Smart Packaging Cache

June 21, 2013

Pol Algueró Raga

GRADUATION / INTERNSHIP REPORT

FONTYS UNIVERSITY OF APPLIED SCIENCES

HBO-ICT: English Stream

Data student:	
Family name , initials:	**Algueró Raga, P.**
Student number:	**2211668**
project period: (from – till)	**February 4th – June 28th**
Data company:	
Name company/institution:	**Pyton Communication Services B.V.**
Department:	**Software department**
Address:	**Schatbeurderlaan 10, 6002 ED Weert**
Company tutor:	
Family name, initials:	**Lind, P.**
Position:	**Senior Technical Application Engineer**
University tutor:	
Family name , initials:	**Schriek, E.**
Final report:	
Title:	**Smart Packaging Cache**
Date:	**21/06/13**

Approved and signed by the company tutor:

Date:

Signature:

Foreword

I'm Pol Algueró and I developed this project abroad during my last semester. I'm from Barcelona, and I went to Eindhoven to finish my last year of career. I developed the project in Weert, a small town in the south of Eindhoven. The company is focused in software developing and it's called Pyton Communication Services, B.V.

During this period I was guided by Patrick Lind, my company tutor and Erick van der Schriek, my Fontys tutor and I also received the help from my workmates. I would like to say thanks to all these people for their advice and support during this project, which helped me to feel more comfortable during my stage.

TABLE OF CONTENTS

Attatchments

Summary

This project has been carried out in Pyton Communication Services B.V., a small company in the south of Eindhoven that develops software for travel agencies providing them with the full website or some functionality related to the website.

The current situation is that Pyton has the Waverunner Smart Shopping system, a system that communicates with Pyton providers in order to get the individual price of each segment, therefore they can offer to their clients packages with flight, accommodation and transfer.

The main problem is that Pyton can only offer to each client a full package with the prices from the client's providers. So the objective of the assignment proposed by the company is to build a cache database, which will contain all the prices of the different segments from all the providers that Pyton has. Therefore, the clients will be able to show more and better prices to their customers. To do that I will have to ask the providers for the prices, and the ones we need in order to not collapse the database. Once the system is ready, the original idea was to build also a best offer system, which was supposed to show the best offer based on the displayed search result.

The process followed to build this project consisted of research, document, implement and test this four parts of the system: the database, where all the price information are stored but also we store some price rules in order to calculate new prices based on the ones in the database, and the information about the most searched cities in order to update them with more frequency. The second part consists of a query generator, which generates a request to the providers depending on the most searched city. The third part is the one in charge of processing all the responses from the providers and stores the price information in the database. Finally, the last part to build is an exporter system, which exports all the data from the database and convert it into an easy-to-read format thus our clients are able to process it and show it in their websites.

But after building the cache database system, we realized that it took too much time to communicate with the providers in order to store the price information, and since it was the providers fault, we couldn't do anything. Thus we decided to implement a new functionality instead of starting the best offer system. This new functionality consisted of an offline mode. Therefore, instead of asking for the price information to the providers, we import this information from text files with all the price information. After some tests, we discovered that the offline mode was much faster than the online mode because we didn't depend on the provider's response time. So at least it was worth to add this offline functionality to the system as the time is critical in this project.

Glossary

C#	C Sharp
Cache	Stores data so that future requests for it can be served faster
CSV	Comma-Separated Values
Dump	Exports the information from the desired input
Segment	Accommodation, Flight or Transfer
SQL	Structured Query Language
XML	Extensible Markup Language

1. Introduction

Can you imagine planning all your holidays, including flight, accommodation and transfer from the airport to your accommodation, with only one click? With this project you will be able to do it!

Pyton, a small company situated in Weert, develops software for travel agencies in order to make them searchable, comparable and bookable in a big digital travel guide.

The project assignment consists in building a system that will request and store all the prices from the flights, accommodations and transfers of all Pyton's providers in a cache database. Therefore with this system, all Pyton's clients will be able to show in their websites a more complete holiday package offer than before, because this system will provide them with real prices, as it's supposed to get updated every day.
The "problem" is that Pyton's clients are only able to show in their website the prices from the segments of their own providers, so if there are others providers with cheaper prices they won't show it. Now, this system will let our clients to show segments from providers that they don't have, so they will be able to show the best prices to their customers, regardless the providers they have, because we will offer them all the prices from all of Pyton's providers.
If the system works as expected, the next step will be developing a system to show the best offers related with the customer search. For example, if one customer is searching flights to Barcelona, this system will show him the best/cheap hotels in Barcelona.

The strategy to follow will consist in two main parts: The first one will consist of all the research and the design of the full system. The second one will consist of all the implementation process and the test that will be done during this implementation.

During this report you will read about my experience working abroad, and which problems did I have to solve to build my project. All this information is organized in different chapters: Chapter 2 provides all the information about the company, chapter 3 gives all the details about the project assignment, chapter 4 discusses the research part of my internship and the following chapters describe all the process I followed during my stage. The last chapter is the conclusions, where I will explain the overall project completion.

2. The Company

Figure 1. Logo of the company

Pyton Communication Services B.V. is located in a small town in the south of Eindhoven called Weert. It was founded on 1992 and consists of 20 workers in Weert. They have an agreement with a company in Poland and another one in Germany that permits them to have extra people for developing some projects. The structure of the company comprises three departments: a software department, a product management department and a sales support department. As the main activity of the company is to develop software for the travel agencies, the software department is the biggest one with 13 people, including myself.

Pyton Communication Services B.V. offers online multichannel portals for tour operators and travel agents in the tourist industry. With over 20 years of experience in information technology, Pyton knows the travel sector like no other. Pyton develop products and services at their own risk and cost, thus providing customers with innovative, tailor-made solutions which offer the best value for money.
As nowadays they have a big influence in the Dutch market, their future intentions is to try to introduce them into the German and Polish market too. [4]

In this organization chart below, you can see the structure of the company and my position on it.

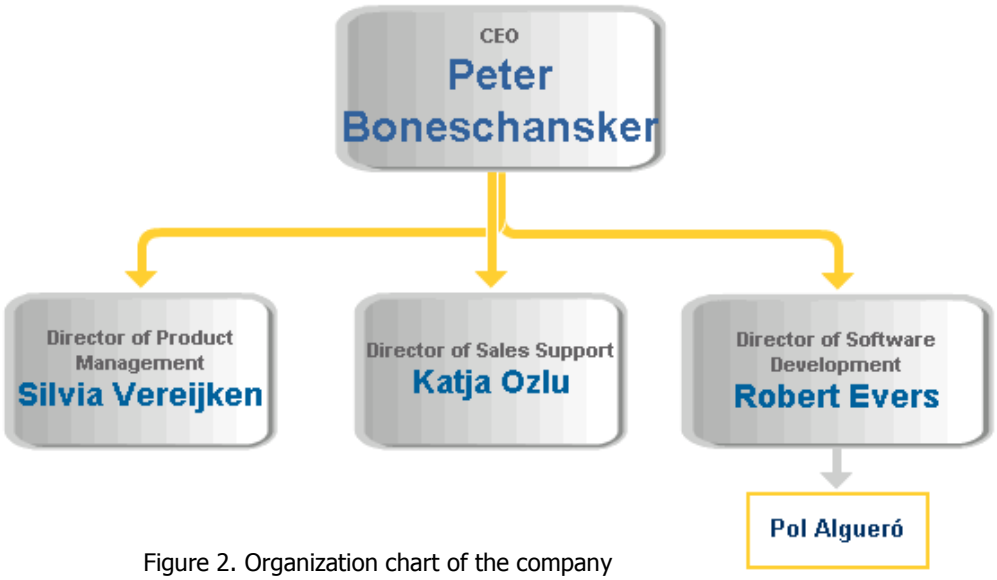


Figure 2. Organization chart of the company

As you can see, I'm working in the software development under the supervision of Robert and also Patrick Lind, who has the role of my company tutor.

3. Assignment Overview

3.1 Cache System

The current situation is that Pyton has the Waverunner Smart Shopping system, which permits Pyton to communicate online with their providers to get the individual price of each segment of the travel products and offer to their clients individual segments or full packages with flight + accommodation + transfer.

Figure 3. Promotional image of the travel pack

Pyton wants to improve this system because now they can only offer to a client, a full package with the segments that the client's provider can offer. To improve it, they ask me to build a cache database where I will store all the prices of all the different segments from all the Pyton providers, so they will be able to offer to their client a full package or individual segments like before, but regardless of the client's provider. Thus our clients will be able to show the best prices to their customers, because they will use all of Pyton providers.
The system will consist of different subsystems as you can see in the image below:

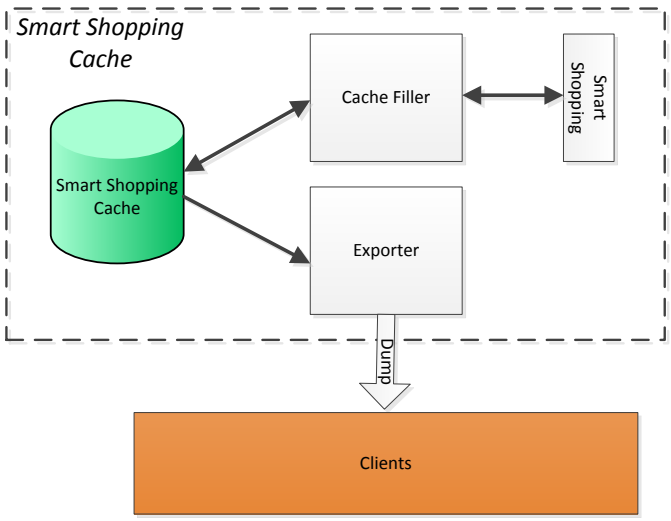


Figure 4. Cache system schema

The first project to be built will be the cache system, as we agreed upon with the project team, the priority is to build first the cache system and after it begin building the best offer system because the best offer system will use the cache database that will contain the first system.

The method of working during all this phases will be an incremental build mode. I decided to work with that method because with this method I can divide the full project in small phases and start researching, designing, implementing and testing each little phase until the product is finished. I also choose it instead of a waterfall method because with a waterfall method I will have to do the researching, designing, implementing and testing with the full project instead of little phases.

Thus the project will consist of these little phases:

- **Initial Research:** The first phase of the project will consist of an initial research of all the resources that exists in the company and a first contact with the templates they use in Pyton in order to start documenting all the project.

- **Database:** The first part of the system that I will focus on is the database because all the other phases will need the database in order to do their job. The database will contain all the price information of all the three segments, the price rules, and the information about the most searched places.
  First, a research will be done to decide which tables are needed and which is the best structure for them. The second step will be implementing it using the SQL Server Management. Finally, some test will be done to measure the performance of the database, and see if it's enough fast for this project.

- **Fill the cache:** This part of the system is the core of the system, so a lot of research will be needed to decide which is the best design in order to performance as fast as possible. It will be the one in charge to communicate with all the providers, in order to receive the information required, and store it into the database. After the implementation a lot of different tests will be done because as the core of the system, it's very important that the communication between the providers and the database works perfectly.

- **Export data:** The last part of the system will be the one in charge to export all the data from the database to a csv file for the clients and convert it into an easy-to-read format for the clients. This is the last part of the system, because it needs all the data stored in the database, therefore the database and the system to store all the data into it is compulsory in order to work.
  The first decision will be which kind of system is the best one to export big amounts of data as fast as possible. After choosing the system and implement it, some test will be necessary in order to test the speed of the export will all the segments and also the conversion format.

