

SOEN-6471

BORG CALENDAR

REPORT

SUBMITTED BY: SUBMITTED TO:

Anjaneyulu Bodepudei (5973775) Assi Prof.Peter C Rigby

Manouchehr Azizi (5232287)

Viet Hung Nguyen (9816240)

Hamid Shahrestani (9729747)

Table of Contents

|  |
| --- |
| **1. INTRODUCTION** |
| **2. PROJECT DESCRIPTION………………………………………………………………….** |
| **3. PROJECT SIZE AND SCOPE …………………………………………………...**  **4. GROUP MEMBERS………………………………………………………………………….**  **5. PERSONAS……………………………………………………………………………………**  **5.1 PRODUCT BACKGROUND……………………………………………………………**  **5.2 HELFUL PERSONA…………………………………………………………………….**  **6. STAKEHOLDERS……………………………………………………………………………**  **7. ACTORS……………………………………………………………………………………….**  **8. USECASES……………………………………………………………………………………..**  **9. SUBSTANTIAL PATTERN RECOGNITION……………………………….....................**  **9.1 MVC PATTERN (**VIET HUNG NGUYEN**)………………………………………………..**  **9.2 COMPOSTE PATTERN (**ANJANEYULU BODEPUDI**)………………………………………………**  **9.3 OBSERVER (LISTENER) PATTERN (**MANOCHERHR AZIZI**)…………......................**  **9.4 FACTORY PATTERN (**HAMID SHAHRESTANI)……………………………………….  **10. BORG UML MODELING …………………………………………………………………..**  **10.1 BORG CLASS DIAGRAM……………………………………………………………….**  **10.2 BORG CONCEPTUAL DIAGRAM…………………………………………………….**  **10.3 REFACTORING AND CODE SMELLS………………………………………………….**  **10.4 IMPLEMENT REFACTORING………………………………………………………….**  **10.5 BORG CALANDER PATCHSET…………………………………………………………**  **10.5.1 PATCHSET 1……………………………………………………………………….**  **10.5.2 PATCHSET 2……………………………………………………………………….**  **10.5.3 PATCHSET 3………………………………………………………………………**  **10.5.4 PATCHSET 4……………………………………………………………………….**  **10.5.5 PATCHSET 5……………………………………………………………………….**  **11. CONCLUSION………………………………………………………………………**  **APPENDIX 1………………………………………………………………………………………….**  **REFERECES…………………………………………………………………………..**   1. **INTRODUCTION**   BORG Calendar, a program that combines a calendar and a task tracking system. The calendar provides day, week, month and year views, a very useful address book, to do list, memos, checklists and allows users to add single or repeating appointments containing any relevant information and enhanced with email and popup reminders. The task tracker can be used to keep track of more complex tasks that run through various states.   1. **PROJECT DESCRIPTION**   BORG is personal organizer software. We use it to manage our everyday appointments, tasks, to-dos, checklists, memos. It provides a calendar so that we can look at our tasks in day, week, month or year views. Besides, it has project management feature so that we can manage our projects.  **We are interested in it because:**   * **Its usefulness**: everyone who have a family, a job and have to go to university as well would definitely agree with this. * **At our first look**, the code seems to be well organized (e.g. with different packages for different layers such as ui, model, db…). So although it is developed by only one author, we think that the author is an experienced developer and we would learn something about the architecture of the software.   BORG is in 1.7.7 version so it should be rather stable. The very first version was in 2003 and the last update was in January 2013. We saw 218 downloads this week and around 30 user reviews so we think it’s not a dead project but quite active.   1. **PROJECT SIZE AND SCOPE**   The whole project has 28K LOC   * Model package has 7K LOC * UI package has 20K LOC   Although it has 28K LOC, UI package takes the majority and normally if the UI is not well organized it often has many boilerplate code. So we think that in general the project is not so large for a term project.   1. **PROJECT GROUP MEMBERS**   **ANJANEYULU BODEPUDI**   * I have good experience in java programming and analytical skills to develop application software systems. I have done two academic project which involved java , J2ee and some advanced technologies. I would like to take refactoring analysis and problem analytical method to solve the requirements of this project. Based on previous of my project I have had some knowledge of the UML and Design models views. So I hope I can give my best to do the UML modeling design to accomplish the requirements of our milestones.   **MANOUCHEHR AZIZI**   * I had some projects in Concordia University with java (Eclipse , Net Beans ). I wrote some web service project in java for other courses. I have some experience in .NET programming (C# , Silverlight). I will try to review BORG to find the refractory technique which is proper for it.   **VIET HUNG**   * I have been working on Java quite a while. Currently I am working web application project with different separated layers. At my first look, I think BORG project is also organized in layers and I think I can extract the architecture of BORG. Besides, refactoring is my interest as well. I will try to scrutinize BORG to see how we can improve it.   **HAMID SHAHRESTANI MEHR**   * The projects I’ve done during the past courses mainly involved Java programming. We created a Game during Advanced Java programming that involved heavy use of design patterns like MVC, Observer, Singleton, etc. For Software Quality Assurance we created a web site to do online banking operations using Asp.net MVC and related technologies, and we tested it. We also created a website to do supply management using web services in Java and managed it using BPEL. I am also experienced in Asp.net and C#.  1. **PERSONAS**    1. **PRODUCT BACKGROUND**   In today’s hectic and stressful society it is becoming more and more common for people to have busy schedules and a lot of things to remember every day. Thus, the need for a mean to organize all this information is bigger than ever. BORG Calendar provides the system to achieve that with a highly versatile calendar which appears in every possible view (i.e. day, week, month and year views), equipped with features such as an appointment editor that enables users to add single or repeating appointments with the ability to incorporate any relevant information and enhanced with email and popup reminders, an address book that can store any kind of information (from names and addresses to phone numbers and birthdays), a to do list, memos and checklists as well as a powerful task tracking system which enables users to keep track of more complex tasks that run through various states. So it becomes quite obvious that BORG Calendar is the ideal application for anyone who wants to organize their everyday schedule in a more sufficient way. It is open source software that is written in Java, so is platform independent as long as JVM is installed on the client machine.   * 1. **HELFUL PERSONA**   The primary beneficiaries of the BORG calendar are people with many appointments and tasks to manage. We have here the case of Edward Smith which is a technology expert for IBM in London. He has appointments everyday with clients to discuss their needs and assess their technology equipment. It is very important that he has access to his calendar everywhere in any platform as he is continuously on the road and he sometimes switch from Windows to Linux. He creates project for each client and generally after meeting them he establishes a list of tasks to address the issues. Edward is a typical user of the BORG Calendar and his ability to better manage his schedule and project tasks will reflect the level of quality of the software. Their inputs and feedback would help to direct the development for the future releases of the software.   1. **STAKEHOLDERS**   **End users/Desktop**  Who only need to have some basic knowledge in the use of computers and are most likely to use the basic features of the application, without getting involved with the more complex ones (e.g. the task/project tracking system), and the default database which requires no special setup. Of course, the program might be more useful to users with a busy daily schedule or those working on a computer quite often.  **Advanced users**  Who have a better knowledge in the use of computers and are probably interested in exploring all of the program’s capabilities and maybe wish to set up a different type of database.  **Developers/Testers**  Who have a deeper knowledge of programming in Java and wish to improve/extend the application and or only test the software.  **Others**  Who wish to contribute to the project in other ways such as getting involved with the program’s documentation or translations? (This user class could be merged with the previous one)  **Project Owner**  He is responsible to make major changes to the project. He can shut it down and stops any support for it. He can initiate a premium version of the software.   1. **ACTORS**   **PRIMARY ACTORS/END USER**   * Intended to use the BORG calendar Features.   **SECONDARY ACTORS**  **Physical System**   * Service provide by the system to operate the BORG calendar   **Organization**   * Maintain/version controlling of the BORG calendar System  1. **USECASES**   **ADD APPOINTMENT**   * The user can create a new appointment/to-do item in the Calendar for a specific date. The appointment can occur once or more than once on a timely manner   **SELECT VIEW**   * The user can open different views in the Calendar as tabs. The available views are : * Month View, Week View, Day View, Year View, To Do List, Address Book, Tasks, Memos, Checklists, Search |

