UDC 004.32

Modeling of Login Procedure for Wireless Application with Interaction Overview Diagrams

Vera Plavšić[1] and Emil Šećerov[2]

[1]M - Rodić d.o.o. Temerinski put 50, 21000 Novi Sad, Serbia
Vera.Plavsic@mercator-rodic.com
[2] Faculty of Technical Sciences, Trg D. Obradovića 6, 21000 Novi Sad, Serbia
secerov@uns.ns.ac.yu

Abstract. In this paper we describe in details UML modeling of login procedure, which is a part of UserBarCodeReader application, developed for large stores and intended for use as a customer support during the shopping session. Login procedure is realized within access control system, in this case over a wireless network. Paper gives the whole modeling and implementation cycle of login procedure, from Use Case diagrams to the Java source code. Login procedure is modeled using interaction overview diagram, new in UML 2.0, which gives concise representation and divides complex sequence diagram into several smaller. The link between these diagrams is modeled with OCL postcondition and precondition expressions.

Keywords: wireless; UML; OCL; network; diagram; login; java; code.

1. Introduction

Purpose of this paper is to describe modeling and realization of Login procedure within UserBarCodeReader (UBCR) application that can be used in large shopping stores. Software modeling and realization process is described from specification to source code coding. As full example, standard Login procedure [1], which is part of UBCR application, is presented. Login procedure is a kind of protocol program that includes three consecutive steps (stages) with correspondent request-reply dialogue. In the simplest case one request-reply dialogue has three possible outcomes (positive and negative answer and occurrence of time-out). That means three dialogues generates 27 (3^3) possible scenarios, what is well known problem of exponential growth of software complexity. Fifth section of the paper shows how modeling of Login procedure can be divided into three disjunctive steps with only four scenarios, reducing the software complexity into linear order of magnitude. In this section generalization and quantification of proposed approach is also presented.

Intention of UBCR application is to help customers during the shopping session to control current bill, which depends on contents of the consumer basket. As in reference [2] motivation is to present the whole software development cycles, from use case (UC) diagram in Unified Modeling Language (UML) ([3], [4]) toward the source code production. New suggestions of the paper are usage of UML 2.0 [5] introduced interaction overview diagrams and constraints (preconditions, invariants and postconditions) modeling in OCL [6] as a link between the sequence diagrams referenced in interaction overview diagram. OCL constraints are top level specification for the programming by contract concept, which should be implemented in a source code. As a modeling tool Enterprise Architect (EA) 6.1 evaluation version [7], [8], was used, since it fully supports UML 2.0.

Architecture of the UBCR application is described in second section, presented in informal description and UML deployment diagram. Third section gives overview of three major UBCR application components: RegisteringNewCustomer, CurrentShoppingSession and VerifyingWith CashierBill. Current Shopping Session is further divided into more detailed components since it contains login UC. All use cases of the CurrentShoppingSession are described briefly.

Detailed use case of Login procedure in informal text, with formal OCL expressions, is described in section four, while section five presents its behavior modeled with three sequence diagrams (EnterUserId, EnterPassword and CreateSession). These diagrams are firstly referenced in interaction overview diagram of login procedure. Glue for sequence diagrams are OCL expressions, used as postconditions and preconditions. In the fifth section generalization of proposed software modeling process is also presented, with software complexity reduction metrics and diagrams that visualize the size of reduction.

Sixth section presents UML class diagrams derived from sequence diagrams in previous section.

UML activity diagram of ClientApplication is in section seven, while in section eight the fragments of source code with implemented OCL expressions are presented.

Conclusion of the paper is given in the section nine.

2. Overview description of UBCR application

Purpose of UBCR application is to help customers during their shopping sessions in large stores (megamarkets). Since this session includes taking a lot of articles it is very hard to control the total bill that will be at the end of the shopping. UBCR application assumes that customer has a portable personal digital assistant (PDA) or mobile phone with bar code reader device and wireless connection to the megamarket article database. After every import of article into the basket, customer scans the article bar code and manually enters its quantity. UBCR client application and megamarket database realize

request-response dialogue, where client application supplies product bar code identification and database returns the unit price. Then client application adds the price to the previous total sum and PDA displays current value of total bill (the price of basket contents).

UBCR application has three-tiered layer architecture consisting of client part, server part and database, as shown in deployment diagram in figure 1. Client device (e.g. PDA supplied with bar code reader) is connected to megamarget database via wireless network (IEEE 802.11 a/b/g) and executing UBCR application. When the consumer scan an article client part of the application request from megamarket information about unit price of the article. If the bar code entry about article exists in database server part of UBCR application returns price of the article. Megamarket database is organized as related SQL tables with alphanumeric fields. With three tiered organization of application direct communication between client part and database is avoided.

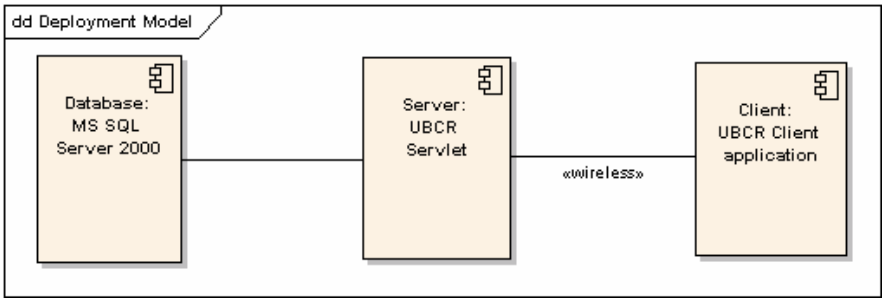


Fig. 1. UBCR application Deployment diagram

Client part of the application is implemented in JAVA programming language, in Net Beans development environment [9] with Mobility Pack add on to support wireless connection. For server application web Tomcat server is used and megamarket database is Microsoft SQL server 2000.

