UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Miran Perič

RAZVOJ TESTNEGA ORODJA ZA SISTEM ZA UPRAVLJANJE PRISTANIŠKIH TERMINALOV

DIPLOMSKO DELO NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU
PRVE STOPNJE

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00108/2011
Datum: 05.04.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MIRAN PERIČ**

Naslov: **RAZVOJ TESTNEGA ORODJA ZA SISTEM ZA UPRAVLJANJE PRISTANIŠKIH TERMINALOV**

DEVELOPING A TESTING TOOL FOR A PORT TERMINAL OPERATING SYSTEM

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

V diplomski nalogi predstavite razvoj testnega orodja za testiranje novega sistema za upravljanje pristaniških terminalov v Luki Koper. V ta namen kratko predstavite sedanji sistem za upravljanje pristaniških terminalov, zahteve za testiranje novega sistema ter postopen razvoj testnega orodja. Nalogo zaključite s predstavitvijo ter analizo uporabe razvitega testnega orodja.

Mentor:

viš. pred. dr. Igor Rožanc

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Miran Perič

RAZVOJ TESTNEGA ORODJA ZA SISTEM ZA UPRAVLJANJE PRISTANIŠKIH TERMINALOV

DIPLOMSKO DELO NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU
PRVE STOPNJE

MENTOR: dr. Igor Rožanc

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00108/2011
Datum: 05.04.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MIRAN PERIČ**

Naslov: **RAZVOJ TESTNEGA ORODJA ZA SISTEM ZA UPRAVLJANJE PRISTANIŠKIH TERMINALOV**

DEVELOPING A TESTING TOOL FOR A PORT TERMINAL OPERATING SYSTEM

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

V diplomski nalogi predstavite razvoj testnega orodja za testiranje novega sistema za upravljanje pristaniških terminalov v Luki Koper. V ta namen kratko predstavite sedanji sistem za upravljanje pristaniških terminalov, zahteve za testiranje novega sistema ter postopen razvoj testnega orodja. Nalogo zaključite s predstavitvijo ter analizo uporabe razvitega testnega orodja.

Mentor:

viš. pred. dr. Igor Rožanc

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani **Miran Perič**,

z vpisno številko **63990113**,

sem avtor diplomskega dela z naslovom: **Razvoj testnega orodja za sistem za upravljanje pristaniških terminalov.**

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom **dr. Igorja Rožanca,**
- so elektronska oblika diplomskega dela, naslov (slov., ang.), povzetek (slov., ang.) ter ključne besede (slov., ang.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 04. 07. 2011 Podpis avtorja:

Zahvala

Zahvaljujem se mentorju dr. Igorju Rožancu za pomoč, usmerjanje in nasvete pri izdelavi diplomskega dela. Zahvala za pomoč in sodelovanje gre tudi kolektivu podjetja Actual-IT in odgovornim v Luki Koper.

Hvala prijateljem za vso vzpodbudo.

Posebna zahvala gre staršema za potrpežljivost, razumevanje, skrb in podporo skozi vsa leta študija.

Kazalo

Seznam uporabljenih kratic in simbolov

C#	C Sharp – objektno usmerjen programski jezik
CNTM	Container Move Confirmation – potrditev gibanja kontejnerja
DOM	Document Object Modeling – jezikovno neodvisna platforma za zastopanje in interakcijo z objekti XML
MSMQ	Miscrosoft Message Queue – protokol za izmenjavo sporočil
ORDA	WorkOrder Acknowledgement – potrditev sprejema/zavrnitve dispozicije
ORDR	WorkOrder – vhodna dispozicija
PB	Podatkovna baza
RegEx	Regular Expression – regularni izraz
RLPL	Rail Plan – nalog za natovarjanje vlakovne kompozicije
SOAP	Simple Object Access Protocol – protokol za izmenjavo sporočil
SQL	Structured Query Language – strukturirani povpraševalni jezik za delo s PB
TinO	Trženje in Operativa – Pristaniški informacijski sistem
TOS	Terminal Operating System – sistem za upravljanje pristaniških terminalov
UML	Unified Modeling Language – poenoteni jezik modeliranja
VSAA	Vessel Call Announcement Acknowledgment – potrditev najave ladje
VSAN	Vessel Call Announcement – najava ladje
XML	Extendable Markup Language – razširljiv označevalni jezik
XPath	XML Path Language – povpraševalni jezik za izbiranje vozlišč v XML dokumentu

Povzetek

Diplomsko delo opisuje pristop k razvoju testnega orodja za simulacijo sistema za upravljanje pristaniških terminalov.

Rezultat diplomske naloge je prikaz razvoja in delovanja testne aplikacije, ki je napisana v programskem jeziku C#. Aplikacija komunicira s pristaniškim informacijskim sistemom in si izmenjuje sporočila, podatke pa shranjuje v podatkovno bazo.

S pomočjo aplikacije lahko pristaniški sistem testiramo še pred implementacijo novega sistema za upravljanje s kontejnerskim terminalom.

Ključne besede:

- sistem za upravljanje pristaniških terminalov
- testno orodje
- pristanišče
- C#
- XML
- XPath

Abstract

This thesis describes the approach for development of a testing tool which simulates the terminal operating system.

Main result of the thesis is presentation of the approach for the development and operation of the application, which was developed in C# programming language. The application communicates with the port information system by messages exchange. Data is stored on a database.

With the support of the application we can test the port information system before of future implementation of the new system for the container terminal management.

Keywords:

- terminal operating system
- testing tool
- port
- C#
- XML
- XPath

1 Uvod

V današnjem času se zaradi svetovnega ekonomskega položaja poskuša na vseh področjih nižati stroške. Dragi energenti, velika konkurenca, vse krajši dobavni roki in visoka frekvenca vplutja in izplutja ladij je le nekaj razlogov, zaradi katerih je potrebno optimizirati tudi logistiko v pristaniščih. Vse to s seboj prinaša tudi veliko količino podatkov, ki jih je potrebno obdelati, zato je potrebno ponavljajoče procese čim bolj avtomatizirati. Tak proces je tudi manipulacija s kontejnerji. Zato se tovorna pristanišča odločajo za uporabo sistema za upravljanje pristaniških terminalov (ang. Terminal Operating System; v nadaljevanju TOS). Nekatere funkcionalnosti TOS-a so:

- sledljivost položaja kontejnerjev,
- beleženje fizičnega stanja kontejnerjev,
- planiranje skladiščnega prostora na tovornih ladjah.

Sledljivost položaja in beleženje stanja kontejnerjev je izjemnega pomena za organizacijo in racionalizacijo prostora na samem terminalu, da ne prihaja do nepotrebnega prekladanja pri pretovarjanju za nadaljnjo pot po kopnem ali morju. Planiranje natovarjanja ladij pa je omogočeno, saj sistem vsebuje tudi sheme skladiščnih prostorov le-teh.