**MANAGE TASKS**

* Users can create/edit/delete tasks. Tasks are complex activities that have start and end dates, and are created in an initial state (default is OPEN) and progress through a user configurable tree of states to reach an end state (default is CLOSED).

**MANAGE PROJECTS**

* Like tasks, users can create/edit/delete a project. A Project is simply a way to group related projects and tasks. A Project can set an overall due date for all child projects and tasks.
* Each project can have any number of child projects and tasks. All child tasks must have due dates on or before the project due date. A project cannot be closed unless all of its child tasks are closed.

**MANAGE SUBTASKS**

* Subtasks are single units of work that do not progress through a set of states. They are either completed or not-completed. Users can create them, edit or delete them.

**MANAGE CONTACTS**

* User can add a contact to the Address Book. Address Book holds addresses and other personal information. There is a birthday field for each record in the address book. If a date is entered in this field, the person's birthday will appear on the calendar on the appropriate day each year. They can change the values for a contact later or even remove it.

**MANAGE MEMOS**

* The memo tab on the main window allows user to create and edit plain text memos.

**CHANGE OPTIONS**

* Options can be changed from the ***Options*** menu. User can change the following settings: Appearance, Fonts, Email Parameters, Popup Reminders, User color Scheme, Task Options, Encryption, Startup Views, Database Information, Miscellaneous

**SEARCH**

* The search option from the *Action* menu will bring up a search window. There are a number of search criteria to choose from. User can also change the category of multiple appointments at the same time or bulk-delete several items from this view.

