Budget Event Management

Design Document

Team 4

Yifan Yin(TL), Jiangnan Shangguan, Yuan Xia, Di Xu, Xuan Xu, Long Zhen

Table Of Contents

Purpose

This document describes important aspects of the implementation of the Budget Event Management system. We will discuss the priorities, important design decisions and issues that will impact the design of our project.

Summary List of Functional Requirements

(See Requirements Document for full descriptions)

- User can browser to the specific web page address and either input a username and password to sign in or select new user to sign up.
- Users will be offered a number of different types of event configuration forms.
- Users will be offered a number of different types of event configuration forms.
- Users will be provided with personal information that need attendees to provide.
- sers are able to add several agenda items for optional workshops, additional talks, meal options, or sports outings with an array of features.
- sers can choose to collect lodging and travel information including the preferences such as hotel room preferences, special requirements, or dietary restrictions of attendees.
- User can request service/item needed for this event by using a drop down list to view the available services/item.
- Users can add confirmation message, registration record header and footer with several optional features such as self-reminder, attendee feedback, location search and map.
- sers are able to add link options at the bottom and the head of every page on the registration form.
- Users can add and edit different page header and page footer for each page in the registration form.
- Users will be provided the basic information, as budget title, and then add the total funding.
- Different events have unique budgeting requirements.
- Event manager can import attendees from any mainstream existing system.
- Event manager can interact with attendees through social media, like Facebook, Google+, Gmail, LinkedIn, and more.
- When a user is idle over a long time, the system will automatically log the user off.

General Priorities

The decisions that we make in this document are based on the priorities that we have set for the project. These are (in order of importance):

Usability

Users of all levels of experience should be able to intuitively use our system. This includes create a new login, create events, apply budgets, manage contacts and etc. Create an event will be a straightforward process (Functional requirement #2) and available in any platform that has a browser.(Non-functional requirements #1)

Accessibility

The system is a web-based application which is not only for a specific platform. In other words, users are able to access our application with almost any kind of platform with a browser, such as Windows, Mac OS, Linux and even Android or iOS.

Availability

The system takes the advantage of cloud computing. The involvement of cloud makes the application available to users anytime anywhere, which is more flexible than a client.

Reliability

All the data is stored in Google's high-quality remote servers. The use of servers is highly more reliable than using the local computer.

Performance

Servers are able to provides accurate events information within the minimum amount of time. Clients will take the advantage of optimized algorithms to rapidly response severs' feedback with limited local system resources.

Stability

The storage solution for the system is the High Replication datastore (HRD). The HRD is a highly available. It remains available for reads and writes during planned downtime and is extremely resilient in the face of catastrophic failure. It provides the highest level of availability for reads and writes and uses approximately three times the storage and CPU cost of the master/slave option, comparing to the master/slave option.

Design Outline

Our system is a client-server system where Google App Engine front-end takes requests from web browser clients and responds with a server generated page(s).

All the client request will be handled by Google App Engine and processed by the Client servlet with possible RPC calls to the backend. These requests are handled directly by the App Engine servers.

Google Datastore will be the backend database for our system which allows transactional queries from server classes. The advantage of the Google Datastore is it saves data objects, known as entities. An entity has one or more properties, named values of one of several supported data types. For instance, a property can be a string, an integer, or even a reference to another entity.

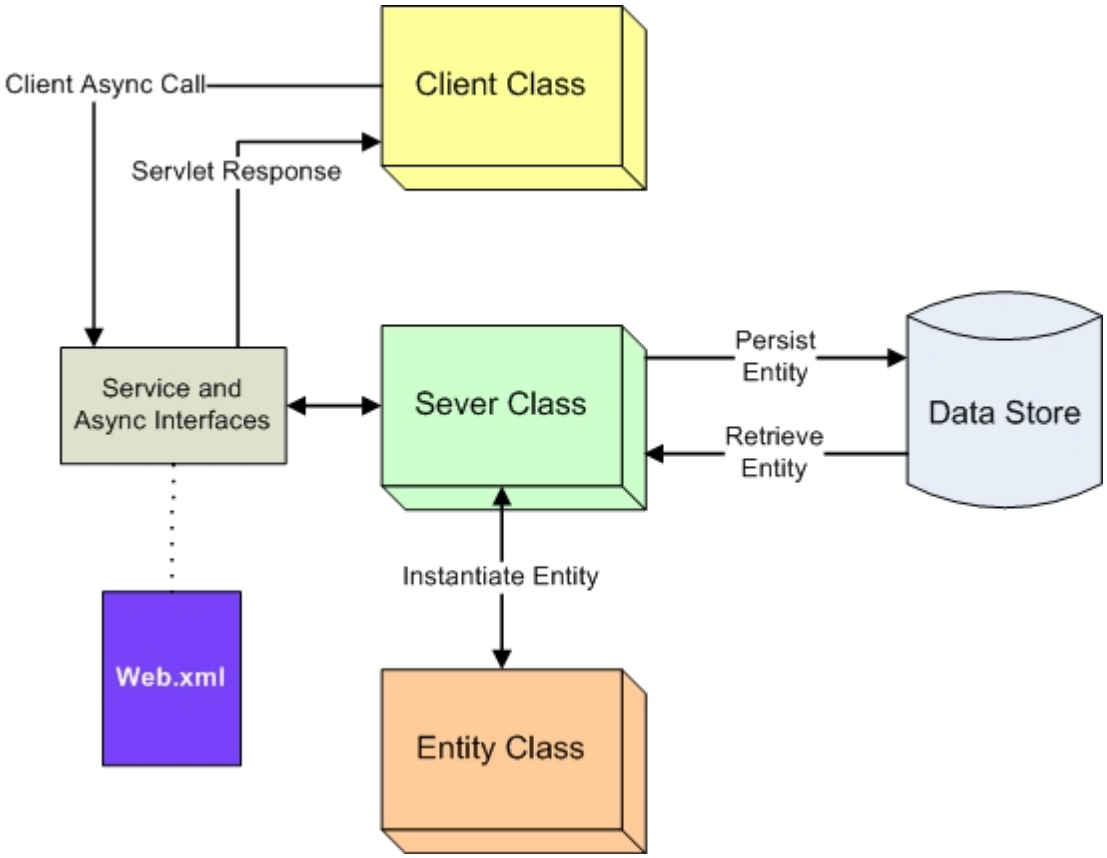

Figure 1: High-level overview of the system architecture

We will use Blobstore service, which allows apps to serve data objects limited only by the amount of data you can upload or download over a single HTTP connection.