3. UBCR application components

UBCR application consists of several components. For each component functional description is in use case diagram followed by informal textual specification. Top level use case diagram of application is in figure 2. There are three major components: Registering NewCustomer, Current ShoppingSession and VerifyingWithCashierBill and four actors: Customer, AccessControlSystem (ACS), UBCR database and WirelessNetwork.

CurrentShoppingSession use case diagram is a composite diagram consisting of eleven more detailed use case diagrams, as shown in figure 3. EA UML tool supports composition and nesting inside use case diagrams, what simplified the modeling process.

Actors in CurrentShopingSession use case are:
Customer,
Access Control System: part of the system that identifies and authenticates customer based on (username, password) pair,
Wireless Network: resources of the wireless network that connects PDA device and megamarket database,
Database: megamarket information system containing data about registered customers, available articles, current user bill, user bills history and bill generated by cashier.

First step in UBCR application modeling was the use case diagrams description of the total functionality for realized system, decomposed into several independent functionalities. Besides the functional description it is desirable to define constraints that should be satisfied before and after the use case execution. With sequential composition of use cases postcondition of preceding use case is a postcondition for the subsequent use case. In the software development cycle, constraints will first be given in informal textual description, then in formal OCL expressions and finally in source code.

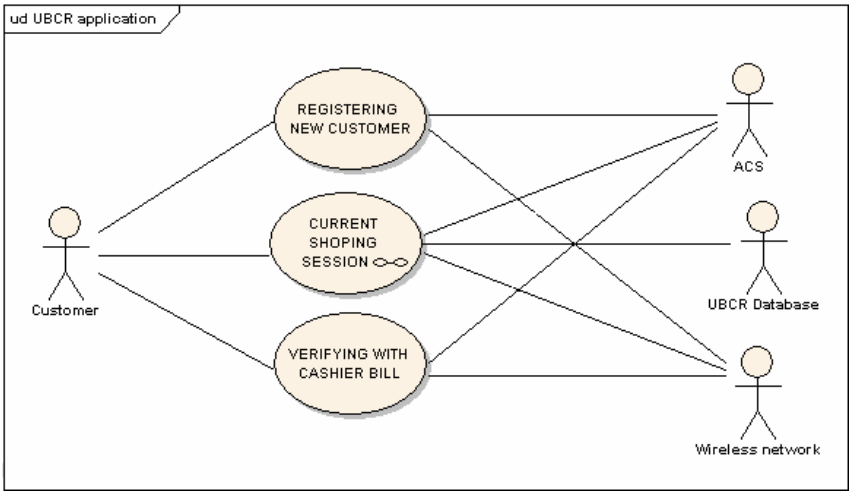


Fig. 2. Overview Use Case diagram of UBCR application

Figure 3 shows use case CurrentShoppingSession decomposed into eleven atomic use cases. Use case Connect, Login and SetupApplication will be discussed in details in the remaining part of the paper. Use case IdentifyProduct, ScanUPC (Unified Product Code) and AcceptManulEntry are responsible for identification of product, use cases AddArticle, CheckPrice and AcceptQuantity adds a new entry into current user bill. DeleteArticle is used to delete withdrawn entry and TotalBill closes the CurrentShoppingSession and they are only referenced in this paper.

Following part of this section gives textual descriptions for use cases connect, login and application setup. These use cases show how OCL

expression are the glue between the successive use cases, what is one of the important aspect of the paper. Postcondition of use case connect is precondition for use case login, while postcondition of use case login is precondition for use case application setup.

