Зборник радова Факултета техничких наука, Нови Сад

UDK: 004.9
DOI: https://doi.org/10.24867/18BE16Jovic

РАЗВОЈ АПЛИКАЦИЈЕ ЗА УПРАВЉАЊЕ РЕСТОРАНОМ РАДИ ПОМОЋИ СТУДЕНТИМА У УСВАЈАЊУ ЗНАЊА ИЗ ПРЕДМЕТА СПЕЦИФИКАЦИЈА И МОДЕЛОВАЊЕ СОФТВЕРА

DEVELOPMENT OF RESTAURANT MANAGEMENT APPLICATION AS A LEARNING EXAMPLE FOR SOFTWARE SPECIFICATION AND MODELING COURSE

Лазар Јовић, *Факултет техничких наука, Нови Сад*

Област – ЕЛЕКТРОТЕХНИКА И РАЧУНАРСТВО

Кратак садржај – *Држећи вежбе на предмету Спецификација и моделовање софтвера, на другој години смера Софтверско инжењерство и информационе технологије, увиђено је да постоји потреба за пројектом који би студентима пружио увид у комплетно градиво примењено на реалан систем. Приказан је пут развоја софтвера, у који су укључени спецификација, моделовање, као и имплементација апликације која представља систем за управљање ресторана. Дијаграми који представљају UML моделе система креирани су у алату Magic Draw, док је имплементација рађена у Java програмском језику, уз потребу Swing библиотеке за графички кориснички интерфејс.*

Кључне речи: *моделовање софтвера, спецификација софтвера, UML*

Abstract – *While holding excercises on the subject of Software Specification and Modeling, on the 2nd year in Software Engineering and Information Technology module, it was realized that there is a need for a project that would provide students with an insight into the complete material of subject applied to a real software system. The path of software development is presented, which includes specification, modeling, as well as the implementation of restaurant management system. Diagrams, which represent UML models of the system, were created using Magic Draw tool, while implementation was done using Java programming language and Swing library for graphical user interface.*

Keywords: *software modeling, software specification, UML*

1. УВОД

Данас изазови развоја софтвера обухватају процес прикупљања захтева од стране клијената и корисника, дизајнирање система, преко саме имплементације до његовог одржавања и адаптације на нове праксе које су стално присутне у области софтверског инжењерства. У свим наведеним фазама најчешће учествују људи различитог искуства и из различитих области индустрије.

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији ментор је била проф. др. Гордана Милосављевић.

Како би сви они могли да прецизно и једноставно комуницирају потребно је користити такав језик који превазилази нејасноће природног језика. То је сврха језика за моделовање софтвера међу којима преовлађује *UML* (*Unified Modeling Language*) [1]. Након израде модела употребом *UML* језика, потребно је пратити добре праксе приликом имплементације система, што олакшава касније одржавање и спровођење евентуалних измена. Рад се управо бави применом горенаведеног на систем за управљање рестораном, који ће студенти на предмету Спецификација и моделовање софтвера моћи да користе приликом савладавања градива и обављања задатака који се пред њих стављају.

2. ТЕОРИЈСКЕ ОСНОВЕ

2.1. Дефиниција и циљеви модела

Реалан систем представља целину интегрисану од више компоненти које се могу посматрати као независне јединице од којих је систем састављен али му заједно омогућавају да испуни своју функцију. Модел дозвољава да на јасан и ефикасан начин представимо реалан систем. Врло је важно да подразумева такву нотацију коју сви на исти начин могу тумачити. Најважнија особина модела јесте **апстракција**, без које бисмо морали да увек узимамо у обзир комплексност реалног света. Према нивоу апстракције деле се на: **дескриптивне** и **прескриптивне**. Дескриптивни се користе за боље разумевање система, документовање и остваривање добре комуникације. Прескриптивни се користе ради специфицирања како неки систем треба да се имплементира [2].

2.2. Изазови у области моделовања и спецификације софтвера

Софтверски системи испуњавају одређену пословну сврху и функцију, које су дефинисане од стране корисника на основу њихових потреба приликом употребе неког софтверског система [3]. Процес развоја софтвера креће од прикупљања доменског знања у виду корисничких захтева. Након тога, следи анализирање захтева и моделовање на основу направљених анализа. Ове две активности се спроводе инкрементално и итеративно до задовољавајућег нивоа разумевања. Следи фаза имплементације која се значајно аутоматизује добро измоделованим системом. Као крајње фазе јављају се верификација

имплементираног и одржавање система пуштеног у примену.