V našem tovornem pristanišču se na kontejnerskem terminalu sedaj uporablja TOS Cosmos. Zaradi zastarelosti in neustreznosti ga bodo do konca leta opustili, nadomestil pa ga bo TideWorks (v nadaljevanju TW). TW ima dolgoročno zagotovljeno podporo s strani proizvajalca, je bolj prilagodljiv in razširljiv. V ta projekt je vključena združba Actual IT d.d., ki bo nudila izvedbo prenove sistema TOS ter nadaljnjo podporo.

Preden se tak sistem preda v uporabo, ga je potrebno preveriti in prilagoditi naročniku v testnem okolju. Ko bo TW pripravljen, ga bo potrebno čim hitreje vklopiti v obstoječi sistem. Komunikacija s preostalimi deli mora biti tekoča, zato se uporablja dogovorjena standardizirana oblika sporočil. Od uporabnikove aplikacije do komunikacijskega strežnika potujejo zapisana v obliki z ločevalnimi znaki, naprej do TOS-a pa kot dokumenti XML (ang. Extensible Markup Language). V drugo smer gredo v obratnem vrstnem redu.

Zato smo se odločili, da razvijemo testno orodje, s katerim bodo interni testni uporabniki simulirali TW komunikacijo. Akcije se bodo izvajale ročno, tako da je vsak korak pod nadzorom. S tako simulacijo lahko pripravimo obstoječi del sistema že pred vgradnjo novega TOS-a. Orodje bo seveda uporabno, tudi ko bo v sistem implementiran nov TOS, za testiranje in odpravljanje morebitnih težav, ki se pojavijo že med utečenim delovanjem.

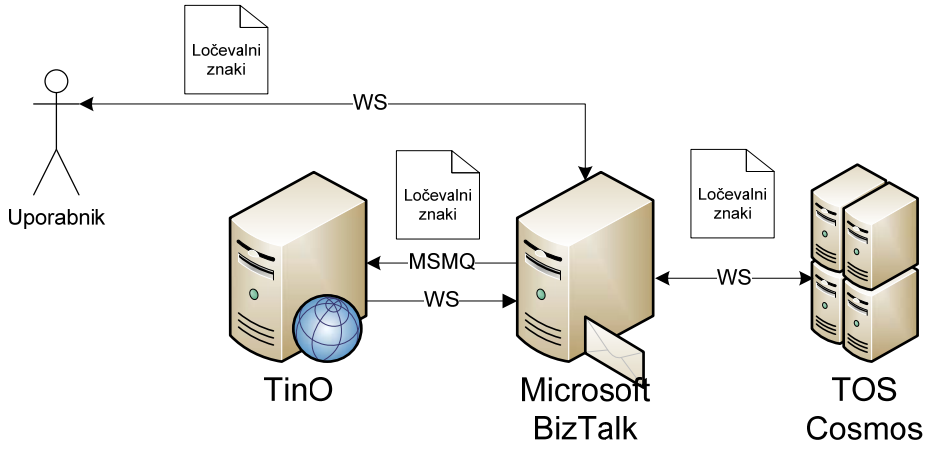
Na naslednjih straneh bomo predstavili pristop do razvoja in funkcionalnosti testnega orodja, ki je bilo uvrščeno med naše naloge.

2 Opis obstoječega sistema in prihajajoče spremembe

Obstoječi sistem je sestavljen iz treh večjih komponent (slika 1):

- Posredniško-komunikacijskega strežnika MS BizTalk,
- strežniške aplikacije TinO,
- TOS-a Cosmos.

V obstoječem sistemu ima strežnik BizTalk (v nadaljevanju BT) svoje mesto med uporabnikom, spletnim aplikacijskim strežnikom TinO in TOS-om Cosmos. Sporočila potujejo zapisana v obliki dokumenta z ločevalnimi znaki. Komunikacija med uporabnikom in samimi strežniki poteka večinoma preko spletne storitve (ang. Web Service) in SOAP protokola, le v primeru, ko strežnik BizTalk posreduje sporočila strežniku TinO, to izvaja preko MSMQ protokola.



Slika 1: Prikaz obstoječega sistema

2.1 Komunikacija med posameznimi podsistemi poslovnega procesa

Med različnimi podsistemi, ki tvorijo poslovni proces, je potrebnega veliko usklajevanja pri sporočanju, saj ni rečeno, da uporabljajo enak komunikacijski protokol. Za premostitev takih težav lahko uporabimo posredniški program, ki se mu reče tudi posredovalec sporočil (ang. message broker).

Tak posredniški program na omrežjih pretvarja formalno dogovorjena sporočila med formalnimi komunikacijskimi protokoli pošiljatelja in prejemnika. Dejansko s svojo prisotnostjo razdeli sistem in posreduje komunikacijo med aplikacijami, ne da bi bile te seznanjene s protokoli aplikacij na drugi strani.

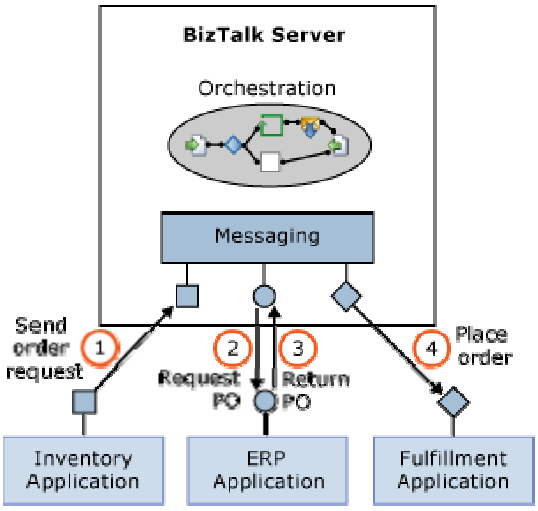
Naloge komunikacijskega strežnika so:

- sprejem pošiljateljevih sporočil v realnem času,
- potrjevanje skladnosti sporočil,
- pošiljanje odgovorov pošiljatelju o prejetju sporočila ali napake v sporočilu,
- preoblikovanje sporočil v format, ki ga zahteva prejemnik,
- shranjevanje sporočil v podatkovno zbirko (ang. message repository),
- pošiljanje sporočil prejemnikom.

```
LKDDLUKA        03095845.DISDISP.NP 0607030958                              0607030956 07973      00

GL#DVH#7973##0##TRT#LKDD#             #MIRELLA#056656910#D#TAR###LKDD#ACKP#AMAO#36351#USMOB#MOBILE, AL#AT
#AVSTRIJA#SIKOP#KOPER#OD#DES3#L-S#####14########70789300,00##MIR############PREMOG####

PB#1#################1#PREMOG##U#VOESTALPINE#VO####70789300,00#######

PR#1#CB#VOESTALPIN#060706#PB#

EM#    5#
```

Slika 2: Primer sporočila v obstoječem sistemu [1]

Za komunikacijo med različnimi podsistemi je za efektiven poslovni proces našega sistema v uporabi komunikacijski strežnik Microsoft BizTalk. Srce strežnika tvorita dva poglavitna dela, kot ju prikazuje slika 2:



Slika 3: Shema BizTalk strežnika [2]

Komponenta za izvajanje logike »Orchestration« je zasnovana nad komunikacijsko komponento in implementira logiko, ki poganja ves poslovni proces ali le del tega. V tem delu strežnika se izvajajo vsa preoblikovanja sporočil in potrjevanja skladnosti le-teh.

