Reputation management for airline companies

The B.O.M.
The Ball Of Mud
Omar Pakker, Chiel Peters, Mary Gouseti, Cindy Berghuizen

September 6, 2013

Introduction

There are currently many outlets on which customers can leave feedback about the quality of an airline: tripadvisor, booking.com, twitter or any social network site where people can post reviews. However these reviews are scattered across the internet. This makes analysis hard for clients, airline companies and other users. Therefore a new system is being developed that collect all the different reviews together.

The purpose of this document is to research the different possibilities of the system's architecture. Section two contains the key quality attributes of the system. In section three the use cases are stated and explained. Finally in section four the solution proposal for the architecture is given.

Key quality attributes

In order to build the architecture first the key quality attributes need to be stated. The quality attributes question how the key components of the system should behave. No answers are given in this section however it points out the main questions which aid in the design and trade offs later.

Availability

Availability is one of the key factors for the cliens such as airline companies. The system should be build according to their needs. Therefore a important question is : How does the system reach the end users?

Scalability

The amount of data gathered by the system is quite a lot. The system should be able to efficiently store and handle this data. For now it is unknown how many users will make use of this review system. However in order for the system to be

a success it should be accessible to a high amount of users. Therefore trade-offs need to be made related the following questions:

- How much resources need to be allocated in order to keep our system up to date with the external sources?
- How many active users should the system be able to handle at the same time?
- How will our system access the external datasources and what will we do with it?
- What kind of (temporary) storage for the rapidly changing data is needed, if any is needed.

Modifiability

The system is depending on external sources. Those external sources are part of the everchanging internet. Also, our stakeholders will have a wide variety of needs which may change over time. The important questions about this are:

- How can our system cope with changes in external data sources?
  - What if another external datasource has to be added or an external datasource changes it's structure?
- How can we modularize our system to keep changes from stakeholders maintainable.

Scenario Analysis

The most important questions from the attribute are:

- How much resources need to be allocated in order to keep our system up to date with the external sources?
- What kind of (temporary) storage for the rapidly changing data is needed?
- How will our system access the external datasources?

In order to answer these questions first an overall figure of the system is given in Figure 1. Throughout an interface the user will request for data (e.g. airline information). Then the purpose of the server is to gather the data and show the data to the user in a given format. The process of data gathering is the key attribute of this system. In this document three different data gathering processes are explored:

1. Immediate request: No storage is allocated and the request is forwarded to all external sources immediately. This process is depicted in Figure 2.

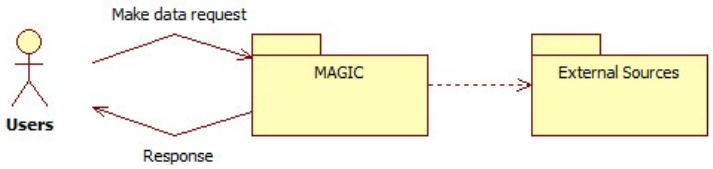

Figure 1: Overall use case

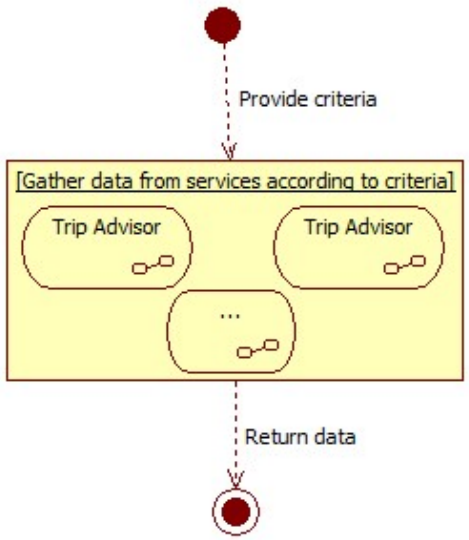

Figure 2: Real time requesting of the airline reviews

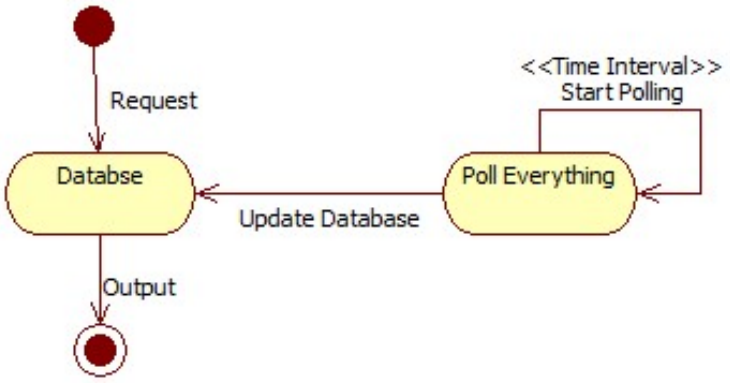

Figure 3: Global polling of the external resources to a database

2. Global polling: The data from the external sources (twitter, tripadvisor, etc.) is stored in a seperate database. A polling mechanism is put in place to synchronize the database with the external sources. One mechanism/process exists to update the external sources in a sequence. Data request from users will result on request on the database. This process is depicted in Figure 3.

3. Independent polling: The data is stored from the external sources into a seperate database. However instead of one process polling the external sources these can be seperated. For example some sources change more frequently than others therefore it makes sense to poll at different time mechanism's. Data request from users will result on request on the database. This process is depicted in Figure 4.

