Design Document

Version 1.0 – 2008.10.03

Created 2008.09.21

Yahoo! Property View

Rob Shaw	- Team Leader
Jacob McDorman	- Project Leader
Robert Read	- Technologist
Brad Van Dyk	- Editor

Table of Contents

[1] Introduction

[1.1] Document Scope

This document is the widget design document. This document contains a detailed outline of design features that will be implemented into the property view widget. The first section of this document will explain the high-level design specifications. It will contain the following: the goals and guidelines of the project, the architectural strategies that will be used, the high-level component view to demonstrate the software components and different dependencies, and a high-level deployment view to model the hardware used. The second section of this document will explain the detailed design specifications which will contain the following: the logical view of class diagrams, a more detailed component view, a process view to describe how the objects interact, and a user interface flow model.

[1.2] Intended Audience

This document is **not** intended for widget users. It will **not** describe how the user uses the system. This document is intended for the stakeholders (persons affected by the project) and the software development team.

[1.3] Project Overview

As stated in the requirements document 1.4.1, this project will provide a solution to Yahoo's difficulty with accessing statics collected from the various systems offered by Yahoo. The domain of this widget is statistic analysis, and as such, the motivation of this software is to aid the user with analysis by providing trending data, volume data, and alerting mechanisms.

[2] High-Level Design

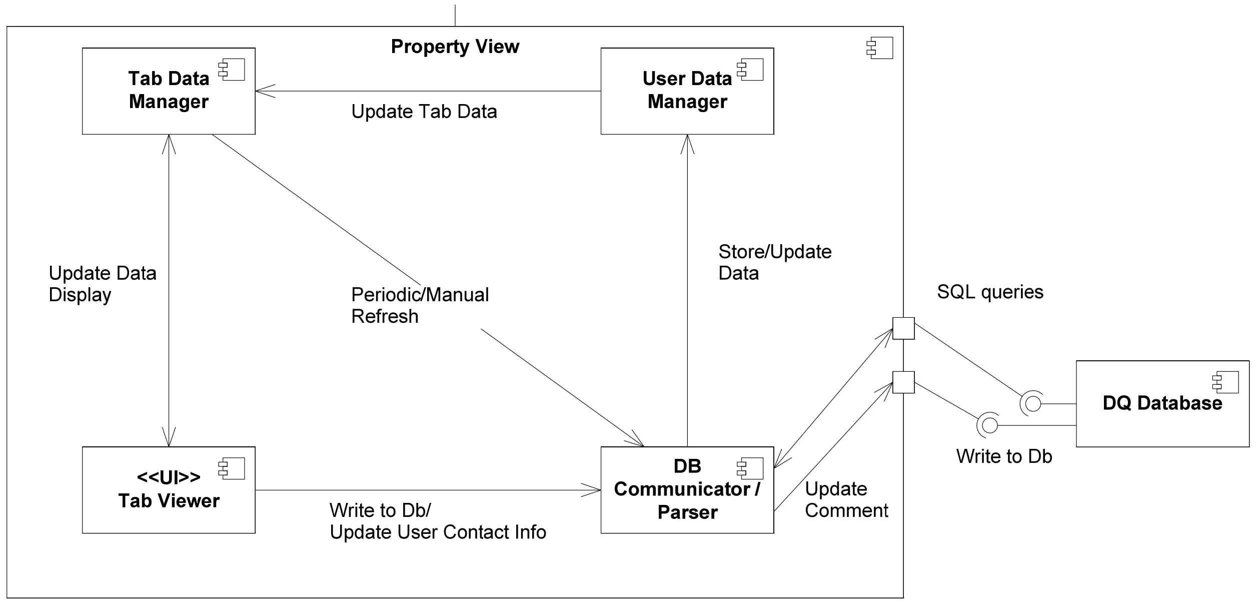
[2.1] Goals and Guidelines

There are no software requirements or objectives that have a significant impact on the architecture.

[2.2] Architectural Strategies

- The overall design of the project is not very complicated.
- There are no such requirements that suggest any special attention is needed to certain styles or models of a user interface.
- The database is managed by another team and the Property View widget will only connect to it and provide queries and a view of the data.
- There is no needed communication between components.
- There is no component to handle errors. All errors are reported directly to the user.

