Seminar Software Engineering, WS 2001/2002

ImmoWarrior
e-Immo

Robert Harald Rescher
E-Mail: rrescher@cosy.sbg.ac.at

in group with
Lucia Fimberger, Marion Nidetzky, Oskar Schobesberger, Elisabeth Strobl.

Paris-Lodron-University Salzburg, Austria.
Institue for Computer Sciences.

Chapter 1

Remarks

1.1 About this Project

This project, ***e-Immo***, is an auction firm for real estates that runs via the Internet. The project was from the very beginning a group work, so noone has really done complete own drafts. But me, Robert Rescher, I was the first one who presented a concept based on three classes. This concept was splitted by all of us and refined until the now existing 10 classes. With the first draft we needed to know that there must be a kind of a system interface, but its responsiblites had changed several times. With the developpment of the HTML page, the classes and method calls were ready, since we now have the classes' and the methods' implementation hierarchical structured.

1.2 Principle of Method Calls

The principle of the method calls is very simple. The methods are called upwardly in the hierarchical structure. Since all the classes are aggregated (not inherited) hierarchically, there can be only one direction. There could also be a circular call, but it would be bad style. Whenever the caller expects from the callee something to be returned, the caller is waiting for an answer. The only caller that must not wait is the finally implemented system interface we will see later.

Chapter 2

Developpment

2.1 Rough Draft and first Refinements

First we had to create any classes that abstract our problem. We had a *system interface*, a *database* and a *representation processor*. This concept had to fall, because the classes were stuffed with methods they never will contain. Next we split the database into its entities and methods. We also knew that there were not only *Real Estates* to handle, but also *Users* and their *Access Rights*. Hence we split the database layer into two databases (a concept that was abandoned later). For both of these databases we had classes to *read* and *write*, so we had duplicates of these classes — this was the point the concept failed later on. Furhteron, the responsiblity of the system interface was very much; it had to call all methods of other classes. The only class that called the system interface was either another system-interface-object or the representation processor. The idea was that the interface to the callers of the system interface were complex data types that contained all the data that were needed to present the next mask or pass the results.

The idea that the application should communicate with another instance of itself was abandoned, although the return values stayed complex data types. The idea for not changing this was that the representation processor can interpret it differently — for example as HTML, WML, etc. Another class was added, the *Terminal*, it is an abstraction of the user's console. Any class had to initiate the representation processor to communicate with the system interface. With it the representation processor was no longer instructed to communicate with the user; it now did what its name said: It processes the representation of the data it gets returned from the system interface (the complex data types).

2.2 Unification of Method Calls

This concept we had created until then worked quite well until the point two user log in. Me, I had the idea to realize this in a main operating method of the system interface. The work the system interface had done before was now given to a new class in a large part. This new class is called *Session*. The main operating method of the system interface now did what its name said: It is the interface between the user the application, without it, the entire e-Immo-system can not work. There is an object of the Session created for each user. The System Interface has only the responsibility to handle a login and create the session's objects. This was also the point, where the concept with the two data bases in the database layer failed. From then on, the database layer contained all the entities (Real Estates, Users, Access Rights) as well as the classes *Enquirer* with only one method to fetch data, and a class *Recorder* with all the methods to write (add, delete, modify).

But now there was another capture word missing in our class list: *Login*. The System Interface can not handle two logins at a time, so we had to do the same with the login as we did with the session. These is the state presented now, and it seems to be perfect: A main operating method in the System Interface that has nothing else to do than to:

- ... instantiate login and session.

- ... wait for login to answer with the input, then call session to check the login.
- ... destroy login and, on logout or login fail, destroy session.

This method must not have modal login and session, otherwise the principle would not work. All sessions and logins must have the chance the run parallel without conflict.
All calls to the database layer run via the Session. This means that for the System Interface (where there is only one instance of it) we have reached that it has as little to do as possible. The calls to the database have unified look (non-circular calls on the underlying undirected graph), the System Interface is not needed during the session.

2.3 Sequence Diagrams

I was the one who had to present the sequence diagrams (together with the state diagrams) in the seminar. Since then, it was clear how the application should run. The main thing that took long to be defined exactly was how the login should work. It is the only point when not a method of the session or the system interface wants a mask to be displayed. The login method runs as long as the the data are being input. On submission, the login method return the data to the system interface which then makes the rest of the login.
The unified runs made it possible to define exactly when the database layer is accessed. A database has an own confliction solving algorithm that must be implemented in the enquirer and recorder. But this problem goes much too deep into file sharing violation and its solvations, they might be part of the implementation. With the sequence diagrams, $ImmoWarrior$ is stable usable so far.