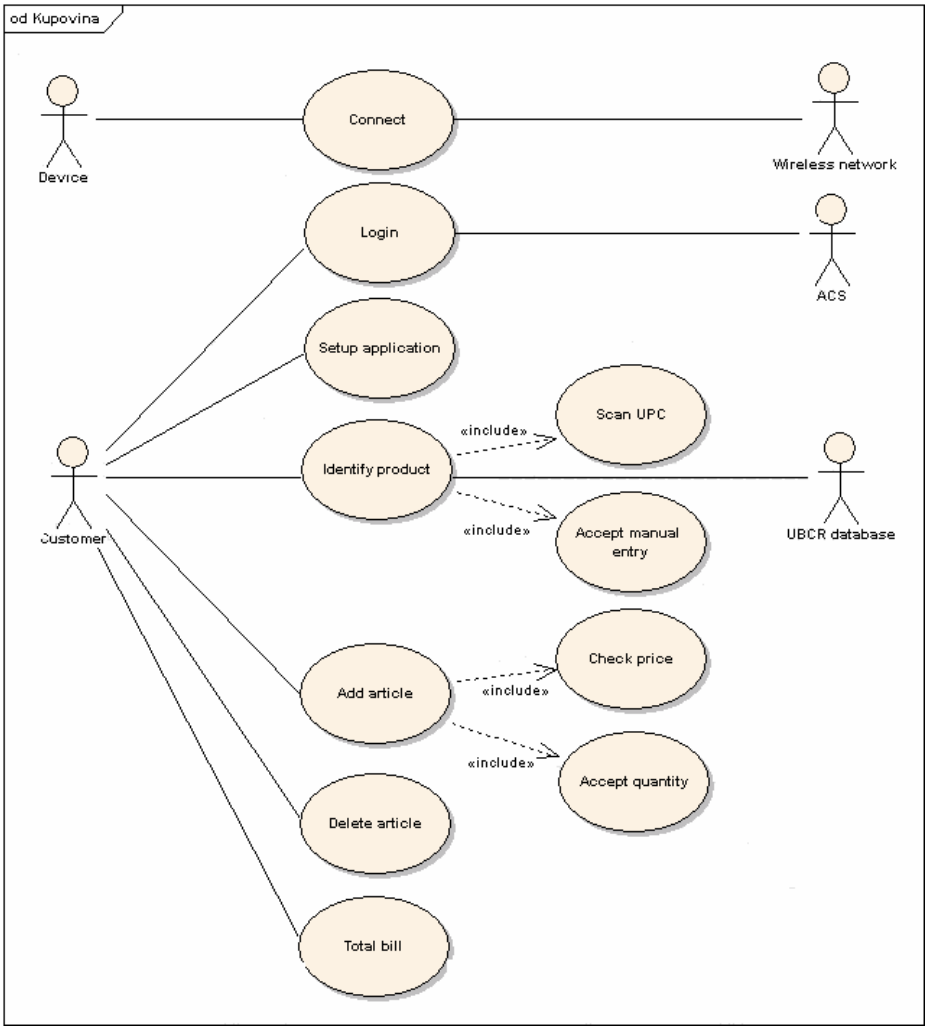


Fig. 3. Use Case CurrentShoppingSession

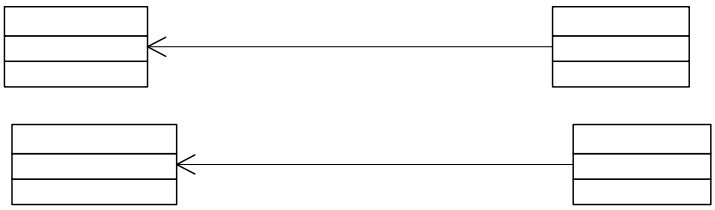
Textual description of Use Case: Connect
Actors: PDA device and Wireless network
Short Description: use case Connect realizes connection of PDA device supplied with bar code reader to the megamerket wireless network.
Preconditions: PDA device is in the wireless LAN area.
Invariant: if time-out occurs restart the procedure.

Postcondition: after the execution, PDA device is connected to the wireless network and authenticated by connection algorithm based on its hardware signature. Only registered devices may access the network.

OCL expression:

```
Context: ClientApplication
pre: self.itsIP_Connection.IsConnected = false
pre: self.itsBarCodeReader.IsAuthenticated = false
post: self.itsIP_Connection.IsConnected = true
post: self.itsBarCodeReader.IsAuthenticated = true
```

Figure 4 gives the fragment of Class Diagram that can be linked to OCL expressions of use case connect.



Textual description of Use Case: Login

Actors: Customer and Access Control System (ACS).

Short Description: Use Case Login verifies customer access to the megamarket database based on (UserId, Password) pair that is entered by user on its PDA device.

Preconditions: PDA device is connected to the wireless network.

Invariant: if time-out occurs restart the procedure.

Postconditions: if both UserId and Passwords are valid user is logged in to the megamarket database.

OCL expressions:

```
Context: ClientApplication
pre: self.itsIP_Connection.IsConnected = true
pre: self.SessionValid = false
post: if( UserId = Registred AND Password(UserId) =
      Valid) self.SessionValid = true
```

Detailed description of use case Login is given in section 4, since the remaining part of the paper deals with entire software production cycle with login procedure as an example.

Textual description of Use Case: SetupApplication

Actor: Customer.

Short Description: This use case assumes that customers enter the UBCR client application initial values: MaxTotalPrice that maximizes customer's total bill for that shopping session and MaxUnitPrice that maximizes the unit price of the articles in the basket. TotalBill is set to zero.

Preconditions: customer must be identified and authenticated.

Invariant: if time-out occurs restart the procedure.
Invariant: wireless connection exists during the session
Postconditions: after the execution MaxUnitPrice and MaxTotalBill variables are initialized and Total bill is set to zero. Also UBCR client application is initialized.

OCL expressions:

```
Context: ClientApplication
pre: Customer::IsAuthenticated = true
pre:  self.IsInitialized = false
post:  self.Application.MaxUnitPrice > 0
post:  self.MaxTotalPrice > 0
post:  self.TotalBill = 0
post:  self.IsInitialized = true
```

4. Use Case Login

Use case diagram Login is presented in figure 4, while its full informal textual description is given below.

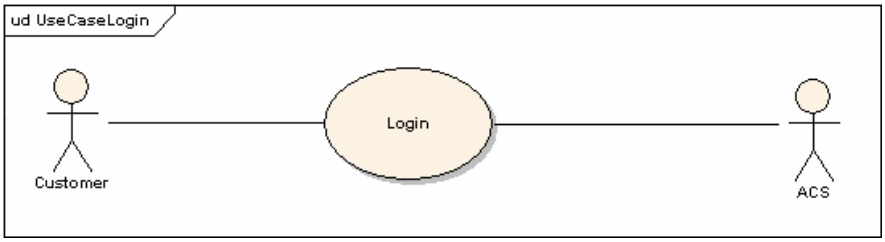


Fig. 4. Use Case Login

4.1. Textual description

1. Customer acquires device (PDA with bar code reader)
preconditions: wireless connection exists.
Bar code reader is authenticated.
invariant: if time-out occurs restart the procedure (go to step 1).

2. Customer starts the application pushing the Start button
precondition: customer unidentified
invariant: if time-out occurs restart the procedure (go to step 1).
2.a. ACS returns prompt "Enter User Id"
2.b. User enters its UserId
2.c. ACS checks UserId

If UserId doesn't exist in ACS database
ACS sends "Invalid User Id"
go back to 2.a
postcondition: customer unidentified
If UserId exists in ACS database
count = 1
postcondition: customer identified

3. Customer Enters Password
precondition: customer identified
precondition: customer is not authenticated
invariant: if time-out occurs restart the procedure (go to step 1).
3.a. ACS returns prompt "Enter Password"
3.b. User enters its Password
3.c. ACS checks Password(UserId)
If password doesn't match
count = count+1
if(count equal 4) go back to 2.a
ACS sends "Invalid Password"
go back to 3.a
postcondition: customer is not authenticated
If password match
go to 4.
postcondition: customer authenticated
4. Customer accessed the system
postcondition: customer authenticated

4.2. OCL expressions for login use case

OCL expressions are derived from the informal textual description of login use case.

