends, the Ruby programming language and the GTK GUI toolkit have been chosen. FreeTime will run on Microsoft Windows, GNU/Linux, and MacOS X, as well as various handhelds such as the Nokia 770. Ruby's excellent community support has a number of benefits. One is that the FreeTime developers can be even more productive in their favorite development environments, such as Emacs and Eclipse. Another benefit is a diverse community of developers and business partners that will allow more flexible product distribution than competing products.

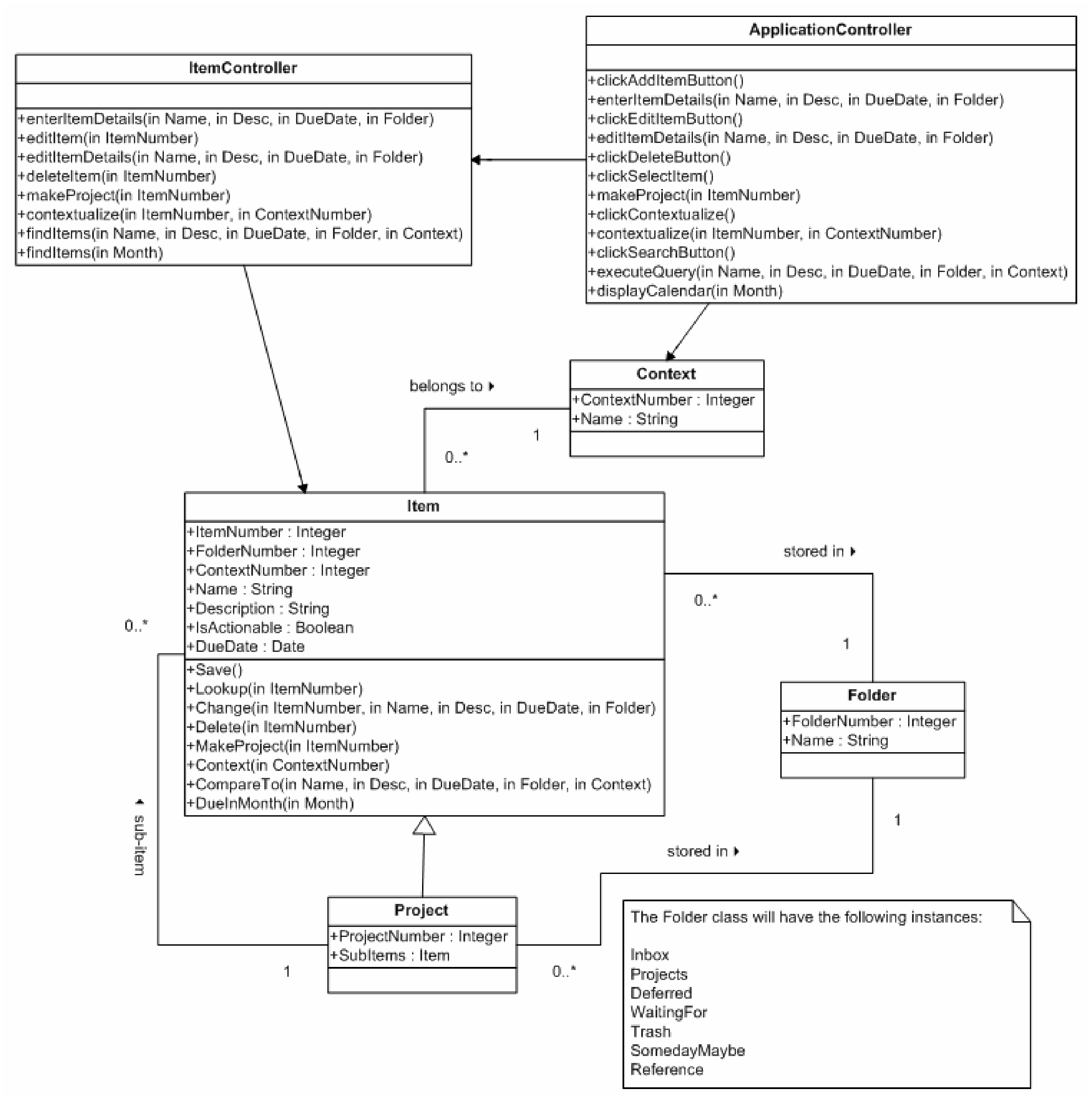
3. Logical Architectural Representation



4. Domain Model



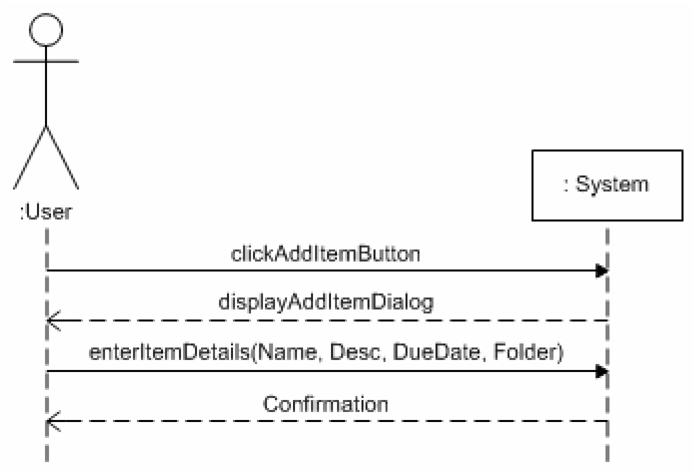
5. Design [Class] Model


ItemController
+enterItemDetails(in Name, in Desc, in DueDate, in Folder)
+editItem(in ItemNumber)
+editItemDetails(in Name, in Desc, in DueDate, in Folder)
+deleteItem(in ItemNumber)
+makeProject(in ItemNumber)
+contextualize(in ItemNumber, in ContextNumber)
+findItems(in Name, in Desc, in DueDate, in Folder, in Context)
+findItems(in Month)
ApplicationController
+clickAddItemButton()
