Description with UML for a Hotel Reservation System

Door Janne Louw

10-05-2006

Introduction

Since 2001 I've been working for a small web-solution provider called Remotion. In 2003 a colleague and I developed an online hotel reservation system called simply HRS. In 2005 the task was given to me to extend the systems functionality. The additions required a complete redesign of the Database Model, and the front-end code. We decided to completely rebuild the systems front-end, do extensive refactoring on the back-end and make a new database model complete with automatic database-conversion, all with future extendibility and easy of maintenance in mind.

This project became HRSv2, and besides some drones work, I've completely done all refactoring, redesigns and programming. My Bachelor Project covers part of the HRSv2 project, specifically the description of the front-end code. Sadly, due to some commercial issues, we've put a hold on the project. It's about 75% done, but it will only be continued if we can find a buyer, or if current or new customers are willing to pay for the development of the new version.

The software itself is a system for hotels where they can deeplink to from their own website to let visitors book a room in the hotel. This means the system is a very single minded system, and no introduction is necessary, because when a customer enters, he has already pressed the 'Book now' button on the hotel's front page. The system will simply display a series of forms that will query the user about dates, rooms, additions, etc.

In this project I will not go into the exact details of the interface, but I will give a description meant for a programmer who needs to perform maintenance on the system, to help him understand the software faster.

So what does the system do? I'll answer that question first in text in this introduction. The first half of the paper will answer it in increasing depth. The second half will cover more of the actual code structure, and finally I'll give some conclusions and recommendations.

I'll sum up what the system does during a typical booking:

- Query the user for date of arrival and length of stay
- Calculate available rooms from the database
- Display the available rooms and query user to choose one or several rooms
- Ask user how many and what kind of guests will be staying in each room
- After that a calculation is shown for the cost of the stay
- In the same screen the user is presented with general and room specific options and extra's to choose from, like diner and breakfast, or the possibility of renting bikes, etc.
- In the next step the user is queried for his name, address etc.

- After that the user will be asked to confirm the booking with a credit card. Hotels can let registered users (usually corporate customers) automatically skip this step
- A final confirmation screen is shown, where the user is shown a complete overview of the booking, can give some comments and confirm the booking.
- Emails are sent to the client and to the hotel receptionist, and the booking is added to the database.

As briefly mentioned, there is also an option to log into the system for registered users. When logged in a customer can view and cancel bookings. Also, hotels may have discount policies for certain registered users.

For the hotels the main strength of the HRS system is that it is very flexible in pricing and availability issues, and can handle a wide variety of special clients, special offers, and any regional price fluctuations, for rooms as well as options, whilst allowing each hotel to have its own look. Of course, a competitive price is also a nice asset. There are three basic types (pricing methods) of selling a room (from now on called item) and the system is set up in such a way, that new methods can added with a minimum of labor. Each pricing methods can be quite different, and they all at least slight variations in one or several screens.

A lot of work has been put in designing this flexible pricing method system, so I hope the project will once be re-started, and my work will not be in vain.

This document is basically a reverse-engineered description of the system. I've done this by using several different UML diagrams to describe in increasing depth a variety of ways to look at the system.

So why is this all necessary? There's a good chance other programmers will have to do maintenance on the system at some point. The system is too complex to easily understand it by just looking at the code and comments. This paper will allow those people to get a good understanding of the system before diving into the code.

Global structure

Figure 1 is the activity diagram of the global structure of the front-end. It shows the relation between user and webserver. The user starts by requesting a page from the webserver. The server first creates a new invocation, then starts a screen. The user has to go through a series of screens. At the last screen the booking is finalized. Within each screen the user is presented with a page containing a form, which he has to fill out and send back.