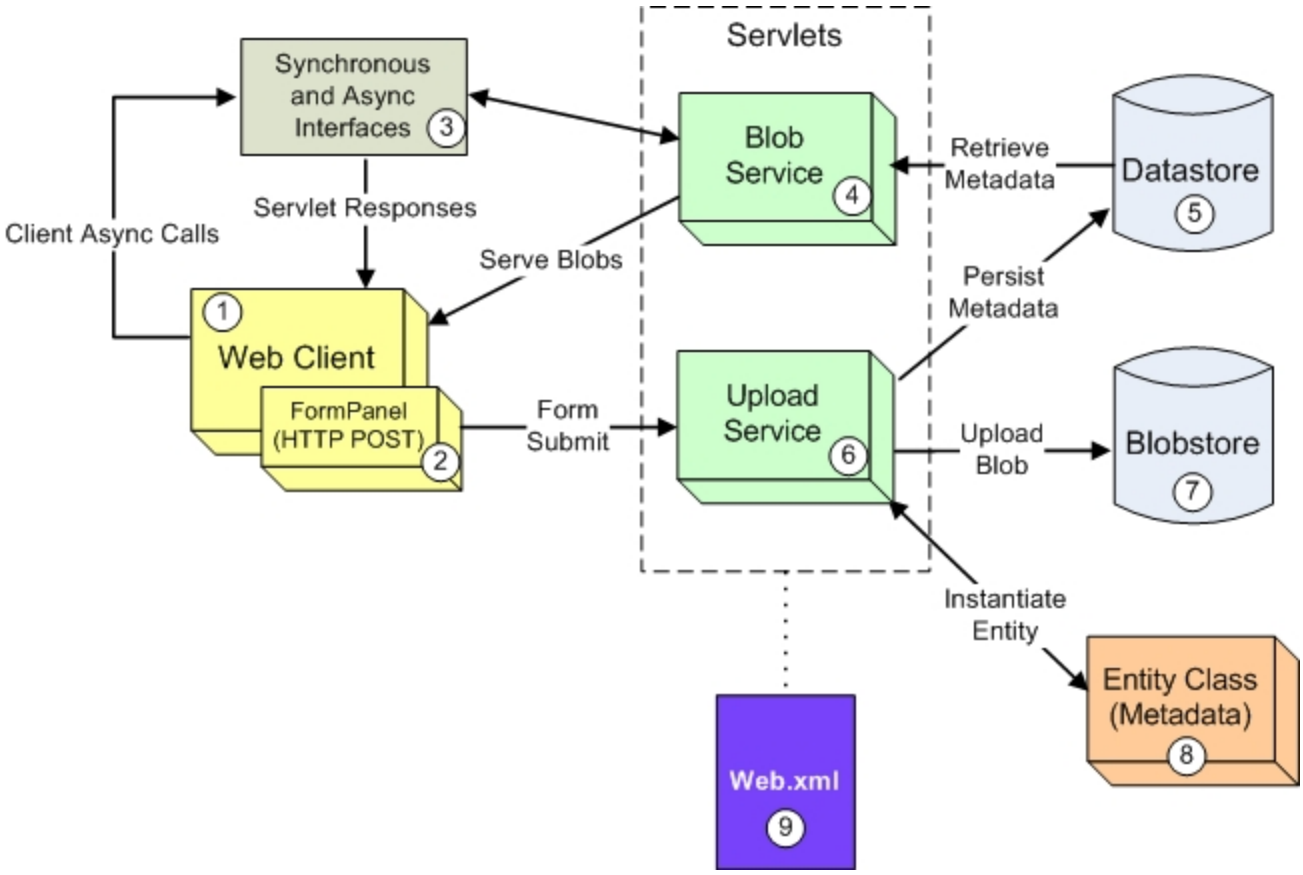

Figure 2: Overview of the document upload process

A attendee in the system (1) wishes to submit a document/spreadsheet, so an RPC call is made through the Interfaces (3) to the Blob Service (4) to start a Blobstore session and get a Blobstore upload URL. This upload URL is returned to the client and applied to the FormPanel (2) and will be called when the FormPanel is submitted.

Submitting the FormPanel (which includes a FileUpload widget) makes an HTTP POST call to the Upload Service (6), which uploads the Blob to the Blobstore (7). In the doPost method of the Upload Service, an Entity (8) is instantiated and put in the Datastore (5) so that some meta-data about the document/ spreadsheet can be stored.

Data reliability will be guaranteed by High Replication datastore provided by Google App Engine, data is replicated across data centers using a system based on the Paxos algorithm. High Replication provides very high availability for reads and writes (at the cost of higher-latency writes). Most queries are eventually consistent. Therefore, we can ensure that data will always be able to recover in the case of a natural disaster or possible disk failure.

Design Issues

Architecture Issues

Issue 1*:* Server decision
Option 1: Dedicated Servers
Option 2: Google App Engine Platform
Decision:
Option 2. This decision is made because of it's scale automatically without worrying about managing machines. With sophisticated APIs we can focus on building differentiating features for our system. We don't need to worry about hardware, patches or backups; and effortless scalability.

Issue 2*:* Object persistence technology
Option 1: Java Data Objects (JDO)
Option 2: Java Persistence API (JPA)
Decision:
Option 1. JDO is datastore-agnostic. JPA isn't. JDO allows fetch groups. JPA doesn't. JDO allows datastore identity. JPA doesn't. JDO allows datastore transactions. JPA doesn't.

Issue 3*:* User interface implementation
Option 1: JSP + HTML + Javascript
Option 2: Google Web Toolkit
Decision:
Option 2. GWT SDK provides us a set of core Java APIs and Widgets. These allow us to write AJAX applications in Java and then compile the source to highly optimized JavaScript that runs across all browsers, including mobile browsers for Android and iPhone.

Issue 4*:* Database
Option 1: Traditional RDBMS
Option 2: Google Datastore
Decision:
Option 2. Therefore, we don't have to run a database server to support our system. And the main advantage of the Google Datastore is its scalability and a tighter guarantee on the amount of time a query will take (sort of like CPU time). The scalability comes from the way records are distributed, if we setup our keys correctly then the data associated with those keys will be closer together physically (the data is distributed so there is no single point of failure).

Issue 5*:* How to store user uploaded files
Option 1: Static files
Option 2: Blobstore
Decision:
Option 2. Blobstore service, which allows apps to serve data objects limited only by the amount of data you can upload or download over a single HTTP connection. These objects are called Blobstore values, or blobs. Blobstore values are served as responses from request handlers and are created as uploads via web forms. Therefore, Blobstore values can be served to the user, or accessed by the app in a file-like stream, using the Blobstore API without file fetching in a single machine.

Issue 6*:* Client Type
Option 1: Fat-client

Option 2: Thin-client

Decision:

Option 2. This decision is made to satisfy the priority of data security and efficiency of data management.

Issue 7 : Platform

Option 1: Design different clients for different platforms.

Option 2: Web App for all platform

Decision:

Option 2. Even though designing specific client for corresponding platform would bring more stability and stronger utility, choosing Web App allow us focusing more energy on designing background components but not multiple clients.

Design Issues

Issue 8*:* Budget and Event

Option 1: A budget can only be associated with a unique event and can not exist with out a event

Option 2: A budget can exist without associating with a certain event

Decision:

Option 2: This decision is made in order to reduce the true dependencies along the entire system. It comes from the idea of “decoupling”. This arrangement allows a budget to exist with out associating with an event. It makes it possible that the organizer may reuse a previously created budget to a new and similar event. A budget will be assigned an ID upon data store and can be associated to any event as ordered.

Issue 9*:* Item and Category

Option 1: Each item has only one category

Option 2: Each item can have multiple categories

Decision:

Option 1. This decision is made to reduce the redundant data storage and make management on items and categories be more efficient and easier.

Issue 10*:* User identification

Option 1: Assign unique UUID for each user for identification

Option 2: Assign user id by using integers

Decision:

Option 1. This decision is made to enable our system to uniquely identify users without significant central coordination.

Issue 11*:* Attendee account

Option 1: Allow attendees to create accounts in the system

Option 2: Only provide attendees event invitation links

Decision:

Option 2. First this online management system goal is to help organizers to get rid of using several different separated systems to manage their events. The system doesn’t provide lots of features to help attendees to manage the events they will attend. Second, statistically speaking, people are more willing to use calendar applications(such as Google Calendar) to manage their events. Those applications provide users more powerful functions. Also, people may receive event invitations from different sites or systems, not only from us. It will be more efficient to merge all the event information in one calendar application.

Issue 12*:* Category creation

Option 1: The system does not allow users to add new service categories.

Option 2: The system provides some default categories and accept new category addition.

Decision:

Option 2. Only allowing users to select from several default categories that are provided by us will make us manage the system more easily. However, there is too many categories in the real life. It will be an

extremely difficult job for us to gather enough category information such that vendors are able to find a proper one for certain service. Otherwise, it may lead much more difficulties to users in finding a proper service in a certain category.

Issue 13*:* Organizer and Event

Option 1: Each event has only one organizer

Option 2: Each event can have multiple organizers

Decision:

Option 1. This decision is made to reduce the work on collaborative synchronization. Option 2 would cause some synchronization problems and probably decrease the efficiency on management.

Issue 14*:* Multiple roles in a single account

Option 1: Allow a user to be organizer and vendor in one account

Option 2: Only allow a user to choose one role from organizer and vendor

Decision:

Option 2. Statistically speaking, there is little users who need to be both organizer and vendor at the same time. It is not worth spending so much time on something that most of users do not want. Also, if the system provided such switching role function and UI, it may cause the same side effect as the BlackBoard System and confuse users.

User Interface Issue

Issue 15*:* Theme hue

Option 1: multiple color

Option 2: few kind of color with simple tone

Decision:

Option 2. option 1 will lead User Interface to be too colorful for user to focus on important information, and it will bring inconvenience to people with color blindness.