3.2 Best Offer System

Also Pyton need to show the best offers based on the displayed search result (i.e. the customer search for a flight to Barcelona, so the system will display also the best hotels in Barcelona). This could be done using the cache system I will be working in the Waverunner Smart Shopping system. With this system the providers will be able to sell more segments easily to their customers.

Figure 5. Best offer - popular hotels

Figure 6. Best offer - popular destinations

The best offers generator generates best offers based on definitions stored in customer management. It is triggered by putting a message on a message queue. This message will trigger a refresh of the specified definition.
A trigger can contain any of the following actions:

- Update.
- Delete.

The frontend (SmartShopping B2C) only uses the best offers that are in the cache. Each item in the cache has a period it is valid. When this period is expired and the frontend reads this item from cache the following will happen in the frontend:

- The frontend still shows the expired item.
- The frontend puts an update request on the queue.

In the image below you can see a schema of the general view of the best offer system and all of its components:

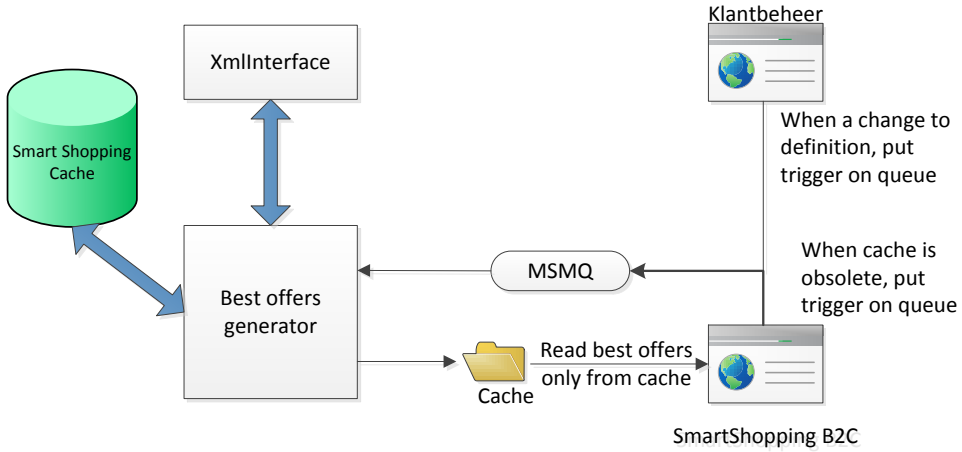


Figure 7. Best offer system schema

The best offer system will have the following phases:

- **Initial research:** The first phase of the best offer system will consist of a global research in order to understand how is supposed to work the system, and starting to make a general design of it.

- **Triger system:** After the initial research I will focus on the trigger system, so it will be possible to start sending the queries and updating or deleting them.

- **Rules:** Thus if the system is able to update or delete the offers, the next step is to design the set of rules that will decide which offers to show.

- **Calendar:** When everything will be finished, it will be time to design and implement a month calendar, that will show the best offer for each day.

3.3 Changes in the project plan

When the cache system was finished and after running some tests, the system was running too slow. The main reason was the time that the system spend trying to communicate with the providers. Because that time was not in the cache system scope, it was providers's fault, so we decided to implement an offline method. This was not planned in the initial project plan, but we had to look for some solution after the result of these tests. This offline method consists of import all the data throught csv files instead of requesting it to the providers. This new function of the system resulted to be much faster than the online request one, so at the end it was worth to add this new functionality to the system instead of begin working with the new offer system.

During my stage at Pyton, I will develop the project using the Microsoft tools that the company has as Visual Studio for coding in C#, SQL Server Management Studio for building the cache database, Microsoft Office for writing all the project technical documentation and also the Fontys documentation.

4. Research

The main research question for the cache system is: "*How can I build an online cache system for the providers that don't have it, and keep it up to date?*" For answering this question I will first have to answer all the different sub-questions that will appear as I do the project, i.e.:

- Which is the best size that the cache database should have?
- Which data should I export? An already made package or the individual segments so the providers can build the packages they want?
- Which is the best way to keep the cache system updated?
- Which is the best data format to use as less space as possible?
- Which is the best query format to communicate with the providers?

My main research question for the best offer system is "*How can I build a system that shows the best offers, of a particular segment, depending on certain rules?*"

As before, I will have to start answering more specific questions before it like:

- Which rules do I have to use to decide the best offers?
- How is going to work the trigger system along with the rules I defined.
- How I'm going to work with the cache obsolete offers.
- How I'm going to communicate with the front end and the xml system.
- Which is the best way to maintain all the offers updated as much as possible?

To answer all of these questions I will have to use different resources. The main resource will be the internet, but I will have to be carefully, because there's a lot of information in there and I have to choose for the right one. The main pages I will use will be the ones you can find in the research chapter.
A part from the internet I will use the knowledge of my work mates, especially the ones that appear in the organization chart of chapter 2.

5. Analyzing and designing the system

5.1 General Overview

The first phase of this project consists of research and document the design and how is going to work the system. This documentation phase will use as a template the one they use at Pyton. At the beginning I was quite lost, because I didn't know what I should start researching as it was my first time doing something like this. But after the first doubts, I start researching and documenting all the important information I found, even if I found 2 or 3 ways of doing the same, I documented it with the pros and cons of each method. I started documenting the cache database system, using as model a document with a brief explanation about what was supposed to be the "Smart Shopping Cache System".
I started to think how should work the full system from a general point of view, without diving too much details. Therefore I began writing a brief description of each component of the system: the cache database, the cache filler, the smart shopping and the exporter. After that I start describing and designing the database.

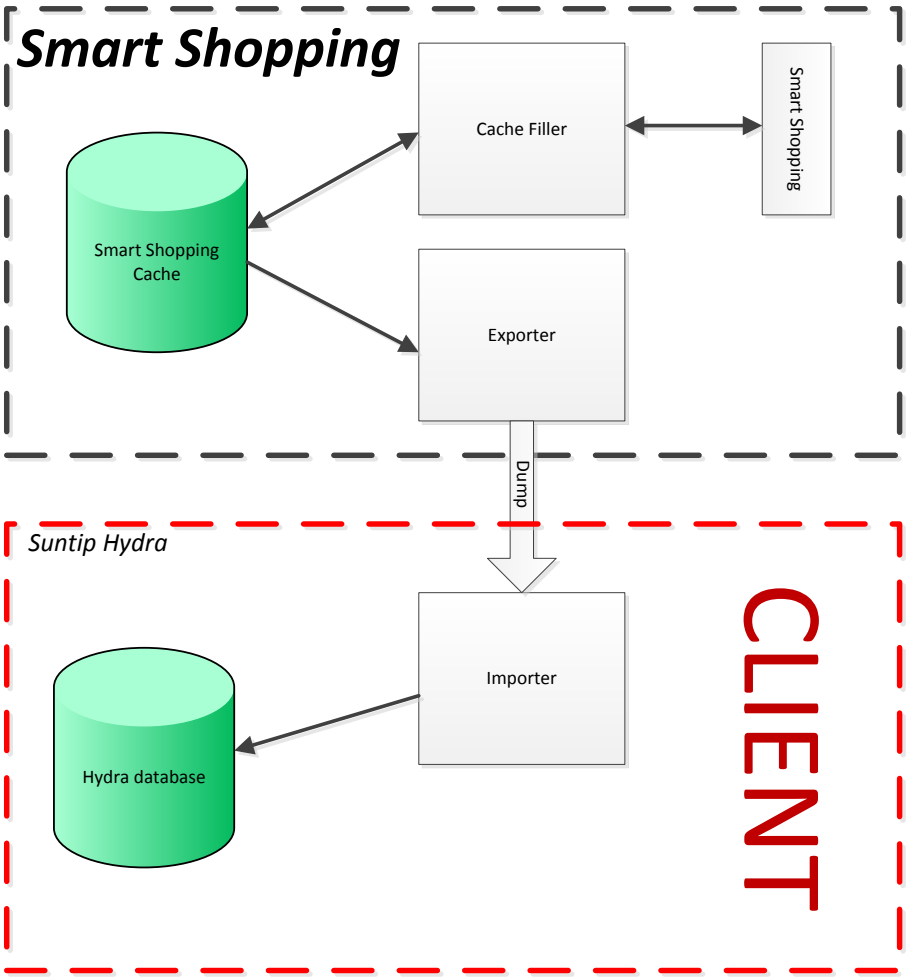
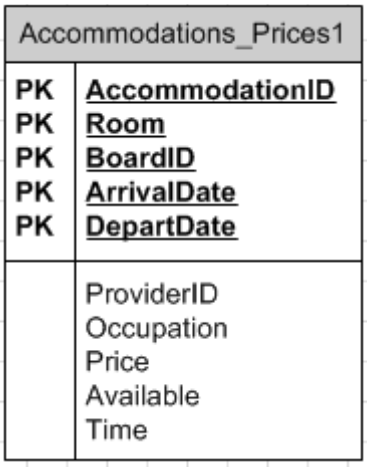


Figure 8. General cache system overview

5.2 DATABASE

The system was supposed to support three types of segments, so the first step to take was to start designing a table for each segment: one for the accommodations, one for the flights and one for the transfers. The main problem at this time was that I didn't know which type of information I should include in each table, because they had to be as small as possible due to the amount of data that I will have to store in each table. So the first design of the database only included the basic information of each segment, i.e. for the accommodations there were set some fields: an id to identify the accommodation, a couple of fields to identify the room and the type of board that was included, the arrival and depart date, the provider, the number of people that booked it and the price. But this table only contained the basic information about the accommodation and this was not enough, because it was needed also to control the quality of this data, as the database was supposed to get updated every day with new or more up-to-date data, so there were added a couple of fields: Available, that indicates if the data was available, and Time, that indicates the time that the data was stored in the database.
After getting some feedback with project members, the design of the database was updated in a way to make it more clear and easy to understand, changing some field names, adding some primary keys, and adding two more fields: "PlaceID" to identify the city of the accommodation and "RoomID" to store the id of the room. So at the end it looked like this:

Figure 9. First design of accommodations_prices table



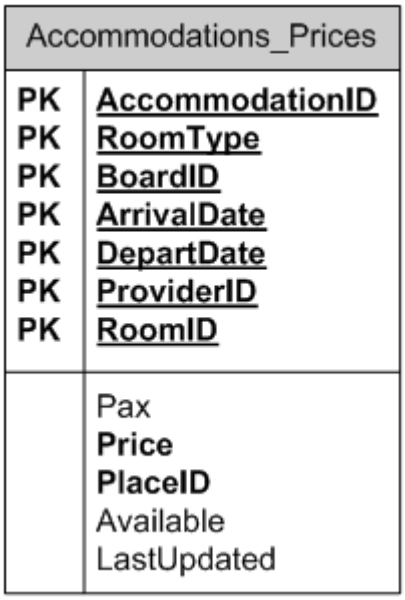


Figure 10. Final design of accommodations_price table

Once the design of the database was more or less defined, it was time to start deciding which type of data would have each field, and as before, as small as possible. So after researching I made a table with each field and its correspondent type, length and a brief explanation. To decide each type it was necessary to think which the range of each field was. [2]

- **AccommodationID**: It's an int because the range of the small int (±32,767) is not enough to identify all the different accommodations that Pyton can have with all the providers.
- **RoomType and BoardID**: These two fields contain letters and numbers so the best option in that case was a varchar as the type. After talking with the providers for the length, the answer was that the maximum length of these two ids was 30.
- **ArrivalDate and DepartDate**: In the accommodation segment, only the date was required instead of the date + time, thus was selected date as the type.
- **ProviderID and PlaceID**: The decision was similar to the AccommodationID, because the smallint type was not enough to embrace it all.
- **RoomID**: The reason of using a varchar 50 was because the id of the rooms contains numbers and letters, and after some researching which length should be used for this field, the conclusion was that with length equal to 30 was not enough to embrace the full id, and it was necessary to use 50 bytes.
- **Price**: The initial idea was to use a smallint instead of an int, but as we should store all the prices in cents, with a smallint it wasn't enough because the maximum price they can store is 32767, in our case 327,67€ (the price is stored in cents) so the right decision was to choose an int, as it has a bigger range, 2147483647 in my case 21474836,47€, more than enough.
- **Pax**: To indicate the number of people, the decision was to use a tinyint because the maximum range of it is ±250, therefore it was more than enough with it.
- **Available**: As the function of this field was to use 1 or 0 to indicate if the data was available, with a bit was enough.
- **LastUpdated**: Unlike ArrivalDate and DepartDate, this field also had the time value on it, so the data type chosen was the smalldate type in order to be able to store the date and the time that the data was updated.