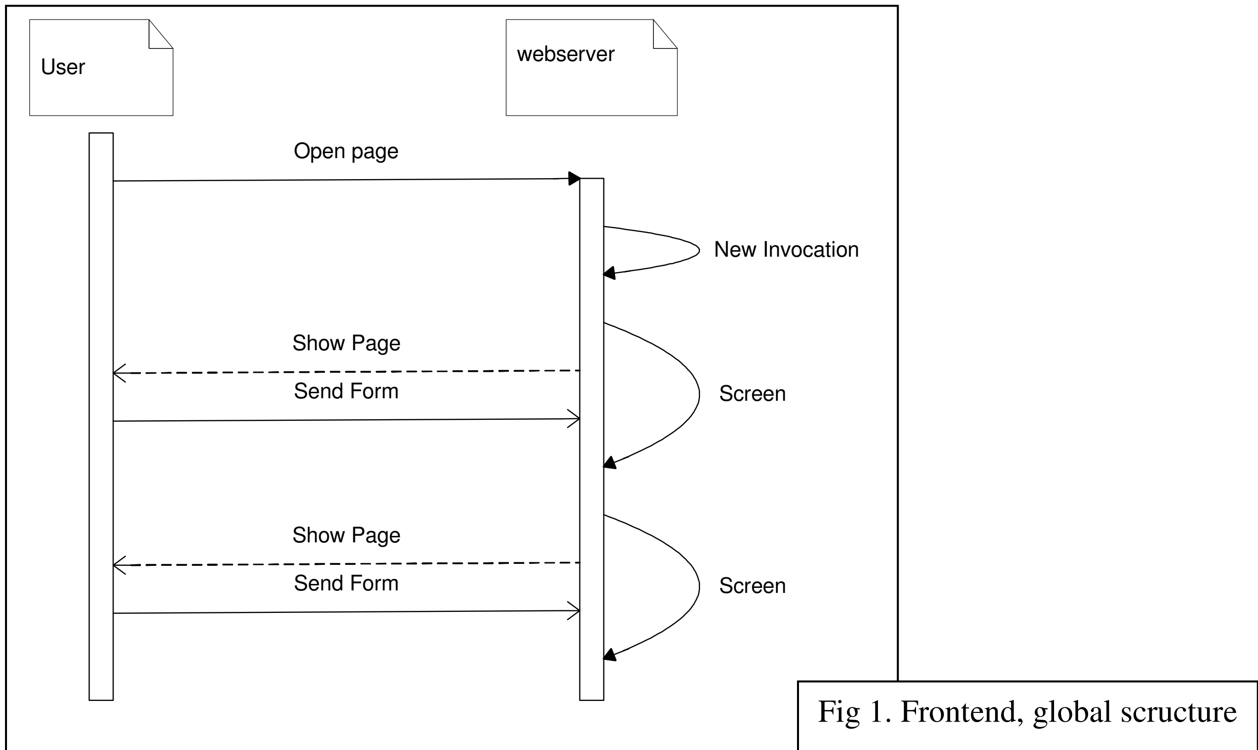

Fig 1. Frontend, global scructure

Figure 2 takes a closer look at the Screen part. The Screen is
actually an object within the webserver. It supports two main methods: writeScreen and handler. The first returns code for the HTML form that will be presented to the user. The handler processes the form data returned by the user, and creates a new Screen object for the next step.