Design Details

Class Diagram

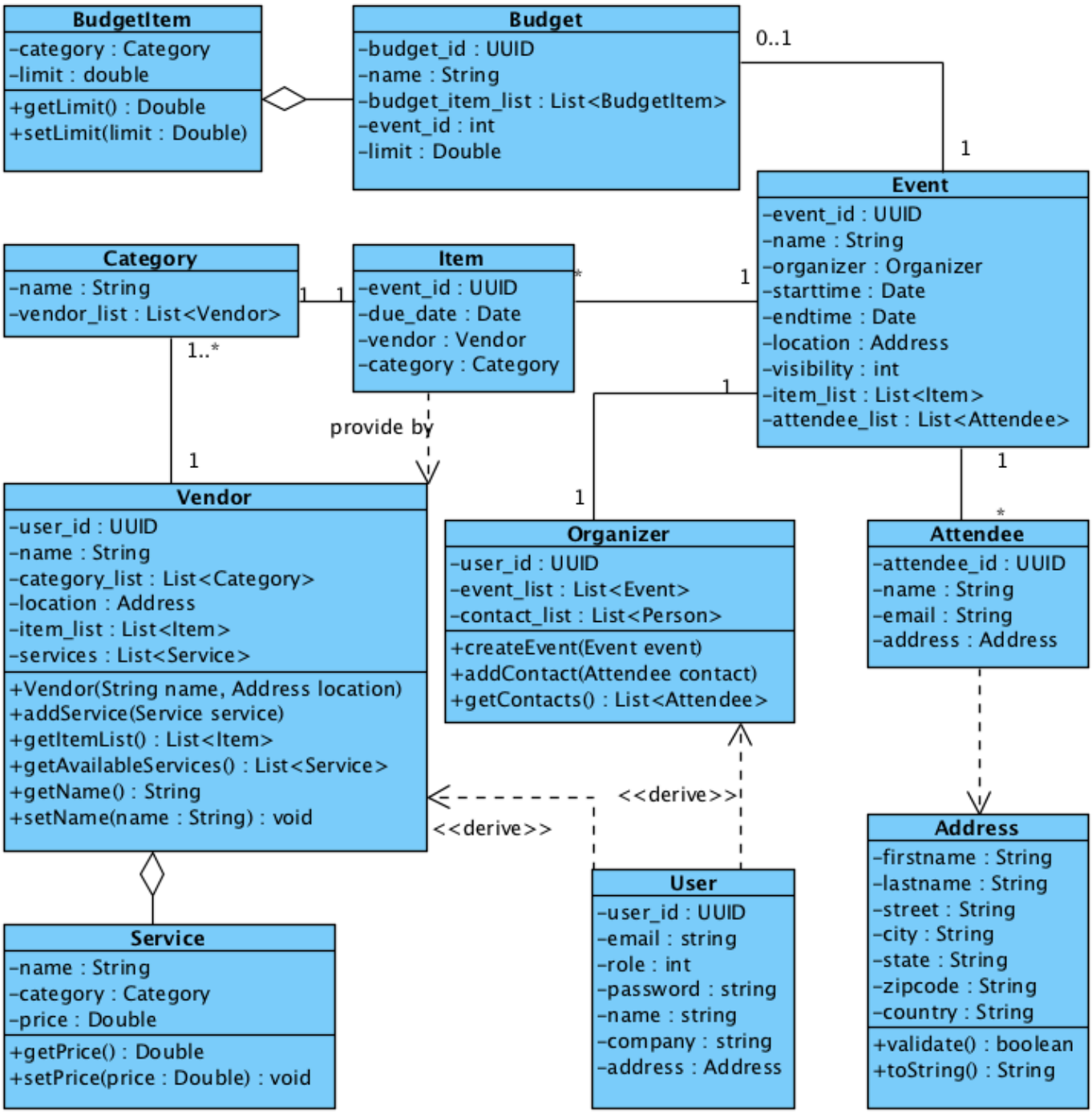

Figure 3: Class Diagram

Description of classes and models

User

Entity that encapsulates the system's main users (organizers and vendors).
For demonstration purposes, additional details of a User that do not pertain directly to our system are left out for simplicity, such as phone number, etc.

Address

Entity that encapsulates the system's all addresses.
Provide convenience methods that process and display address easily.

Attendee

Entity that encapsulates all attendee information.
For demonstration purposes, additional details of a User that do not pertain directly to our system are left out for simplicity.
Only the event organizer can access or organize the attendee he/she own.

Event

Major entity that encapsulates a event which includes detailed information.
It holds all the content a organizer might need to access/modify after the system generates an event(dynamic web page) for Attendees.
Only an organizer have the permission to modify the details of a event he/she created.

Item

Entity for a particular item which added by a event organizer for a event.
Item can be created only by the organizer of a particular event.

Category

Entity for a particular category which used to categories services, items and vendors.

Budget

Entity for a budget template defines a list of limits for a list of Budget Items.

BudgetItem

Entity for a budget limit for one category.

Service

Entity for a particular service provided by a Vendor which includes detailed information.
Service can be created only by the Vendor for a particular category.

Sequence Diagram

Event Creation

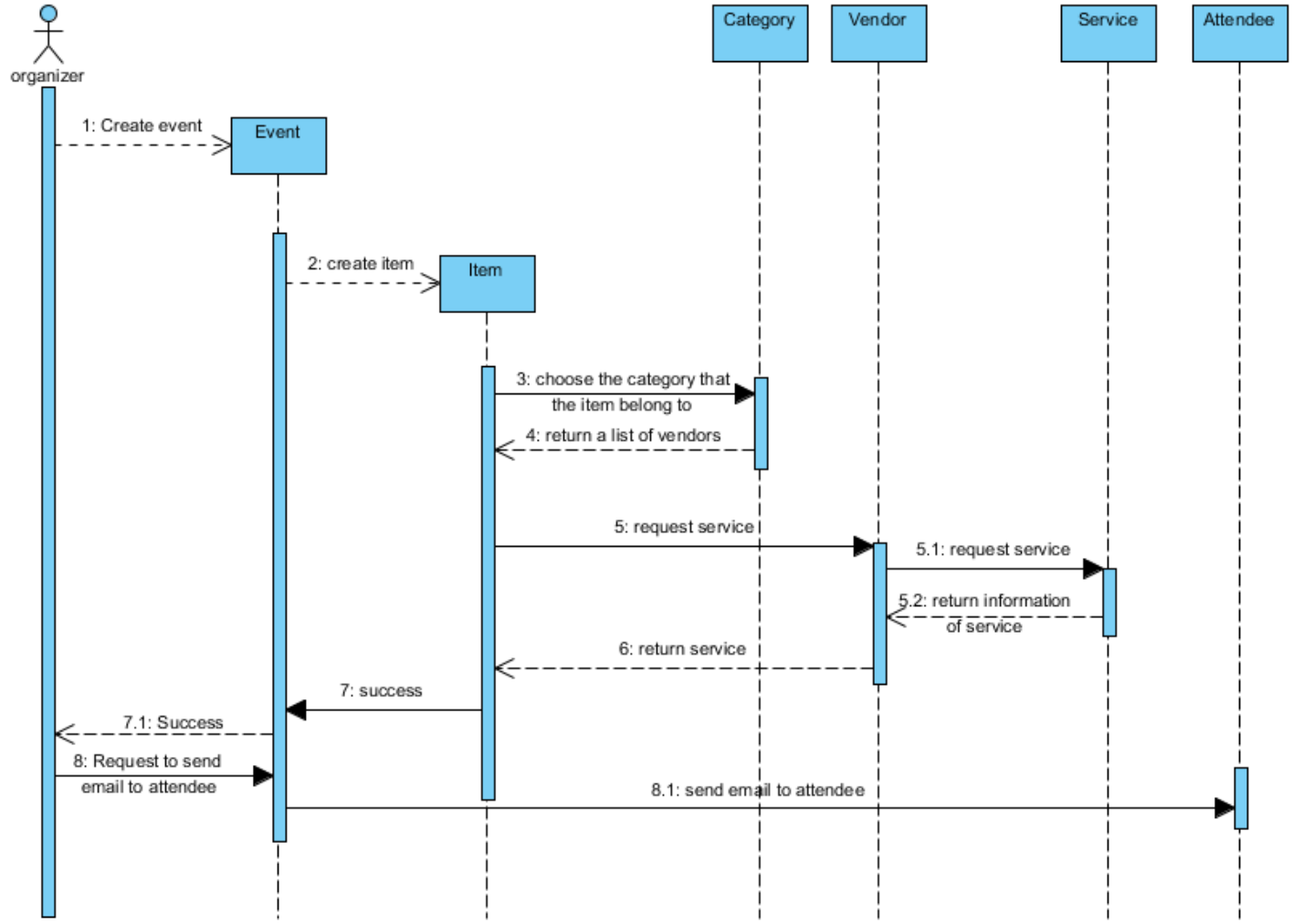

Figure 4: Sequence Diagram for Event Creation

Event organizer create an event object and input information of the event. while organizer add items(e.g. food, hotel, airline tickets etc) to the event, the event object create the item object which then search in the category object to find the corresponding category that the item belongs to. After that, the category object return item object a list of vendor from which organizer choose appropriate vendor as service supplier. In the process of requesting service from vendor, vendor first check its service list then return the information(e.g. price, inventory etc) back to item object for organizer to compare. Finally, the item object return an success signal back to Event object. After organizer input all the information needed, the event object will return an successful information then the organizer can choose to send invitation to attendees through email.