[2.3] High-Level Component View



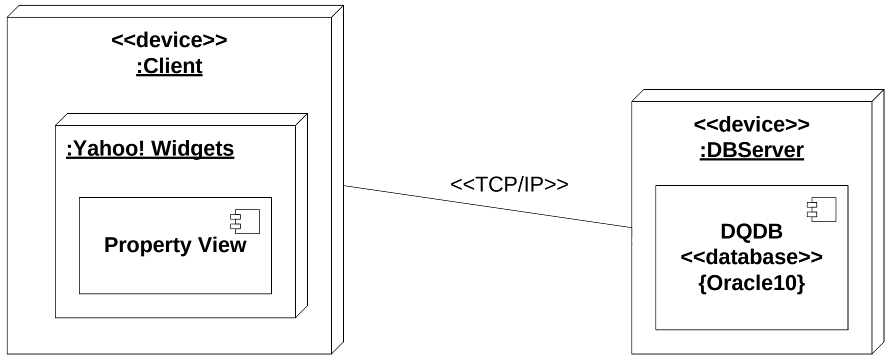
The property view is the widget created to connect to the Data Quality database where the data will be queried and updated if necessary.

The interaction between the Tab Viewer and the DB Communicator/Parser is the writing of any updatable values to the database. An example would be the user writes in the comment field for the bug tab, which is specified to be updateable by the MMT. Any other values able to be written to the database will be editable through the Tab Viewer and when entered by the user will be sent to the Communicator which will update the values in the database. The user can choose to tell the Communicator to query the database again for any information changed in the Contacts table for the user.

Data coming out of the DB communicator goes through the User Data Manager first so any specified options by the user that get stored on the client will be applied to the data before sending it to the appropriate tab.

The interface is query driven.