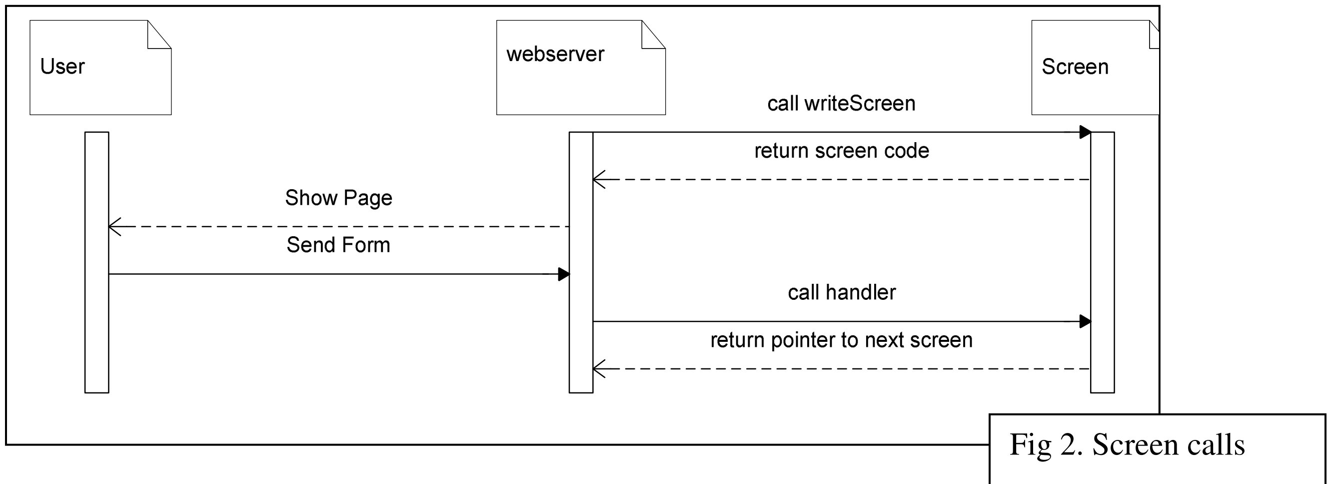

Fig 2. Screen calls

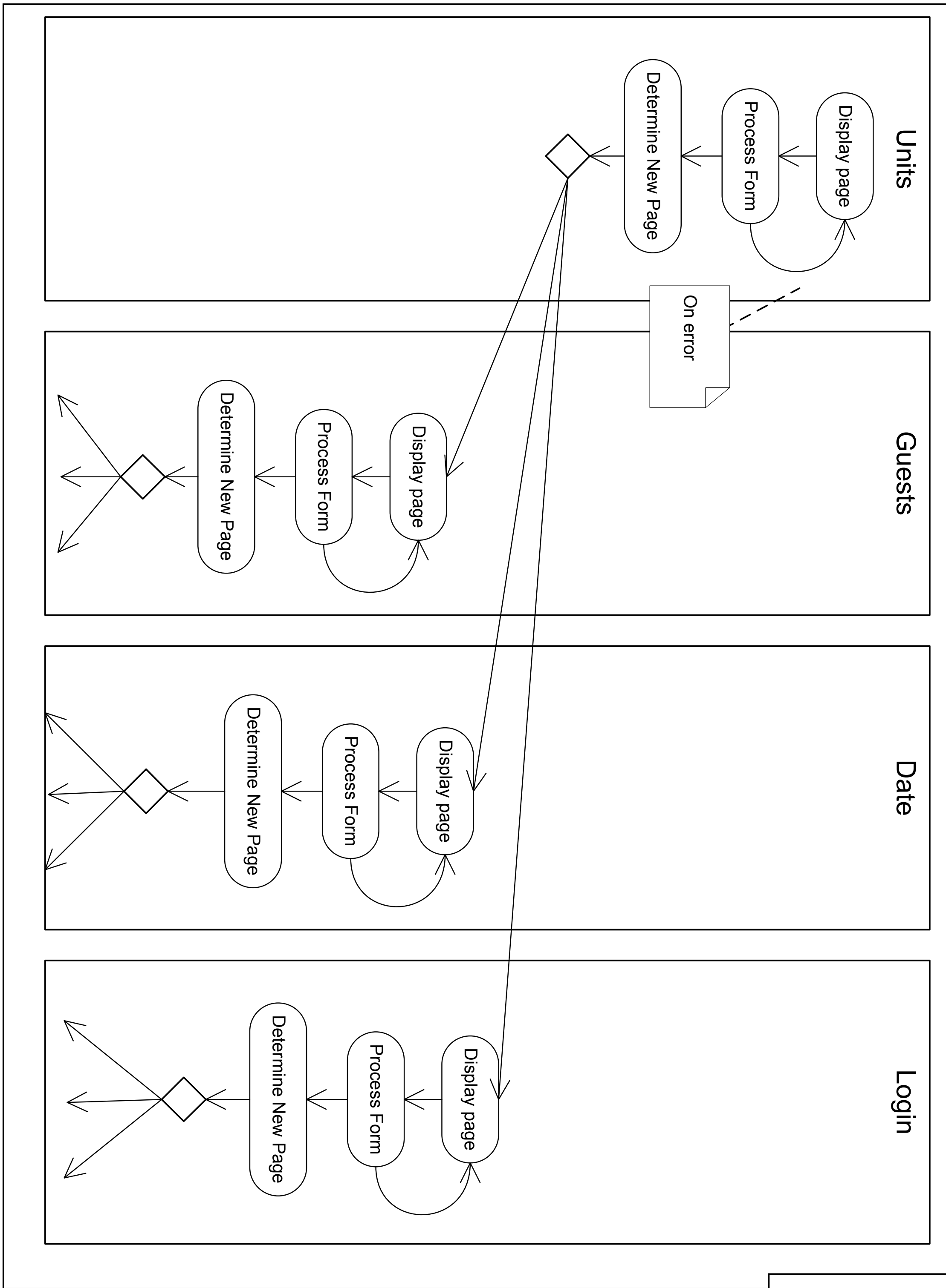

Fig 3: Activity in swimlanes

The way different screens relate to each other is displayed in figure 3. At the final step of each screen a decision is made what the next screen will be. This depends on the button pressed, or in some cases, some database retrieved variable.

Screen sequences

As became clear from the last part, the application is a sequence of screens to which the user has to provide responses.
There are three main choices for the next screen. The first is through for submission. The user provides all information, and is redirected through to the next page in the normal sequence.
The second way is through direct navigation. In this way the user can go back to previous pages in the booking sequence, or go to the login page or to the login part.
The final possibility is that the user has provided incorrect input. In that case the current screen is kept.