Although the real time architecture would be ideal towards the clients, it is however not feasible within the scalability quality attribute. As this would require non-stop forwarding and collecting data requests from the external sources and would lead to a high amount of bandwith. Therefore to avoid non-stop polling of the same data a database was added. However this also introduces problems related to keeping the database up to date. The database can be updated from the external sources by a polling mechanism. A global polling mechanism would be the most simple solution however this does not account for the differences between the external sources. For instance, twitter is more volatile than tripadvisor. Hence an independent polling mechanism would be more appropriate as it can account for these differences and can also occur asynchronously. Independent polling also enhances the modifiability of the software. A small polling module for a new datasource can be added instead of making larger changes in the module layer. Also, when an external datasource changes we can modify the module it is linked to.

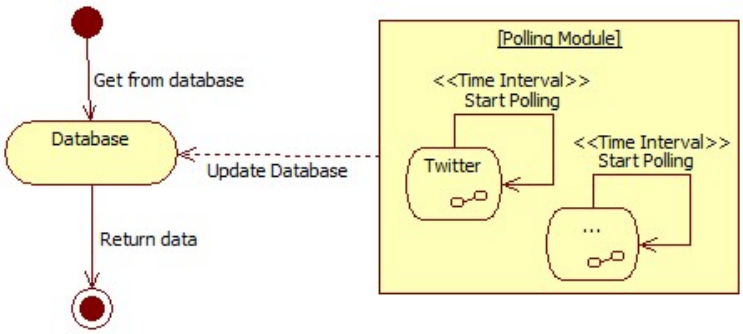

Figure 4: Independent polling of the external resources to a database

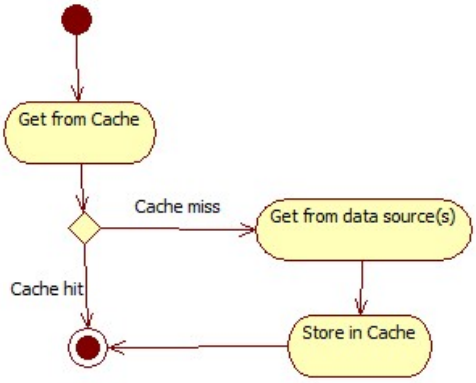

Figure 5: Cache structure

Getting data from the database to the user can also be done in different ways

1. Send a request to the database which will return the data
2. Send a request to a cache, which will see if the data is still (partly) in the cache. If the data is present it will check it's validity. When the data is not in the cache or is not valid anymore the data will be gathered from the database and stored in the cache. This procedure be seen in Figure 5

Since there will be a lot of data present in the database, searching through the database can be a slow process. The cache structure will speed up the searching process if the data is already present in the cache.

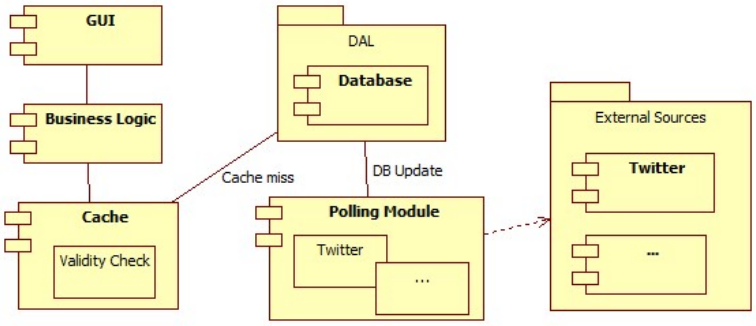

Figure 6: Architecture proposal

Architecture proposal

In order to answer the questions concerning the key quality attributes we propose the architecture presented in Figure 6.

The proposed architecture consists of separate layers to increase modifiability and maintainability. For instance, the GUI and the Business Logic are separated, as a result when the GUI has to change it will have no effect on the rest of the program; this will make it easy to a add a new interface e.g. a mobile interface. The Business Logic layer will handle all the requests from the GUI layer by processing the data coming from a temporary storage.

From the Business Logic Layer the request will be directed to the Cache. If there are parts of the data in the cache it will check if the data is still valid using one of the algorithms currently available for cache invalidation. If the data is not valid or the data is not in the Cache the request is directed to the database. By using a Cache and reducing the number of requests sent to the database we reduce the workload and achieve better perfomance and scalability.

In order to gather data the database is connected to a polling module. The polling module will handle the polling of the data from the external sources. The polling module consists of different smaller modules. Each module is optimised to harvest data from a specific external source according to its characteristics. Additionally, the Data Access Layer will be responsible to transform the harvested data to a universal format, e.g. xml, JSON etc, and send the data to update the database.

References

[1] Edmon Begoli and James Horey, *Design principles for effective knowledge discovery from big data*, http://blackboard.uva.nl/bbcswebdav/pid-4358526-dt-content-rid-5411006_1/courses/2318N001.5364SOAR6Y.S1.1.2013/Begali-wicsa2012.pdf.

[2] Ashish Thusoo et al and Facebook, *Data warehousing and analytics infrastructure at facebook*, http://borthakur.com/ftp/sigmodwarehouse2010.pdf.

[3] Sergey Melnik et al and Google, *Dremel: Interactive analysis of web-scale datasets*, http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/nl//pubs/archive/36632.pdf.

[4] Jason Evans, *Scalable memory allocation using jemalloc*, https://www.facebook.com/notes/facebook-engineering/scalable-memory-allocation-using-jemalloc/480222803919.

[5] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, *The google file system*, http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/nl//archive/gfs-sosp2003.pdf.

[6] Oracle, *Oracle information architecture: An architects guide to big data*, http://www.oracle.com/technetwork/topics/entarch/miscs/oea-big-data-guide-1522052.pdf.

[7] Patterns and practices Developer Center, *Microsoft application architecture guide, 2nd edition*, http://msdn.microsoft.com/en-us/library/ee658098.aspx.

[8] Randy Shoup, *Scalability best practices: Lessons from ebay*, http://www.infoq.com/miscs/ebay-scalability-best-practices.