1. User acquires device
Context: ClientApplication.Start()
pre: ConnectionExists = true
pre: BCR authenticated = true
inv: on time-out go to 1
2. User starts the UBCR client application
Context: ClientApplication.EnterUserId()
pre: UserValid = false
inv: on time-out go to 1
post:
if (ACS.UserValid() == false)
UserValid = false
else
UserValid = true

3. Password verification
Context: ClientApplication.PasswordVerification()
pre: UserValid = true
inv: on time-out go to 1
post:
if (ACS.PasswordInvalid() = false)
UserAuthenticated = false
else
UserAuthenticated = true
4. User gets the access to the system
Context: ClientApplication.SetupApplication()
pre: UserAuthenticated = true

5. Interaction overview diagram for Login procedure

In UML, sequence diagrams are used to define scenarios of interaction between the environment and system classes (objects) during the program execution. Before UML version 2.0, it was required one sequence diagram for each path of execution that depends on possible decisions in program, e.g. if(exist(CustomerRecord(Id))). To avoid this unnecessary redundancy UML 2.0 interaction overview diagrams may be used.

From UML version 2.0 the interaction overview diagrams can be used to surpass these unnecessary redundancies. Interaction overview diagrams show how atomic sequence diagrams are combined into more complex scenarios avoiding drawing every possible path.

Login procedure interaction overview diagram is shown in figure 5. Diagram includes three interaction occurrence objects: EnterUserId, EnterPassword and CreateSession. These objects are references for the existing sequence diagrams. Corresponding sequence diagrams are in figures 6, 7 and 8 respectively.

It is clear to see that interaction overview diagram gives concise representation of whole login use case, while referenced sequence diagrams are connected with logical expressions. These expressions are used in further software implementation process as OCL expressions and finally implemented in the source code without any significant change, except only syntax.

Modeling with sequence diagrams would require eight diagrams for three logical expressions ($2^3 = 8$), while in proposed approach there are only four diagrams, one for overview and three for atomic scenarios.

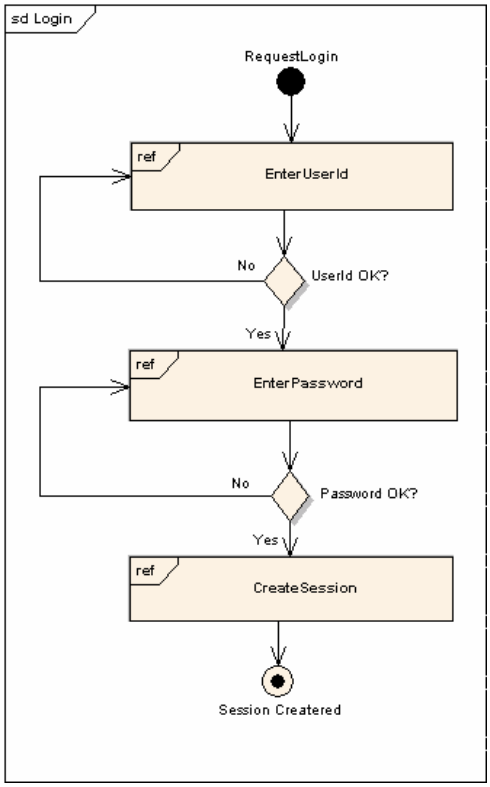


Fig. 5. Interaction Overview diagram for Use Case Login

Logical condition that connects former and latter sequence diagram is postcondition and precondition constraint. OCL formalization of these expressions leads to source code generation, as shown in section 8. From figure 5 it is clear which activities are realized in use case login. First of all, customer requests login, and enters user identification (UserId). If UserId is registered in database, the procedure continues, if not procedure goes back to the first step. Similar is password authentication and if all supplied data are correct UBCR client application session is initiated.

5.1. Sequence diagram: EnterUserId

EnterUserId sequence diagram is represented in figure 6 giving details for EnterUserId object first referenced in interaction overview diagram. At the diagram there is complete information about message/signal flow between the customer, client application and megamarket database. Customer starts the client application pushing the Start button, then application returns prompt “EnterUserId“ and expects customer’s entry. When the customer enters his

user identification, client application, in dialogue with database, verifies user input. If verification is successful UBCR application continues to the next step, illustrated with sequence diagram EnterPassword.