Komunikacijska komponenta »messaging« je sposobna komunicirati preko različnih protokolov z velikim naborom programske opreme na tržišču. Komponenta omogoča različne načine komunikacije preko velikega nabora adapterjev (vtičnikov). Med njimi najdemo tudi adapterje, ki jih uporablja naš sistem, to so:

- Web Services (spletne storitve) adapter, ki komunicira preko SOAP (Simple Object Access Protocol) protokola za prenos sporočil,

- MSMQ adapter, ki omogoča pošiljanja in sprejem sporočil po protokolu MSMQ (Microsoft Message Queue),

- SQL Adapter, ki piše ali bere informacije in sporočila na podatkovno bazo.

Prednost takega posredniškega strežnika v kompleksnih sistemih je, da za komunikacijo z ostalimi aplikacijami ni potrebno popravljati aplikacijskih komunikacijskih vrat. Število povezav se zmanjša, ker se koordinacija in posredovanje dogajata v eni točki. Ob morebitni zamenjavi aplikacij ali dodajanju novih, na posredniškem strežniku dodamo primeren adapter uporabljenega protokola in ga povežemo z vrati aplikacij, po katerih poteka izmenjava sporočil.

V luškem sistemu posredniški strežnik opravi veliko število izmenjav sporočil s pristaniškim informacijskim sistemom, slednji v sistem vključuje celo logistično verigo (špediterji, terminali, ladjarji, prevozniki, ...).

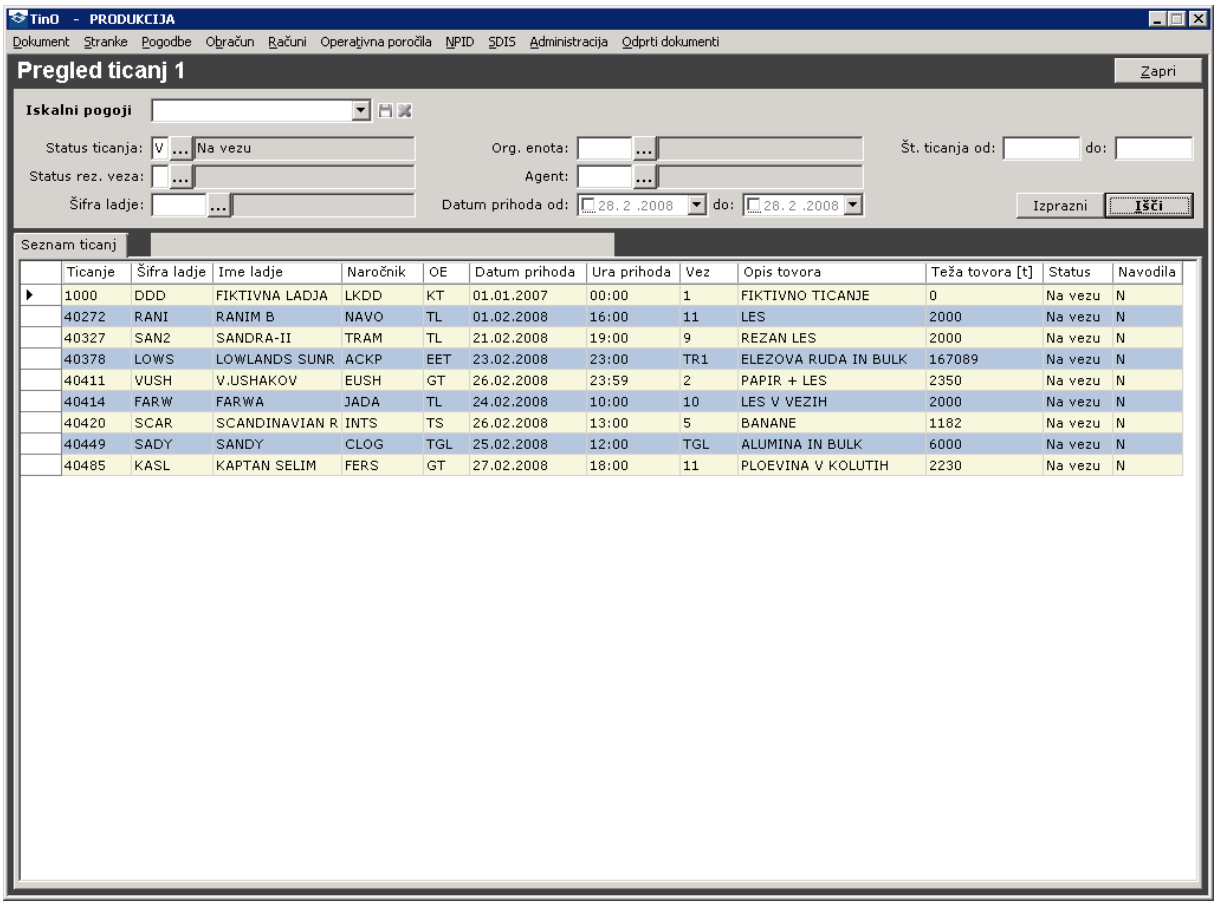
2.2 Pristaniški informacijski sistem

Zahteve po vedno večji prepustnosti blaga skozi tovorna pristanišča so vsako leto večje. Ladijski prevozniki pričakujejo vedno krajše čase pretovora blaga, informacije o blagu ter o storitvah, ki so povezana z njim. Med pristanišči se vije boj, kdo bo omogočal hitrejši servis. Zato se trudijo implementirati vedno novejše generacije učinkovitejših in avtomatiziranih pristaniških informacijskih sistemov (v nadaljevanju PIS). Zelo je pomembno, da imajo PIS-i orodja za integracijo okoliških sistemov, zato je podpora mrežnim tehnologijam nujna. Glavne funkcije v mrežni podpori v pristaniščih so:

- podpora izmenjavi podatkov z vsemi subjekti v logistični verigi,
- zmožnost hitrega prilagajanja spremembam, ki se dogajajo v okoliških sistemih,
- učinkovita integracija z informacijsko komunikacijskimi sistemi in rešitvami v pristanišču.