So at the end the table fields definition was like this:

Field	Type	Length	Explanation
AccomodationID	INT	4	Id of the accommodation
RoomType	VARCHAR	30	Type of the room
BoardID	VARCHAR	30	Indicates the type of Board included
ArrivalDate	DATE	3	Arrival date
DepartDate	DATE	3	Departure date
ProviderID	INT	4	Id of the provider of this accommodation
RoomID	VARCHAR	50	Id of the room
Pax	TINYINT	1	Number of occupants
Price	INT	4	Price of this accommodation, in cents
PlaceID	INT	4	Id of the place
Available	BIT	1	Indicates if the data is available/usable 0 = not available 1 = available
LastUpdated	SMALLDATETIME	4	Indicates the last time data was updated

Figure 11. Accommodation fields table

5.3 XML Request and Response

When the design and description on each table was done, it was the right moment to start researching about the xml request [3] that should be sent to the providers. As I had some idea about the data I was supposed to handle in each segment after designing the database, I used it to design the xml request. I also took some examples that were already in other Pyton projects, so I could follow the general schema of these xml requests. As with the database, the xml schema was supposed to use as less data as possible, so the first step was to start thinking which information was needed to look for an accommodation. At least it's needed the city where one would like to stay, the dates and the number of people. Therefore, after some researching in some similar xml request, the result looked like this:

```
- <SearchRequest>
  - <Provider>
      <Type>Accommodation</Type>
      <ID>1003</ID>
    </Provider>
  - <SearchAccommodations>
      <Destination Type="City" ID="6542"/>
      <Arrival Date="2013-08-04"/>
      <Depart Date="2013-08-11"/>
      <Occupation>2</Occupation>
    </SearchAccommodations>
</SearchRequest>
```

Figure 12. Accommodation request XML schema

Now that the xml request schema was ready, it was time to start thinking about the response schema. The response xml is the xml schema that will be created after the system receives a response from our providers, selecting only the important data that it's necessary to store, because normally they will send us an xml response with a lot of information. Now it was time to research[3] how was the response of one of Pyton's accommodation providers and thus decide which was the important information, i.e. they included links to images of the hotel and the room that were not important for the system. After some researching and discussion, the conclusion was to store the same information as in the database. Finally, with all the information it was needed, I started with the design of the xml response schema. In the accommodation case, I came up with two different options:

- Option 1: For each accommodation, it would list all the available rooms, and for each room, all the available boards with the own price addition. After all the boards of each room it would list the price information of the room along with the number of people that the price was calculated for, and the currency type of the price.

```
<Accommodation id="3310">
  <Provider>1003</Provider>
  <Rooms>
    <Room type="2PK" id="$MTM4NjEjREJUVyNEQiMqIzIjMDA=$DBTW">
      <Boards>
        <Board type="AB" addition="15"/>
        <Board type="LG" addition="5"/>
        <Board type="LO" addition="10"/>
        <Board type="HP" addition="0"/>
      </Boards>
      <Priceinfo>
        <Price currency="EUR" priceper="TP">115</Price>
        <Occupation>2</Occupation>
      </Priceinfo>
    </Room>
  </Rooms>
</Accommodation>
```

Figure 13. Accommodation response option 1 XML schema

- Option 2: Similar to option 1, but instead of listing all the boards types and its own price addition, it would list each board type with the final price including the room price.

```
<Accommodation id="3310">
  <Provider>1003</Provider>
  <Rooms>
    <Room type="2PK" ID="$MTM4NjEjREJUVyNEQiMqIzIjMDA=$DBTW">
      <Board type="LG">
        <Priceinfo>
          <Price currency="EUR" priceper="TP">120</Price>
          <Occupation>2</Occupation>
        </Priceinfo>
      </Board>
      <Board type="LO">
        <Priceinfo>
          <Price currency="EUR" priceper="TP">125</Price>
          <Occupation>2</Occupation>
        </Priceinfo>
      </Board>
      <Board type="AB">
        <Priceinfo>
          <Price currency="EUR" priceper="TP">130</Price>
          <Occupation>2</Occupation>
        </Priceinfo>
      </Board>
    </Room>
  </Rooms>
</Accommodation>
```

Figure 14. Accommodation response option 2 XML schema

At the end, the final decision was to implement the first one because it was the most similar from the one that the provider was sending to us as a response. Also it was the most organized and easy-to-read one.

5.4 Rules

The main point of the system is to ask for prices and store them into the database and to make this database useful for Pyton, it must contain a lot of prices. The perfect database would contain all the possible combinations for each week of the year. As it's impossible to do it, mainly because the system spends a lot of time each time that sends a request and receives an answer for this request, and also because it's too much data for the database. So some research about the way to store as much prices as possible spending the less time so it doesn't affect to the load of Pyton servers was required. Thus research leads us to some rules to make it possible. A decision was taken, and the system was required to ask for the prices for 2 and 8 days for 1 year period in order to apply these rules. Therefore after asking for all these prices it was the right time to apply some rules. I came up with 2 types of rules:

- **Simple**: The simple rules will calculate the prices for 'x' days depending on the prices in the database. So if we have a room that costs 300€ for 7 days and 2 people it means:

  300€ / 7days = 43€(2 people) -> 43€/2 = 21,5€ per person per day
  If we want to know the price for 6 days:
  21,5€ * 6 = 129€ per 6 days for 1 person.
  129€ * 2 = **258€ per 6 days and 2 people**.

- **Complex**: The complex rules will take 2 real prices (prices that we get from provider instead from rules), calculate the price per day and person, and at the end, make an average to know a more realistic price that with simple rules. So if we have these two different prices for the same room than before:

  300€ for 7 days and 2 people -> 21.5€ per person per day
  100€ for 2 days and 2 people -> 25€ per person per day

  We want to know the price of 4 days:
  $\frac{((21.5€ * 4 \text{ days}) + (25€ * 4 \text{ days}))}{2}$ = **93€ per 4 days 1 person**.

5.5 Cache Filler

The cache filler is the core of the system, so I had to pay attention to a lot of details to make it as more efficient as possible. The main function of the cache filler is to communicate with the providers in order to get the required price information, and after that, store it into the database if necessary. So the first task that was made was to research about the communication between the cache filler and the providers and I choose MSMQ (Microsoft Message Queuing). Why was MSMQ the chosen option?

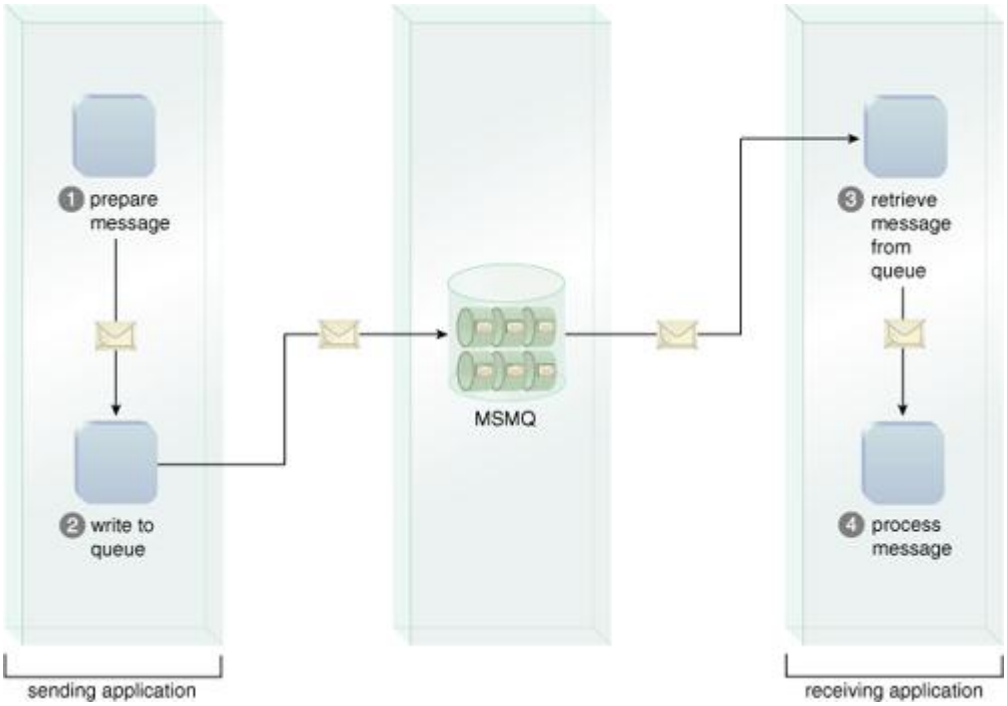


Figure 15. MSMQ functionality (source: mscerts) [5]

- **Asynchronous message processing:** The entire system request will be send to a virtual queue that will make sure that can process it one by one. And if servers go offline, it won't lose the messages; instead it will be able to read them when they turn online again.
- **Server load**: If there's a moment during the day that there is a peak in the server load, nothing will happen as the system will read the messages one by one and this supposes no extra load.
- **Microsoft friendly**: As MSMQ it's a Microsoft product and here in Pyton they use Microsoft products, there won't be any problems to integrate it with the rest of the system. [6]

After deciding which communication protocol should be used, it was time to decide which requests to send. The idea was to request the most searched cities, so that the system could update them more frequently than the cities that are not so popular. Thus an extra table was built to store all the cities, therefore the table contained a list from the most popular city until the less popular.