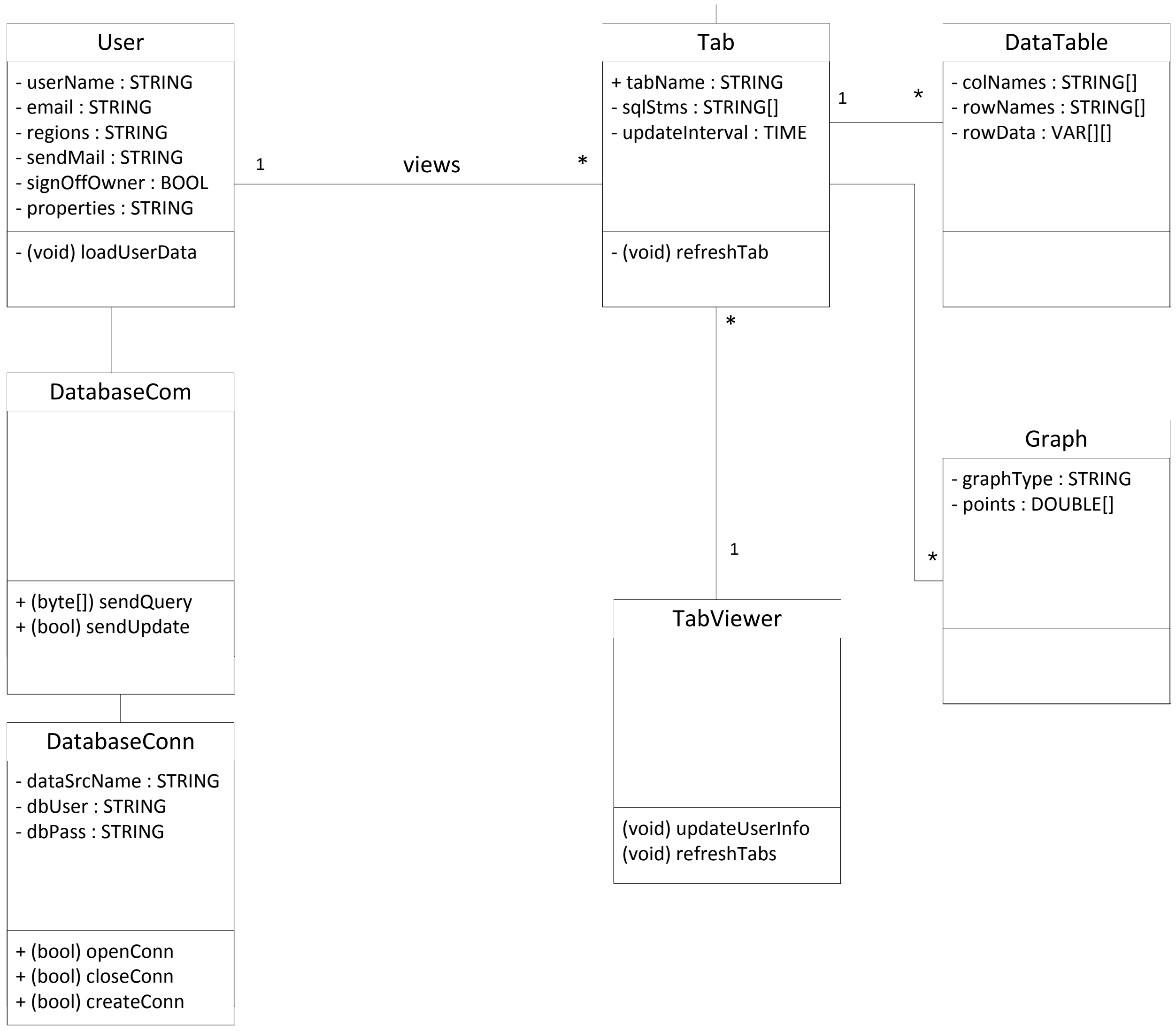
[2.4] High-Level Deployment View



The Property View component is what is being developed for this project. It will be deployable on any client able to run Yahoo! Widgets and a network connection allowing it to access Yahoo's data quality database.

[3] Detailed Design

[3.1] Logical View



[3.1.1] User

The user class is where the options specified by the user and the user's info from the contacts table will be stored. The class will be responsible for writing certain data to the client's hard-drive when needed for persistence between sessions. Any data coming from the database will be filtered according to any user settings through this class before going to the appropriate tab data manager.

[3.1.2] DatabaseCom

This class is responsible for communication with the database. The class will parse data coming from the database into a useable format before passing it to the User, which will ultimately pass the data to a Tab. The DatabaseCom will also take data for updating the database and make queries for collecting the data needed for the tabs.

[3.1.3] DatabaseConn

The DatabaseConn class stores the information needed to connect to a specified database. It will be responsible for maintaining the connection with the database.

[3.1.4] Tab

The Tab class stores the data needed for each tab. The Tab class will be responsible for keeping the data in the correct format whether it needs to be in a graph or in a table. The tab class also keeps track of when it needs to automatically update the tab and if it's time to update it does so.

[3.1.5] DataTable

A DataTable is a formatting of certain data to be easily interpreted by the TabViewer so it can be displayed in a table.

[3.1.6] Graph

A Graph is the formatting of certain data into a certain type of graph, which can be easily interpreted by the TabViewer.

[3.1.7] TabViewer

The TabViewer is where the GUI functionalities will be handled. This is where the user will specify options such as to turn off alerts for certain tabs, be able to manually refresh certain or all tabs, and edit updateable fields for tabs. The user will be able to specify which tab to view and see any Graphs or DataTables associated with that tab.

