BORG CALENDAR – M1

Group Members

* Anjaneyulu Bodepudei (5973775)
* Manouchehr Azizi (5232287)
* Hermann Sonfock (5986052)
* Viet Hung Nguyen (9816240)
* Hamid Shahrestani (9729747)

Date: Friday, Jan 25th

We have compiled and assessed this project a believe it to be of a reasonable size for a term project.

# 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.

# 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.

# 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  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.

**Hermann Sonfack**

I have a good experience in software acquisition and deployment. I used Java in many projects and I think I can contribute in understanding the architecture and represent it. In a previous course we actually developed  a “task management” software which is very similar to this one using the MVC model for the design and some design patterns. We also had to do the architecture diagram in UML so I will be able to use the skills acquired there and do some code review.

**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 webservices in Java and managed it using BPEL. I am also experienced in Asp.net and C#.

BORG CALANEDER – M2

Group Members:

* Anjaneyulu Bodepudi (5973775)
* Manouchehr Azizi (5232287)
* Hermann Sonfack (5986052)
* Viet Hung Nguyen (9816240)
* Hamid Shahrestani (9729747)

# PERSONAS

## 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.

## Helpful 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.

## 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.

## 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

# Use Cases

|  |
| --- |
| **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. |

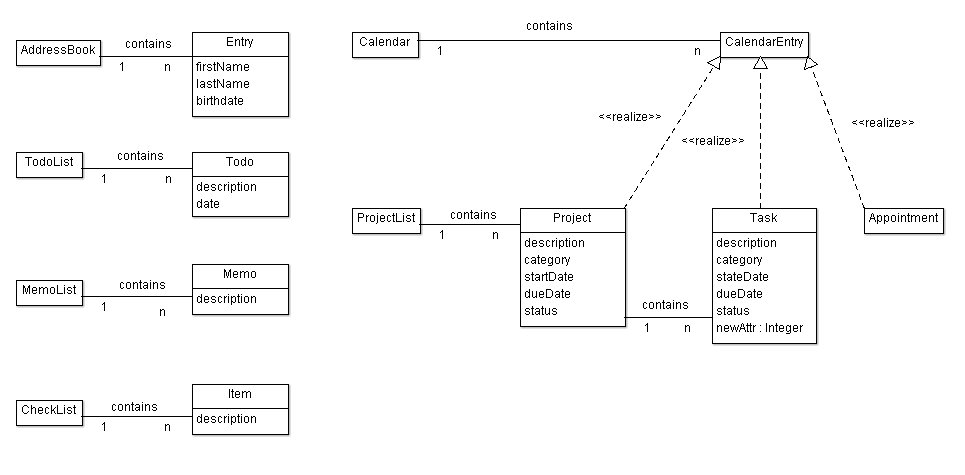
|  |
| --- |
| **Change Options** |
| Options can be changed from the ***Options*** menu. User can change the following settings:  Appearance, Fonts, Email Parameters, Popup Reminders, User color Scheme, Task Options, Encryption, Startup Views, Database Information, Miscellaneous |

|  |
| --- |
| **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. |

# Conceptual Model



BORG CALENDAR – M3

Student Name and Number

* Hamid Shahrestani (9729747)
* Anjaneyulu Bodepudi (5973775)
* Manouchehr Azizi (5232287)
* Viet Hung Nguyen (9816240)

# Description of the software architecture

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” and a few classes within “net.sf.borg.model.entity” package. The main correspondence of our modeling is with the classes contained in these packages. The other 16 packages are mostly taking care of UI tasks (10 packages), tools (3 packages), database tools (2 packages), 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.

Examining our use cases closely, there are several main entities in the calendar that can be added, removed, edited, etc. These entities are: Project, Task, Subtask, memo and ToDo.

Within our entities package we have data classes that hold data without any functionalities. For example the following classes are included in this package: Project, Task, Subtask, and Memo. We also included these classes in our “Conceptual Class Diagram”. These classes along with classes contained in Model package, are in close relationship that provide core functionalities of the software.