Now it was the moment to decide how to manage the responses. At least they had to be stored in the database because of the price information, but this was not the only function, as all the data from the responses, was supposed to be shown to our customers via Smart Shopping. For the first part the wise decision should be build some system that extracts the information we need from the response, and stores it into the database through some kind of data filter, because if we request the same data more than 1 time per day, it means that the database will receive a lot of duplicate data. For the second part, an "exporter" system that will export all the data to our customers in a unified format seemed to be the best solution. Therefore, after these decisions, the final design of the cache filler was this one:

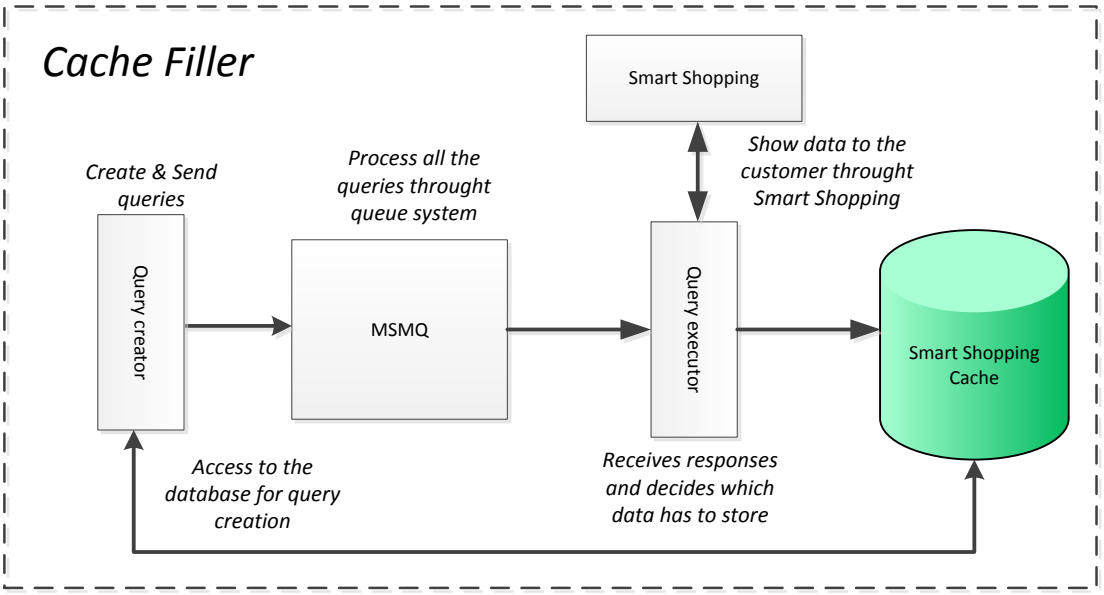


Figure 16. Cache filler design

At the end, the cache filler will consists of a query creator that will generate all the request depending on the most searched cities, and will send them to the MSMQ that will process it. The next step will be receive the responses with the query executor and extract from them the important data, and store (if necessary) it into the database. Finally, at least one time per day, an exporter function will export all this data to the customers.

5.6 Exporter

The main function of the exporter is to export all the existing data in the database to a csv file, and convert that file to a format that the customer is able to understand in order to send them all the data. As I had no experience with exporting big amounts of data, I asked my workmates if they knew any software or method that could satisfy my requirements. They told me to research for SSIS packages and so I did it:

- SSIS package is a Microsoft tool that it's easy to use once you understand the basics of how to handle it. It permits you to create packages via a visual toolbox. SSIS packages are visual work flows rather than a coding tool, so one can easily understand what is going on within a package by looking at the work flows instead of going through hundreds of lines of code.

- SSIS packages are built mainly to perform ETL (Extract, Transform, and Load) jobs. It is fine tuned to handle that functionality really well especially with SQL Server, which is the software we use in Pyton for managing the database.

- SSIS let you use different data sources/destinations as text files, excel files, and databases, so it fulfilled all my requirements. [1]

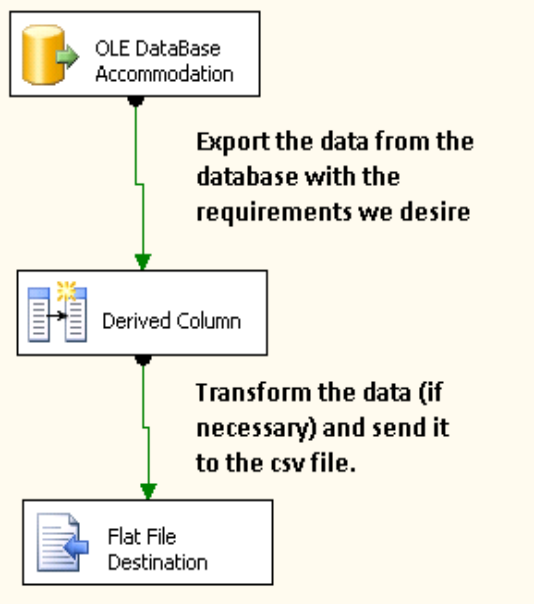


Figure 17. SSIS functionality example

In the image above you can see a brief example of the functions of a SSIS package.
Finally, after having a method to export all the data to a csv file, it was time to research how to convert all this data to a standard format and send it to the customers. As Pyton already had a similar project that converted a big amount of data stored in a csv file to a standard format, I decided to research how this project was working in order to understand it and try to adapt it to my requirements.

6. Developing and testing

6.1 Database

The first sub system meant to be built was the database. The existing software at Pyton for managing databases was the SQL Server Management Studio 2008 R2, so all the database was implemented with this software. The initial idea was to build only 4 tables, one per each segment and one extra for the most searched cities. But as time went by it was necessary to add more tables to satisfy all the requirements so at the end the cache database was composed by this 9 tables:

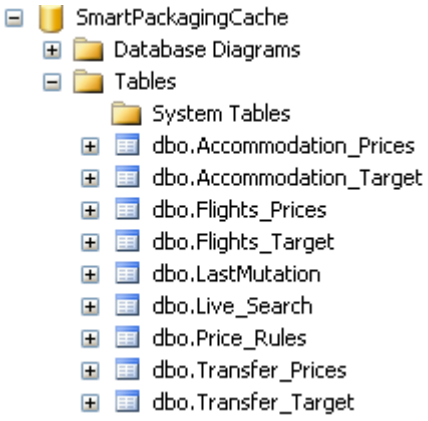


Figure 18. Cache database tables

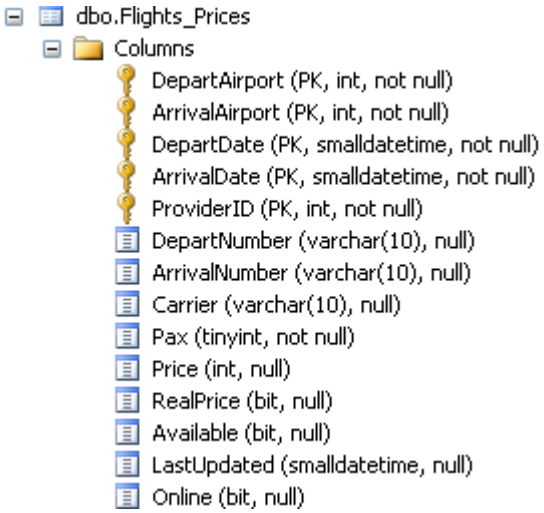


Figure 19. Flight_Prices columns design

- **Accommodation, Flights and Transfer prices**: These are the tables that were used to store all the price information of each segment. When the data needed to be extracted for the dumps, these were the tables used to extract the data from the database.

- **Accommodation, Flights and Transfer target**: These tables are the ones that are used when it is necessary to update the price information. All the new data received from the providers, is stored into these tables and at the end it is merged with the "prices" tables in order to store it properly, which means, without duplicating data. Also, when new data was received, the price rules were applied and at the end were merged with each correspondent in the price tables. This merge process, adds the new data to the price table, updates the existing data with the new prices, or sets to not available the data that we did not receive in the update but that was in the price table. After the merge, all the data from this target tables is deleted in order to prepare it for the next update.

- **Last Mutation**: Stores the date and time when the last mutation was processed for each segment.

- **Live Search**: Stores all the cities with a count field so the system can know which cities have to ask for the price information.

- **Price Rules**: This table contains all the price rules, so they can be executed from the code in an automated way. The table contains 2 kinds of rules: simples and complex.

6.2 Query Creator

When the database was implemented and working, it was time to start coding the cache filler. The first part that was meant to be coded was the "*Query Creator*" because it's main function is the creation of the queries. The *query creator* is the one that communicates with the database in order to get the most searched cities so they can be sent as a request to the queue. The initial idea was to create the xml request in the *query creator*, but it made no sense because it was better if the xml was built in the *cache executor*, and leave the *cache creator* with the only mission of sending requests with the basic information to the virtual queue, and build the xml request in the *query executor* after receiving it from the queue. Thus a test queue was created to start sending the request with the basic information.

Figure 20. Test queue

6.3 Query Executor

When it was finished, the "*Query Executor*" was started because the *executor* it's the one that process the queries from the *query creator*, so it was necessary to finish first the creator and after it, start with the executor in order to process the queries from the *creator*. The *query executor* had to receive all the queue messages, one by one, and process them. The first step is receiving the message and building the xml request with the information inside the message. Once the xml request is build, it was time to send it to the providers depending on the type of segment and wait for a valid response. When an answer was received, the system had to process it in order to extract all the important information that was required. Therefore there were 3 different cases, one per each segment, as the important information would vary depending on the segment. As the database had a limited space, it was necessary to choose some criteria in order to store the less possible data:

- **Accommodations**: For each accommodation, it will store only the cheapest room for each type of board.

- **Flights**: It will store only the cheapest flight regardless the carrier and the stopovers.

- **Transfers**: It will only store the cheapest transfer, regardless the type of transport and the duration.

Once this part of the system was ready, it was time to start working with some data, so the first tests were ready to be made. The first provider that was used for the test was TransHotel. TransHotel as its name indicates it's an accommodation provider. The first thing that was meant to be done was to adapt our request to TransHotel data format so when our request was send to them, their system was able to understood it, and therefore they would be able to answer us with a valid response.
Thus I decided which cities I would use for the testing, and as you can see I choose these four cities:

	PlaceID	PlaceName	Count
1	207	Amsterdam	5
2	561	Barcelona	15
3	956	Budapest	30
4	1772	Eindhoven	4

Figure 21. Live_Search table example

In the image above, you can see the four cities with its count field. This means, that the first city that should be requested would be Budapest, and the last one Eindhoven. The first tests were made asking only for a 7-day-period during the month of August (4 weeks).
In the picture below you can see an example of the request that was send to TransHotel. In this case the requested city was Barcelona (ID = 561) from the 1st until the 8th of August for 2 people.

```
<SearchRequest>
  <Provider>
    <Type>Accommodation</Type>
    <ID>1003</ID>
  </Provider>
  <SearchAccommodations>
    <Destination ID="561" Type="City" />
    <Arrival Date="2013-08-01" />
    <Depart Date="2013-08-08" />
    <Occupation>2</Occupation>
  </SearchAccommodations>
</SearchRequest>
```

Figure 22. Request to TransHotel example

And here you can see a brief example of the response after process and selecting the important data needed to be stored.

	Accommo...	RoomTy...	Board...	ArrivalDate	DepartDate	Provider...	RoomID	Pax	Price	PlaceID
1	3310	2PK	LO	2013-08-01	2013-08-08	1003	$MTM4...	2	62937	561
2	3319	2PK	LG	2013-08-01	2013-08-08	1003	$MTM4...	2	81997	561
3	3322	2PK	LG	2013-08-01	2013-08-08	1003	$MzM0N...	2	71154	561
4	3567	2PK	LG	2013-08-01	2013-08-08	1003	$MTM4...	2	9167	561
5	3570	2PK	LG	2013-08-01	2013-08-08	1003	$MTM5...	2	59835	561
6	3975	2PK	LG	2013-08-01	2013-08-08	1003	$MTM4...	2	110195	561

Figure 23. Accommodation_Prices table content example

In the image above you can see 6 different accommodations in Barcelona, all of them with the same arrival and depart date, with its own room type and board type and the final price in cents. You can also see the identifier of each accommodation, provider and room.

Now that this part of the system was working, it was the moment to start researching about the time duration that the software was spending for requesting and processing the responses. In that case there were two "types" of spending the time:

- **Request time**: This is the time that the software spends communicating with the providers in order to send the request and receive a valid response. This time cannot be improved because depend on the providers.

- **Database time**: This is the time that the software spends communicating with the database. This time can be improved (if possible) because it depends on our database.

Request Time	Database Time	Total Time
16,66	1,3	18,24
16,58	1,5	18,1
21,1	1,35	22,43
16,1	1,3	17,4
70,44	**5,45**	**76,42**

Figure 24. Time test result

After running the same test as before, but this time debugging the time that the system spend in every part of the code, it was time to build some kind of table and analyze the time. As you can see in the image above, during the most part of the time, the software spends the time contacting with the providers, which means that I cannot do much about it as it's a provider "problem". In the other hand we have the database time, that represents a 7% of the total time, that's the time I'm able to improve, but in that case is not that high.
When the accommodation testing was finished, it was time to do the same for the flights and transfers using their respective providers.