Category Creation

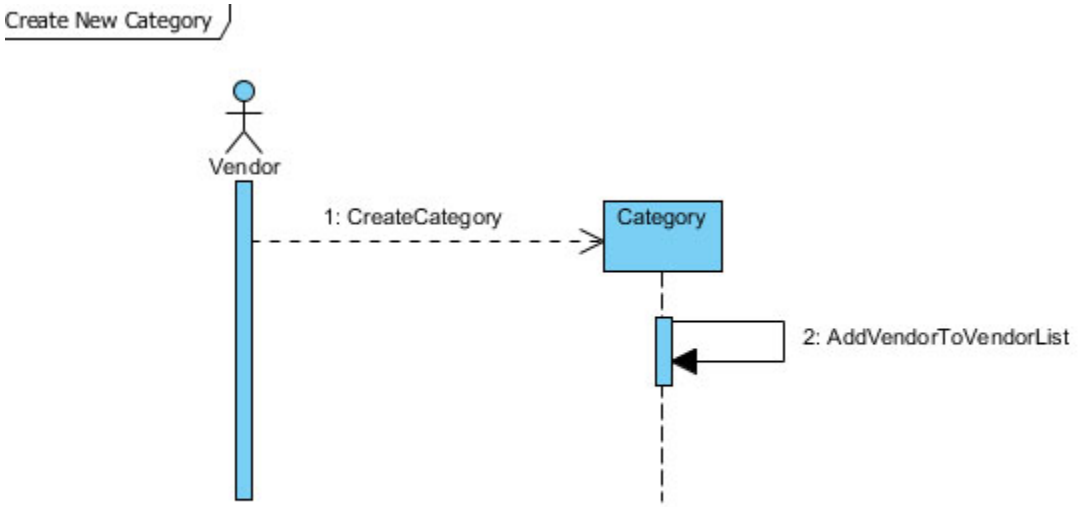

Figure 5: Sequence Diagram for Category Creation

User creates a new category of the service. This is generally performed by Vendors. When Vendor request to add a new category, it will create a new Category object, and add the vendor this the VendorList of this category.

Service Addition

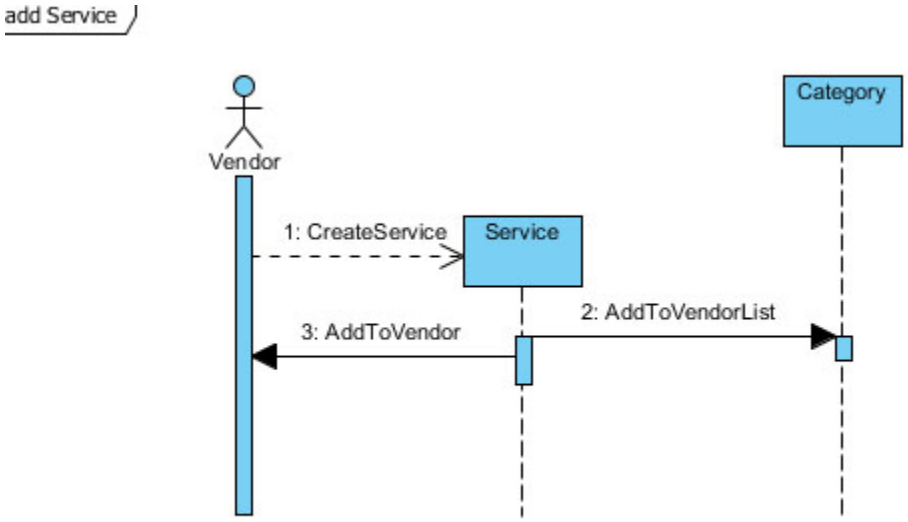

Figure 6: Sequence Diagram for Service Addition

User adds new Services. This is generally performed by Vendors. When Vendor request to add a new service, it will create a new Service object, then Vendor requires to assign the service to a specific category. The selected category object will add the vendor to the vendor list after that.

Budget Creation

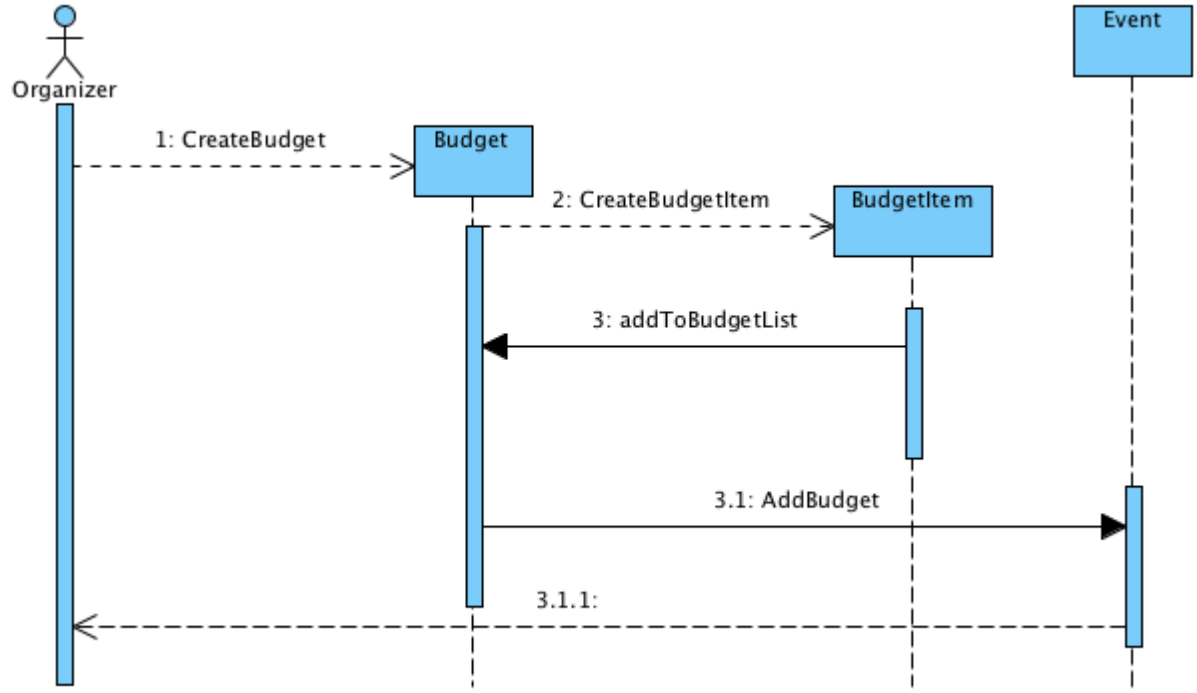

Figure 7: Sequence Diagram for Budget Creation

User can create budget. This is performed by Organizers. Organizers can create a budget object, in order to describe the detail of the budget, organizer will create several BudgetItem objects, these BudgetItems will be added to the BudgetList, after that user need to assign the created budget to an existing event object. then this budget will be asked to assign to a specific event.