**PRINT**

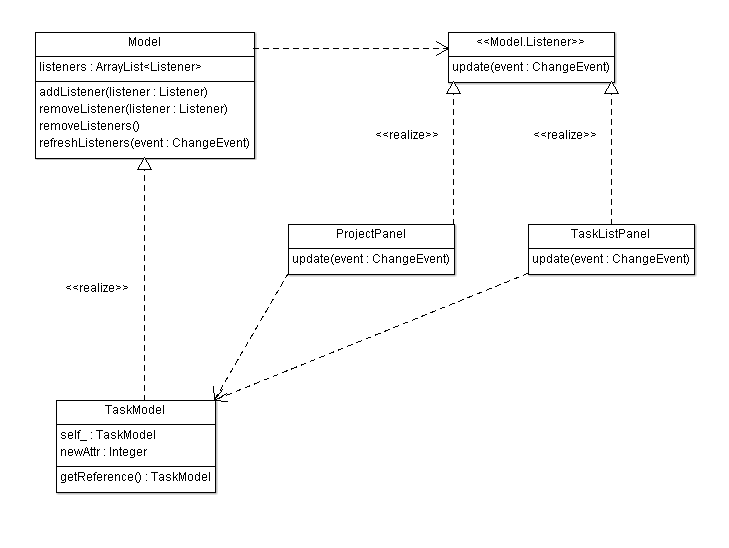
* Users can select to print any of the views of the software. They can select various options for the printing

**IMPORT/EXPORT TO XML**

* BORG can export/import its data to/from XML. The import/export actions can be found on the month view Action menu. Import/Export provides a good way to back up data in human readable/editable form. The auto back up feature also uses an XML file to back up the data in a ZIP file

1. **SUBSTANTIAL PATTERN RECOGNITION**
   1. **MVC PATTERN (VIET HUNG NGUYEN)**

MVC design pattern [3] is implemented in BORG to separate between models and UI. However it combines view and control into a UI class. A UI class get data from a model and display data to users. From user interaction, UI class manipulates its model. Then the model will then notify all listening UI about its changes.

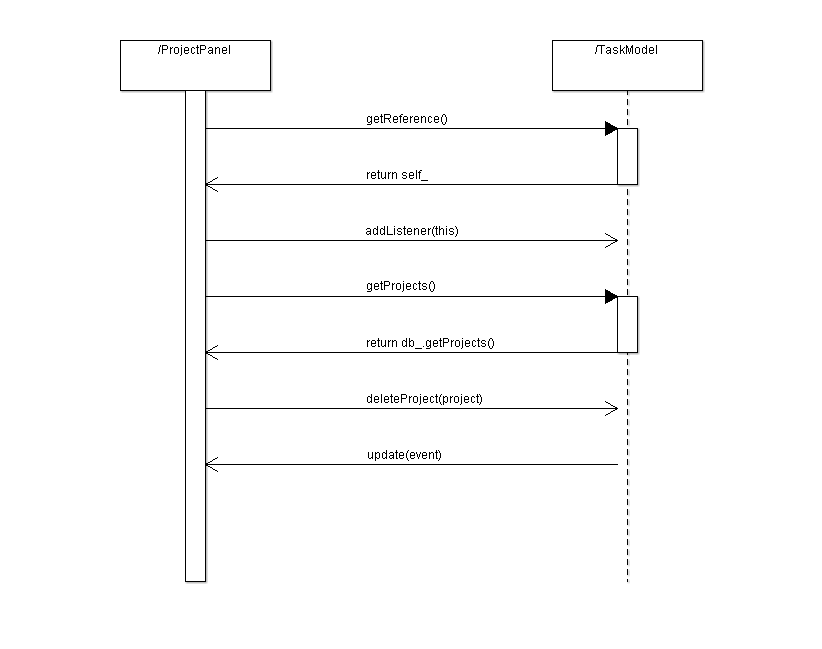


* Observer pattern is used to allow views getting notified by model changes [1].
* Singleton pattern is used to allow UI classes have access to models [2].
* The
* Way of applying MVC design pattern in BORG with combined view and control is similar to the Document View pattern in MFC framework from Microsoft [4].
* Using MV
* C in BORG allow separating between models and UI classes so that we can easily write unit tests for model classes and protect models from UI changes.
* In general, the separation between view and control allows having different views, for example supporting different environment such as desktop, web browsers, mobile browsers etc., with the same control logic and allows testing controller’s logic independent with views. In BORG, view and controller are combined. So testing control logic would not be easy for BORG.