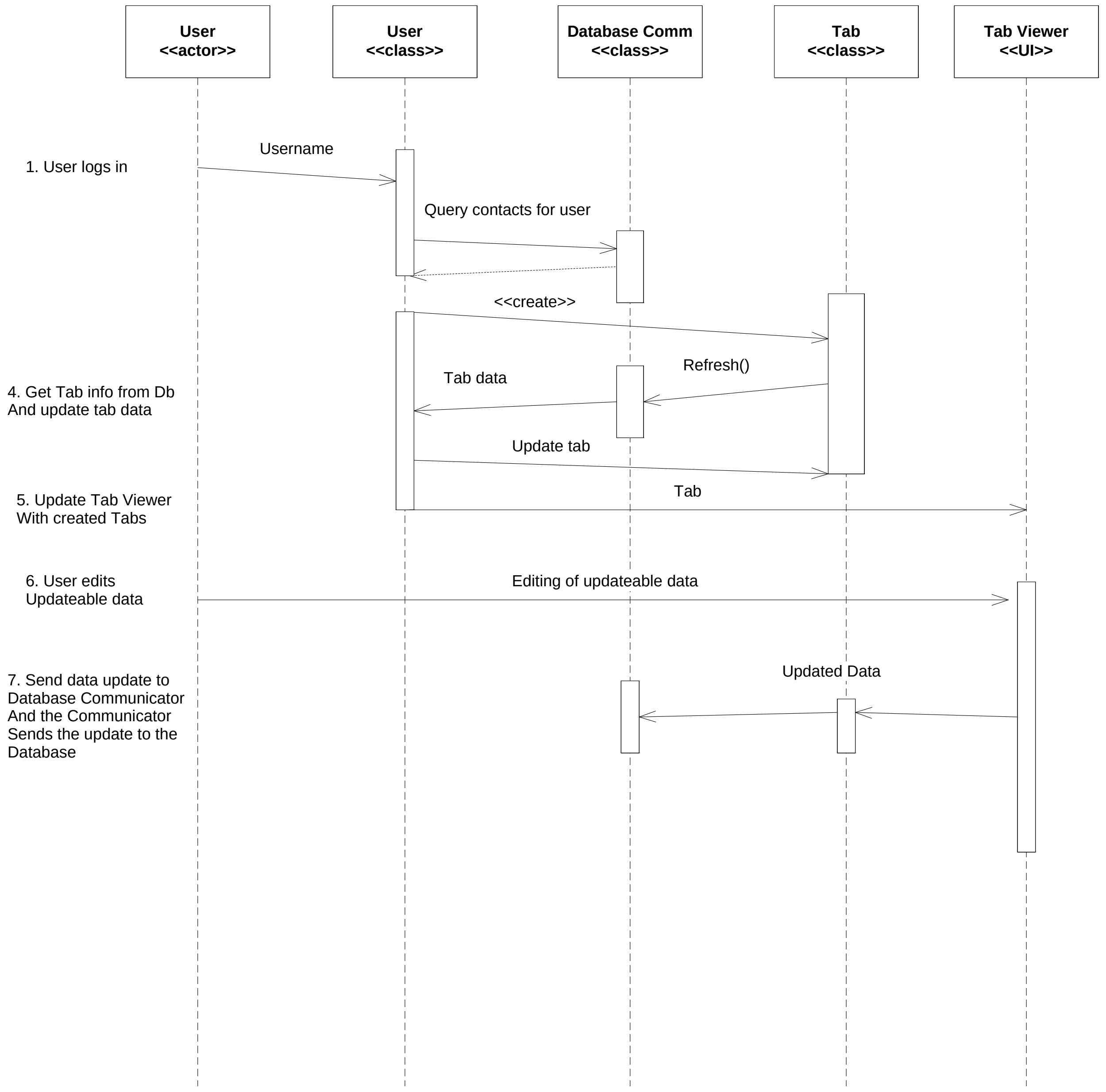
[3.2] Detailed Component View

The DatabaseCom and DatabaseConn classes work together to create the DB Communicator/Parser component.