6.4 Exporter

So now it was time to start implementing one of the last parts of the cache system, the exporter. There were two types of export:

- **Full dump**: the full dump will be generated once per day, and will export all the available data from the database
- **Mutation dump**: the mutation dump will be generated as many times as possible during the day and will export all the data since the previous dump.

With the project team members we decided to have two types of dumps in order to make them as efficient as possible. While the full dump will be executed during the nights, and once per day, it means that during the day we can have another dumps but not as big as the full dump, because the load from the servers is quite big during the day but much smaller during the night. Therefore we decided to use the mutation dump during the day, and with many cities as possible, but always trying to not increase too much the server load.

Therefore the first thing to be coded was a SSIS package as you can see in the image below. This package implements the full dump export, and its main function is to export all the necessary data from the database. When this is finished, it updates the *LastMutation* table with the date and time when the export was done, and at the end, it deletes all the data that was not available from the original table.

Figure 25. SSIS package full mutation example

The main problem during the implementation was trying to select the necessary data for the mutation dump, thus I decided to implement one extra table for each segment and that helped a lot managing the new data. Therefore each time one segment needed to be updated, the system inserted all these new data to the new table (*target* tables), and after it merged with the original one. The merge process had to control 3 possible cases:

- **Insert**: When the new data received didn't exist in the original table. This means that it has to be inserted into the original table as new data.

- **Update**: When new data is received, but these already exists in the original table, mean that it has to be updated with the new price, if it is different from the old one, and make it available.

- **Delete**: When data that already exists in the original table is not received. That means this data needs to be set to not available, so in the next mutation the client will know that they have to delete it.

At that time, the system was able to export all the data through both dumps to a csv file. In the picture below you can see an example of a full dump:

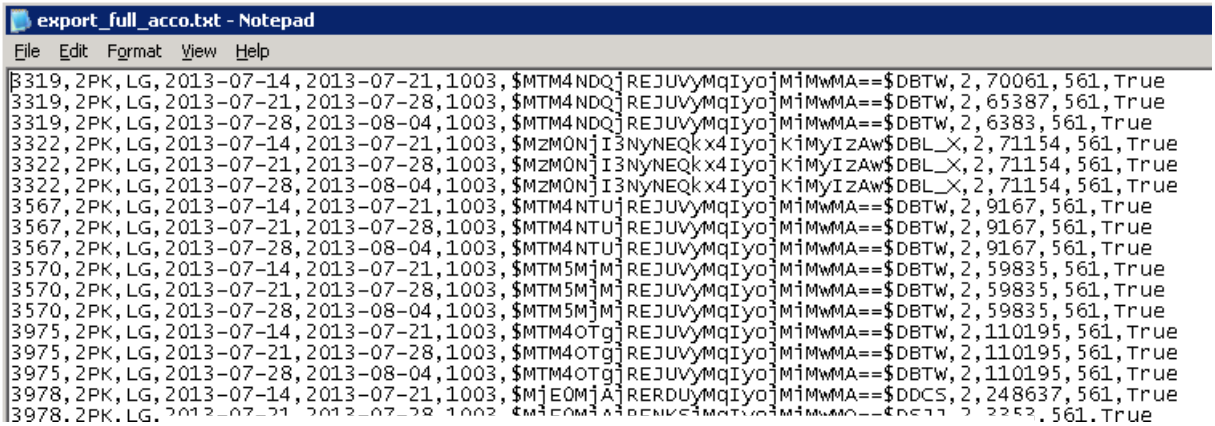


Figure 26. Accommodation full dump export example

6.5 Convertor

Now that the system was able to extract the data I was interested in, it was time to convert it into a format that the client was able to process it.
So the first thing that needed to be done was research one of Pyton's projects that were also converting some data. I tried to adapt it to my needs but it wasn't as easy as expected. The main problem was that the system needed to process a lot of data from the files and try to organize it as much as possible in the output. Unfortunately not all the data needed was in the csv file, because this file was as small as possible, thus the system had to access also to the database to recollect all the data that was needed and that wasn't in the csv file; i.e. in the accommodation example that you can see above, I don't have the name of the hotel, the country, the region, etc. It was also better to show this data to the customer in a way that makes it easy for them to read it. I organized the data in different groups, i.e.: In the accommodation example I put all the rooms of the same hotel together with each price, each type of board, type of dump, dates, etc. So at the end it looks like this:

```
<Accommodations>
  <Name>Atenea Calabria</Name>
  <Country>SPANJE</Country>
  <Region>BARCELONA</Region>
  <Place>BARCELONA</Place>
  <AccomodationID>3319</AccomodationID>
  <RoomType Type="2PK">
    <Room Provider="1003">
      <MutationType>New</MutationType>
      <RoomID>$MTM4NDQjREJUVyMqIyojMiMwMA==$DBTW</RoomID>
      <ArrivalDate>2013-07-14</ArrivalDate>
      <ReturnDate>2013-07-21</ReturnDate>
      <BoardType>LG</BoardType>
      <PriceInfo>
        <Price Currency="EUR" PricePer="PA">70061</Price>
        <Occupancy>2</Occupancy>
      </PriceInfo>
    </Room>
    <Room Provider="1003">
      <MutationType>New</MutationType>
      <RoomID>$MTM4NDQjREJUVyMqIyojMiMwMA==$DBTW</RoomID>
      <ArrivalDate>2013-07-21</ArrivalDate>
      <ReturnDate>2013-07-28</ReturnDate>
      <BoardType>LG</BoardType>
      <PriceInfo>
        <Price Currency="EUR" PricePer="PA">65387</Price>
        <Occupancy>2</Occupancy>
      </PriceInfo>
    </Room>
```

Figure 27. Accommodation dump conversion example

6.6 Price Rules

At this time the cache system was almost finished, so it was time to start coding the last part left: *Price rules*. I began with the design of the table that will contain all the rules. As I came up with two types of rules, simple and complex, I decided to build the table with 4 fields:

- **DesiredDays**: This field indicates the duration of the new price calculation.
- **FirstParam**: This field indicates the first condition to select the data, i.e. in the picture below you can see that the rules will apply to data that has 3 or 7 days duration.
- **SecondParam**: This field indicates the second condition to select the data. Only works for complex rules, because with the simple ones you only use one condition. I.e. in the picture below we will select the data that has 3 and 7 days duration, and will calculate the price for 4 days.
- **Type**: This field indicates the type of rule. This is helpful because the rules work different depending on the type.

	DesiredDays	FirstParam	SecondParam	Type
1	4	3	7	Complex
2	6	7	0	Simple
3	8	7	0	Simple

Figure 28. Price_Rules table example

Here you can see a transfer example after applying the rules in the image above. Rows 1 and 4 contain real prices, prices that were extracted from a valid response. Row 2 is the result of applying rule 1. Row 3 is the result of applying rule 2, and row 5 is the result of applying rule 3. In that case the price for 3 and 7 days is the same, so that's the reason that the price in row 2 is cheaper than the price in row 1.

	Airport...	Destinat...	DateTimeInbound	DateTimeOutbound	TransferID	Provide...	Price	RealPrice	Pax	Travel
1	22	561	2013-08-05	2013-08-08	355557	3001	2400	1	1	50
2	22	561	2013-08-05	2013-08-09	355557	3001	2284	0	1	50
3	22	561	2013-08-05	2013-08-11	355557	3001	2052	0	1	50
4	22	561	2013-08-05	2013-08-12	355557	3001	2400	1	1	50
5	22	561	2013-08-05	2013-08-13	355557	3001	2736	0	1	50

Figure 29. Rules application example

6.7 Offline Mode

At this point the cache system was finished so it was time to start with the best offer system. But after talking with Patrick we decided to improve the cache system and add some functionality instead of starting with the best offer one. We did this because the actual system was too slow therefore the time between the request and the response of the providers was taking too long, so we decided to implement an offline function. This offline function consists of importing to the database a series of csv files with all the segment information. Thus was similar to the exporter function that was already implemented, but in the opposite direction. Some files have more than 2 million lines, but it's even much faster to import it, instead of asking the provider for the same amount of information.
Another reason was that the cache system is the important system, and the best offer is a small system that will use the cache database that I created, therefore it was better to improve as much as possible the information stored in this database, so after it someone can build the other system using the improved cache database.

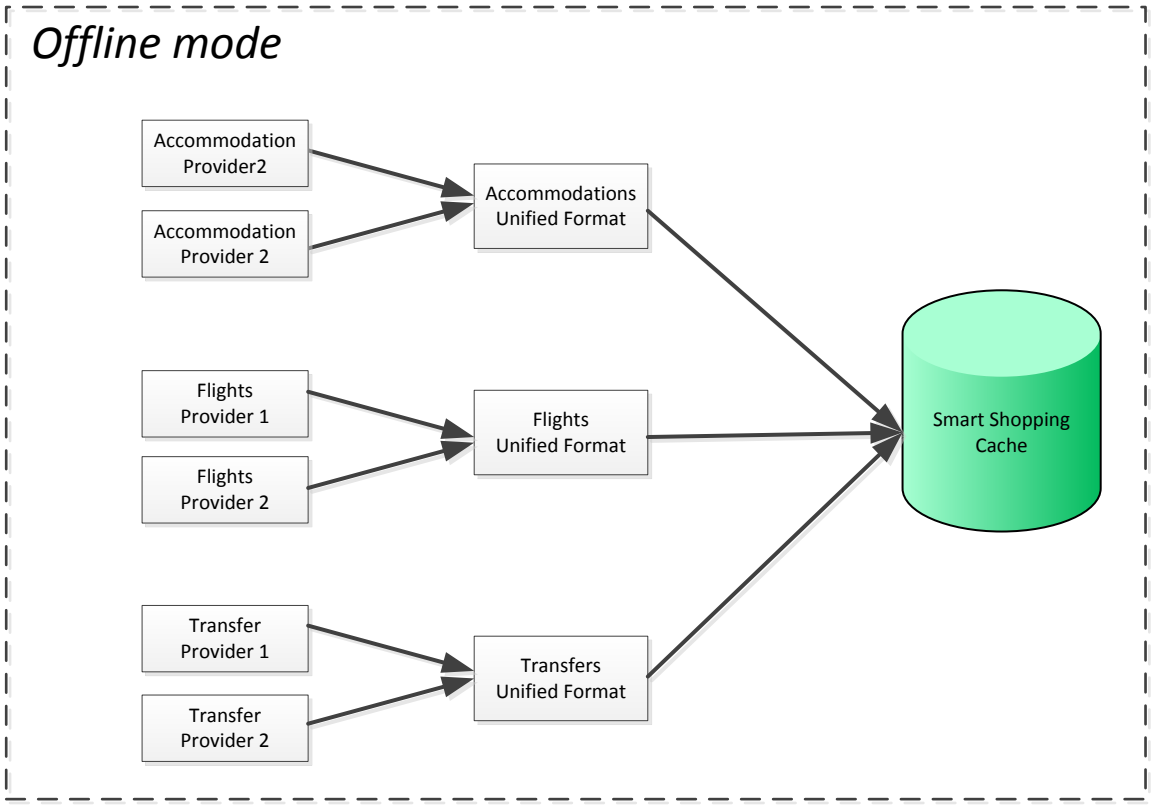


Figure 30. Offline mode design

In the picture above you can see the three step operation of the offline mode. The first step consists of designing small software for each provider, to be able to read all the data files they provide us. This small software will process all the information in the files, and will convert it into a unified format. There will be 3 types of format, one per each segment, and this format will be as similar as possible to the database design, because in the next step we will import this unified formats files to the database, so if they have a similar format will be much fast and easy to export. The second step consists of importing these files to the database. The one in charge to perform this operation will be a SSIS package, similar to the one used in the *export* part. Finally, the third step consists of controlling which data must be inserted or updated in the database, similar to the previous *merge* method.