**TOOLS FOR REVERSE ENGINEERING:**

* AgroUML: a tool to draw class diagrams
* Intellij Idea: an IDE with strong support for navigating around the code

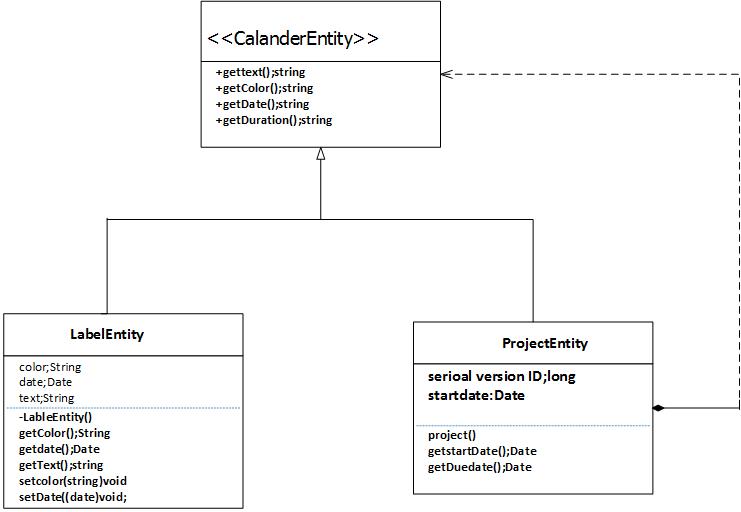
**SEQUENCE DIAGRAM**



* 1. **COMPOSTE (ANJANEYULU BODEPUDI)**

COMPOSITE PATTERN

Abstraction: Composite in Java: Model-configurable "Entities" of a Composite

In BORG calendar borg.model.entity package project class and Label Entity classes’ files are implements the << Calendar Entity>> interface. An entity is a lightweight persistence domain object. Typically an entity represents a table in a relational database, and each entity instance corresponds to a row in that table. The primary programming artifact of an entity is the entity class, although entities can use helper classes. The persistent state of an entity is represented either through persistent fields or persistent properties. These fields or properties use object/relational mapping annotations to map the entities and entity relationship. The below class are composite abstract. Its part whole relationship of the Calendar Entity and project Entity classes.s to the relational data in the underlying data store.

**TOOLS FOR REVERSE ENGINEERING:**

* AgroUML: a tool to draw class diagrams

Code snippets

**Package** net.sf.borg.model.entity;

* Interface that needs to be implemented by any entity that can appear on the Calendar.
* This interface would more properly belong in the UI package - but it would take some extra wrapper
* classes that would be a waste

**Code Reference to Mile stone 4**

* 1. **OBSERVER (LISTENER) PATTERN (**MANOCHERHR AZIZI**)**

In “net.sf.borg.common.profs “ class, Listener class is implemented which is responsible to keep list of listeners and Notify listeners of a pref change (following notifyListeners method).the point is that prefsChanged method is going to be override in all subscriber that we will explain in subscriber section.

PARTIAL CLASS DIAGRAM

**Publisher**

In “net.sf.borg.common.profs “ class, Listener class is implemented which is responsible to keep list of listeners and Notify listeners of a preferences change (following notifyListeners method).the point is that prefsChanged method is going to be override in all subscriber that we will explain in subscriber section.

**Remark:** prefsChanged method is going to be override to do proper action for each of listener member in case of preference changes.