+enterItemDetails(in Name, in Desc, in DueDate, in Folder)
+clickEditItemButton()
+editItemDetails(in Name, in Desc, in DueDate, in Folder)
+clickDeleteButton()
+clickSelectItem()
+makeProject(in ItemNumber)
+clickContextualize()
+contextualize(in ItemNumber, in ContextNumber)
+clickSearchButton()
+executeQuery(in Name, in Desc, in DueDate, in Folder, in Context)
+displayCalendar(in Month)
Context
+ContextNumber : Integer
+Name : String
belongs to
1
0..*
Item
+ItemNumber : Integer
+FolderNumber : Integer
+ContextNumber : Integer
+Name : String
+Description : String
+IsActionable : Boolean
+DueDate : Date
+Save()
+Lookup(in ItemNumber)
+Change(in ItemNumber, in Name, in Desc, in DueDate, in Folder)
+Delete(in ItemNumber)
+MakeProject(in ItemNumber)
+Context(in ContextNumber)
+CompareTo(in Name, in Desc, in DueDate, in Folder, in Context)
+DueInMonth(in Month)
stored in
0..*
1
Folder
+FolderNumber : Integer
+Name : String
0..*
sub-item
1
stored in
Project
+ProjectNumber : Integer
+SubItems : Item
1
0..*
The Folder class will have the following instances:
Inbox
Projects
Deferred
WaitingFor
Trash
SomedayMaybe
Reference