Activity Diagram

User Registration

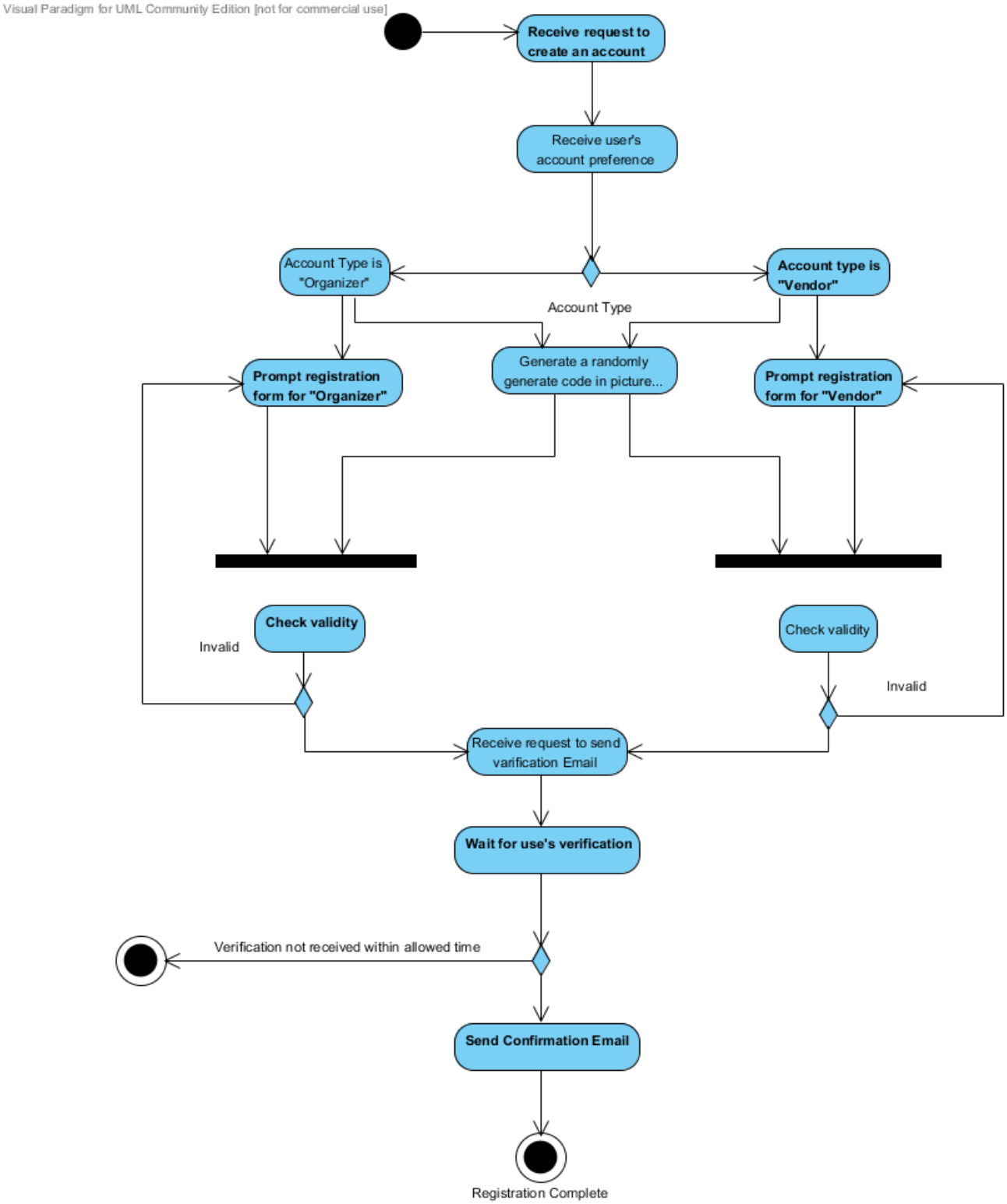

Figure 8: Activity Diagram for User Registration

This allows a new user to create an account associate with a unique email and is of certain role(either the organizer or the vendor). The user need to provide valid information and email address to finish the registration.

User Login

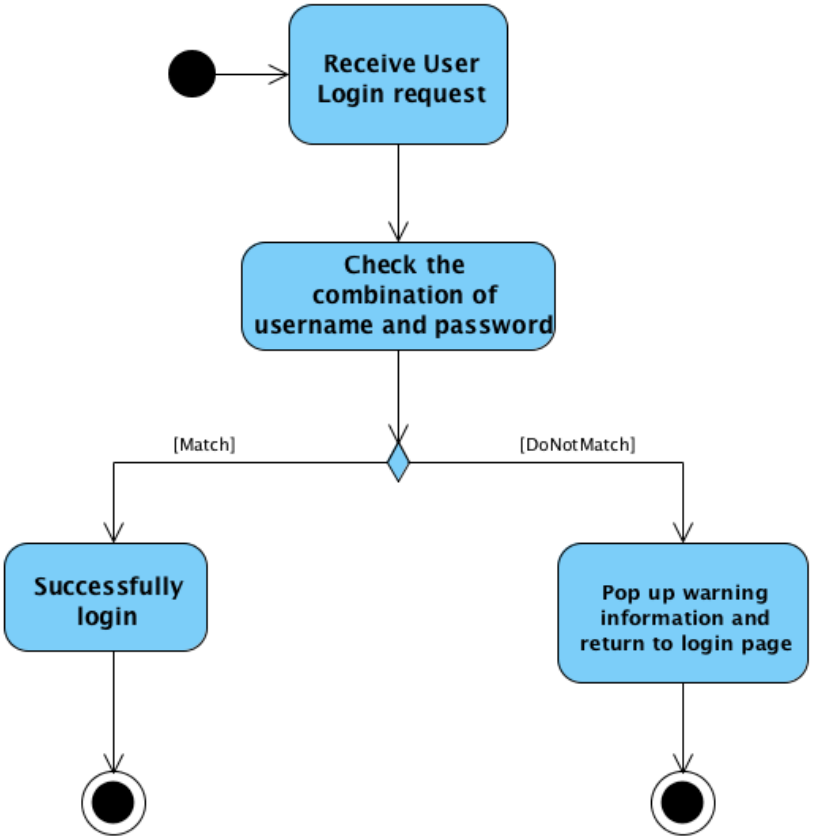

Figure 9: Activity Diagram for User Login

An existing user enter his/her user-name and password to log in the management system.