Model classes all extend a model interface. Each model class is a singleton class which provides functionalities to access/manipulate entities for a feature. Their primary function is to do CRUD methods. For example: TaskModel provides functionalities to access/manipulate entities for project management feature; includes: Porject, Task, SubTask.

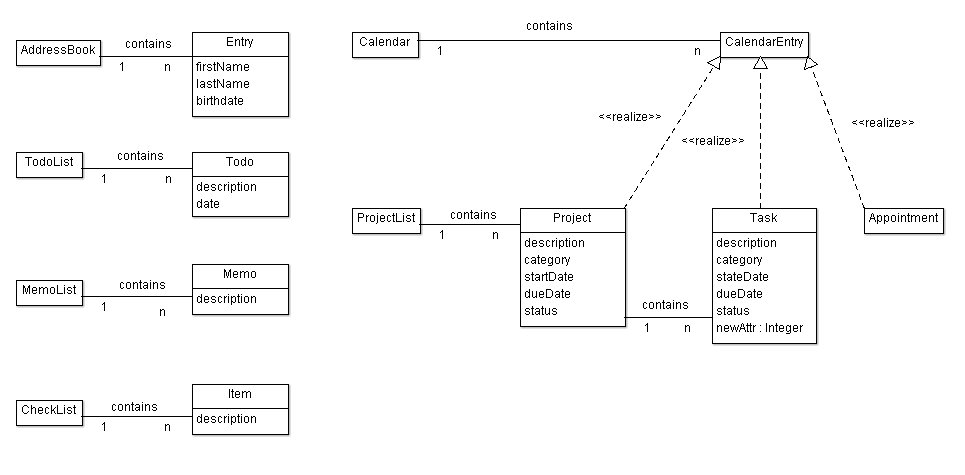
The following classes are included in the actual class diagram:

* Model: Abstract class that all the other model classes implement it. It has basic operations like addListener(), removeListener(), import(), export().
* TaskModel: It is a realization class of Model and is used to access/manipulate projects, tasks and subtasks.
* MemoModel: It is a realization of Model with CRUD methods to manipulate Memo objects
* AppointmentModel: It is a realization of Model with CRUD methods to manipulate Appointment objects
* AddressModel: It is a realization of Model with CRUD methods to manipulate Address objects
* CheckListModel: It is a realization of Model with CRUD methods to manipulate CheckList objects
* Other classes: In total there are 18 classes. The other classes included in this package perform utility tasks like searching, import, export, them provider and etc.

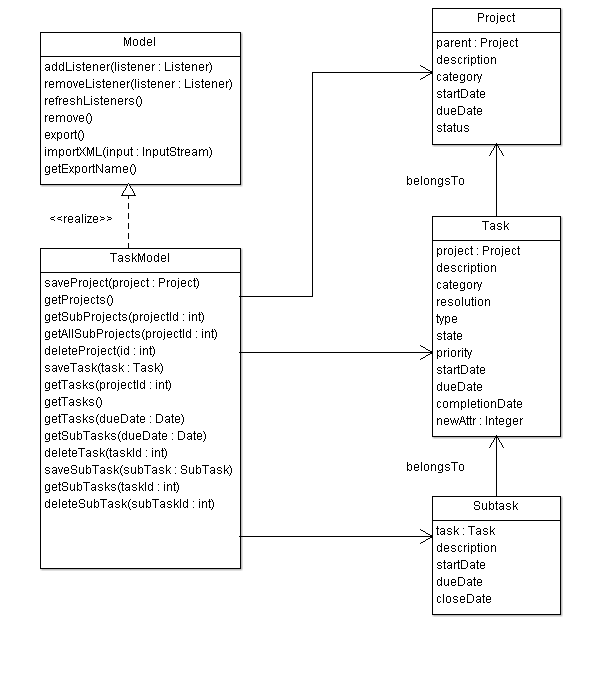
Each of these model classes are in association with their corresponding data class e.g. MemoModel with Memo, TaskModel with Task and etc.