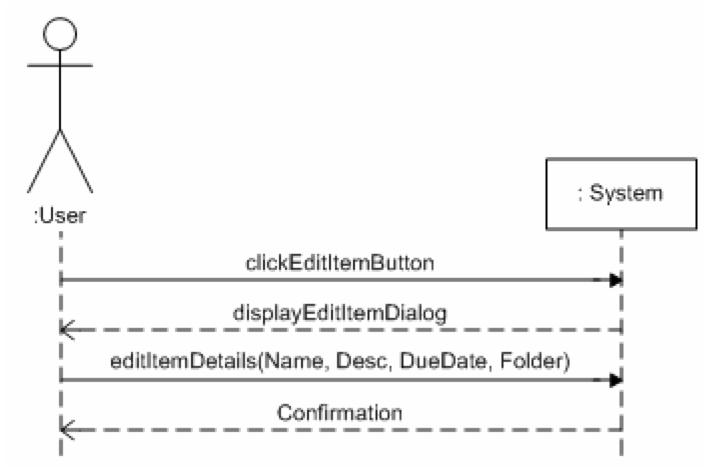
6. Use-Case Views

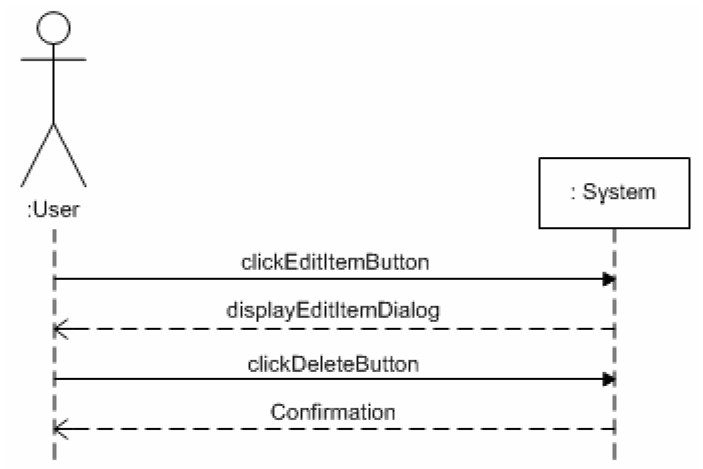
6.1 System Sequence Diagrams

6.1.1 Add Item

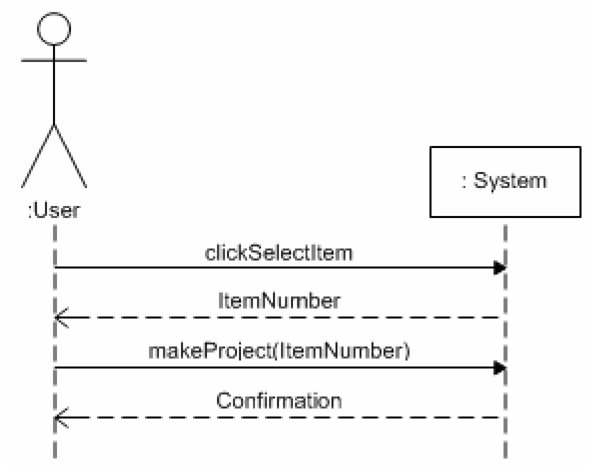


6.1.2 Manage Item

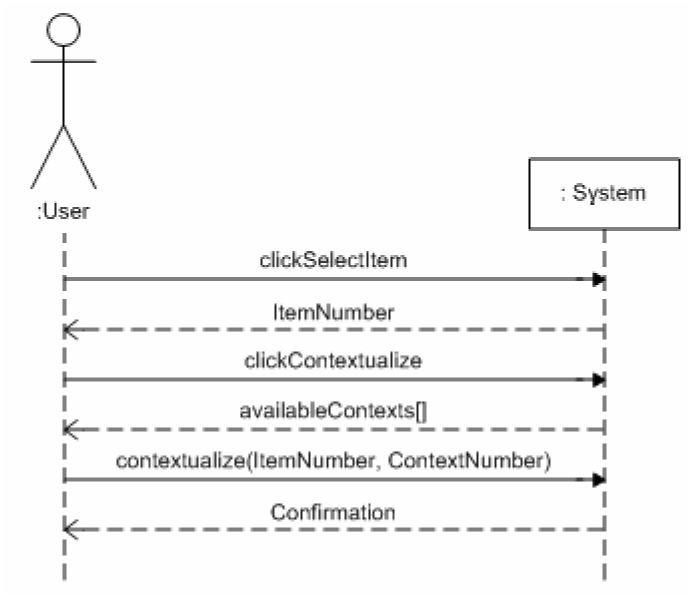




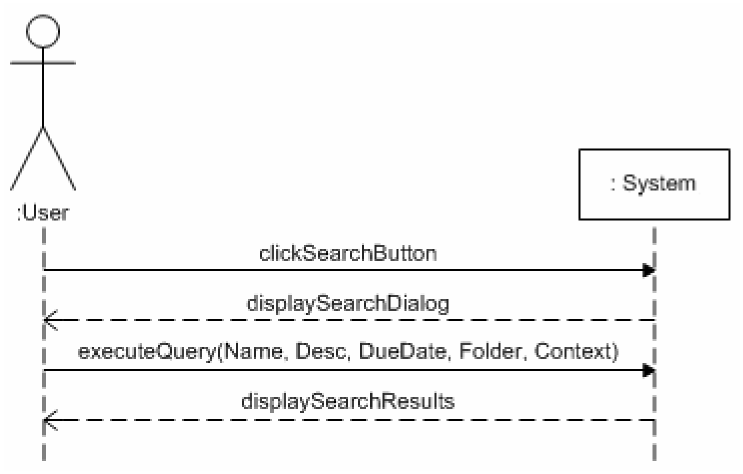