Event Creation

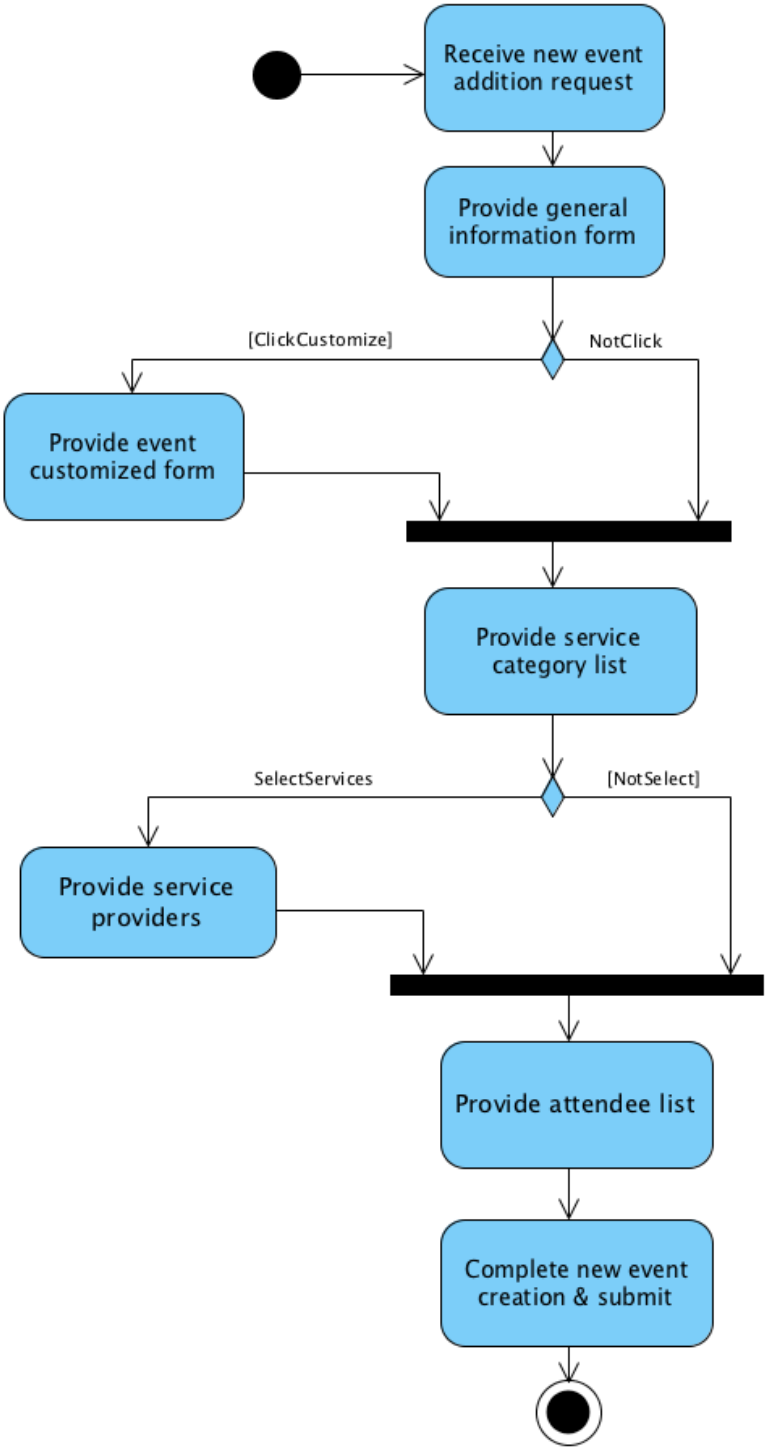

Figure 10: Activity Diagram for Event Creation

Organizers will create their events and initialize some basic information, such as event content, time, location, attendee list, necessary services and so on. Also, organizers can choose to send email notifications to attendees or even order some services at this time.

Budget Creation

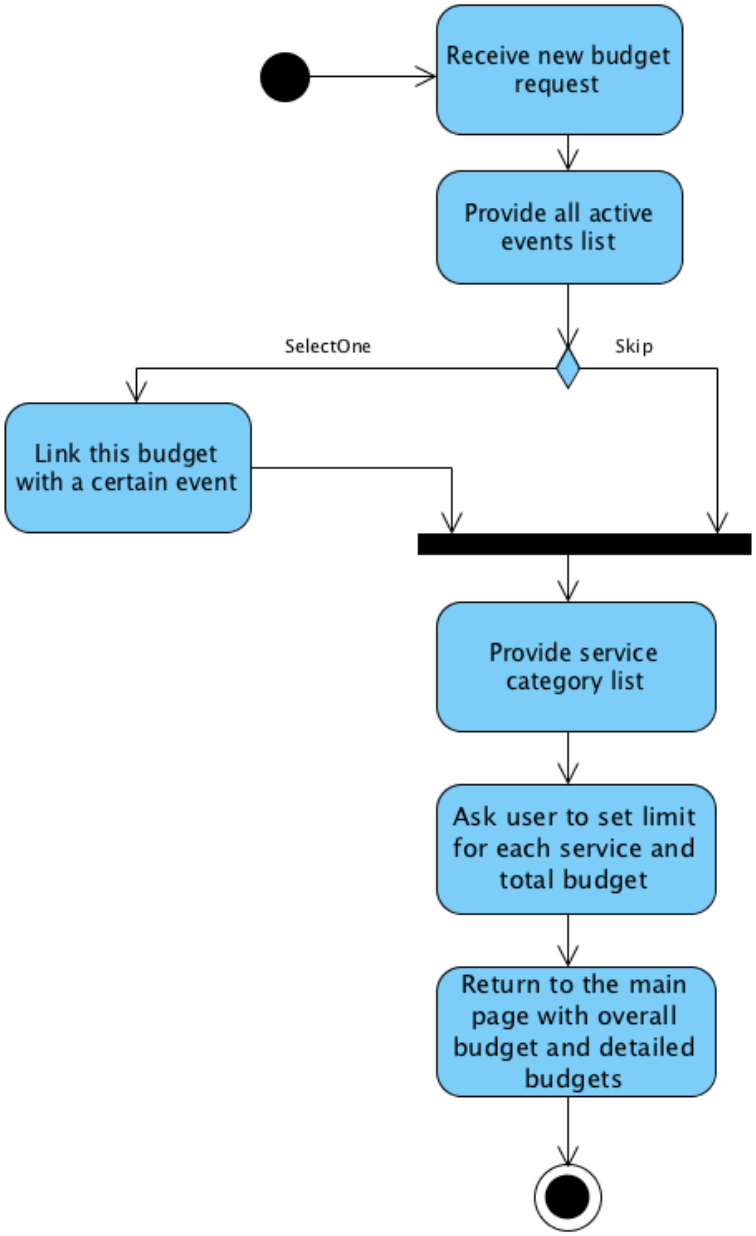

Figure 11: Activity Diagram for Budget Creation

Users can create a new budget and choose to link it with one of the active events in their event lists. After finishing the budget creation, the system will redirect users back to the main page and show the overall budget chart and chart for each category budget.

Budget Modification

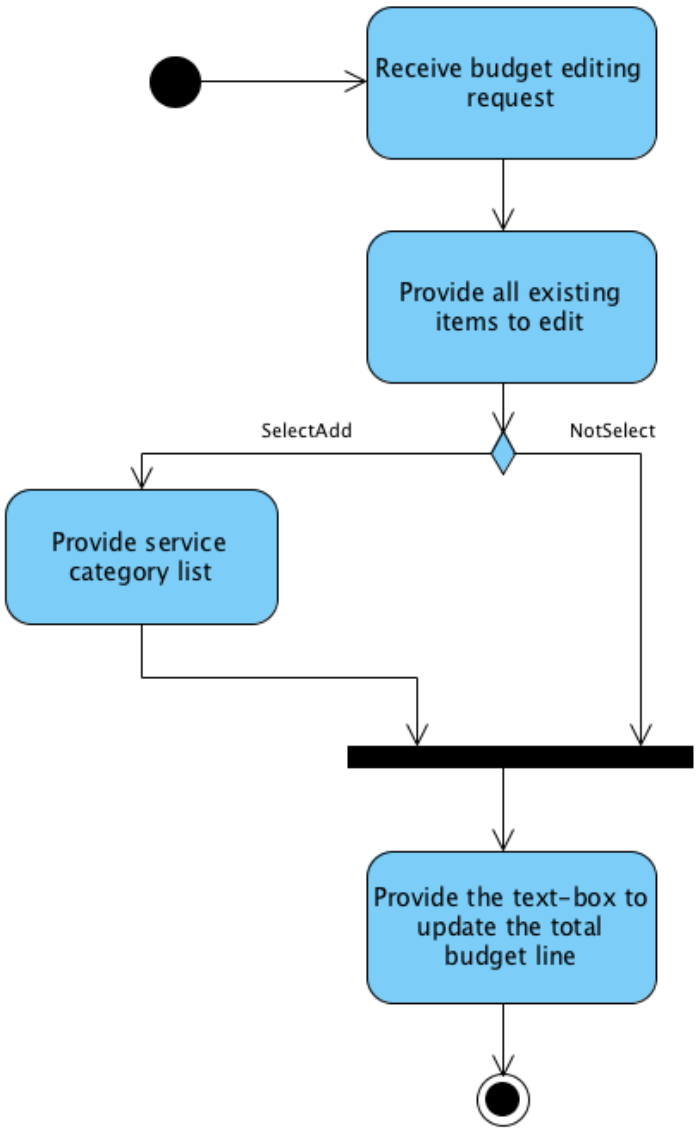

Figure 12: Activity Diagram for Budget Modification

Users can update current items in the existing budgets or add more service items to budgets.