V našem pristaniškem informacijskem sistemu je poglavitna komponenta aplikacijski strežnik TinO (Trženje in Operativa) [3], ki nudi podporo večini procesov v pristanišču na področju trženja in operative. TinO tako pokriva:

- naročanje storitev,
- evidentiranje opravljenih storitev,
- evidence stanja tovora v skladiščih (luške, carinske, trošarinske),
- obračun storitev in izdajo računov,
- pogodbe,
- najavo ladij,
- naročanje dela in železniških vagonov,
- proces izmenjave vagonov s Slovenskimi železnicami,
- razpis in obračun delovnih nalogov,
- sprejem in obdelavo sporočil iz ostalih podsistemom oz. od naročnikov,
- tvorjenje in pošiljanje sporočil v podsisteme ali k naročnikom,
- pripravo podatkov za podatkovno skladišče in prenos podatkov v ERP sistem SAP.



Ticanje	Šifra ladje	Ime ladje	Naročnik	OE	Datum prihoda	Ura prihoda	Vez	Opis tovora	Teža tovora [t]	Status	Navodila
1000	DDD	FIKTIVNA LADJA	LKDD	KT	01.01.2007	00:00	1	FIKTIVNO TICANJE	0	Na vezu	N
40272	RANI	RANIM B	NAVO	TL	01.02.2008	16:00	11	LES	2000	Na vezu	N
40327	SAN2	SANDRA-II	TRAM	TL	21.02.2008	19:00	9	REZAN LES	2000	Na vezu	N
40378	LOWS	LOWLANDS SUNR	ACKP	EET	23.02.2008	23:00	TR1	ELEZOVA RUDA IN BULK	167089	Na vezu	N
40411	VUSH	V.USHAKOV	EUSH	GT	26.02.2008	23:59	2	PAPIR + LES	2350	Na vezu	N
40414	FARW	FARWA	JADA	TL	24.02.2008	10:00	10	LES V VEZIH	2000	Na vezu	N
40420	SCAR	SCANDINAVIAN R	INTS	TS	26.02.2008	13:00	5	BANANE	1182	Na vezu	N
40449	SADY	SANDY	CLOG	TGL	25.02.2008	12:00	TGL	ALUMINA IN BULK	6000	Na vezu	N
40485	KASL	KAPTAN SELIM	FERS	GT	27.02.2008	18:00	11	PLOEVINA V KOLUTIH	2230	Na vezu	N

Slika 4: Grafični vmesnik aplikacije TinO [1]

Tipičen scenarij pri pretovarjanju na terminalu, ki vključuje strežnik TinO (slika 4), je naslednji:

- Uporabnik v spletni aplikaciji, ki zna tvoriti sporočila po dogovorjenem standardu, pripravi naročilo o natovarjanju ali raztovarjanju in ga pošlje komunikacijskemu strežniku.
- Komunikacijski strežnik v prvem delu sporočilo semantično pregleda, odstrani nepotrebne podatke, kot je ovoj sporočila, in ga posreduje sistemu za izvajanje logike.
- Sistem za izvajanje logike pregleda komu je sporočilo namenjeno, ga pretvori in pošlje prejemniku, v večini primerov strežniku TinO.
- Ko strežnik TinO sprejme sporočilo, se izvrši število preverjanj, da preverijo, če se sporočilo ujema s poslovnimi pravili. Če je rezultat pozitiven, se sporočilo vpiše v podatkovno bazo in nazaj komunikacijskemu strežniku, da ta prepošlje partnerju oz. sistemu na določenem terminalu.
- Ko delavci na terminalu opravijo vse manipulacije z blagom, jih zapišejo v TinO. Če so v manipulacije vključeni še ostali pristaniški sistemi, TinO pošlje sporočila preko komunikacijskega strežnika tudi njim. V našem primeru se taka sporočila pošljejo sistemu na kontejnerskem terminalu.
- TinO obvešča partnerje o stanju blaga ob vsaki spremembi.
- Ob zaključku premikov delavci zaprejo delavni nalog, TinO obvesti partnerje in pripravi fakture za opravljeno delo in če je potrebno, tudi preglede in poročila za carinske postopke.

Kontejnerska manipulacija na osnovi sporočil je dober primer integracije med različnimi sistemi.

2.3 Sistemi za upravljanje kontejnerskih terminalov

Na pristaniških terminalih se vrstijo aktivnosti, za katere je potrebno poskrbeti v organizacijskem smislu. Če kje poznajo rek »čas je denar«, je to zagotovo na luških terminalih, kjer se pretovarja praktično štiriindvajset ur na dan, sedem dni na teden, vse leto. Za organizacijo in avtomatizacijo, da bi vse dejavnosti tekle tekoče, tudi v tem prostoru potrebujejo podporo informacijske tehnologije.

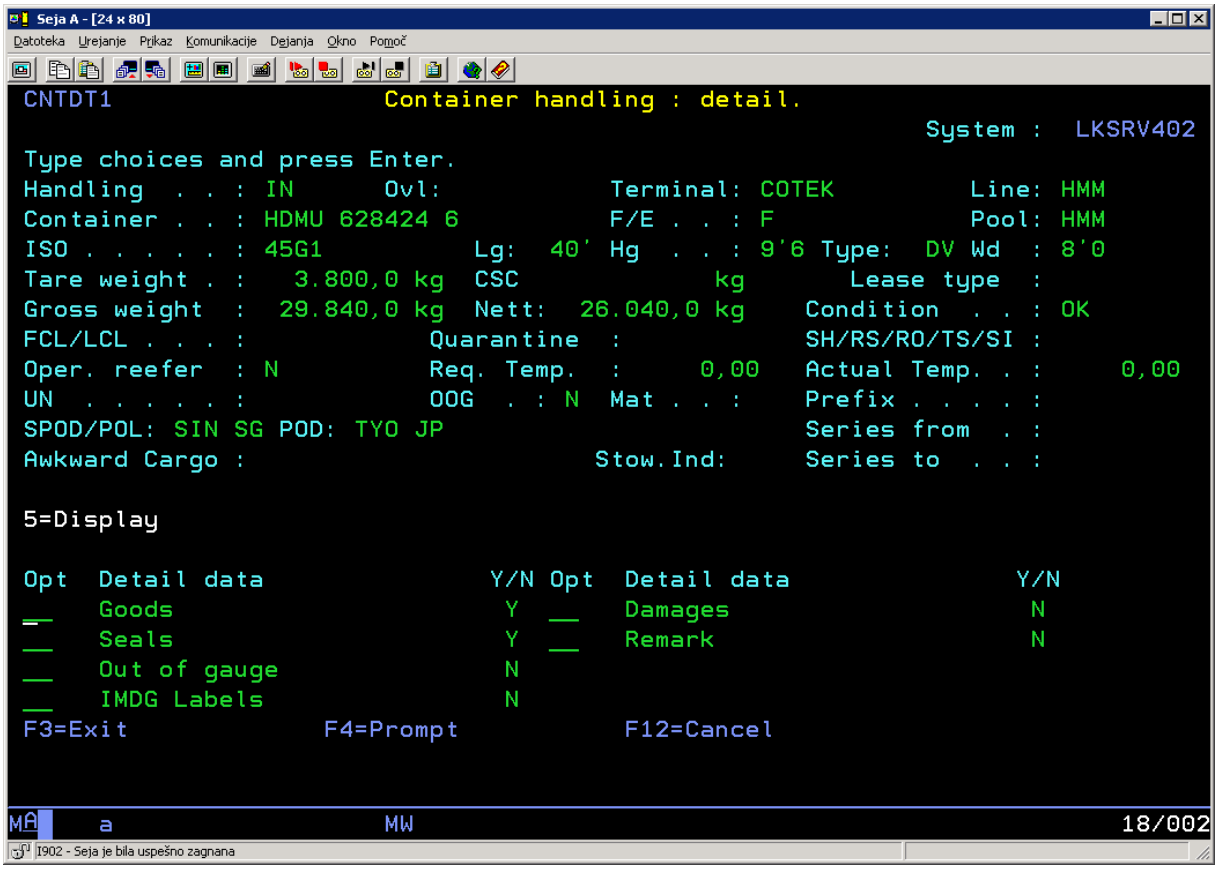
Na kontejnerskih terminalih imajo prav poseben izziv, saj morajo spremljati kontejnerje od raztovora ladje, mimo skladišč, skladiščenja na terminalu in vse do odhoda iz pristanišča. Včasih pa je potreben tudi servis; kontejnerje na zahtevo praznijo ali polnijo z blagom. Vse te premike in stanja blaga ter prevoznih sredstev morajo delavci na terminalu beležiti. In za to uporabljajo sistem za upravljanje pristaniških terminalov (ang. Terminal Operating System). Za upravljanje kontejnerskega terminala je vidna smotrnost takega sistema. Sestavlja ga več komponent, saj omogoča celovit pogled nad administracijo in dejavnostmi na terminalu. Tak sistem sestavljajo naslednje komponente:

Slika 5: Shema komponent TOS Cosmos [4]

Naše pristanišče trenutno upravlja kontejnerski terminal s sistemom TOS Cosmos. Sistem je sestavljen iz več komponent. Preko centralnega sistema upravlja vse operativne in administrativne procese na terminalu, temeljna naloga pa je sledenje vsem storitvam, ki se izvajajo nad kontejnerji, in beleženje le-teh v centralno podatkovni bazi. Sistem je sestavljen še iz komponent (slika 5):

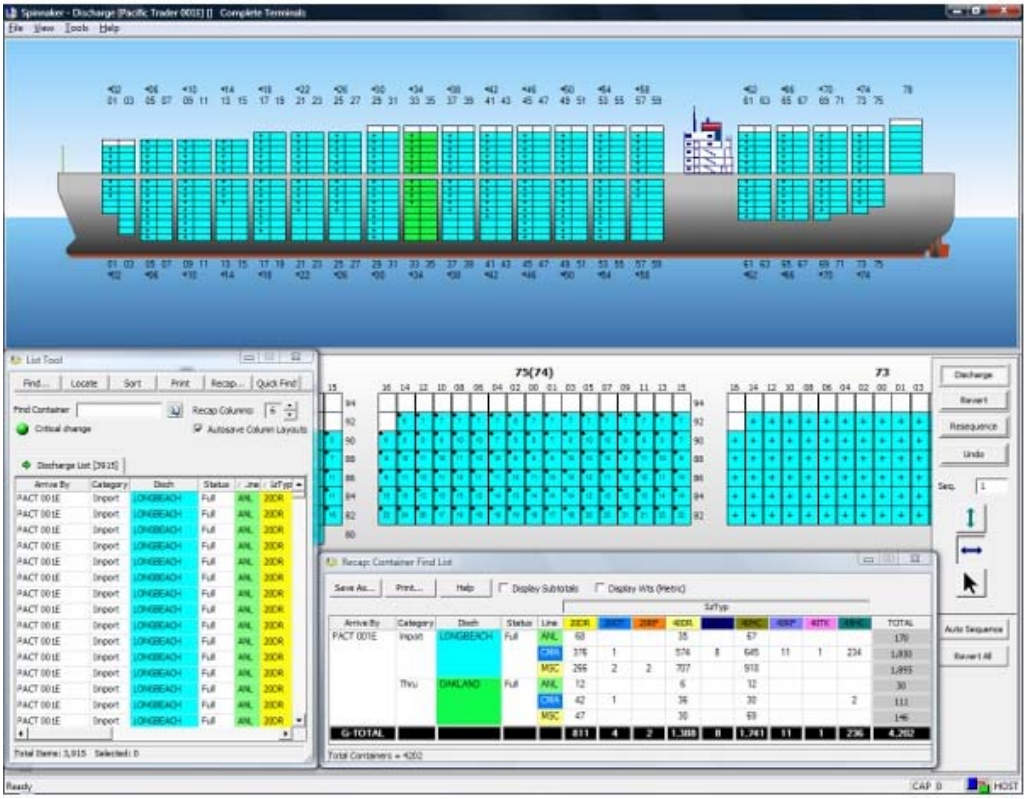
- za planiranje optimalne pozicije kontejnerja na terminalu (slika 6),
- za vodenje in optimiziranje dela z mehanizacijo na terminalu,
- dostava oz. odvoz kontejnerjev na praznjenje oz. polnjenje,

- avtomatizirano planiranje, nakladanje in razkladanje ladij,
- avtomatičen obračun storitev in vodenje pogodb,
- vmesnik za elektronsko izmenjavo podatkov,
- vpogled v podatke preko spleta.



Slika 6: Uporabniški vmesnik TOS-a Cosmos [1]

TOS Cosmos pa bo do konca leta 2011 v pristanišču zamenjan z novejšim sistemom TideWorks. Tudi TW je sestavljen iz več komponent, ki omogočajo enake funkcionalnosti kakor Cosmos. Ena poglavitnih prednosti je v tem, da je TW razširljiv. Preko enega centralnega sistema je tako možno nadzirati, organizirati in administrirati še ostale pristaniške terminale. Tako bo v bodoče možno poenotenje sistema. Grafični uporabniški vmesnik je pri TW bolj dodelan in privlačnejši za oko. Uporabnik lahko enostavno z miško ter z vleci-spusti gestami hitreje pride do želenega rezultata, posebej uporabno je pri planiranju transporta in skladiščenja.