The user can take several paths through the system, but there is one main path, and one side path. All other paths are similar to those paths.

The main path is a normal booking. The user will go through screens Date, Units, Guests, Addition, Client, CreditCard and Confirm, and finally to Ready, where the session is cleaned. In figure 3 these are coded in yellow.
In between he can go back to a previous Screen, but never skip a subsequent screen.

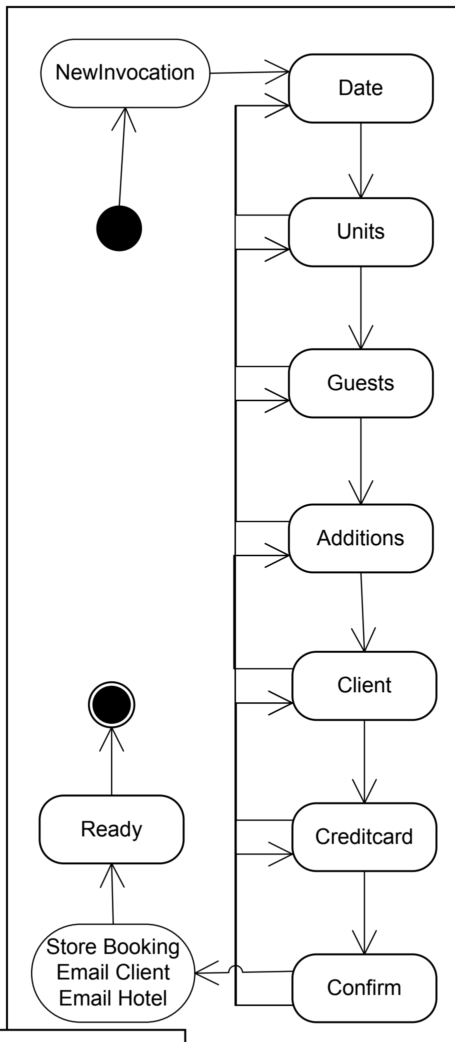

Fig 4. Normal flow

In a certain case the creditcard screen is skipped. This will only happen if the user is logged in, and the user has the appropriate paymethod allowed by the hotel. This is seen in figure 5.

The total flow is shown in figure 6. Also added is the login path. The user goes either directly, or at any other step, to the login page. Logged in users can go to any of the user screens, coded in blue, or they can logout, or they can start a booking. If a user is logged in and on the normal booking track, he can go to the BookingList page. From there he has the same options as when he just logged in. Information about the booking is retained.