Event Modification/Deletion

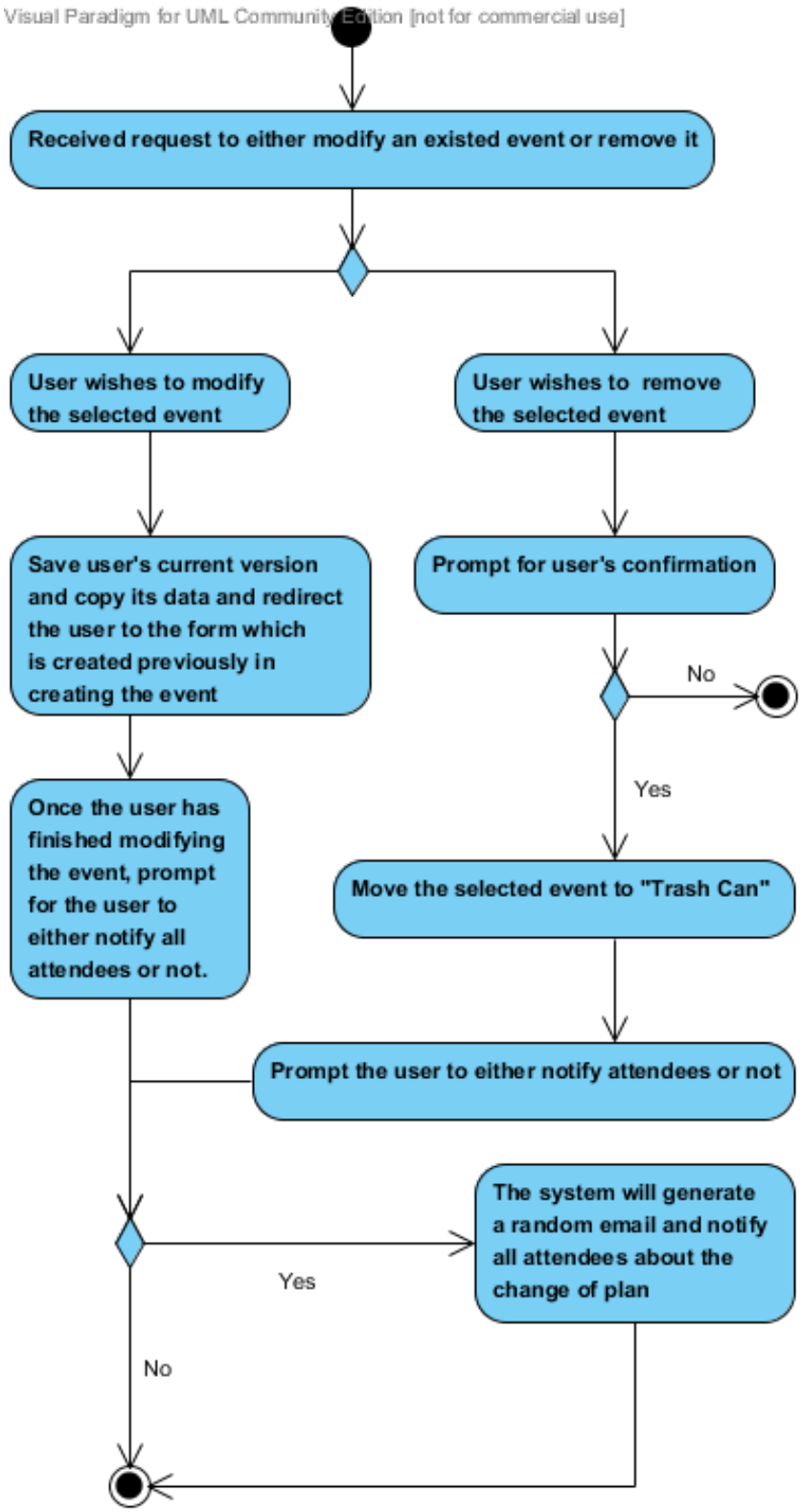

Figure 13: Activity Diagram for Event Modification/Deletion

This allows the user to modify or delete an existing event which has been created already. Once the user have selected an event, the user will have the options to either modify it or simply remove it. If the user chooses to modify, the user will be redirected to the form of the event which the user has made previously.

Upon the submission of the modified version, the user will be given a choice to notify the attendees associated with the event. If the user wishes to delete the event, the system will prompt for the user's confirmation to proceed. Upon user's verification, the event will be removed to a trash can where the user can recover the deleted event.