6.1.3 Make Project



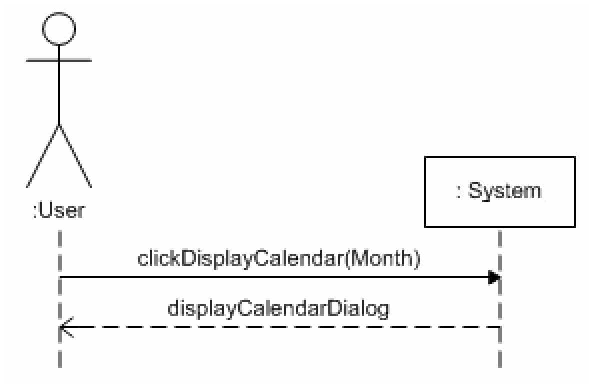
6.1.4 Contextualize Item



6.1.5 Search Item

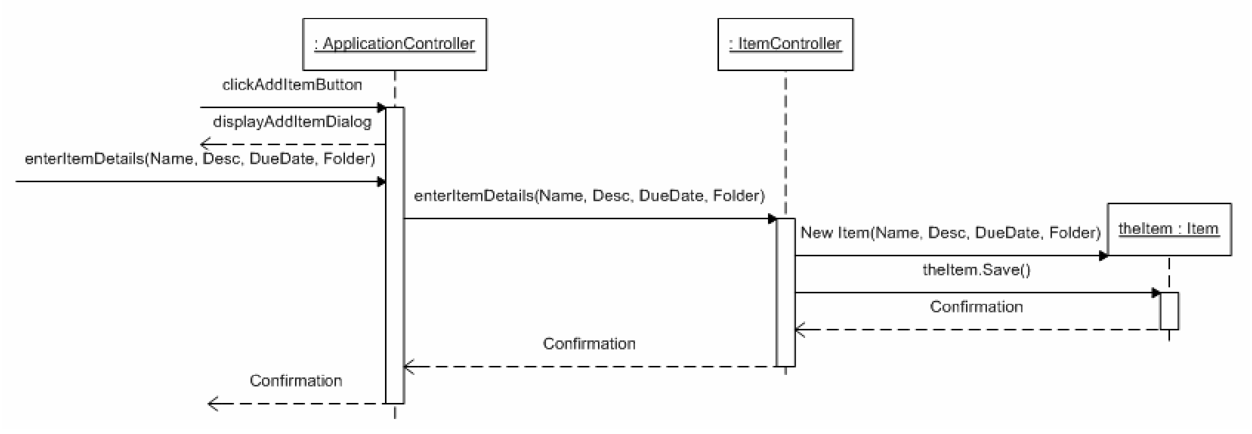


6.1.6 Display Calendar

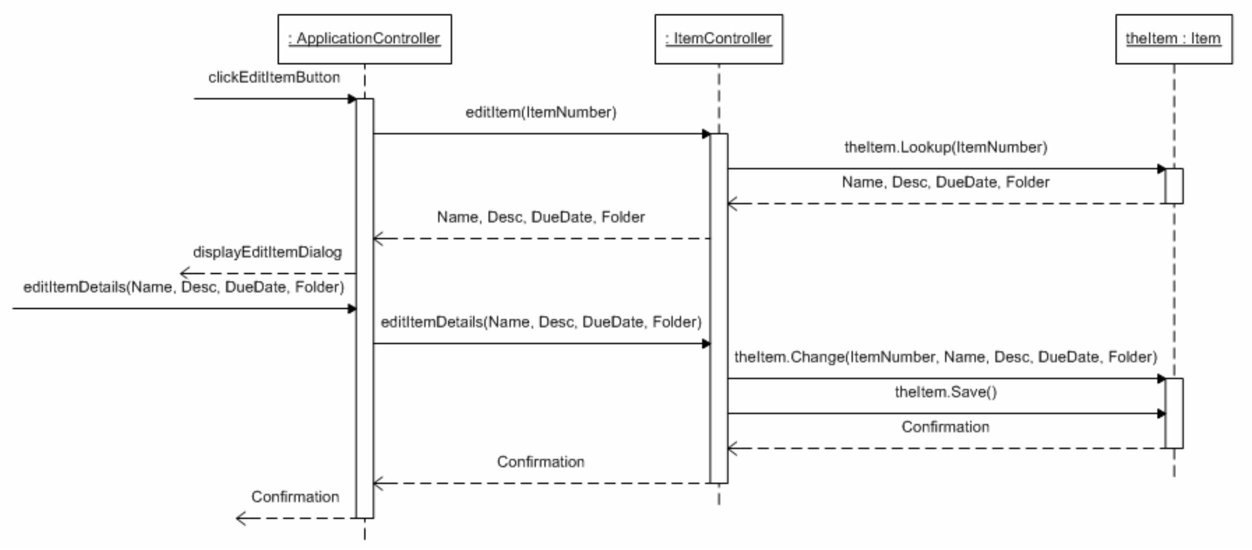


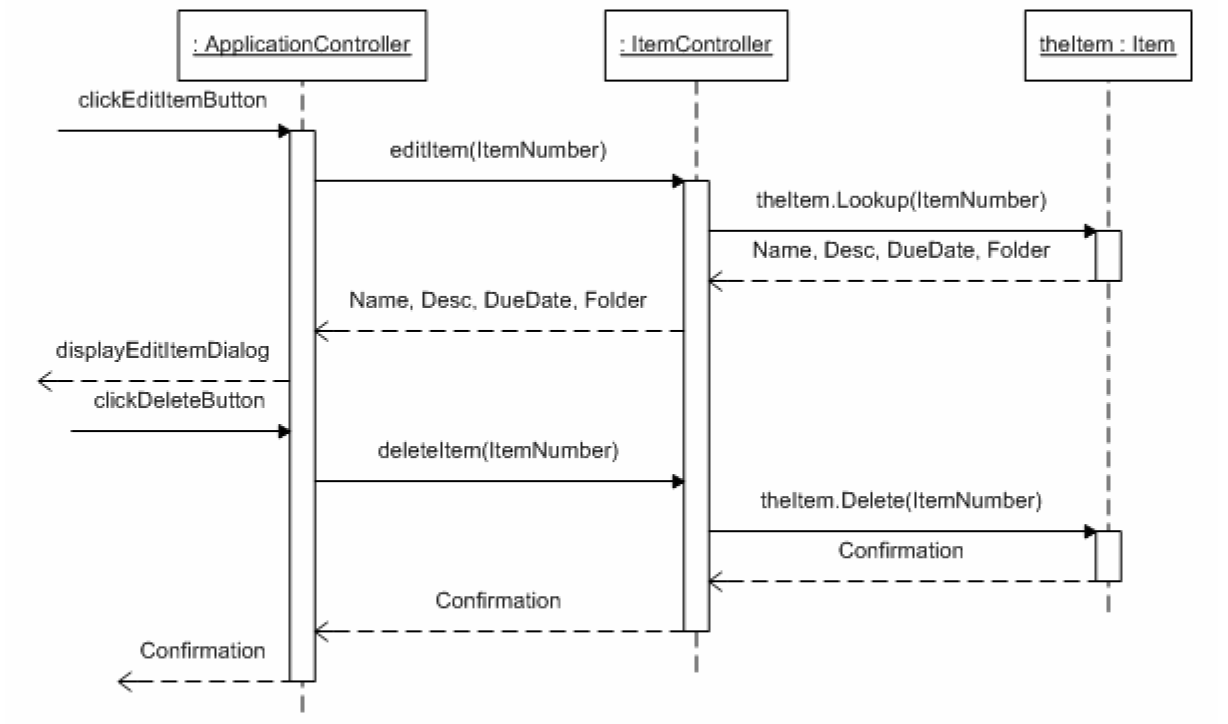