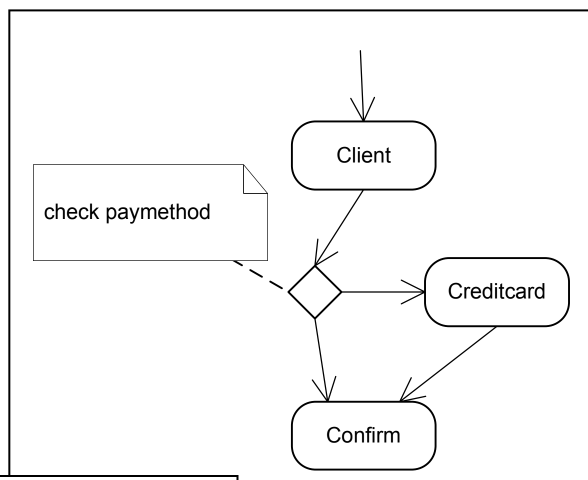

Fig 5.CreditCard

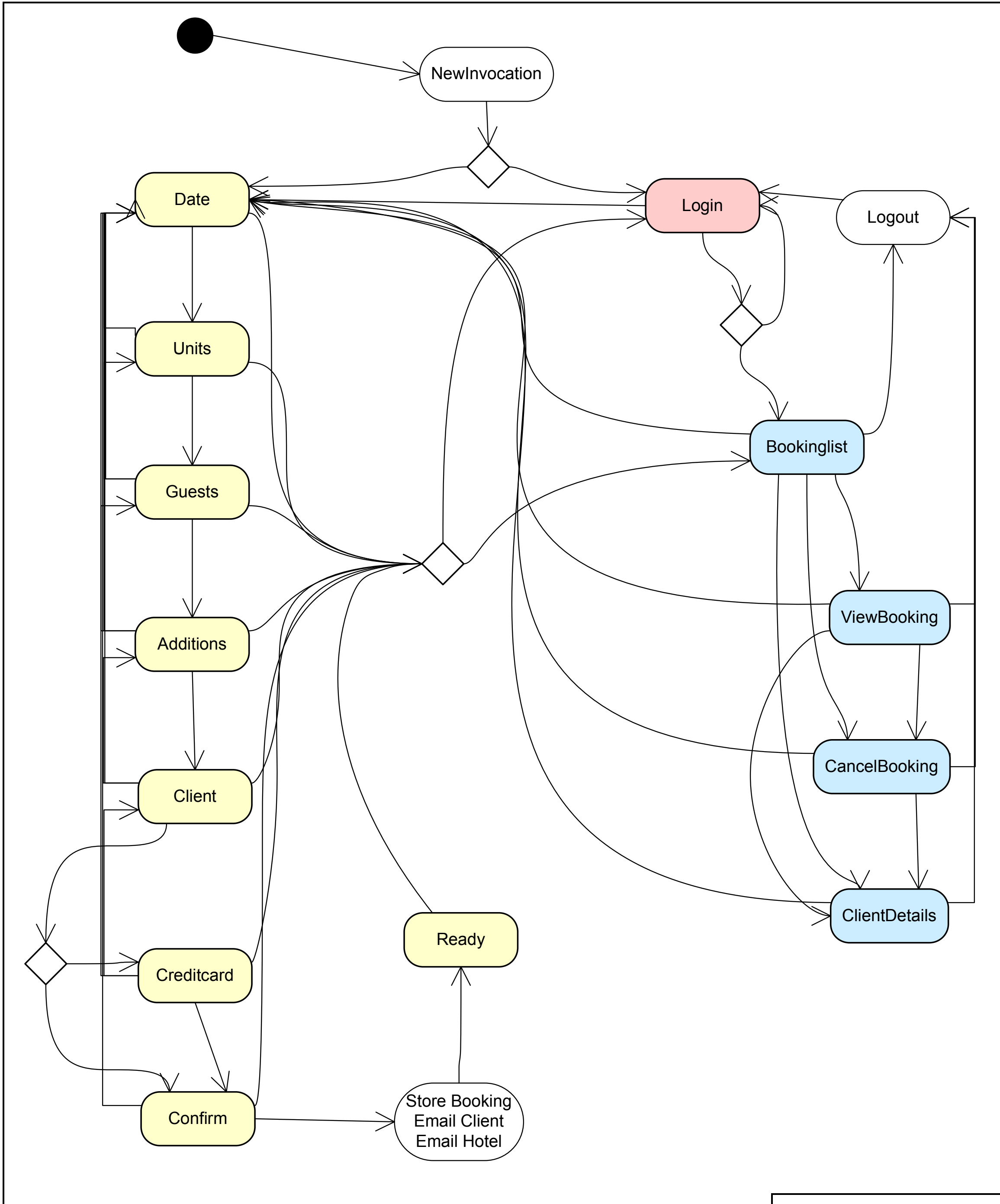

Fig 6.Complete flow

Code-level

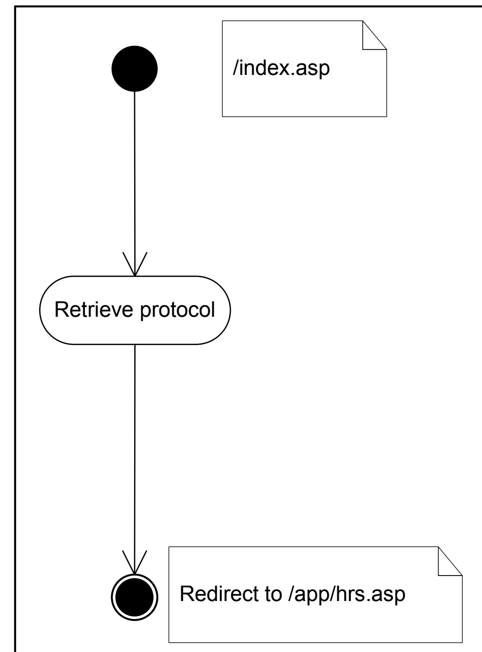

Fig 7. front-end pages

Most of the systems functionality is put in a single requestable page, “app/hrs.asp”. This page is opened every time a screen is submitted, and every time a new screen is shown all functionality is included and called through prototypes (objects), procedures or functions.

The initial landing page is the only other requestable page, “root/index.asp”.This page will only retrieve the correct protocol from the database (http for development, https for live), and immediately redirect to the main page.