**Notify**

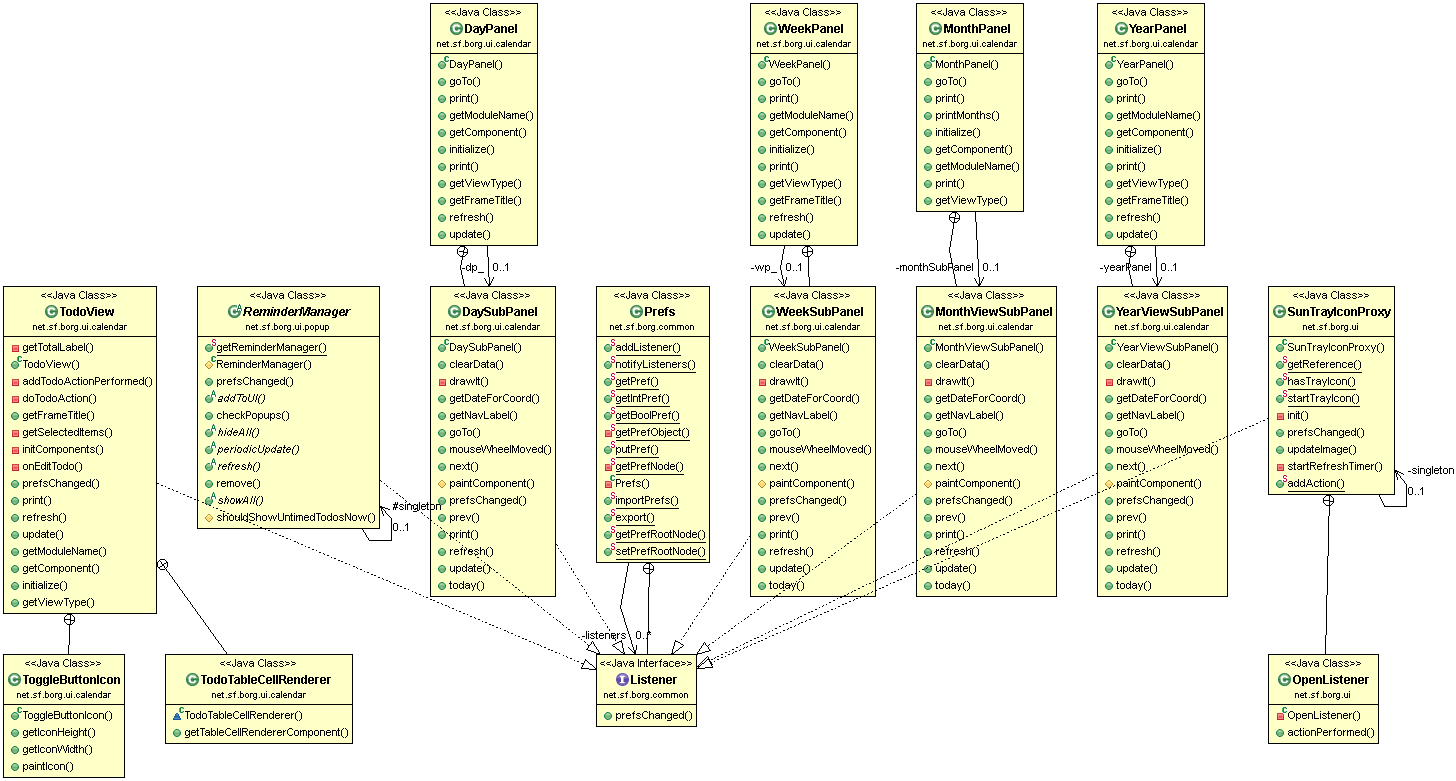
**“Prefs.*notifyListeners*();”** is used in two following class to apply proper action to Observer members:

**Subscriber**

Following class with “Prefs.*addListener*(**this**);”add themselves to Listener list:

* net.sf.borg.ui.calendar.DayPanel
* net.sf.borg.ui.calendar. MonthPanel
* net.sf.borg.ui.calendar. TodoView
* net.sf.borg.ui.calendar. WeekPanel
* net.sf.borg.ui.calendar. YearPanel
* net.sf.borg.ui.popup. ReminderManager
* net.sf.borg.ui.task. ProjectTreePanel
* net.sf.borg.ui. SunTrayIconProxy

**Remark**: in all of upcoming class, “prefsChanged” method is override to do proper action in case of prefrences change.

****

**Code reference to Mile Stone 4**

**9.4 FACTORY PATTERN (**HAMID SHAHRESTANI)

**Factory Pattern**

**Intro**:

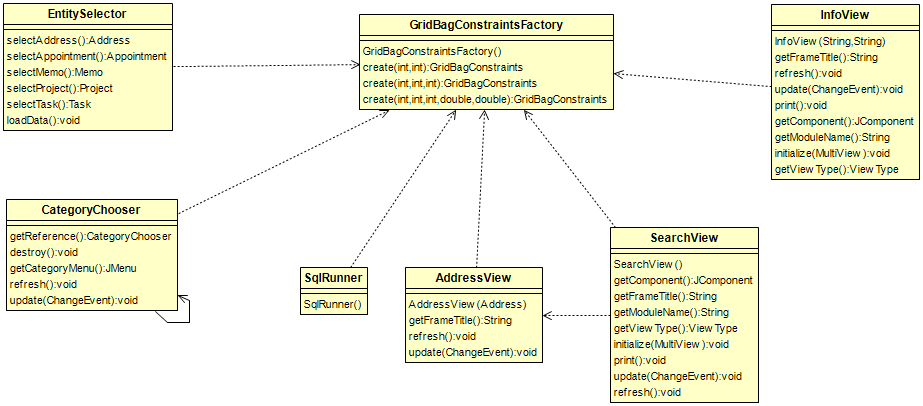
According to Wikipedia “Factory is a creational design pattern to implement the concept of factories and deals with the problem of creating objects (products) without specifying the exact class of object that will be created. The essence of this pattern is to "Define an interface for creating an object, but let the classes that implement the interface decide which class to instantiate. The Factory method lets a class defer instantiation to subclasses.”

In BorgCalendar, GridBagConstraints class is used numerously in many classes. The GridBagConstraints class specifies constraints for components that are laid out using the GridBagLayout class. To facilitate instantiation of objects from this class, a factory class is designed. This class is used 39 times within UI packages to instantiate GridBagConstraints objects. GridBagConstraints object can be created in several ways with different fields populated. Factory method encapsulates the process of creation and saves lots of time.

To capture the following interactions I used Object Aid UML tool.