Now it's the right moment to do some tests and try to see if the offline mode is faster than the online one. The test will be done with the flight segment. The offline test will consist of converting 46.372 rows to the unified format and import it to the database. The time for converting it has been around 53 seconds, and the time to import it to the database is about 4 seconds, so its almost 1 minute time for nearly 50.000 rows.

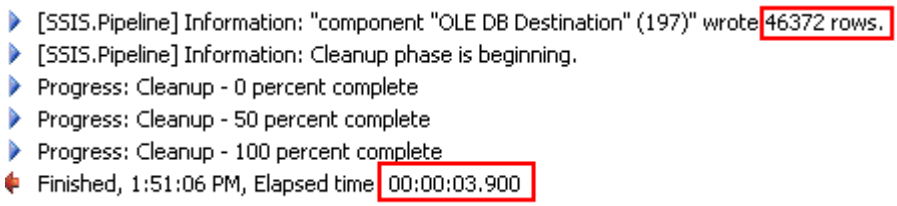


Figure 31. Offline mode test

With the online mode, the test was performed asking the providers for flights from Amsterdam to Barcelona for a period of 53 weeks (1 year). The result was 53 rows, (the system only stores the cheapest flight per day), with an execution time of 2 minutes and 30 seconds.

Therefore as you can see, the offline method is much faster than the online one, mainly because the offline method don't spend time asking the providers and the online one has to ask each time it sends a request, in that case 53 times.
At this moment the best solution seems to be a mix of both methods, using the offline to get the most part of the price information, and the online one to keep it as much updated as possible.

Conclusions

The main reason to build this system was that Pyton needed a cache database full of prices information, in order to offer to their clients the best possible combinations of accommodations, flights and transfers. The main issue since the beginning was the execution time needed in order to get all the necessary data. After implementing the first version of the system, the execution time was not as good as expected, mainly because the time that the system spends communicating with the providers was too high, therefore a solution must be found as we cannot improve this time because it depends on the providers.

So we decided to implement an offline method, that instead of sending queries to the providers and store the prices information, it consisted of a set of csv files with all the price information from the providers. The system will process all this files, and will select the necessary information, following the same criteria as before, and store it into the database. We realized that this offline method is much faster than the old one, because it doesn't depend on the provider response time, therefore this new functionality of the project makes it worth after testing it and comparing the results to the online method.

Finally we have the cache database with three different types of prices: the prices from the online requests to the providers, the non-real prices after applying the price rules to the real prices from the providers, and the prices from the offline method. This cache database consists of 10 tables to control everything with the smaller data types to use as less space as possible. The database will update itself as much as possible, depending on the server load and the time spend updating each city. The export system included, will export the different data in individual segments therefore the clients can build the packages they want.

For these remaining days, the goal is to work improving the execution time of the system, improving the price rules method so we can add more rules and the system applies them in an automated way and make it compatible to as many providers as possible. As recommendation for the future, some extra functionality can be added, like one to build the packages in addition to send the individual segments to the client and add another segment like car rental.

Evaluation

These 5 months working at Pyton has been my first ICT related work experience. So as you can imagine I've learned a lot of things in a very short time period. The fact of doing this internship in the Netherlands in a small Dutch company, made me improve my communication skills in English as well as my capability of adaption to a different working environment from what I am used to. It has been a great challenge working 8 hours per day developing a software project that at the beginning I didn't think I would be able to do it. One of the most important things that I've learned developing this project is to solve the problems I got during this period by myself.

Furthermore, regarding the technical part, this assignment required a lot of coding that make me improve a lot my skills after more than 1 year without coding. Also building a system that big by my own during 5 months taught me a lot about the different phases of the software development cycle: documentation, implementation and testing.

At the end of the period, all information, details, results and conclusions obtained about the project have to be gathered in a complete report that must reflect all the steps followed to achieve what has been done so far. From my point of view it was the most difficult part because after working such a long time on the same thing it becomes more difficult to reflect all the information in a clear and understandable way through a report. Plus, writing a technical report in English, which is not my mother tongue, added another handicap to this task. However, I am satisfied with the work done in this internship concerning the report and the work at the company.

Regarding the company, Pyton Communication Services B.V., I think that they provided me with a good and positive environment to carry out this project therefore my experience here has been very positive.

References

- [1] http://stackoverflow.com/questions/6283689/why-would-i-use-an-ssis-package-in-sql-server-2008-as-opposed-to-some-other-tech: Stack Overflow is a question and answer site for professional and enthusiast programmers. It helped me a lot because most of my coding questions were answered here with different solutions, so I could decide which the better one for me was.

- [2] http://msdn.microsoft.com/en-us/library/ms187745.aspx: Microsoft Developer Network it's the official Microsoft page where I can find a lot tutorials about c# and SQL server. They also provide a forum where I can find some answers to my Microsoft-coding related questions.

- [3] http://www.dotnetperls.com/xml: A c# dedicated website with a lot of tutorials.

- [4] pyton.nl: The website of the company. I used to extract some information about the company itself.

- [5] http://mscerts.programming4.us/windows/soa%20with%20_net%20and%20windows%20azure%20%20%20microsoft%20messaging%20queue%20(msmq).aspx: Webpage with a lot of information related to Microsoft software programmers.

- [6] http://stackoverflow.com/questions/9077598/key-benefits-of-msmq

Attachments

Attachment I – Project plan

Attachment II – Project Survey

Attachment I - Project Plan

Smart Packaging Cache

Project Plan v 4.0

04/04/2013

Written by:
Pol Algueró

Document Distribution	
Customer	**Pyton Communication Services**
Pyton Communication Services	Robert Evers
	Development team

Document Release acceptance	
Approved by:	Approved by:
Customer	**Pyton Communication Services**
Main contact	Project manager
- Signature	- Signature

Document Information

Document name: Project Plan
Filename: Project Plan.doc

Document History

Date	Changed by	Brief description of changes
20/02/2013	P. Algueró	Document Creation (1.0)
15/03/2013	P. Algueró	Fontys tutor feedback (2.0)
20/03/2013	P. Algueró	Fontys tutor feedback (3.0)
28/03/2013	P. Algueró	Patrick Lind feedback (4.0)

Table of content

1 Project Statement

1.1 Formal client

My graduation project client will be the company itself, Pyton Communication Services B.V., because it's an internal project.

1.2 Project leader

The project leader will be myself, as I'm the only one working on this project.

1.3 Current situation

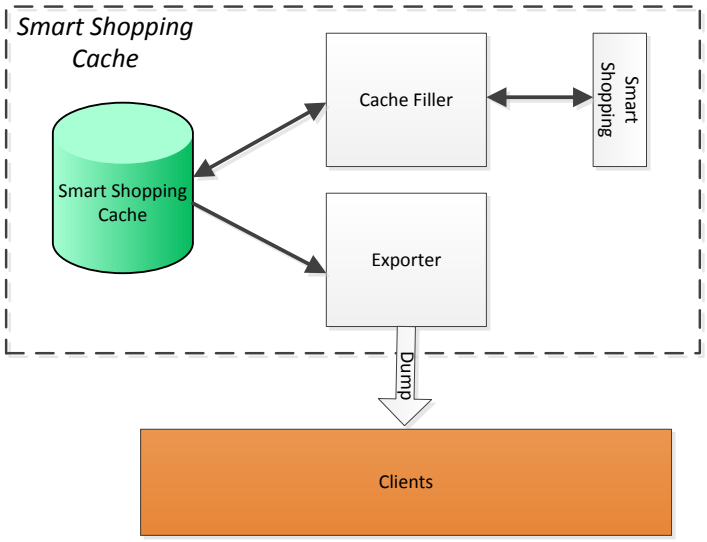
The current situation is that Pyton has the Waverunner Smart Shopping system, that permits us to communicate online with our providers and get the individual price of each segment of the travel products that they can offer to their clients, and offer them full packages with flight + accommodation + transfer.

1.4 Project justification

The main problem is that now we can only offer to each client a full package with their own segments, so what we want with the new cache system, is to offer each client all the individual segments we have from our providers, regardless which provider is the owner, thus our clients will have a better offer to show to their customers.
Also Pyton need to show the best offers based on the displayed search result (i.e. the customer search for a flight to Barcelona, so the system will display also the best hotels in Barcelona). This could be done using the cache system I will be working in the Waverunner Smart Shopping system. With this system the providers will be able to sell more segments easily to their customers.

1.5 Project product

The cache system will consist on a cache database were we will store all the available prices of all flights, accommodations and transfer from all of our providers. This will be done through a system that will periodically dump all the prices to the clients.
In the picture below we can see the request system (cache filler) that will communicate with the providers via the smart shopping platform and will store the price information in the smart shopping cache database. Finally we will have an exporter system that will export all the data from the cache to a file.



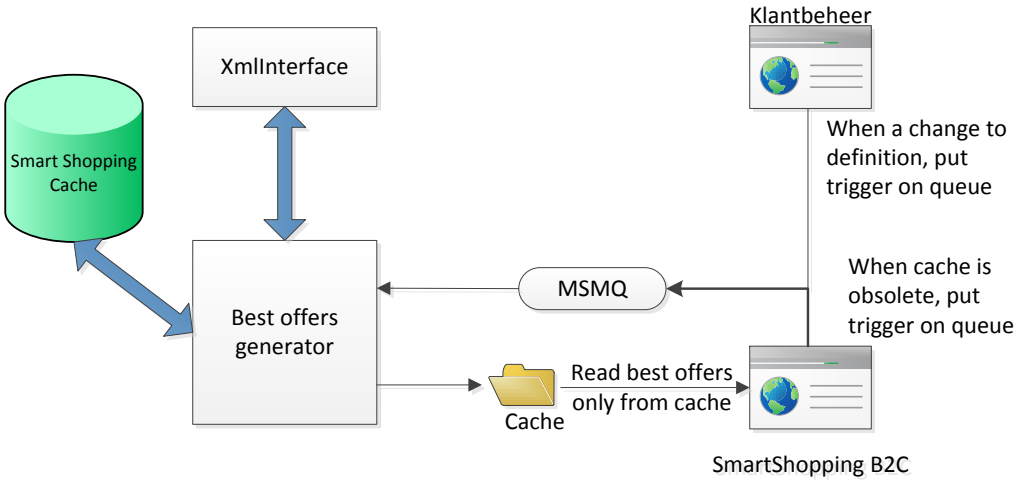
The best offer system will consist on a cache based system that will show to the customers the best offers (of flight, accommodations and/or transfer) depending on certain rules. For example, if the customer is looking for flights to Germany, the system will show the popular hotels in popular places of Germany.

The best offers generator generates best offers based on definitions stored in customer management. It is triggered by putting a message on a message queue. This message will trigger a refresh of the specified definition.
A trigger can contain any of the following actions:

- Update.
- Delete.

The frontend (SmartShopping B2C) only uses the best offers that are in the cache. Each item in the cache has a period it is valid. When this period is expired and the frontend reads this item from cache the following will happen in the frontend:

- The frontend still shows the expired item.
- The frontend puts an update request on the queue.



1.6 Project deliverables and non-deliverables

At the end of the project I will arrange a meeting with Robert and Patrick to explain and show all my deliverables.
In this project I will have to deliver:

- All the technical documentation about both projects that I will be developing during my assignment project, which means, I will describe what and how I'm going to do, and the possible alternatives I have to implement everything.
- The final code of the 2 projects that I will have implemented.
- The final product, that means the 2 working systems.