Using a tool in the backend the hotel’s employees can create links. These links contain, besides the domain and path information, additional information in the query string part of the url. The employee can add information like the hotel id, type of booking, language, arrangement, etc. The hotels will put these links on a part of their own website. These links are the only way for a client to enter the reservation system.

I will follow with a description of what happens at the main page upon a page request.

Firstly a check is done to see if the request is a new invocation of the system. This is done by inspecting the query string. If so, the session persistent objects and globals are created and stored. After that the page is refreshed. If the data was incorrect, an error message is shown.

If no new invocation is detected all session persistent objects and globals are retrieved, and several other globals are retrieved.

Secondly a check is done to see if there is a POST-form submitted. A POST form is one of the two types of html forms. In a GET form the information that is submitted in the form is put into the querystring, and in a POST form the submitted information is send with the HTML request header. If such a form is detected the handler of the screen is called. The handler alters the persistent objects, and redirects the page.

Finally, if the page is neither a new invocation nor a submitted form, the layout is constructed and then responded to the user. Submission of the form will reload the page.

This structure is shown in figure 8.

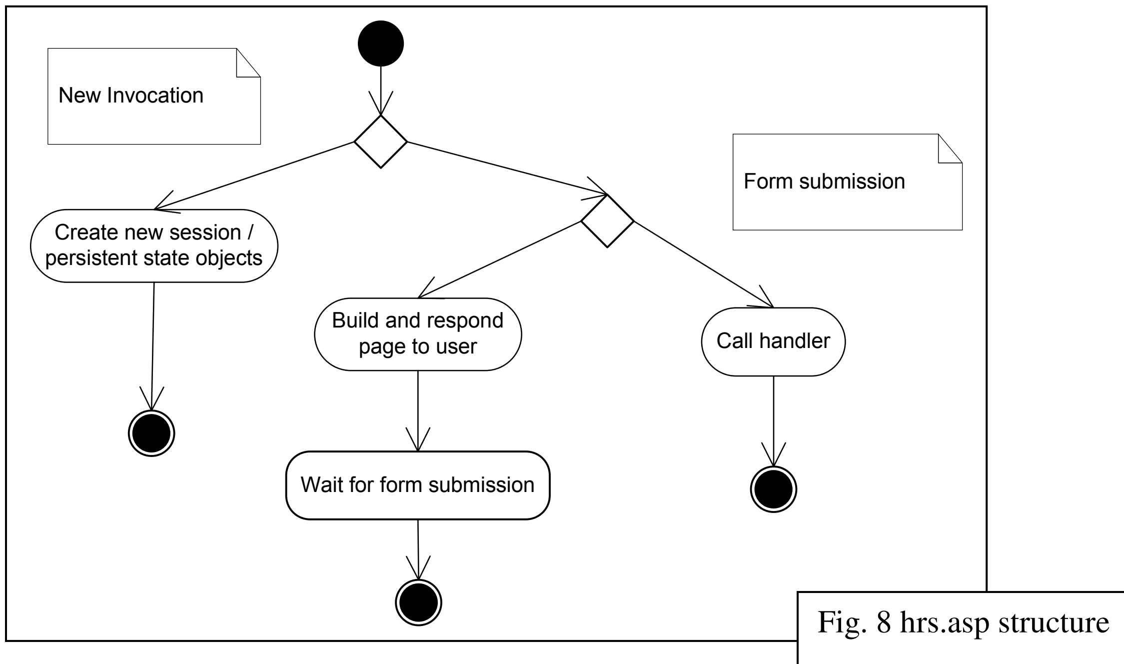

Fig. 8 hrs.asp structure

Object structure

The system has two main persistent objects. The first is the Booking object. This object contains all information gathered about the current booking. It will also write this information to the database if the booking is confirmed, and it can fill itself from the database if existent information is used. This is typically a user who logs in (client information is filled) or for display of an existing booking.