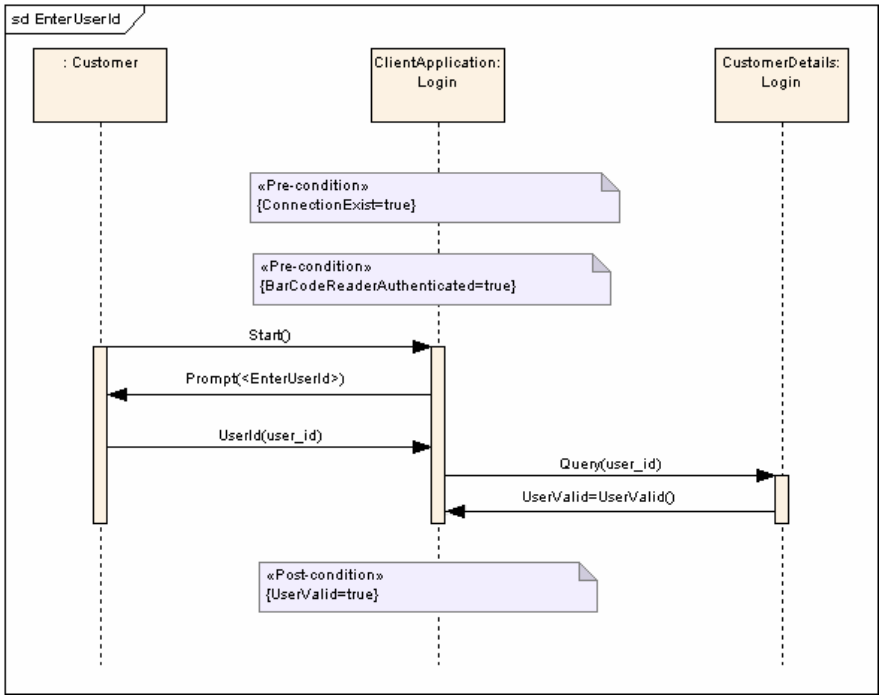


Fig. 6. EnterUserId sequence diagram

OCL constraints in EnterUserId sequence diagram define preconditions and postconditions that should be satisfied at the beginning and at the end of execution. Precondition for EnterUserId is existence of connection between the client application and megamarket database (ConnectionExists = true). After the execution, if there is a record in ACS database for entered user identification, user identification has to be valid (UserValid = true).

5.2. Sequence Diagram: EnterPassword

Enter Password sequence diagram represented in figure 7 gives details for EnterPassword object referenced in interaction overview diagram. This diagram is sequential continuation of EnterUserId sequence diagram. First ClientApplication sends prompt “EnterPassword” and expects customer entry. After that, customer enters password and application compares user entry with password in megamarket database for that customer. If passwords are identical, client application continues.

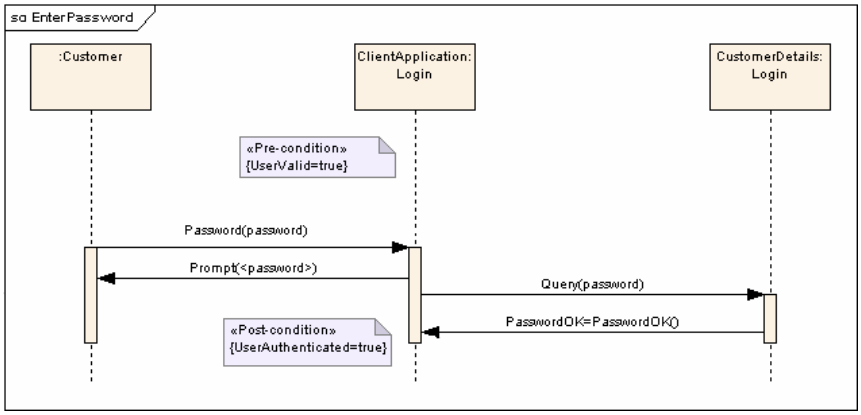


Fig. 7. EnterPassword sequence diagram

Precondition for EnterPassword is that entered username must be valid (UserValid = true). Postcondition for EnterPassword is that customer must be authenticated for use of application (UserAuthenticated = true).

5.3. Sequence diagram: CreateSession

CreateSession sequence diagram represented in figure 8 is also first referenced in interaction overview diagram. In the diagram, there are messages which flow between customer, application and database when customer session is created.

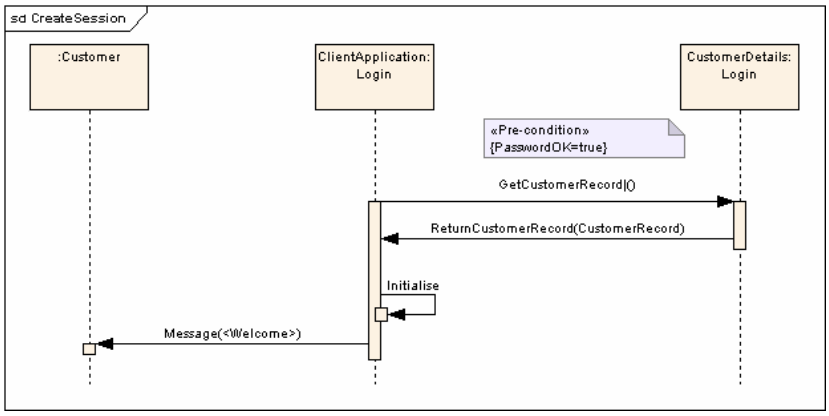


Fig. 8. CreateSession sequence diagram

ClientApplication retrieves initial program data about authenticated user from database and initiates program variables for use in started session. After that, Welcome message is sent to client PDA device.

Precondition for CreateSession sequence diagram is that the user of system must be identified and authenticated. Outcome of CreateSession execution is initialization of user data and welcome message is seen at the screen of the user's device.

5.4. Generalization of proposed software modeling approach

Reduction of software complexity in the modeling phase is important since it directly affects coding and testing phases and it is well known that these phases of software development cycle generates major cost of software production. This section gives generalization for software reduction based on usage of interaction overview diagrams. In the following text some simplification are made to show qualitative aspect of the method instead of accurate quantitative approach. In any case exact quantification is an artifact for each software project itself.

We shall use the following notation in the rest of section:

S – signal, A – answer, S_i – i-th signal, n – number of signals, m – number of interaction overview diagrams, NSD – number of sequence diagrams.

If there is no software modeling reduction:

$$NSD = \prod_{i=1}^{n} A(S_i) \tag{1}$$

While with reduction

$$NSD = \sum_{i=1}^{m} \left(\prod_{j=1}^{k_j} A(S_j) \right) \tag{2}$$

where k_j is the number of signals in the j-th interaction overview diagram.

In the most simple case when there are two possible answers for each signal, or message, (yes and no) and time constraint (time-out occurrence) $A(S_i) = 3$ (yes, no and time-out).

For n signals

$$NSD = 3^n \tag{3}$$

Equation 3 imposes exponential dependency what means that after few input signals software will be complex.