Non-deliverables:

- Some extra documentation with research that won't be included in the technical documentation.

1.7 Project constraints

The constraints I have is to use the resources that the company can provide me, in this case, all the Microsoft Technology. For example, I will have to code in C# using Visual Studio and also I will have to use Microsoft SQL Server Management Studio to build the database.
I also have to follow the constraints that the company follows when they document a project or when they code(style), and Fontys constraints for writing their documentation.
For building the cache system I will have some constraints related to not have a huge load on the system, i.e. request only the prices for 7 days and calculate the price instead of requesting each day of the year.

1.8 Project risks

Risk	Description	Probability	Impact	Mitigation
Patrick Lind	Something can happen to Patrick so he won't be able to guide me with the project.	rare	major	nothing
Providers	Providers give as wrong data.	rare	major	investigate which type of data they will provide us
Servers	Servers stop working.	unlikely	major	nothing
No testers	I don't have testers to test my project	rare	major	nothing
Testers not available	I have testers, but they are not available when I need it.	moderate	moderate	Have different testers if someone's fails.
Extra functionalities	The customer ask me for extra functionalities	moderate	moderate	change project plan to fit the new functionalities
Time	Run out of time	moderate	major	nothing

2 Project Phasing

I will start with the cache system, as we agreed upon with the project team, the priority is to build first the cache system and after it begin building the best offer system because the best offer system will use the cache database that I will built in the first system.
My method of working during all this phases will be an incremental build mode. I decided to work with that method because with this method I can divide the full project in small phases and start researching, designing, implementing and testing each little phase until the product is finished. I also choose it instead of a waterfall method because with a waterfall method I will have to do the researching, designing, implementing and testing with the full project instead of little phases.

2.1 Cache System

The main research question is: "*How can I build an online cache system for the providers that don't have it, and keep it up to date?*". For answering this question I will first have to answer all the different sub-questions that will appear as I do the project, i.e.:

- Which is the best size that the cache database should have?
- Which data should I export, an already made package or the individual segments so the providers can build the packages they want?
- Which is the best way to keep the cache system updated?
- Which is the best data format to use as less space as possible?
- Which is the best query format to communicate with the providers?

2.1.1 Initial research

- activities

  Research how does the new tools I will use in the company works and also how can I start building the database.

- sub-deliverables

  A first version of the technical documentation with all the information I investigated on the internet or asking the workmates, and with the different options I have to start building the cache database.

- dependencies:
  - Time
  - Servers
  - Patrick Lind

2.1.2 Database implementation

- activities

  Implement the cache database

- sub-deliverables

  The cache database I build with the Microsoft SQL Server Management Studio.
  Update the Technical report with the final version of the database.

- dependencies:
  - Time
  - Servers
  - Patrick Lind

2.1.3 Database testing

- activities

  Start introducing dummy data to the database.

- sub-deliverables

  A file with all the data I've introduced to the database for testing proposals and the test results.

- dependencies:
  - Time
  - Servers

2.1.4 Research how to fill the cache research

- activities

  Research how is going to work the cache filler.

- sub-deliverables

  Update the technical document with all the information I found/ask related with the cache filler.

- dependencies:
  - Time
  - Servers
  - Patrick Lind

2.1.5 Fill the cache implementation

- activities

  Implement the cache filler.

- sub-deliverables

  The software I coded using C# via Visual Studio.
  Update the technical document with the final implementation of the cache filler.

- dependencies:
  - Time
  - Servers
  - Patrick Lind

2.1.6 Fill the cache testing

- activities

  Test the cache filler.

- sub-deliverables

  All the different files that I've used to test the cache filler with the results of the test.

- dependencies:
  - Time

- Servers

2.1.7 Research how to export the data

- activities

  Research how is the exporter going to work.

- sub-deliverables

  Update the technical document with all the information I've searched/asked about the exporter.

- dependencies:
  - Time
  - Servers
  - Patrick Lind

2.1.8 Implement data exporter

- activities

  Implement the exporter.

- sub-deliverables

  The software I coded using C# via Visual Studio.
  Update the technical document with the final implementation of the exporter.

- dependencies:
  - Time
  - Servers
  - Patrick Lind

2.1.9 Test the data exporter

- activities

  Test the exporter.

- sub-deliverables

  All the different files that I'll use to test the exporter with the results of the test.

- dependencies:
  - Time
  - Servers

2.1.10 Cache system testing

- activities

  Test the full cache system.

- sub-deliverables

  All the files used to test it and the test result.

- dependencies:
  - Theo Seuren(Tester)
  - Providers
  - Servers
  - Time

2.1.11 Scalability system testing

- activities

  Test the full cache system with different amounts of data.

- sub-deliverables

  All the files used to test it and the test result.

- dependencies:
  - Theo Seuren(Tester)
  - Providers
  - Servers
  - Time

2.2 Best offer system

My main research question is "*How can I build a system that shows the best offers, of a particular segment, depending on certain rules?*"
As before, I will have to start answering more specific questions before it like:

- Which rules do I have to use to decide the best offers.
- How is going to work the trigger system along with the rules I defined.
- How I'm going to work with the cache obsolete offers.
- How I'm going to communicate with the front end and the xml system.
- Which is the best way to maintain all the offers updated as much as possible.

2.2.1 Best offer research

- activities

  Research how can I build the best offer system.

- sub-deliverables

  A first version of the technical documentation with all the information I research on the internet and asking the workmates, and the different options I found for building it.

- dependencies:
  - Time
  - Servers
  - Patrick Lind

2.2.2 Trigger system implementation

- activities

  Implement the trigger system that will permit us to send all the queries.

- sub-deliverables

  All the code I used to implement it.
  An updated version of the technical documentation about the trigger functionality.

- dependencies:
  - Time
  - Servers
  - Patrick Lind

2.2.3 Trigger System test

- activities

  Test the trigger system.

- sub-deliverables

  All the files I will use to test it and the test results.

- dependencies:
  - Time
  - Servers

2.2.4 Rules implementation

- activities

  Implement the rules that will decide which offers to show.

- sub-deliverables

  All the code I will use to implement it.
  Update the technical document with the final rules implementation.

- dependencies

  Silvia: She will asses me to get the best rules.

- dependencies:
  - Time
  - Servers
  - Patrick Lind
  - Silvia

2.2.5 Rules test

- activities

  Test the best offer generator.

- sub-deliverables

  All the code I will use to test it and the test result.

- dependencies:
  - Time
  - Servers
  - Silvia

2.2.6 Calendar implementation

- activities

  Implement the best offer calendar that we will show to the user.

- sub-deliverables

  All the code I will use to implement it.
  Update the technical documentation with the final implementation of the calendar.

- dependencies:
  - Time
  - Servers
  - Patrick Lind

2.2.7 Calendar test

- activities

  Test the calendar.

- sub-deliverables

  All the code I will use to test it and the test result.

- dependencies:
  - Time

- Servers

2.2.8 Best offer system test

- activities

  Test the full best offer system.

- sub-deliverables

  All the code I will use to test and the final test result.

- dependencies:
  - Time
  - Servers
  - Theo Seuren
  - Providers

2.2.9 Scalability system testing

- activities

  Test the full best offer system with different amounts of data.

- sub-deliverables

  All the files used to test it and the test result.

- dependencies:
  - Time
  - Servers
  - Theo Seuren
  - Providers

3 Management plan

3.1 Money

It's an internal project, all the resources I will use are the internal resources from the company.
The main benefits will be the cache database full of data, because this will mean the company will have more data than the suppliers itself. So the company will be able to use it for future projects.

3.2 Skills

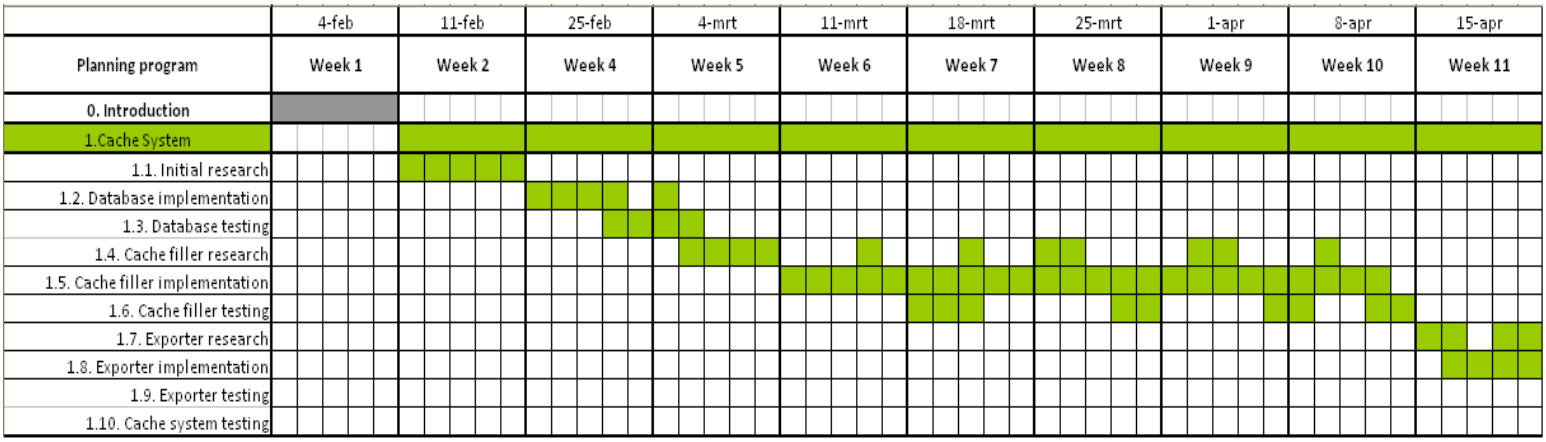
For the documentation phases I will need project organisation skills, writing skills and foresee problem skills too.
During the implementation phases, I will need knowledge about the software I will use, mainly Microsoft Visual Studio and Microsoft SQL Server Management.
I will use C# for the most part of the coding so I will have to research on the internet and get some help from the workmates of the software department (see 4.5 Organisation).
For building the cache database I will also need SQL skills.