2.3 Врсте *UML* дијаграма

Захтеви прикупљени од корисника могу бити функционални и нефункционални. Функционални захтеви дефинишу функције које систем треба да обезбеди корисницима и другим системима који су са њим у интеракцији. Нефункционални дефинишу додатна ограничења која се постављају пред систем [2]. *UML* нуди **дијаграм случајева коришћења** као средство дефинисања функционалних захтева које систем треба да испуни. Случај коришћења (енгл. *use case*) представља функцију система којом се пружа одређена услуга. Дијаграм такође дефинише и учеснике у функционисању система. Веза између случаја коришћења и учесника се зове асоцијација. Такође, случајеви коришћења могу бити међусобно повезани везама проширивања (енг. *extend*) и укључивања (енгл. *include*).

Дијаграм активности користи се за моделовање процеса било какве врсте. Реализују се управљањем тока активности и података који чине одређени процес и то усмереним графом где су чворови различити елементи дијаграма активности повезани гранама које моделују смер извршавања.

Основни елемент јесте акција која представља један корак активности (процеса) која се моделује. Свака акција је јединствено идентификована називом. Могу бити обичне и сложене, где сложене садрже више корака који се користе у оквиру главне активности. Контрола тока представља скуп грана које повезују елементе дијаграма. За сваку грану може бити везан услов њеног извршавања. Поред наведених, остали елементи дијаграма су условно извршавање, симболи за ток објеката (служе за дефинисање преноса података у оквиру моделоване активности), разделник, слање и пријем сигнала, партиције (моделују груписање акција које се врше од стране једног корисника) итд.

Постоје концептуални и имплементациони **дијаграм класа**. Концептуални имају задатак да моделују домен везан за софтвер који се развија и садрже мање детаља од имплементационих, трудећи се да ставе акценат на односе између његових делова. Имплементациони дијаграм може да садржи детаље до нивоа обележја и метода класа које ће се користити приликом имплементације, тј. налази се на нижем нивоу апстракције.

Класа је основни елемент овог дијаграма и представља шаблон за креирање објеката који чине моделовани систем. Свака класа поседује обележја и методе за које постоји дефинисан начин специфицирања. Основна веза референцирања између класа је асоцијација, одређена пре свега називом и кардиналитетом који представља број објеката са друге стране везе. Специјална врста ове везе је агрегација која моделује однос „целина-део“, док је композиција посебан тип агрегације код ког класа „део“ не може да егзистира самостално. Генерализација дефинише наслеђивање између класа, тј. моделује однос „родитељ-дете“.

Дијаграм секвенце моделује комуникацију између елемената и учесника система. Комуникација подразумева размену различитих врста порука које представљају садржај интеракције између учесника [2]. Улоге репрезентују учеснике моделованог дела комуникације. Свака улога је одређена својом животном линијом, која означава опсег времена у којем је дата улога активна, што подразумева примање и слање порука. Размена порука репрезентована је симболом активације. Постоје различити типови порука: синхроне, асинхроне, креирање објекта, ослобађање објекта. Такође, могу да се користе фрагметни, који моделују различите контроле тока (услове, петље и слично).

Дијаграм стања користи се за моделовање коначног скупа стања у којима се део система или цео систем може наћи, где је прелаз између стања иницран догађајима. Представљен је као граф чији су чворови стања, а гране су транзиције између стања. Стање описује услове у којима се објекат система налази у одређеном временском тренутку. Објекат се у сваком тренутку мора налазити у неком од стања. Транзиција моделује реакцију на догађај који изазива прелазак из једног стања у друго или повратак у полазно стање. Транзиција може имати и скуп услова који дефинишу могућност извршавања транзиције. Акција представља последицу окидања одређене транзиције или налажења објекта у конкретном стању.