6.2 Use-Case Realizations

6.2.1 Add Item

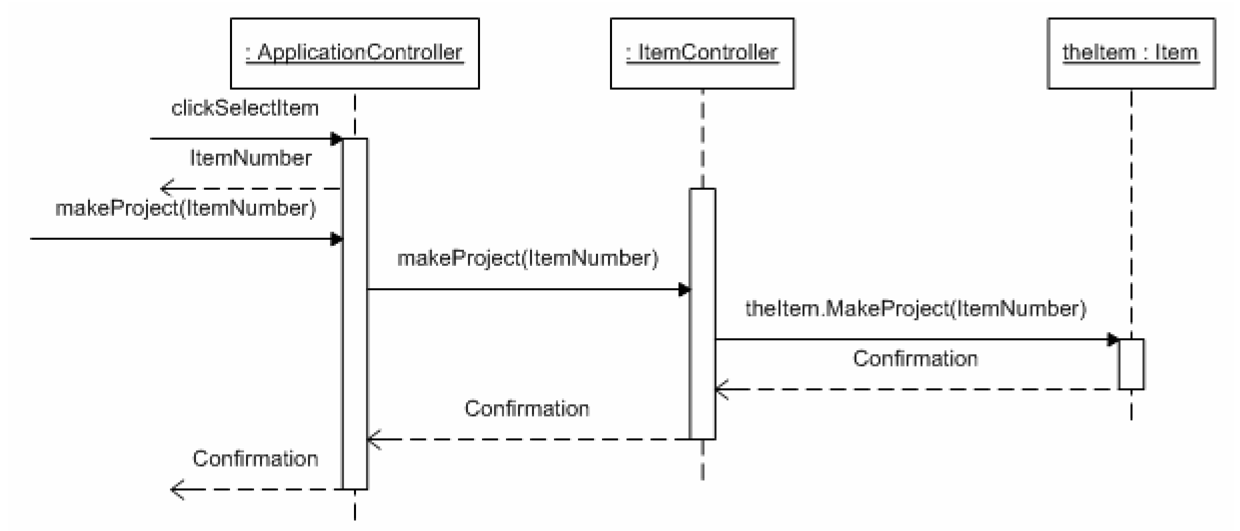


6.2.2 Manage Item

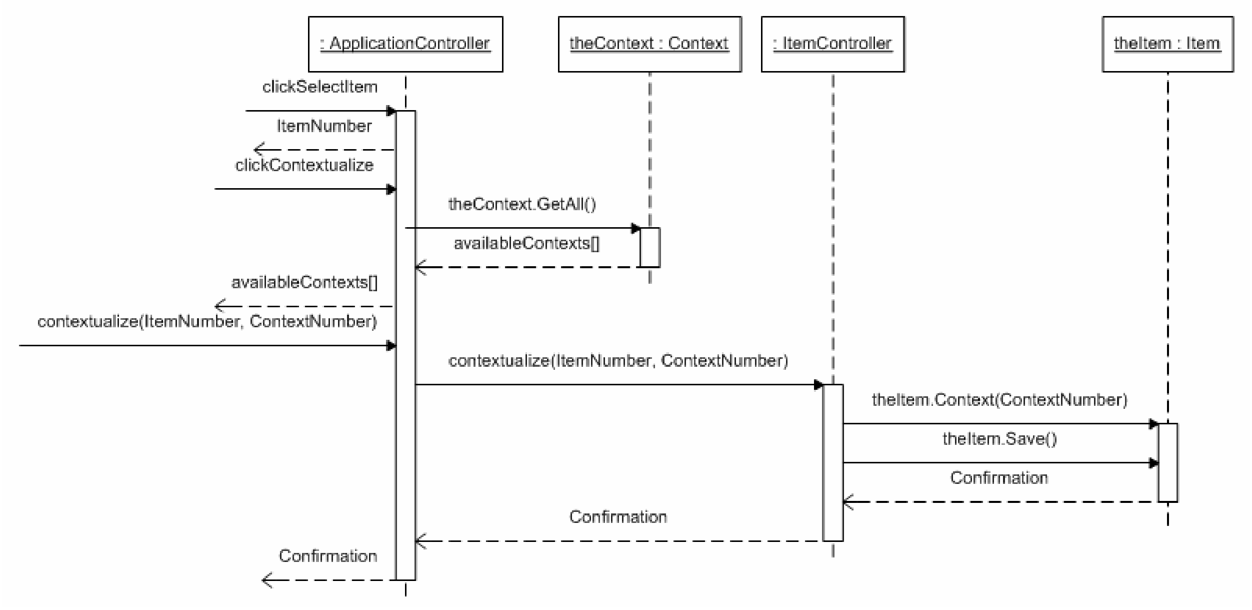

: ApplicationController
: ItemController
theItem : Item
clickEditItemButton
editItem(ItemNumber)
theItem.Lookup(ItemNumber)
Name, Desc, DueDate, Folder
Name, Desc, DueDate, Folder
displayEditItemDialog
editItemDetails(Name, Desc, DueDate, Folder)
editItemDetails(Name, Desc, DueDate, Folder)
theItem.Change(ItemNumber, Name, Desc, DueDate, Folder)
theItem.Save()
Confirmation
Confirmation
Confirmation