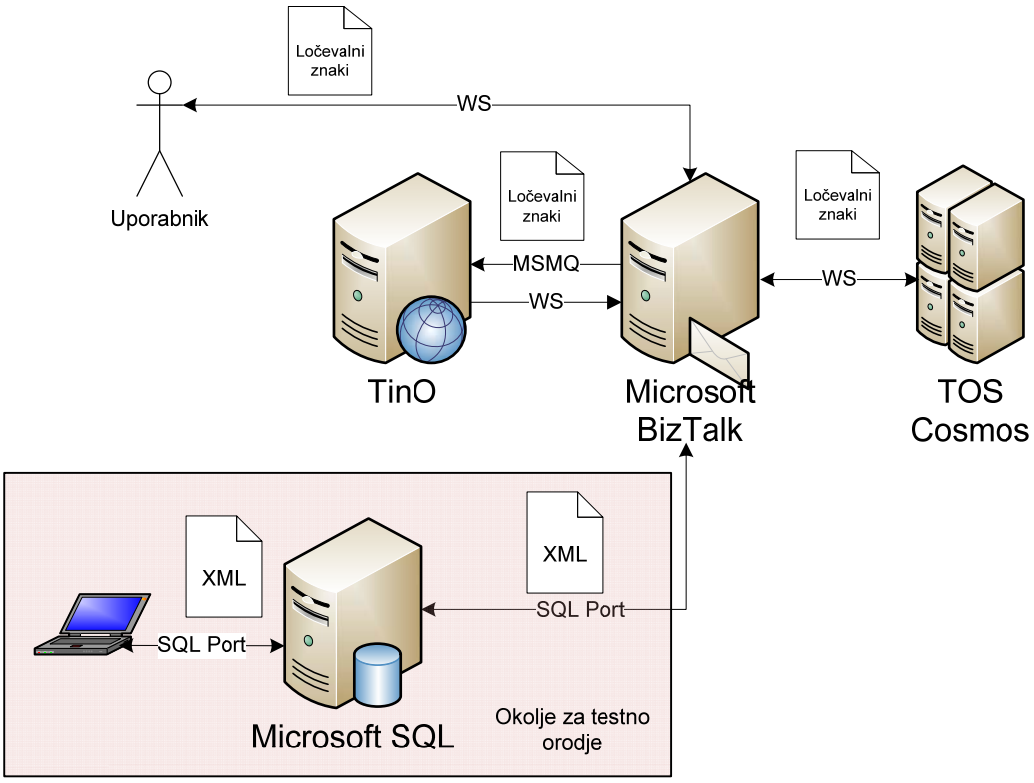


Slika 7: TideWorks komponenta za planiranje skladiščenja in prevozov [5]

2.4 Spremembe nad obstoječim sistemom

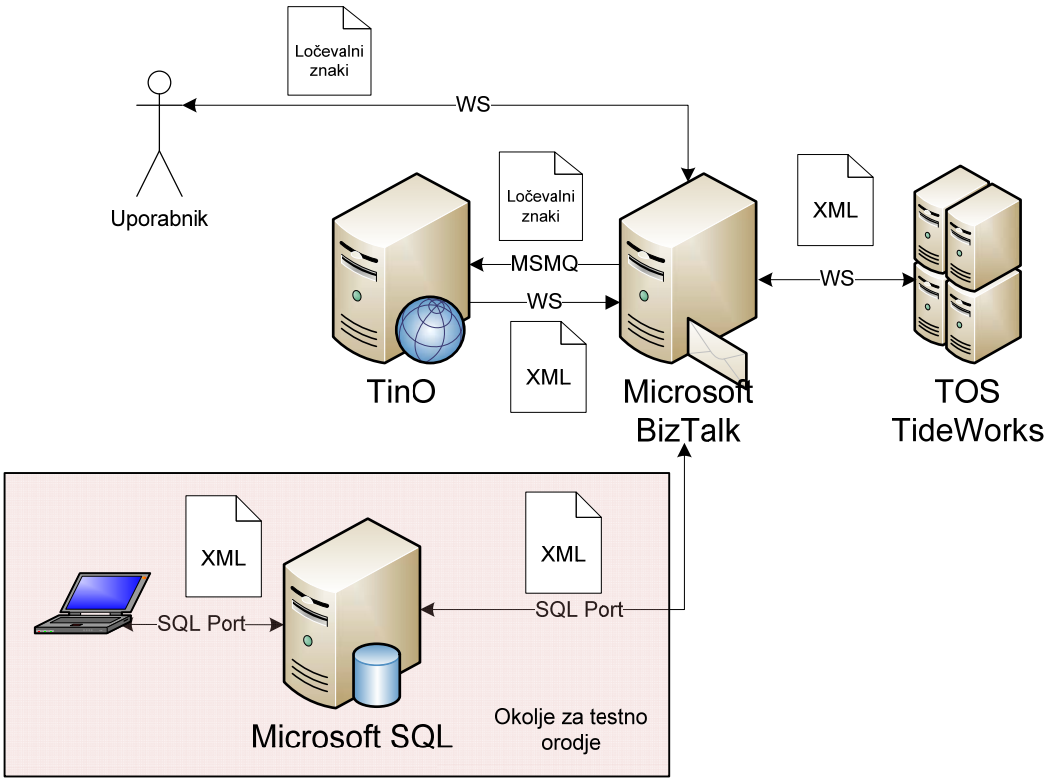
Celoten informacijski sistem že sedaj nudi večino funkcionalnosti. Razlog, da se nadomesti obstoječi TOS s TideWorks sistemom je, da obstoječi sistem ne ponuja dovolj zagotovitev glede dolgoročne podpore in nima možnosti razširitve v okviru enakega sistema tudi na druge terminale v pristanišču.

V prvi fazi se je obstoječemu sistemu dodalo samo manjše testno okolje, ki vsebuje podatkovni strežnik SQL, preko katerega bo s sistemom komuniciralo naše testno orodje. Komunikacija bo potekala preko SQL adapterja. Vsa sporočila, ki so namenjena tudi obstoječemu TOS-u, se bodo vpisovala v tabelo v XML obliki. Prav tako tudi odgovori testnega orodja.



Slika 8: Prikaz obstoječega sistema z okoljem za testno orodje

Ko se bo TOS Tideworks dokončno implementiral, se bo nekoliko spremenila komunikacija med strežnikom TinO in komunikacijskim strežnikom BizTalk in naprej do novega TOS-a. Ukinja se namreč format sporočil z ločevalnimi znaki. Tako bodo sporočila od TinO strežnika do TW potovala v XML obliki. Po potrebi se bodo sporočila vpisovala tudi na podatkovno bazo testnega okolja, na primer, ko bi se reševala kakšna izjema, ki je med testiranjem ne bomo ujeli.



Slika 9: Prikaz končnega okolja z implementacijo sistema TideWorks

2.5 Predstavitev uporabljenih tehnologij

V celotnem sistemu se srečamo z velikim številom tehnologij, a smo se odločili, da naštejemo le tiste, s katerimi smo imeli največ opravka. Za kodiranje smo uporabljali Microsoft Visual Studio 2010 in SQL Server Management Studio, vendar ju ne bomo posebej prestavljali. Tehnologije, s katerimi smo imeli pri razvoju največ stika, so večini ljudem s stroke dobro poznane.