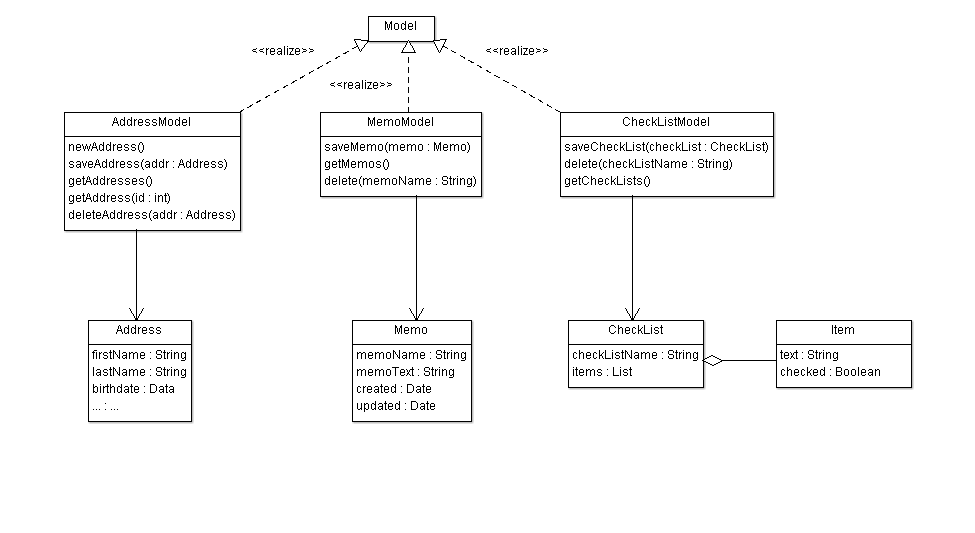
* TaskModel in some degrees acts as ProjectList in conceptual diagram. But they are not totally the same: TaskModel not only contains projects, but also tasks and sub tasks.
* In both conceptual diagram & actual class diagram, a project may contain other projects. But in conceptual diagram, a task cannot contain other tasks. Actual class diagram has sub task class and a task may contain sub tasks.
* In actual class diagram, we don’t have TodoList and Todo: Borg implementation use appointments and tasks with due dates.

## Our Conceptual Class Diagram from M2



## Actual System Class Diagram





## Example of source

We intend to portray the relationship between the following classes: Task, Subtask, and Project in the following code snippet. The comments in the code provide more detailed information.

|  |
| --- |
| **public** **class** Task **extends** KeyedEntity<Task> **implements** CalendarEntity {  **private** String Description;  **private** String Resolution;  **private** String Category;  **private** Integer Project;    **public** **boolean** isTodo(){ **return** **true**; }  @Override  **public** Date getNextTodo(){ **return** **null**; }  @Override  **protected** Task clone() {  Task dst = **new** Task();  dst.setKey( getKey());  dst.setDescription( getDescription() );  dst.setResolution( getResolution() );  dst.setCategory( getCategory() );  dst.setProject( getProject() );  // The above line shows that Task is in relation with Project  **return**(dst);  }  } |

|  |
| --- |
| **public** **class** Subtask **extends** KeyedEntity<Subtask> **implements** CalendarEntity {  **private** **static** **final** **long** *serialVersionUID* = -5794908342032518360L;  **private** Integer Task;  **private** String taskDesc = **null**;  @Override  **protected** Subtask clone() {  Subtask dst = **new** Subtask();  dst.setKey(getKey());  dst.setStartDate(getStartDate());  dst.setCloseDate(getCloseDate());  dst.setDueDate(getDueDate());  dst.setDescription(getDescription());  dst.setTask(getTask());  // The above line shows that Task is in relation with Subtask  **return** (dst);  }    } |

# Code Smells and Possible Refactorings

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\_todo

delete

repeatSet

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(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

1. 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:
2. Unmarshal : it encapsulates the top part of the importXml method.
3. 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.
4. handleOldImports: The middle part of the method can be wrapped in this method.
5. importIntoEmptyDb: the last 40 lines of the method can be wrapped in this method.