If we use m interaction overview diagrams:

$$NSD = m \cdot 3^{\frac{n}{m}} \tag{4}$$

what lead to linear increase of complexity during the software growth.

In the more general case number of possible answer messages depends on input signal/message, but linear (equation 5) or product (equation 6) mean are the good approximation for the average number of answers:

$$\overline{A_a} = \frac{\sum_{i=1}^{n} A(S_i)}{n} \tag{5}$$

$$\overline{A}_g = \frac{\prod_{i=1}^{n} A(S_i)}{n} \tag{6}$$

Then we can write:

$$NSD = \overline{A}^n \tag{7}$$

While modeling with interaction overview diagrams diagram gives:

$$NSD = \sum_{i=1}^{m} \overline{A}^{\frac{n}{i}} \tag{8}$$

Some examples of reduction metrics are shown in section 5.4.1.

Conclusion for this section is that software modeling with interaction diagrams reduces software complexity from exponential into linear degree, what is important for management and costs of software production cycle.

Further steps in developing of the proposed method will lead to form a repository of standard interaction overview diagrams. Every diagram will be described with number of signals and answers correspondent to each signal. In that matter complexity of interaction overview diagram is describer, while a complete program is composition of diagrams in repository. That can lead toward parameterized program or protocol realization (modeling and coding).

5.4.1 Examples of metrics

Table 1 shows the number of scenarios according the number of signals and number of answers. Graphic at figure 9 shows decrease a number of scenarios according the relation $\frac{NSD\ [B]}{NSD\ [A]}$ when $A(S_i) = 3$

Table 1. Example for different number of signal

	n = 2	n = 3	n
S_1	Signal 1	Signal 1	Signal 1
S_2	Signal 2	Signal 2	Signal 2
S_3	-	Signal 3	Signal 3
…	-		…
S_n	-		Signal n
$A(S_1)$	$A(S_1)=3$	$A(S_1)=3$	$A(S_1)=3$
$A(S_2)$	$A(S_2)=3$	$A(S_2)=3$	$A(S_2)=3$
$A(S_3)$	-	$A(S_3)=3$	$A(S_3)=3$
…	-	-	…
$A(S_n)$	-	-	$A(S_n)=3$
NSD [A]	$A(S_1)\cdot A(S_2)$ $=3\cdot 3=3^2=9$	$A(S_1)\cdot A(S_2)\cdot A(S_n)$ $=3\cdot 3\cdot 3=3^3=27$	$A(S_1)\cdot A(S_2)\cdot\ldots\cdot A(S_n)$ $=3\cdot 3\cdot\ldots\cdot 3=3^n$
NSD [B]	$A(S_1)+A(S_2)$ $=3+3=3\cdot 2=6$	$A(S_1)+A(S_2)+A(S_3)$ $=3+3+3=3\cdot 3=9$	$A(S_1)+A(S_2)+\ldots+A(S_n)$ $=3+3+\ldots+3=3n=12$
$\frac{NSD[B]}{NSD[A]}$	$=\frac{6}{9}=0{,}67$	$=\frac{9}{27}=0{,}33$	$=\frac{3n}{3^n}$

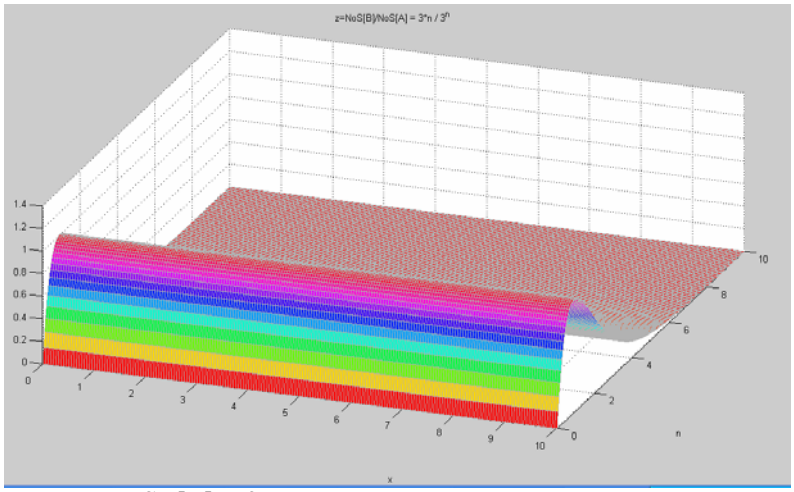


Fig. 9. Graphic $z=\frac{NSD[B]}{NSD[A]}=\frac{3n}{3^n}$

6. Class diagram from sequence diagrams

From the sequence diagrams, UBCR client application UML class diagrams can be directly obtained. For each sequence diagram there is exactly one class diagram. Every communicating object in sequence diagram represents a class while message flows between classes form an association. In following three class diagrams (figures 9, 10 and 11) same classes are addressed, but with different attributes and methods/operations representing different point of view. With three different and disjunctive views it is easier to capture and model atomic structure of the application. Complete class definition (figure 12) is merged from these diagrams, what is automatically supported by the modeling and code generation tool.

6.1. Class Diagram - Sequence Diagram: EnterUserId

Class diagram in figure 10 corresponds to the figure 6 sequence diagram. There are two classes at the diagram: ClientApplication and CustomerDetails, representing the UBCR client part of application and part of the megamarket database containing information about registered customer. ClientApplication is an active class, what means it is the controlling process. This class has UserValid attribute and four operations: InputUserId (String), Prompt (const String “EnterUserId”), QueryUserId (String) and Start (). CustomerDetails class has only UserValid (String) operation that returns value depending on the existence of user identification.