**Class Diagram**

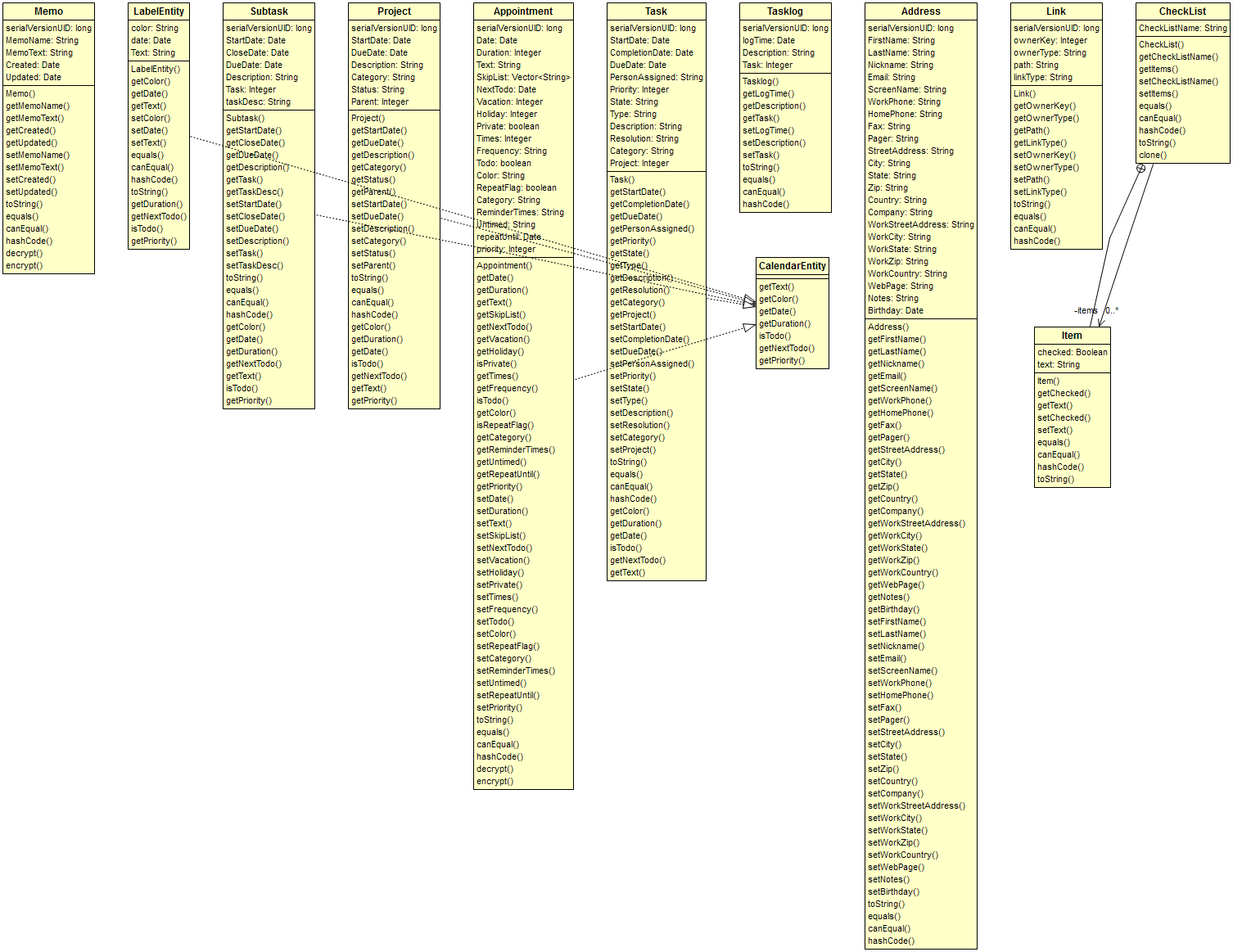
For example the picture below displays some of the classes that use this factory class



**Code reference to Mile Stone 4**

1. **.BORG UML MODELING**

**10.1 BORG CLASS DIAGRAM**



**COMPARIOSN OF CONCEPTUAL AND ACTULA CLASS DIAGRAM**

To create the Actual System Model we used a Reverse Engineering tool namely “Object Aid UML Diagram” which is an Eclipse Plugin. The source code contains 18 packages, but for simplicity we only examine the core entity classes contained within “net.sf.borg.model.entity” package. The main correspondence of our modeling is with the classes contained in this package. The other 17 packages are mostly taking care of UI tasks (10 packages), tools (3 packages), database tools (2 packages), model(1 package) and one controller class (pure fabrication) which is responsible to starting up the model and spawning various threads, including the main UI thread and various timer threads. It also handles shutdown. In total there are 148 classes of which 85 classes are UI classes.