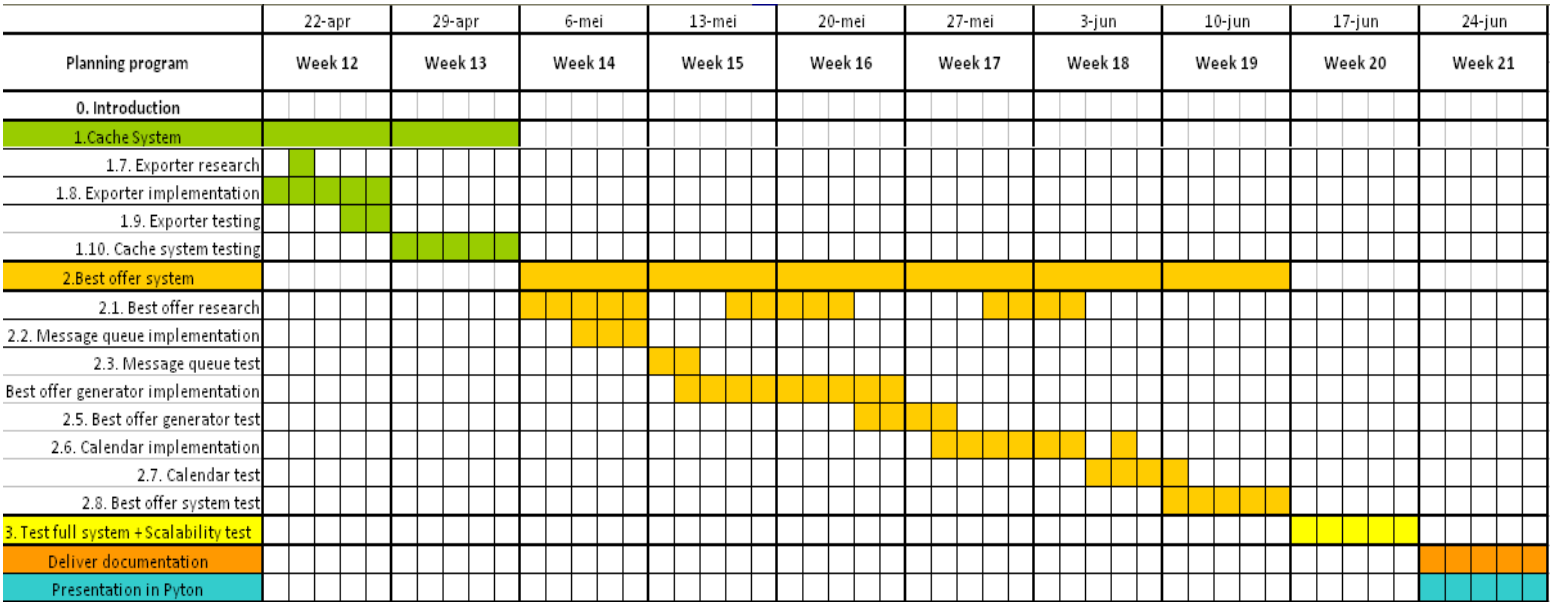
3.3 Quality

For the technical documentation the quality will be given by the templates of the company, that will help me to keep all the documentation in the correct format and will also help me to control all the different versions I will made.
For the implementation phases the quality will be given by the visible results. For example if my system is able to process all the data that I will receive from the providers and show them the expected results it will mean that the system is, at least, working as expected. After that I will try to make it as efficient as possible, I mean, I will try to make the software to work as optimal as possible because it will process a lot of data and the time will be crucial.

3.4 Time

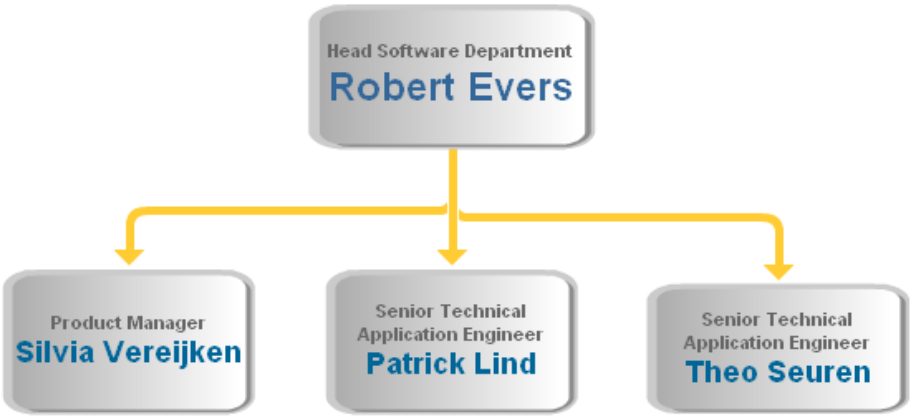
Here you can see the Gant Chart I will try to follow:





4 Organisation

During my stage Patrick will be the one who will guide me, but in some phases I will require some extra help.
In the testing phases, Theo will be the one who will help me. In the phases that will require some special knowledge about how does the product works, Silvia will be the one who will help me.
All of this under Robert's supervision, as he is the one in charge of the project.



5 Communication plan

My mail is: p.algueroraga@student.fontys.nl
My Dutch phone: +31 685 25 34 57

My Fontys tutor is Erick van der Schriek (e.vanderschriek@fontys.nl).
I will receive his visit to the company twice: one at the beginning and one at the end of the stage.
During the stage, I will contact him at least every two weeks, because I will send him a brief explanation about my work during this 2 weeks and a Gantt chart showing what I did.
My communication with him will be via mail, but if the matter is important/urgent I will call him or make an appointment with him.

Attachment II - Project Survey

Fontys

University of Applied Sciences

Appendix A : Graduation Project Survey
HBO-ICT: English Stream

Data student:
Name student : **Initials:** P.A.R **Name:** Pol

First name: ALGUERÓ **Studentnumber.:** 2211668

Telephone: +31 685253457 **E-mail.:** polalguero@gmail.com

Data company:
Name company/organisation: PYTON COMMUNICATION SERVICES BV

Visiting adress : SCHAEBEURDERLAAN 10, 6002 ED WEERT

Company mentor : **Initials:** RPM **Name:** EVERS

Telephone: 0495-577777 **E-mail.:** R.EVERS@PYTON.NL

Department/ position: SOFTWARE DEVELOPMENT

Startdate Graduation project : 4/2/2012

Duo Graduation project : ~~Yes~~ / No

If duo name of buddy: _____

Accepted by student: date: 25/1/2012 **signature:**

Accepted by company: date: 25/1/2012 **signature:**

Hand in date Graduation Project Survey:

Approved by graduation project coordinator: yes/no **date:** **signature:**

Remarks : _____

PLEASE SEND THIS FORM BY EMAIL TO THE INTERNSHIP COORDINATOR IMMEDIATELY AFTER THE INTERNSHIP INTERVIEW HAS TAKEN PLACE.

Description of the graduation project:

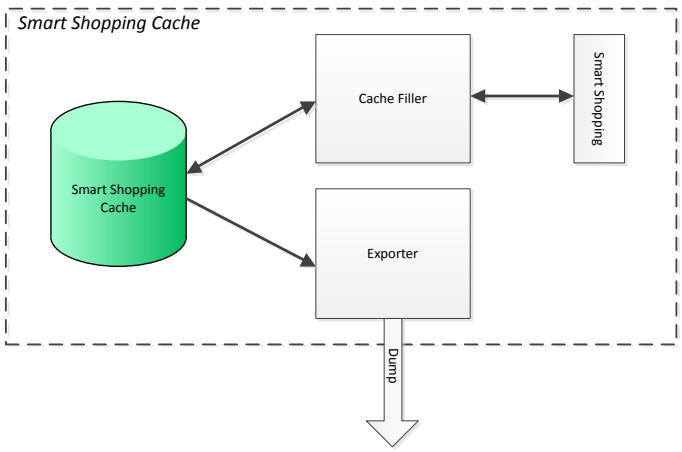
1. **Describe the problem analysis:**
   (what is the reason for the internship company to inititiate this assignment? What is for the company the added value of this assignment? Can you describe the starting situation and starting points: introduction and problem definition.

   Pyton company want me to work in a project they offer called **Waverunner Smart Shopping:** The Waverunner Smart Shopping environment provides a dynamic shopping basket platform for travel products which lets customer create his own travel package composed of flights, accommodation, transfers, and car rental.
   They have an online system to communicate with providers and get the price and availability (of the flight, hotel, transfer, car rental; depending on the type of the provider) but most of them don't use a cache system so they don't know their prices because they use an online system to communicate with their companies so they make a request and they receive the prices. So I need to build a cache system to store the prices from that providers that don't have cache data so we will be able to combine the full package and show it to customers more efficiently.
   Also Pyton need to show the best offers based on the displayed search result (eg. the customer search for a hotel in Barcelona, so the system has to display also the best flights to Barcelona).
   Thus could be done using the cache system I will be working in the Waverunner Smart Shopping system.

2. **Describe the graduation assignmen**t.
   (especially the objectives, results to be delivered and final products to be realized. Also indicate what you want to achieve for the internship company. Give a clear description of the graduation assignment).

   I will start making the technical documentation for both projects and after that I will begin building all the system as it appears in the documentation. The priority is to build first the cache system and after it begin building the best offer system.
   So at the end both system have to be working as expected.

   Cache system:
   The cache will be based on net prices, or gross prices if they are the same for all parties.
   In the system will be a list of places that Pyton will gather data for. This list is maintained by Pyton, but a client can request places to be put on this list.
   Not all prices will be "real" prices, meaning they are calculated. For example the system will request only prices for 7 nights, and calculate the prices for 1 – 6 nights based on rules set-up by Pyton. The online interface will be linked to this cache, so also real-time availability is added to the system.



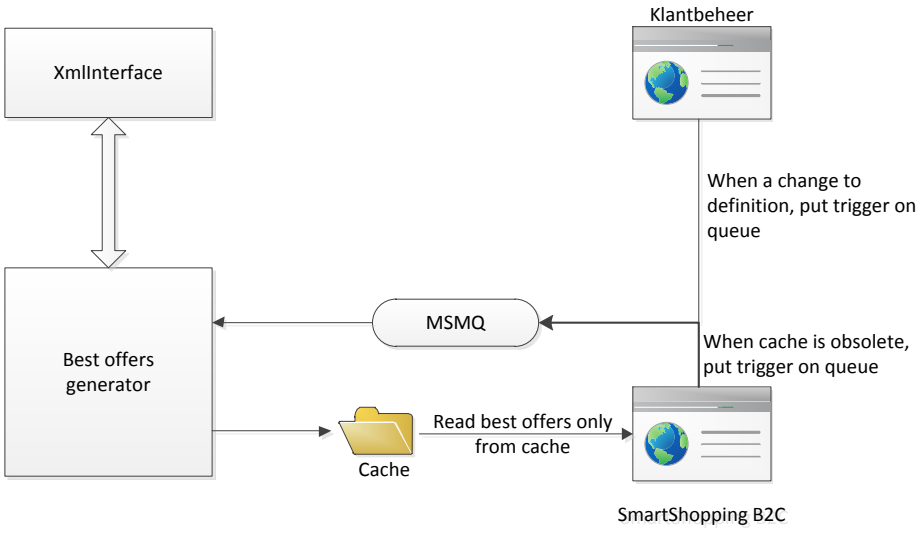
The following data should be returned for an accommodation:
Accommodation
\---- Unit
\-- Board
\--- Date / Duration / Price
The following data should be returned for a flight:
Depart airport
\--- Arrive airport
\--- Date / Duration / Price

Best Offer System:
The best offers generator generates best offers based on definitions stored in customer management. It is triggered by putting a message on a message queue. This message will trigger a refresh of the specified definition.
A trigger can contain any of the following actions:

- Update.
- Delete.

The frontend (SmartShopping B2C) only uses the best offers that are in the cache. Each item in the cache has a period it is valid. When this period is expired and the frontend reads this item from cache the following will happen in the frontend:

- The frontend still shows the expired item.
- The frontend puts an update request on the queue.



3. **What is the research component of this assignment?**
   (What are your research topics? If necessary, draw up a research plan).

My main research will be about how the providers system work so we can get the full cache system with all the data we need(prices and availability) from the providers that don't use a cache system. Also when the cache system is finished I'll have to research how we can update it every day/hour. I'll also make a database design and a performance analysis.

4. **What are the methods and tools?**
   (What operating procedure and means can you made use of during the internship period at the company? What facilities will be made available by the company?

I'll be able to use all the tools and facilities that the company can provide me such as: Microsoft Office, Microsoft Visual Studio, Microsoft SQL Server Management Studio, Internet, etc.
Pyton will also provide me an office with a desktop computer.

5. **How and by whom will you be guided by the company?**

I will be guided by Patrick Lind (p.lind@pyton.nl). He will be working in the same office as me.

6. **What fields of Study play an important factor in realizing the graduation assignment?**
   (for example information analysis, design, realization, monitoring and security).

databases; system development; information analysis; object oriented design; client server application;

OTHER DETAILS:

Preference university tutor: **1.**

2.