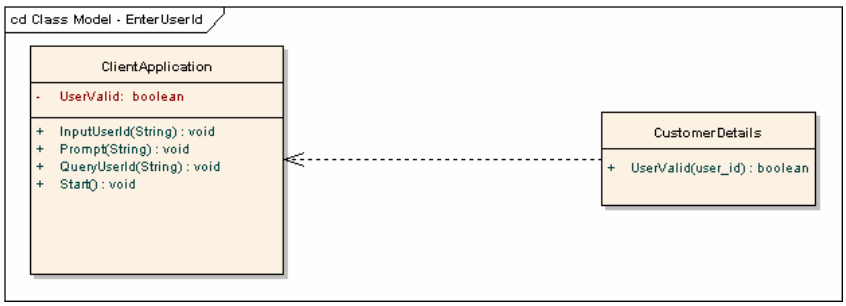


Fig. 10. EnterUserId class diagram

From sequence diagram it is clear that ClientApplication::UserValid attribute is set to the return value of CustomerDetails::UserValid() operation during the execution of scenario described by the diagram.

6.2. Class Diagram - Sequence Diagram: EnterPassword

EnterPassword class diagram in figure 11 is derived from the sequence diagram in figure 7. ClientApplication class has a Password attribute and PasswordValid attribute. ClientApplication class has two operations of void type: Prompt (const String “EnterPassword”) and QueryPassword (String). CustomerDetails class has only PasswordValid () operation of Boolean type. During the ClientApplication program execution, attribute PasswordValid:: ClientApplication is set to the return value of CustomerDetail:: PasswordValid( ) operation.

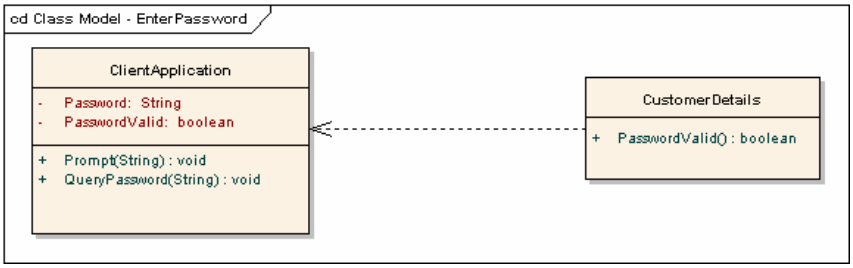


Fig. 11. EnterPassword Class diagram

6.3. Class Diagram - Sequence Diagram: CreateSession

Class diagram from figure 12 is related to the sequence diagram from figure 8 with same two classes as in previous class diagrams. ClientApplication class has customer_record attribute of CustomerRecord type and Message (const String “Welcome”) operation of void type. CustomerDetails has ReturnCustomerRecord (String customer_record) operation of CustomerRecord type. That operation returns customer’s data to ClientApplication.

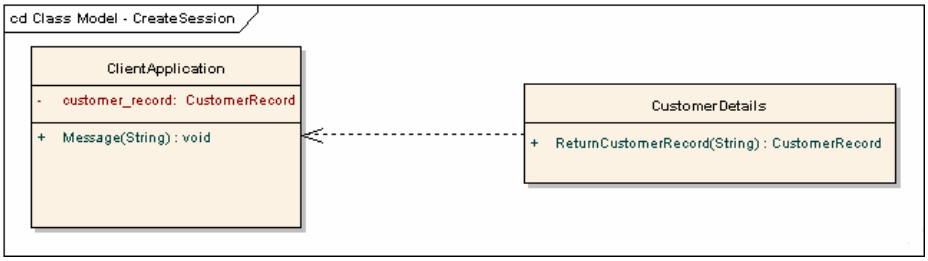


Fig. 12. CreateSession Class diagram

6.4. Composed ClientApplication class diagram (on the basis of three class diagram)

ClientApplication class which is shown in previous three class diagram, in each of those diagrams there are attributes which are important for observed sequence diagram. However, on the basis of those three diagrams, it is possible to generate code. Result of code generation is one ClientApplication class which has attributes and operations from all three class diagrams. Composed class diagram, with complete definition of ClientApplication class in use case login, is shown in figure 13.

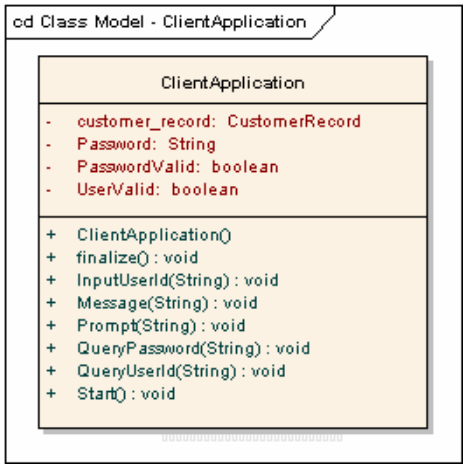


Fig. 13. Composed ClientApplication class

Figure 13 was automatically generated by EA UML modeling tool that merged class diagrams from figures 10, 11 and 12.

7. Activity Diagram for Class ClientApplication

In UML an activity diagram is used to display the sequence of activities and it is usual in business process modeling. Activity diagrams show the workflow from a start point to the finish point. Activity diagram for class ClientApplication is shown in figure 14, and it is practically a copy of interaction overview diagram from figure 5.