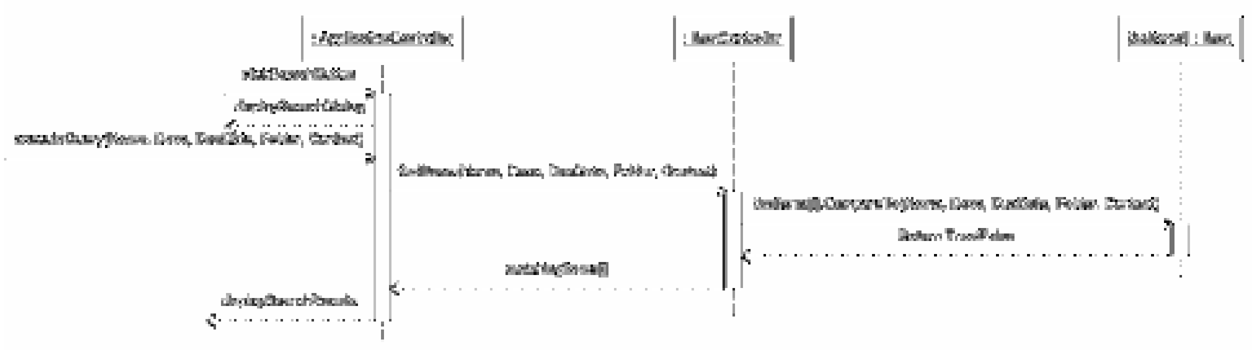
6.2.3 Make Project



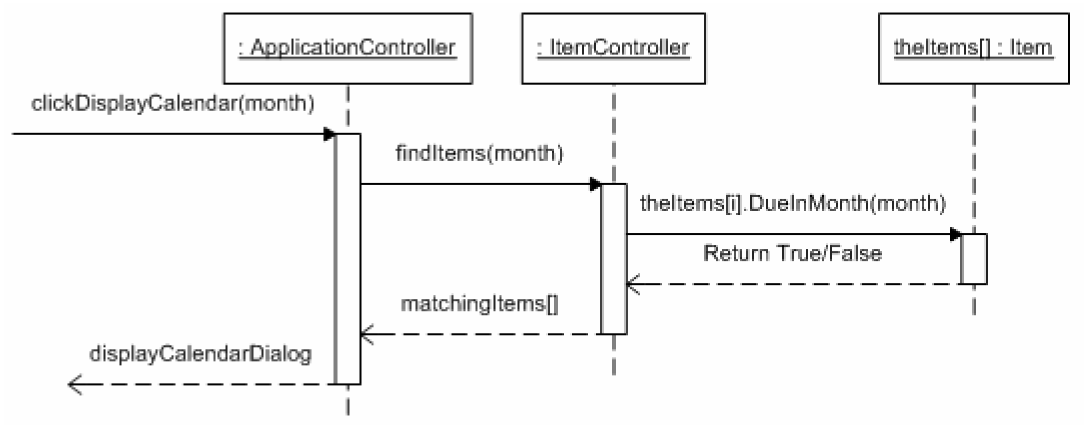
6.2.4 Contextualize Item



6.2.5 Search Item


: ApplicationController
: ItemController
theItems[] : Item
clickSearchButton
displaySearchDialog
executeQuery(Name, Date, DueDate, Folder, Context)
findItems(Name, Date, DueDate, Folder, Context)
theItems[i].CompareTo(Name, Date, DueDate, Folder, Context)
Return True/False
matchingItems[]
displaySearchResults

6.2.6 Display Calendar


: ApplicationController
: ItemController
theItems[] : Item
clickDisplayCalendar(month)
findItems(month)
theItems[i].DueInMonth(month)
Return True/False
matchingItems[]
displayCalendarDialog