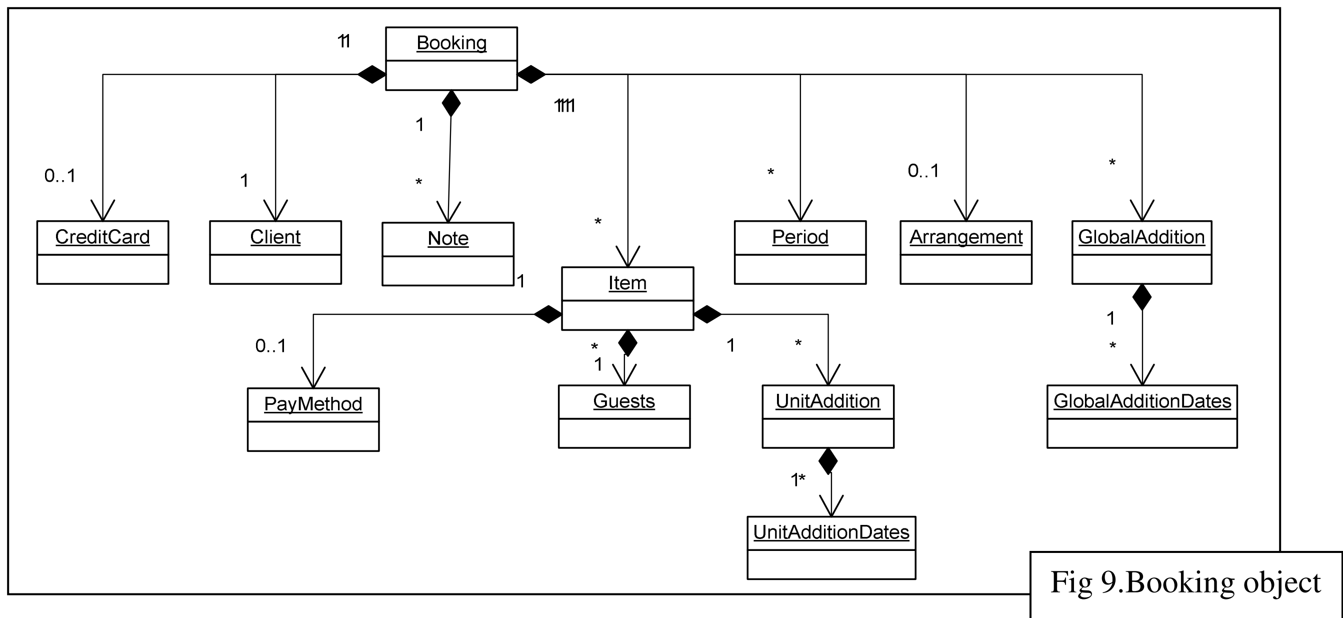

Fig 9.Booking object

For each relevant table in the database a separate object prototype exists. All table objects have methods for getting and setting the fields of the corresponding database table, checking if a field is set, and Boolean methods indicating whether a record can be found matching the filled primary keys, and whether the record was actually retrieved. For each aggregated relation the object also has methods for adding, removing, getting and, if applicable, counting the contained objects.

Finally the object has methods for reading the data from the database, and writing to the database. These methods are recursively called to all contained objects. In case of a write this is done bottom-up. Objects on the bottom are written, and the newly created record id's are passed to, and inserted into the parent record, until finally the entire booking is written. Reads are done top-down. A query is done on for childs using the parent id. New child objects are created, and the retrieved ids are used to fill the primary key fields, after which read is called for that object.

Code reuse is accomplished not by inheritance, but by aggregation and delegation. Each table object instance contains an instance of the DBTable object. This object contains the actual code for the database manipulation, the getting and setting of fields, etc. The actual table objects only contain code for delegation and aggregation functionality. This structure is shown in figure 10.