3. КОРИШЋЕНЕ ТЕХНОЛОГИЈЕ

3.1 *MagicDraw*

Представља визуелни *UML* алат за моделовање. Служи за анализу и дизајнирање објектно оријентисаних система и база података. *UML* тип пројекта омогућава креирање различитих врста дијаграма. Креирање дијаграма се врши на радној површини која графички представља све елементе из одговарајућих палета [4].

3.2 *Java* и *Java Swing*

Приликом имплементације система за управљање рестораном коришћен је програмски језик *Java*, а пре свега библиотека ***Swing*** коришћена за реализацију графичког корисничког интерфејса (енгл. *GUI*). Подржава и *MVC* архитектуру која је примењена приликом имплементације. Контејнер је основни појам који се спомиње у раду са *Swing* библиотеком и има такву особину да може да садржи друге контејнере унутар себе. Постоје контејнери на највишем нивоу (енгл. *top-level*) који не могу бити садржани у другом контејнеру, нпр. класе *JFrame* и *JDialog*. Друге коришћене класе су *JPanel* за груписање елемената, *JButton* која представља дугме или *JTextField* за приказ поља за унос текста.

3.4 Перзистенција података

За потребе рада имплементиране су две апликације система за управљање рестораном које се разликују по начину перзистенције података.

Прва користи ***MySQL*** релациону базу података која је заснована на језику *SQL* за управљање подацима чуваним у табелама дефинисане базе, где се у

колонама табела чувају подаци везани за ентитет који табела представља. *JDBC* (енгл. *Java Database Conectivity*) је *API* коришћен за приступ бази података из апликације и омогућава прављење различитих врста упита над базом.

Друга апликација користи библиотеку ***XStream*** која податке серијализује у *XML* формат који се ослања на маркап, тј. описну ознаку дела садржаја.

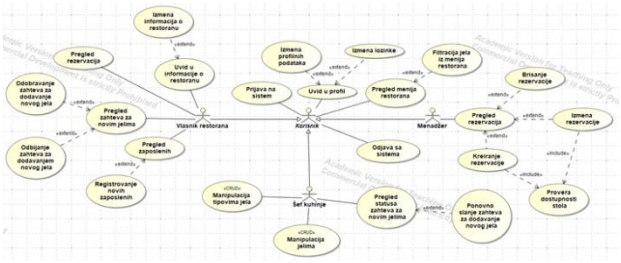
Употребом алијаса је могуће подешавати назив под којим ће се нека класа или њени атрибути чувати.

4. СПЕЦИФИКАЦИЈА И ИМПЛЕМЕНТАЦИЈА

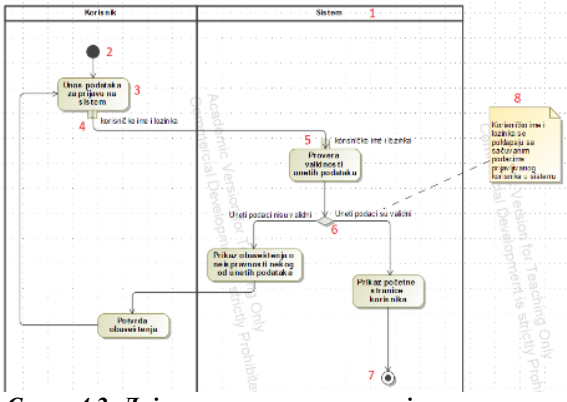
4.1 Спецификација система

Дијаграм случајева коришћења приказује функционалности које корисници могу да обављају и то власник ресторана, шеф кухиње и менаџер. Функционалности које су везане за улогу *Korisnik* могу да буду вршене од стране свих врста корисника (Слика 4.1).

Дијаграм активности на Слици 4.2 даје приказ функционалности пријаве на систем.



Слика 4.1: Дијаграм случајева коришћења



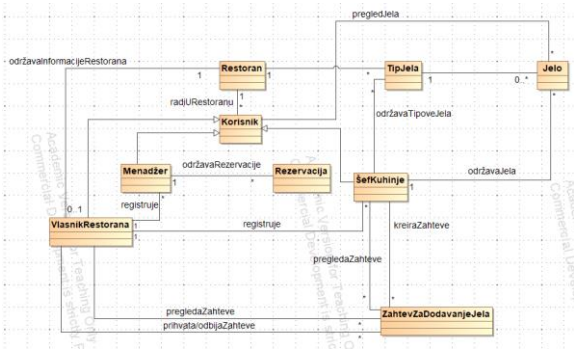
Слика 4.2: Дијаграм активности пријаве на систем

Концептуални дијаграм класа указује на делове и учеснике система и њихове повезаности (Слика 4.3).