Objektni programski jezik C#

C# (ang. C Sharp) [6] je objektno usmerjen programski jezik, ki je s pomočjo microsoftove tehnologije .NET Framework zelo razširjen za razvoj Windows aplikacij, spletnih servisov (ang. Web services), aplikacij strežnik-odjemalec (ang. client-server applications), aplikacij za upravljanje s podatkovno bazo (ang. database application) ter ostalih. S podporo projekta Mono [7] je prisoten tudi na operacijskem sistemu Linux in MacOSX platformah.

Sintaksa samega C# je zelo jasna, zato je primeren za učenje. Prav tako se zelo hitro prilagodijo vsi, ki so seznanjeni s programiranjem v jeziku C, C++ ali Java. C# poenostavlja številne kompleksnosti C++ in ponuja zmogljive funkcije ter neposredni dostop do pomnilnika, ki jih ni mogoče najti niti v Javi.

Kot objektno usmerjen jezik C# podpira:

- enkapsulacijo,
- dedovanje,
- polimorfizem.

Vse spremenljivke in metode, vključno z glavno, so opredeljene v razredu. Razred lahko deduje od enega od staršev, lahko pa implementira poljubno število vmesnikov.

Dedovanje smo uporabili pri podatkovnih strukturah za prenos podatkov iz razčlenjevalnika (ang. parser) sporočil v podatkovno bazo. Polimorfizem, kjer uporabljamo en vmesnik za več klicev, smo uporabili pri pošiljanju odgovorov.

Razširljiv označevalni jezik XML

XML [8] je okrajšava za angleški izraz Extensible Markup Language, razširljiv označevalni jezik, ki ga pogosto srečamo, če brskamo po Internetu. XML je preprost računalniški jezik, ki nam omogoča opisovanje strukturiranih podatkov. Zelo primeren je za prenos in izmenjavo podatkov po mreži med strežniki in aplikacijami v različnih sistemih. Zaradi preglednosti in preprostosti se da XML enostavno razširiti, saj sami določimo imena etiket (ang. tag).

XML je razdeljen na 3 dele:

- podatkovni (vanj shranimo podatke v določeni obliki z želenimi etiketami),
- deklarativni (skrbi za to, da lahko pri dodajanju novih podatkov vidimo kaj predstavlja določena etiketa),
- predstavitveni (oblikujemo izpis podatkov).

Primer XML dokumenta:

```
<student>
  <Ime>Janez</Ime>
  <Priimek>Novak</Priimek>
  <Kraj_bivanja>Ljubljana</Kraj_bivanja>
  <status>RED</status>
  <vpisna_st>69110001</vpisna_st>
</student>
```

Sintaktično pravilnost XML-a lahko preverjamo s pomočjo predhodno določene XSD (XML Schema Definition) sheme.
XSD shema [9] lahko določa:

- katere etikete (tag) lahko vsebuje XML,
- kako so etikete gnezdene,
- kaj pomenijo,
- katerega tipa so,
- kolikokrat se lahko pojavijo,
- kakšen vrstni red imajo, če se pojavijo,
- kakšen tip vsebine naj bi imele.

Vse vrste sporočil v testnem orodju se pojavljajo v XML obliki. Za branje in pisanje sporočil smo uporabili XPath (XML Path Language) [10], poizvedbeni jezik za izbiranje in iskanje poznanih vozlišč po XML dokumentu.

Primer XSD sheme:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Ime">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:maxLength value="20" />
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name="Priimek">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:maxLength value="20" />
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name="Kraj_bivanja">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:maxLength value="20" />
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name="status" fixed="RED">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:maxLength value="3" />
        <xs:enumeration value="RED" />
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name="vpisna_st" default="0">
    <xs:simpleType>
      <xs:restriction base="xs:integer">
        <xs:totalDigits value="8" />
        <xs:maxInclusive value="99999999" />
        <xs:minInclusive value="0" />
        <xs:pattern value="[0-9]*" />
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name="student">
    <xs:complexType>
      <xs:sequence>
```

```
        <xs:element ref="Ime" />
        <xs:element ref="Priimek" />
        <xs:element ref="Kraj_bivanja" />
        <xs:element ref="status" />
        <xs:element ref="vpisna_st" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Branje in pisanje XML sporočil v testnem orodju

Za branje in pisanje XML dokumentov je na voljo več metod. XPath (ang. XML Path Language) [10] je povpraševalni jezik, ki omogoča branje ali pisanje XML dokumentov. Za osnovo uporablja vozlišča, s pomočjo katerih potuje po drevesni strukturi XML dokumenta in s pomočjo raznih kriterijev zna prebrati vsebino elementov. Z XPath metodo smo se ukvarjali pri naši rešitvi.

Branje XML dokumenta poteka tako:

- v spomin naložimo XML dokument: `xml.LoadXml(xmlString);`
- izberemo vozlišče, iz katerega bomo brali elemente:
  ```
  XmlNode xnMsg = xml.SelectSingleNode("/ORDR/message");
  ```
- lastnosti objekta priredimo vrednost elementa iz XML dokumenta:
  ```
  message.MessageType = xnMsg["message_type"].InnerText;
  ```

Potek tvorjenja XML dokumenta:

- ustvarimo nov XML dokument:
  ```
  XmlDocument doc = new XmlDocument();
  ```
- ustvarimo nov element in ga pripnemo dokumentu kot vozlišče, ker ne vsebuje vrednosti:
  ```
  XmlElement msg = doc.CreateElement("message");
  doc.DocumentElement.AppendChild(msg);
  ```
- ustvarimo nov element, mu dodamo vrednost (`ORDA`) in ga pripnemo starševskemu vozlišču:
  ```
  XmlElement msgtype = doc.CreateElement("message_type");
  msgtype.InnerXml = "ORDA";
  msg.AppendChild(msgtype);
  ```

Povpraševalni jezik SQL

SQL (ang. Structured Query Language) [11] je najbolj razširjen in standardiziran povpraševalni jezik za delo s podatkovnimi zbirkami in programskimi stavki, ki posnemajo ukaze v naravnem jeziku. Določen je z ANSI/ISO SQL standardom. Pri razvoju testnega orodja smo implementirali veliko število povpraševalnih (ang. query) in nepovpraševalnih (ang. nonquery) izrazov SQL. Kjer je bilo potrebno zagotavljati konsistenco podatkov, smo uporabili več izrazov v transakciji. V primeru, da se le eden od izrazov ne izvede pravilno, se tudi uspešni izrazi ne izvedejo oz. se transakcija »stornira« (ang. rollback).

Povpraševalni izraz je:

```
SELECT Ime, Priimek, Naslov
FROM Student
WHERE vpisna_st = 69110001
```

Nepovpraševalni izrazi so za vpisovanje (`INSERT`), brisanje (`DELETE`) ali posodabljanje (`UPDATE`) zapisov v podatkovni bazi.