There’s a close correspondence between the actual system classes and those of our conceptual model. Of 15 classes we captured in our model, 8 of them exist in the actual system model. These classes are:

|  |  |  |  |
| --- | --- | --- | --- |
| * CalendarEntity * Address * Project * Task * Memo * Checklist * Item * To-do |  |  |  |

There are 7 classes in the actual system model that are not included in our conceptual model and there are 5 classes in our conceptual model that are not part of the actual system model.

The following are the classes not included in our conceptual model:

**Option:**

* Each Borg Option instance holds a single row from the options table in the database. This is an entity we overlooked.

**Subtask:**

* It represent a task within another task. We figured in our design that we can model this entity with “Task” class. The use of a separate Subtask class makes the design more cohesive but instead there will be more coupling.

**Task log:**

* We didn’t really take into account any logging feature but rather captured the main core functions of the software.

**Link:**

* Any association between an entity and another entity will be represented with a Link object. This will increase coupling but might be more cohesive.

**Label Enti**ty:

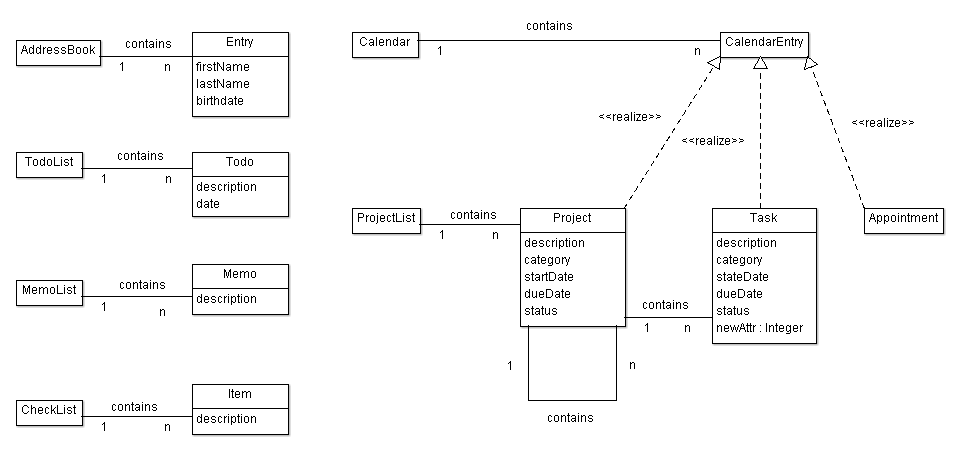
* According to the documentation this class is a non-persisted entity that is used when the model needs to package a transient, calculated entity for the UI, such as a calculated holiday or birthday based on the address book. We didn’t really predicted the design that far.

**Keyed Entity:**

* Abstract base class for all Entities that are keyed by a simple integer key. Again we considered all entities equal without defining any key for them.

**Encrypt able Entity:**

* Abstract base class for entities that can have encrypted fields. It is up to the entity specific Decrypt and Encrypt methods to determine which fields are to be encrypted. We overlooked the encryption feature.
* These extra classes are mainly not included in our design because we started from scratch and captured a simple basic set of entities based on Gall’s Law. This is a complex system that evolved for many years and these features are added in later versions
  1. **BORG CONCEPTUAL DIAGRAM**



There are 5 classes in our conceptual model that are not seen in the actual system model. These classes are:

* **Address Book,**
* **To-do List,**
* **Project List,**
* **Memo List:**

These classes wrap a collection object that holds a reference to the individual objects. E.g. Address, To-do, Project and Memo. We intended to present collection objects in their respective entity classes as opposed to using arrays or vectors which is the solution for the actual system design.

* **Calendar:**

This represents the calendar itself that is in association with calendar entity. In the actual system design this class is not present.

* 1. **REFACTORING AND CODE SMELLS**

1. Looking at TaskModel class we can see that it’s quite long: there is Large Class code smell here. Refactoring to fix this code smell in TaskModel:

* daysBetween(Date start, Date dd), daysLeft(Date dd) is not the responsibilities of TaskModel and can be moved out of it.
* We can refactor by creating a utility class like TaskUtil with two static methods: daysBetween() and daysLeft().

1. In AppointmentModel class , method do\_todo

**public** **void** do\_todo(**int** key, **boolean** del, Date date) **throws** Exception has two different polymorphic methods which are distinguished with “if then else”:

* Delete : delete the todo when all done
* repeatSet: date date of the repeat that is being marked as done. If null, then the next todo is the one. If set, then all todos up to and including the date are marked as done

We can Extract two classes and instead of “if then Else” we implement it with

Polymorphism (strategy):

<<Interface>> do to do

Delete

Repeat Set

1. In Day class, there’s an addToDay method that is quite long. We can reduce its size by introducing shorter methods within it. It also lacks enough comments, so by introducing self –explanatory methods, we make the code more comprehensible.

**private** **static** **void** addToDay(Day day, Collection<Integer> l, **int** year,

**int** month, **int** date) **throws** Exception

It can be shortened by introducing at least three short methods:

The top part of the method consists of the code to indicate whether a flag is public or private. We can introduce the method setAccessLevel (or something like that) to refactor this part of the code. This will increase the cohesion.