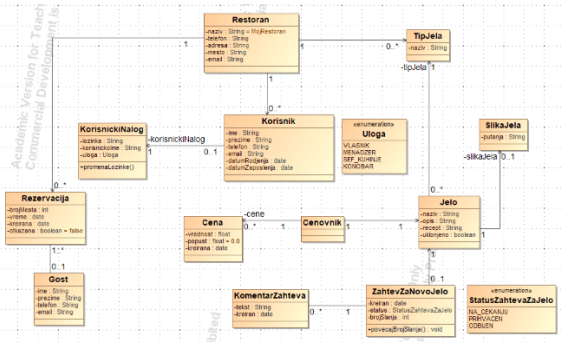
Слика 4.4 приказује **имплементациони дијаграм класа** заједно са спецификацијом обележја и метода.

Дијаграми секвенце креирани су за функционалнсти креирања резервације ресторана (Слика 4.5) и додавања јела.

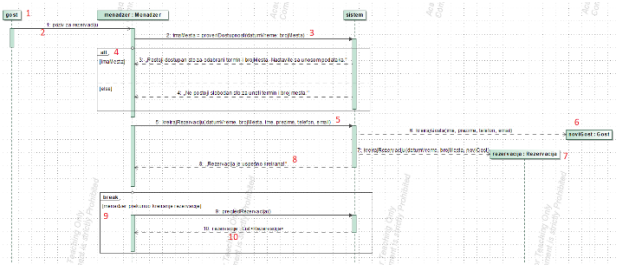
Дијаграм стања може да репрезентује животни циклус неког објекта или целе апликације у одређеном контексту. Слика 4.6 приказује животни циклус захтева за додавање новог јела у систем.



Слика 4.3: Концептуални дијаграм класа



Слика 4.4: Имплементациони дијаграм класа



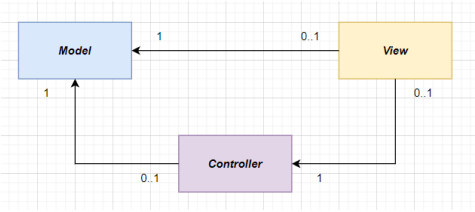
Слика 4.5: Дијаграм секвенце за креирање резервације

Слика 4.6: Дијаграм стања захтева за додавање јела

4.1 Имплементација система

Приликом имплементације примењен је архитектонски шаблон ***MVC*** (енгл. *Model-View-Controller*).

Подразумева поделу одговорности између класа корисничког интерфејса (поглед) и доменских класа (модел) које су повезане класама контролера. Класе погледа су имплементиране употребом неке графичке библиотеке (*Java Swing*) и омогућавају кориснику интеракцију са апликацијом. Акција корисника се састоји у активирању методе контролера којој се прослеђују неопходни подаци из погледа. Контролер затим проверава валидност података и уколико је све успешно прошло, ажурира се одговарајућа класа модела. Уколико се догодила грешка, контролер бацањем изузетка јавља погледу који потом обавештава корисника о типу грешке.



Слика 4.5: Коришћени MVC шаблон

Приликом имплементације поштоване су добре праксе и принципи програмирања. Један од њих је ***DRY*** (енгл. *Don't repeat yourself*) који подразумева да се један исти код не понавља на више места у апликацији са чим је уско везана поновна искористивост кода (енг. *reusability*). Пример наведеног огледа се у класи *PogledUtil* која садржи све методе које чине да интерфејс апликације буде конзистентан за корисника.

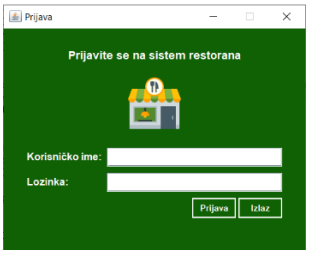
Дизајн шаблони представљају типична решења на честе проблеме који се јављају приликом имплементације софтвера. ***Observer*** омогућава имплементацију механизма „претплате" где се објекти посматрачи (енгл. *observers*) региструју на промене објеката који се посматрају (енгл. *observables*). Шаблон је примењен на функционалности измене профила корисника и додавања новог јела. ***Factory*** омогућава креирање нових објеката без залажења у саму логику њихвоог креирања. Примењен је приликом креирања почетног прозора и менија на основу улоге пријављеног корисника.