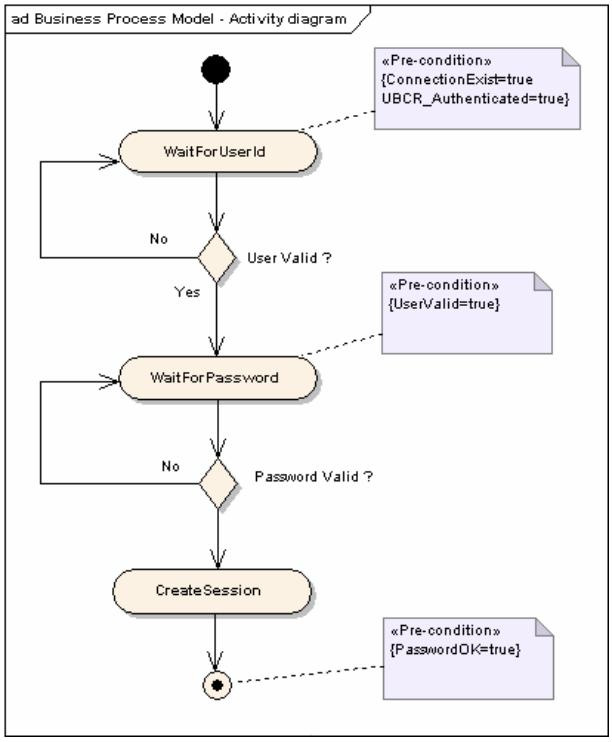


Fig. 14. Activity diagram for class ClientApplication

8. Code generation

On the basis of class diagram, it is possible to generate fragment of code in appropriate programming language. In this project, JAVA is used as a programming language.

Fragment of code for class ClientApplication is shown in listing 1.

Listing 1:

```
public class ClientApplication {
```

```
            private CustomerRecord customer_record;
            private String Password;
            private boolean PasswordValid;
            private boolean UserValid;

            public ClientApplication(){
              //OCL: Customer gets BarCodeReader
              Assert(ConnectionExists = true);
              Assert(BarCodeReaderAuthenticated = true);
            }

            public void InputUserId(String user_id){
              //OCL: Input UserID
              Assert(UserValid = true;
              if( ACS.PasswordInvalid() = false ) then
                Assert(UserAuthenticated = false);
              else
                Assert(UserAuthenticated = true);
            }

            public void Prompt(String EnterPassword){
              //OCL: Customer access to the system
              Assert(UserAuthenticated = true);
            }

            public void Start(){
              //OCL: Customer starts application
              Assert(UserValid = true);

              if( ACS.PasswordInvalid() = false ) then
                Assert(UserAuthenticated = false);
              else
                Assert(UserAuthenticated = true);
            }
}
```

Listing 1 contains attributes from all three class diagrams: EnterUserId, EnterPassword and CreateSession, from figure 10, 11 and 12 respectively. It contains operation from class ClientApplication, which is shown in three class diagrams. Code has appropriate constraints which are stated in OCL language. OCL validation is made in UML modeling tool which has support for OCL.

9. Conclusion

In this paper, for purpose of modeling login procedure within UBCR application we used different kind of UML diagrams: from deployment diagram, use case diagrams, interaction overview diagram and sequence diagrams toward class diagrams. Code was generated from the class diagrams. All presented diagrams model the login procedure in different way.

The use case diagram is starting point to define actors and analyze system functionalities. It also shows the interaction between the system and external entities. Relation between use cases helps system structure analysis. Informal textual description of use case is also a first step to define system behavior. However, use cases do not describe system structure, class and object existence and details of behavior.

Therefore, for modeling application behavior and detailed structure we use interaction overview diagram and sequence diagrams. On the basis of those diagrams, it is possible to follow messages that are exchanged between objects and realize system functionality. Paper also gives a general approach to software modeling technique, based un use of interaction overview diagrams, which significantly reducing software complexity and therefore also reduces software development costs.

Next step toward code generation is to generate class diagram from each sequence diagram. Class diagram that contains all functionalities of the system is obtained with merging of simple class diagrams.

Program source code can be generated from composite class diagram.

Through all of those diagrams we describe constraints, in informal language and OCL. OCL expressions are carried directly to the program source code.

Also using OCL is an implementation of programming by contract paradigm, which is shown explicitly.

UBCR application development will include more sophisticated login procedure, that include cryptographic facilities, what means that presented procedure will be extended with new dialogues. Software modeling approach presented in this paper guaranties that new properties will require only linear increase of software production (modeling, coding and testing) process.

10. References

1. USA Department of Defense, Orange book, 1985
2. Fawzi Albalooshi: Computer-Aided Software Artifacts Generation at Different Stages within the Software Development Process, University of Bahrain, Kingdom of Bahrain, 2005
3. Grady Booch, James Rumbaugh, Ivar Jacobson: The Unified Modeling Language User Guide, CET Computer Equipment and Trade, Belgrade, 2000
4. Ivana Stanojević, Dušan Surla: Introduction in the unified modeling language, Group for information technologies, Novi Sad, 1999
5. Unified Modeling Language: Superstructure version 2.0 formal/05-07-04, www.uml.org, 2006
6. Object Constraint Language Specification OMG-UML V1.3, www.uml.org, 2006
7. Online, http://www.sparxsystems.com/EAUserGuide/index.html , 2007
8. Online, http://sparxsystems.com.au/resources/uml2_tutorial/ , 2007
9. Online,http://developers.sun.com/prodtech/javatools/jsenterprise/learning/tutorials/index.jsp , 2007

Vera Plavšić received her M. Sc. (5 year, former Diploma) degree at the University of Novi Sad, Faculty of Technical Sciences in 2000 at the Department of Telecommunication. In 2000. she started postgraduate study in Computer Sciences at the University of Novi Sad, Faculty of Technical Sciences. She passed all exams and now works on her master thesis. Her research interests are in the area of telecommunication network, wireless network, network security. Currently, she works as IT Manager at "M Rodić" d.o.o. company.

Emil Šećerov received his Ph. D. degree at the University of Novi Sad, Faculty of Technical Sciences in 1998, department of computer science. In 2000. he works as associated professor at the department of telecommunication, Faculty of Technical Sciences. His research interests are in the area of telecommunication network, computer systems, wireless network and network security.

Received: December 18, 2006; Accepted: January 18, 2008.