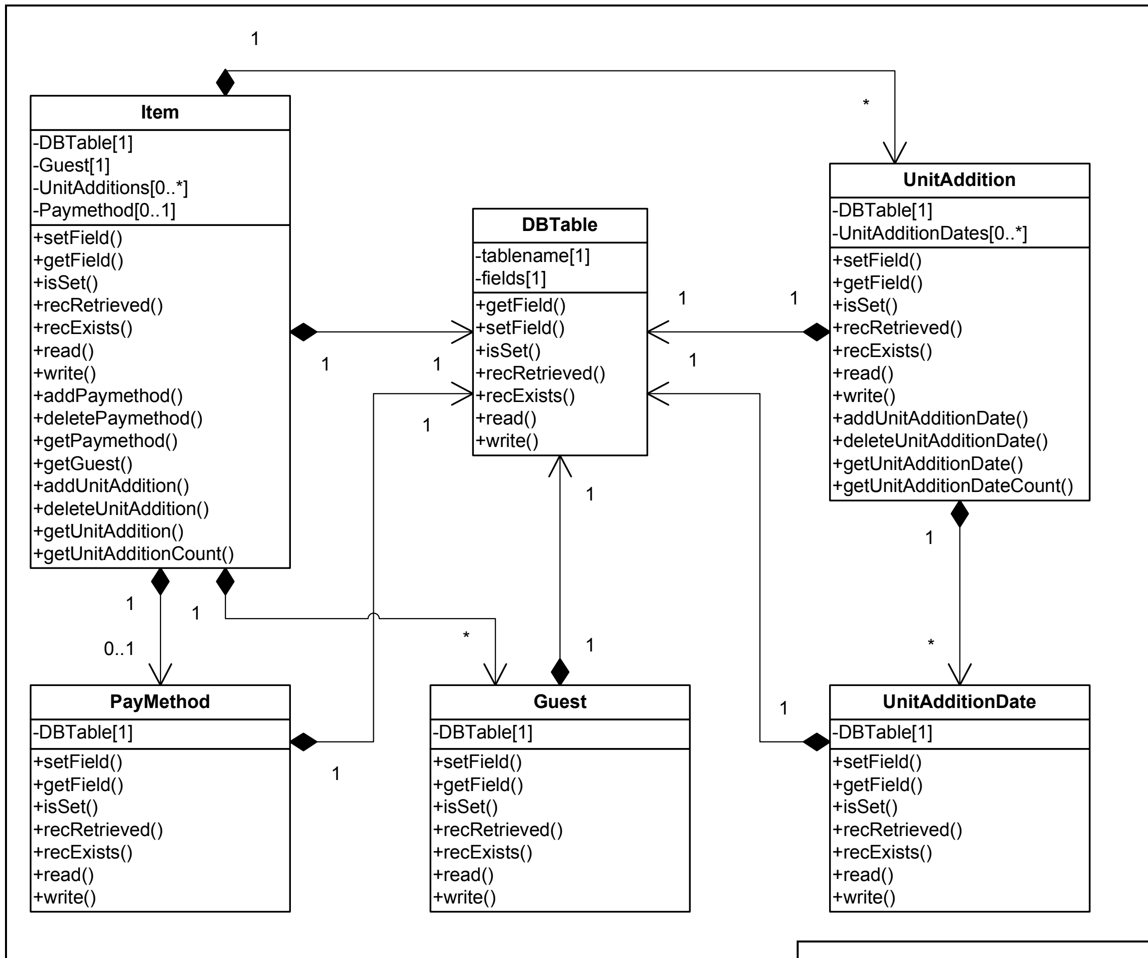

Fig 10. Booking object detail

The second of the two persistent objects is the Screen object. For each screen there is a different Screen object prototype. All these prototypes have the same methods, so the objects are polymorphic. The main methods are handler(), which is called after form submission, and writeScreen() which is called when a screen is loaded. The handler will process the information sent by the user and use it to fill the Booking object, and it will initialize a new Screen object if the submitted information was correct. If an error is detected, a flag is set, which will trigger a response when the form is written again. The writeScreen will write the form part of the displayed page. To do this it will typically start by retrieving information from the database, combining this with information previously entered in the Booking object, and putting this information into a specially designed data structure. This data structure is then used to build the actual HTML form, and at this to the response stream.

Each object has an important private member which is initialized as the object is created, and never changes (with one exception). This variable is a map, which contains all information about all buttons which link all the other Screens, like if a button is visible, the correct link, the style of the button, etc. The screen objects have a lot of other methods, which I will not discuss here. Most of them return parts of the button map, and other data about the display of certain items.

Conclusion

My goal at the start of this project was to learn to understand UML better, and get some insights into the designing of a software system in general, and this HRS in particular. I've made good progress to both goals. I will conclude with some of the findings I made on these subjects.

By making this paper I've made it a lot easier for other programmers to understand my code. UML was a very important tool to achieve that. The visual tools provided by UML give a clear and quick understanding of some mechanisms. I wasn't able to describe everything in UML though. Particularly the aggregations I was not able to completely visualize. There is no way to describe that all instances of a class A use a separate instance of a class B, and that an instance B is never shared between two instances of A, but more importantly, that it is never shared with an instance of any class.

Another thing I've missed in UML is a way to clearly visualize a decision process. Sometimes decisions are more complicated and cannot be captured in a single sentence. I don't have any clear ideas how this can be properly implemented, but I think it could be a lot clearer.

The hotel reservation system version two is sadly discontinued for the moment, so I probably cannot put my insights to very concrete use. I have made a lot of progress in the way I think about the system, particularly the distinction between the global workings and the details. In actually coding the software, this same distinction should be very clear. That will make the code clearer to new programmers, and really help future maintenance.

Even though this is an exceptionally big project in my work, I will be trying to incorporate UML designing into my design process of future projects. Especially the advantages during maintenance by different programmers might be a good way to win over my employers into investing time in this design. Off course this doesn't apply to every project, but I really think it will help some projects.