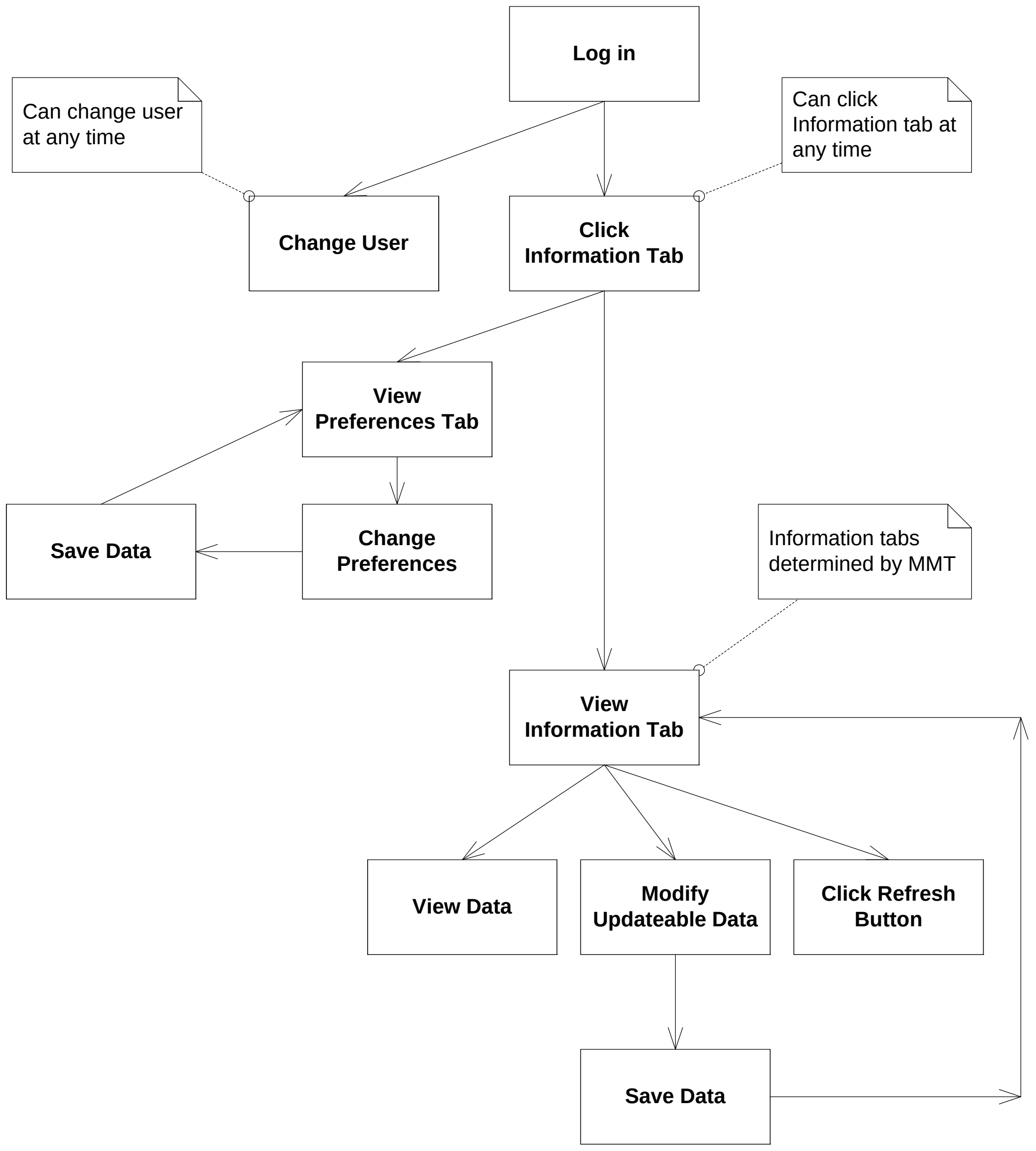
The User class functions as the User Data Manager component.

The Tab, Graph, and DataTable classes are all part of the Tab Data Manager component, and the TabViewer class functions as the Tab Viewer component.

[3.3] Process View


User <<actor>>
User <<class>>
Database Comm <<class>>
Tab <<class>>
Tab Viewer <<UI>>
1. User logs in
Username
Query contacts for user
<<create>>
Refresh()
Tab data
4. Get Tab info from Db And update tab data
Update tab
Tab
5. Update Tab Viewer With created Tabs
6. User edits Updateable data
Editing of updateable data
Updated Data
7. Send data update to Database Communicator And the Communicator Sends the update to the Database

[3.4] User Interface Flow Model



A field is only editable in the TabViewer if it is specified as updateable by the MMT. This restriction should remain in place because allowing the user to edit fields not able to be written to in the database would not serve any purpose and may lead to confusion.

[4] Appendix

No additional information.

[5] Glossary

Property View → The name of the widget.
DatabaseCom → Database Communicator Class
DatabaseConn → Database Connector Class

[6] Revisions

No revisions have been made at this time.