Mock User Interface

User Registration

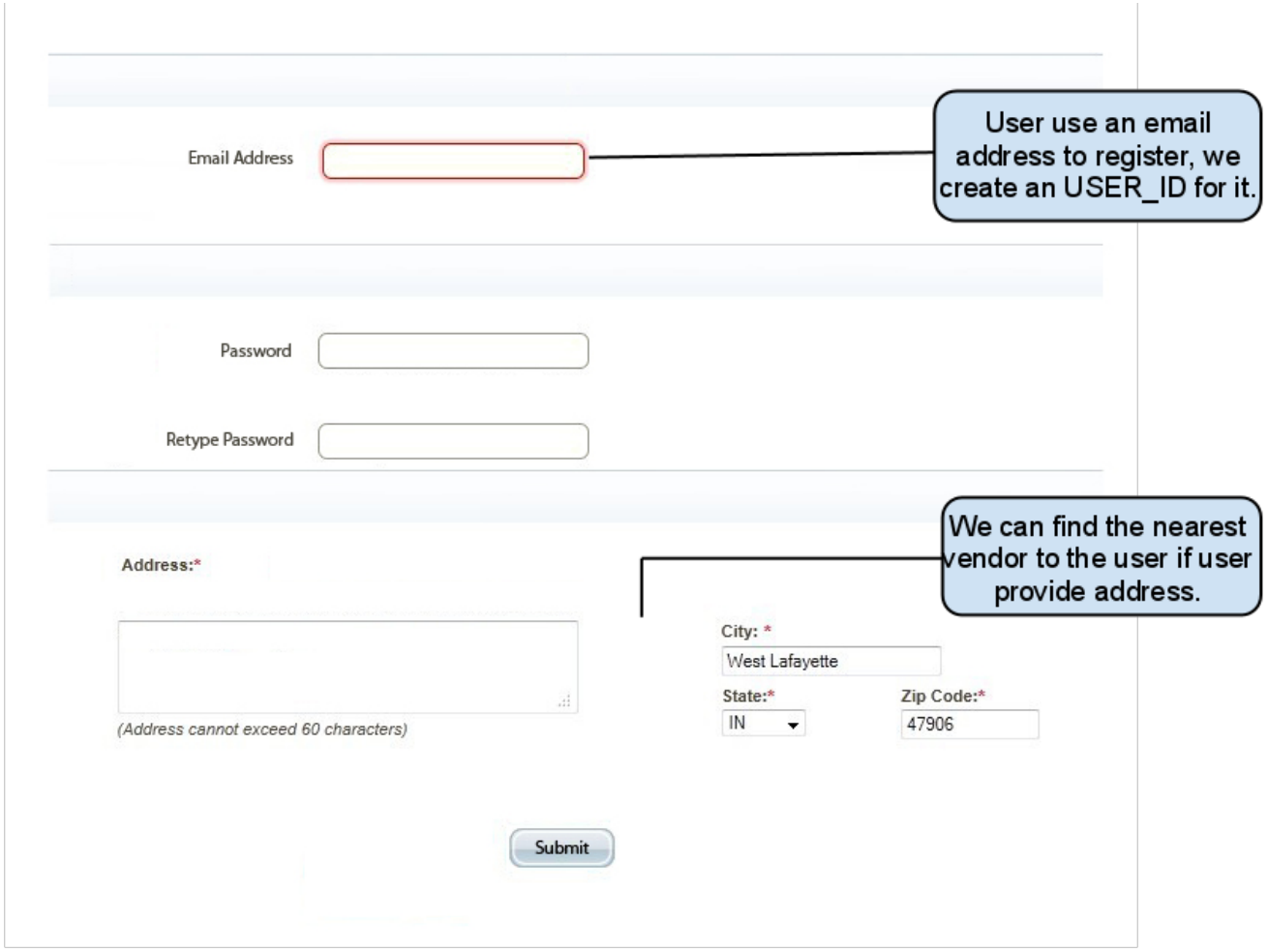

Figure 14: Mock UI for User Registration

User Login

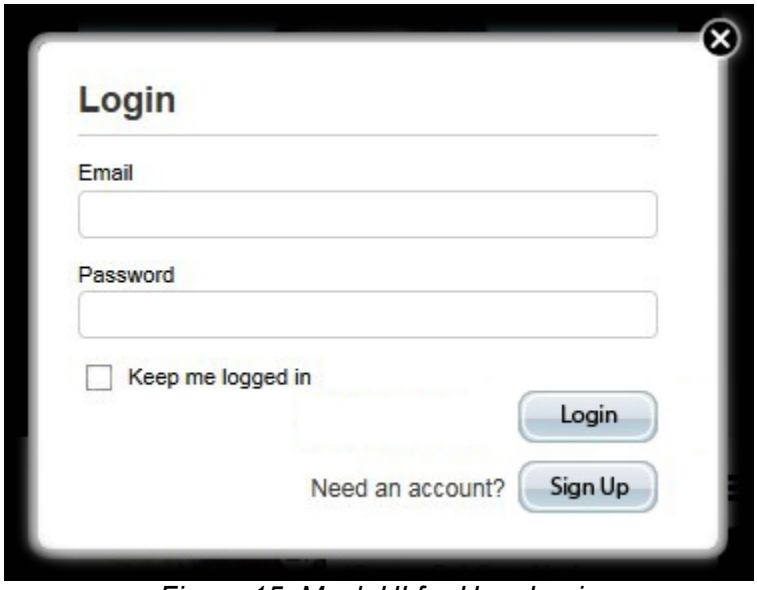

Figure 15: Mock UI for User Login

Event Creation

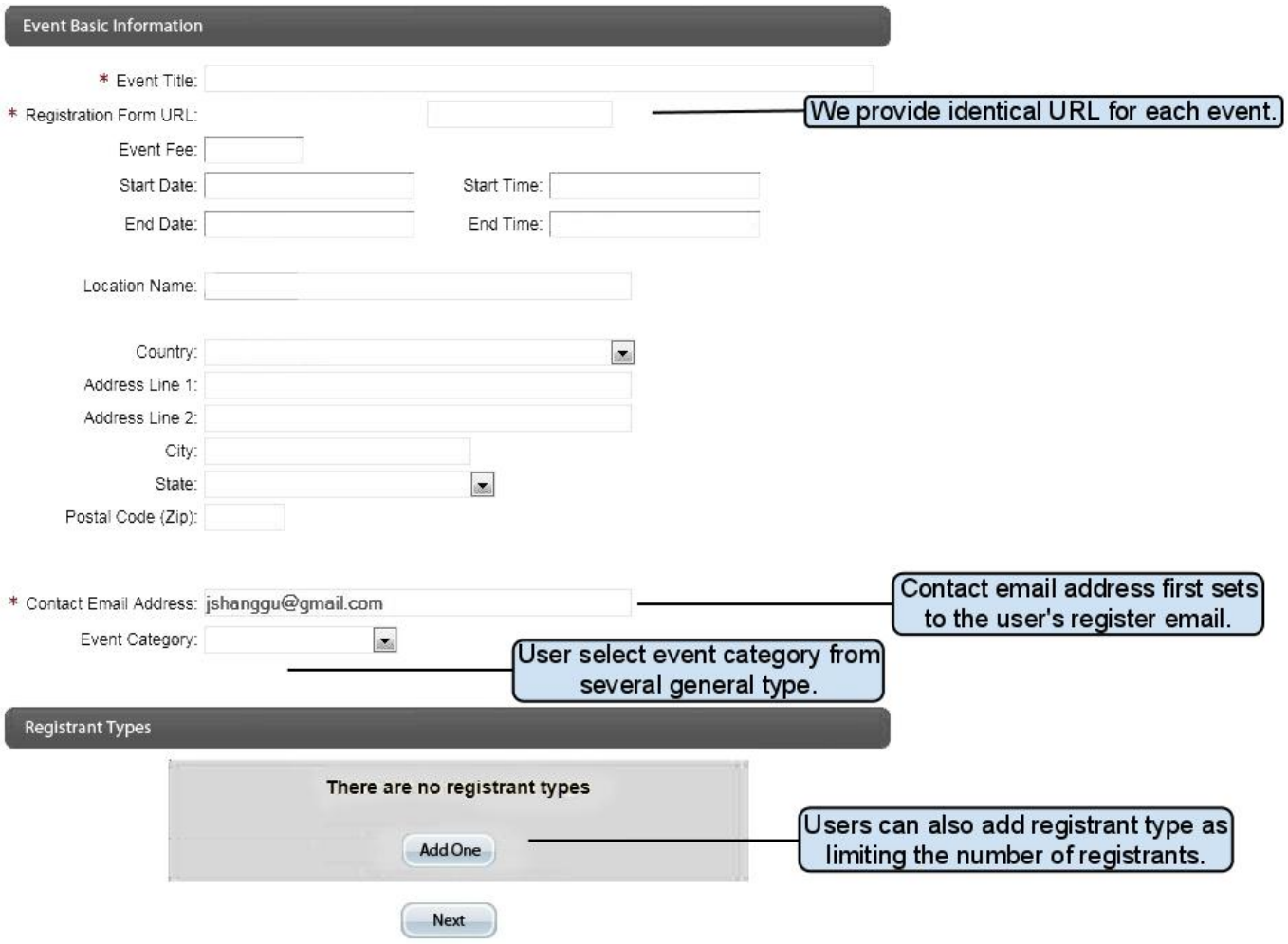

Figure 16: Mock UI for Event Creation

Portal

Figure 17: Mock UI for Portal

Budget

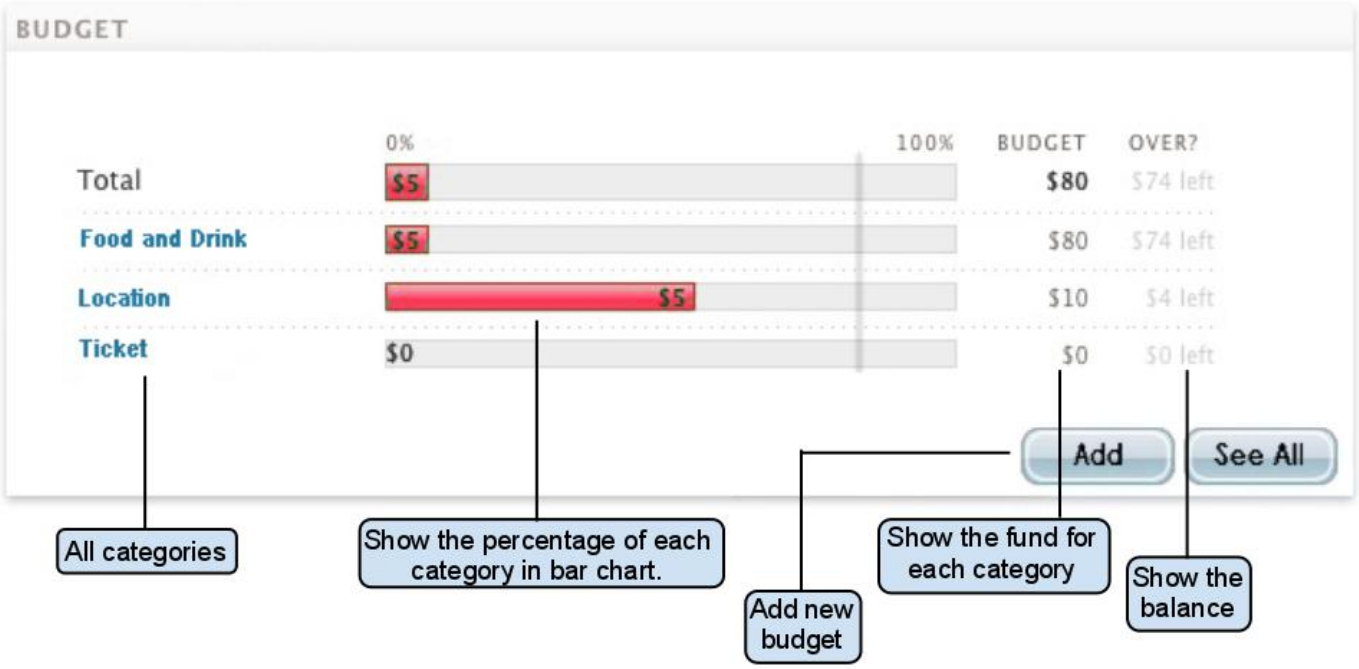

Figure 17: Mock UI for Portal