In the middle of the code there’s a very complicated method to indicate whether the loop should be continued or not. This is a complicated logic, because of the use of Boolean flags. We can simplify it by introducing another method and creating a class that contains the access level flags. Finally the bottom part can be shortened by adding three more methods.

* + - addAppointmentToDay(Appoin Appointment apt)
    - setVacation(Appointment apt)
    - setHoliday(Appointment apt)

1. In TaskTypes class, we have a toXml method that is reducing the cohesion of this class.

We move this method to a new class called TaskTypeSerializer and delegate this task to this class. We just need to introduce an instance of this class in our TaskTypes class and call its toXml method.

1. getInfo method in TaskModel class is not making the class cohesive. We move it to another class called TaskModelInformation to make the class more cohesive. We delegate the task to an instance of the TaskModelInformation class that we introduce in TaskModel class.
2. There are some database related methods in TaskModel class that are making this class too big. We need to move all these methods to another class and delegate all the responsibilities of these methods to an instance of the newly created class. We can call it TaskModelDB and move the following methods to that class:
   * + beginTransaction
     + commitTransaction
     + rollbackTransaction
     + addLog
     + saveLog
3. There’s an importXml method in TaskModel class that is not cohesive at all. We want to move it to a TaskModelXmlImporter class and delegate the responsibility of importing xml to this class. Beside lack of cohesive structure, the method is too long and it is using methods that take care of database related tasks. We first need to shorten the method by introducing shorter methods, and delegating the database related tasks to some other objects and classes. We can introduce the following methods:
   1. Unmarshal : it encapsulates the top part of the importXml method.
   2. executeSql: it wraps the following 5 lines that does database related tasks. Next, we move these tasks, to a different class that only does database related tasks to increase cohesion.
   3. handleOldImports: The middle part of the method can be wrapped in this method.
   4. importIntoEmptyDb: the last 40 lines of the method can be wrapped in this method.

|  |
| --- |
| * 1. **IMPLEMENT REFACTORING**   In TaskModel class, we notice that its two methods: daysBetween() and daysLeft() is not responsible of TaskModel itself, and there is another class, DateUtil, that seems to be more suitable to contain those methods. So we decide to do this refactoring: moving daysBetween() and daysLeft() from TaskModel to DateUtil.  The patch file is patchset.patch on github at https://github.com/nvhung/soen6471/  **PATCHSET 0/5**:  Move TaskModel.daysBetween() and TaskModel.daysLeft() to DateUtil  Commit 19df2def2e13630945b07a833c084e63614335ed  There are 5 steps to do so:  - Step 1: Copy TaskModel.daysBetween() and TaskModel.daysLeft() to DateUtil  - Step 2: Replace usages of TaskModel.daysBetween() by DateUtil.daysBetween()  - Step 3: Remove TaskModel.daysBetween()  - Step 4: Replace usages of TaskModel.daysLeft() by DateUtil.daysLeft()  - Step 5: Remove TaskModel.daysLeft()  **PATCHSET 1/5:**  Copy TaskModel.daysBetween() and TaskModel.daysLeft() to DateUtil  Commit 88cab62dd16b688c814fb2836082e2acf2b4ebd4  diff --git a/Source  **PATCHSET 2/5:**  Replace usages of TaskModel.daysBetween() by DateUtil.daysBetween()  Commit d6b407565898daeec389e3166d9d1207ee0175ea  diff --git a/Source  **PATCHSET 3/5:**  Remove TaskModel.daysBetween()  Commit 25b093018d557c48b226798ee6ee1493db4580d2  diff --git a/Source  **PATCHSET 4/5:**  Replace usages of TaskModel.daysLeft() by DateUtil.daysLeft()  Commit cb79d02e2c783245f9e77b14e179b59a2259bbd6  diff --git a/Source  **PATCHSET 5/5:**  Remove TaskModel.daysLeft()  Commit 22bbbbf65d6335ca56f750c95529e74b6af93c29  diff --git a/Source |
| **CONCLUSION**  Coming to end, we explore different software engineering activities by using BORG calendar. BORG calendar was developed by using MVC frame work. Identifying the substantial patterns are very challenging tasks. Finding the code smells and refactoring are also challenging task in BORG calendar |
| **APPENDIX 1**  CODE REFRENCE TO MILE STONE 4  **REFERENCES**  <http://sourcemaking.com/design_patterns/observer>  <http://sourcemaking.com/design_patterns/singleton>  <http://en.wikipedia.org/wiki/Model-viewcontroller>  <http://msdn.microsoft.com/en-ca/library/k9kb0kba.aspx>  <http://java.uom.gr/~nikos/pattern-detection.html>  <http://sourcemaking.com/design_patterns/composite> |
|  |