Као што је речено, перзистенција података је имплементирана употребом *MySQL* базе података у једној, а *XStream* библиотеке за серијализацију у *XML* формат у другој верзији апликације. Употребом одговарајућег драјвера и *JDBC API*-ја се комуницира са базом података како би се одговарајући подаци добављали, креирали и брисали из система.

Такође, класа *Serijalizacija* садржи методе за учитавање и чување података у *XML* формат код апликације која користи *XStream* библиотеку.

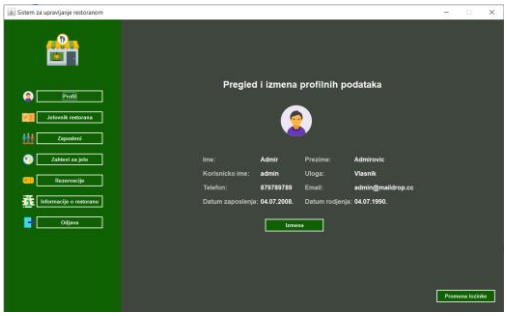
5. ИЗГЛЕД АПЛИКАЦИЈЕ

Пријава на систем подразумева уношење одговарајућег корисничког имена и лозинке корисника. Слика 5.1 приказује прозор за пријаву корисника на систем.

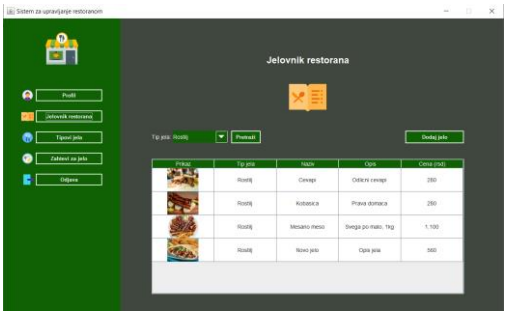


Слика 5.1: Прозор за пријаву на систем

Слика 5.2 даје приказ почетног прозора апликације власника ресторана и панел за преглед профилних података, одакле је могуће отварање дијалога за измену профилних података. Панел за преглед менија који садржи табелу са подацима свих регистрованих јела, одакле се може приступити дијалогу за унос новог јела, дат је на Слици 5.3.



Слика 5.2: Почетни прозор и преглед профила корисника



Слика 5.3: Панел за преглед менија ресторана

5. ЗАКЉУЧАК

Задатак рада је био обрада улоге моделовања у процесу развоја софтвера, уз приказ дијаграма *UML* језика, као и имплементација на основу претходно креираних модела уз примену добрих пракси програмирања. Мотив за реализацију рада и апликације за управљање рестораном јесте креирање свеобухватног примера на основу градива на предмету Спецификација и моделовање софтвера који ће моћи да користе студенти приликом савладавања градива. Представљени су описи свих креираних дијаграма, како би се повећао степен разумевања одређених принципа моделовања. Такође, у циљу усвајања добрих начина програмирања, примењене су устаљене праксе и принципи који воде до добро одрживог и употребљивог кода.

6. ЛИТЕРАТУРА

[1] M. S. C. H. G. K. Martina Seidl, UML@Classroom An Introduction to Object-Oriented Modeling, Heidelberg, Germany: Springer, 2012.

[2] G. Milosavljević, Увод у моделовање софтвера, Нови Сад: Универзитет у Новом Саду, Факултет техничких наука, 2020.

[3] B. Unhelkar, Software Engineering with UML, Taylot & Francis Group, 2018.

[4] „MagicDraw Architecture Made Simple, User Manual," No Magic, Inc., 2013.

Кратка биографија:

Лазар Јовић рођен је 12.1.1998. године у Сремској Митровици. Након завршетка Митровачке гимназије 2016. и дипломирања на Факултету техничких наука 2020. године, уписује мастер академске студије на смеру Рачунарство и аутоматика.