Primer vpisovanja v podatkovno zbirko:

```
INSERT INTO Student (Ime, Priimek, Kraj_bivanja, status,
vpisna_st)
VALUES ('Janez','Novak','Ljubljana','RED',69110001)
```

2.6 Uporaba agilnih načel pri razvoju testnega orodja

Pogosto, ko govorimo o metodologijah, pravzaprav ne vemo dobro, kaj metodologija sploh je. Ni res, da številna podjetja pri svojem delu ne uporabljajo nobene metodologije, saj je ta v neki obliki prisotna praktično pri vsakem organiziranem delu. Metodologija namreč zajema vse, kar redno počnemo, da bi dosegli želen rezultat, torej izdelek ali storitev, ki je cilj našega dela. Poleg formalno opredeljenih elementov, ki so običajno zapisani v obliki postopkov, pravil, napotkov, smernic, standardov itd., vsebuje tudi številne nedokumentirane elemente. Posebno mesto med njimi nosi znanje, ki ga člani organizacije uporabljajo pri svojem delu [12].

Na prvi pogled se nam je zdelo, da je razvoj testnega orodja potekal na osnovi neformalno opredeljene metodologije, saj je mejo med formalnim in neformalnim delom metodologije težko postaviti. A bi se lahko na podlagi nekaterih dejstev najbolj približali agilnemu pristopu. Agilne metodologije temeljijo na štirih načelih:

- posamezniki in njihova komunikacija so pomembnejši kot sam proces in orodja,
- delujoča programska oprema je pomembnejša kot popolna dokumentacija,

- vključevanje (sodelovanje) uporabnika je pomembnejše kot pogajanje na osnovi pogodb,
- upoštevanje sprememb je pomembnejše od sledenja planu.

Na osnovi teh načel se lahko opredelimo z agilnim pristopom razvoja naše aplikacije. Ob skromnejši dokumentaciji smo med razvojem orodja z naročnikom veliko komunicirali in se posvetovali o smotrnosti nekaterih prvotnih idej in rešitev. Te niso bile vse popolnoma ovržene, a so bile implementirane, ko je bila aplikacija že zelo funkcionalna, torej proti zaključku projekta. Pomembneje je bilo, da smo s pomočjo testiranja dosegli pravilno delovanje poglavitnih funkcionalnosti. V nasprotnem primeru bi imeli razdrobljeno in nedelujočo aplikacijo do konca razvoja.

3 Razvoj testnega orodja

3.1 Zajem zahtev

Osnovni namen zajema in specifikacije zahtev pri razvoju je opredeliti osnovno funkcionalnost ter tehnološke in druge nefunkcionalne zahteve in omejitve za izgradnjo želene informacijske rešitve [13].

Funkcionalne zahteve so zahteve, ki se nanašajo na želeno funkcionalnost sistema [13].

1. Interpretacija vhodnih sporočil:
    a. `ORDR.xml`
    b. `VSAN.xml`
    c. `RLPL.xml`
2. Pregledovanje strukture vhodnih sporočil.
3. Uporaba predhodno definirane logike (beri: algoritmov) sprejemanja/zavračanja vhodnih sporočil.
4. Tvorba ustreznih odgovorov:
    a. `ORDA.xml`
    b. `VSAA.xml`
    c. `CNTM.xml`
5. Beleženje vhodnih in izhodnih sporočil.
6. Vodenje zgodovine uspešnih in neuspešnih dogodkov.
7. Komunikacija z osrednjim luškim sistemom preko t.i. spletne storitve (ang. web service).

Nefunkcionalne zahteve so zahteve, ki se nanašajo na tehnične in druge nevsebinske zahteve sistema [13].

- Sistem naj bo narejen v dvonivojski arhitekturi, odjemalec-podatkovni strežnik.
- Podatki naj se hranijo v podatkovni bazi MS SQL.

3.1.1 Uvod

Testna orodja navadno niso popolne aplikacije, posebej ne v smislu estetike. Njihova primarna naloga je uporabniku omogočiti pregled nad procesom in funkcijami, ki jih mora sistem zagotoviti končnemu uporabniku. Za iskanje napak je dobro, če ima testno orodje čim manj avtomatike, da je možno v vsakem koraku odpravljati morebitne napake. Ni zmeraj nujno, da eno orodje nudi vse funkcionalnosti. V nekaterih primerih morajo testni uporabniki poganjati celo več testnih orodij, za vsak del sistema posebej, da bi lahko testirali celoten nabor funkcionalnosti.

Podobno orodje sicer obstaja za sistem Cosmos, vendar je pisano v starejšem razvojnem okolju in Visual Basicu. Ker ni bilo smiselno prirejati stare aplikacije za delovanje v novejšem sistemu, smo dobili nalogo razvoja aplikacije iz ničelne točke, v zadnjem ogrodju .NET (ang. .NET Framework 4.0) in jeziku C#. Ker smo se spopadli s prvo samostojno nalogo v objektno usmerjenem jeziku, se nam je zdela naloga zanimiva. Velik izziv pa je bila tudi zato, ker je bilo potrebno aplikacijo sprogramirati v najkrajšem možnem času in je problem iz realnega sveta, in sicer Luke Koper.

3.1.2 Naloge aplikacije

Aplikacija ima nalogo testnega orodja. Je vrsta simulatorja večjega sistema, ki ima zelo omejeno avtomatiko. Potrebna je uporabnikova interakcija za vsak korak, da so spremembe, ki jih uporabnik sproži z akcijami, jasno sledljive.

Zahtevane so bile naslednje funkcionalnosti aplikacije:

- sprejemanje XML sporočil iz sistema in ob vsaki osvežitvi branje morebitnih novih,
- razbiranje (ang. parsing) XML sporočil,
- strukturirano shranjevanje sporočil v podatkovni bazi,
- prikaz vhodnih sporočil na grafičnem vmesniku po objektih,
- izvajanje operacij nad zapisi in posameznimi objekti, ki se jim spremeni nadaljnja uporaba,
- tvorba izhodnih XML sporočil (t.i. odgovorov),
- prikaz odgovorov na grafičnem vmesniku v takem formatu, da ima uporabnik možnost ročno spremeniti podatke,
- pošiljanje odgovorov nazaj v sistem.

3.1.3 Predpogoji

Predpogoji za poganjanje aplikacije so:

- nameščeno najsodobnejše ogrodje za poganjanje aplikacije,
- povezava s sistemom, ki nam pošilja sporočila,
- za uporabo aplikacije je potrebno dobro poznavanje gibanja sporočil in osnovno znanje uporabe računalnika.

3.2 Reševanje in pristop k razvoju testnega orodja

Razvoj testnega orodja je bil naš prvi samostojni projekt z objektnim pristopom. Ker smo pred razvojem delali v sistemski administraciji, nismo imeli realnih izkušenj na področju programiranja z objektno usmerjenimi jeziki. Razvoj testnega orodja je bil tako naš prvi samostojni projekt. Pomanjkanje dokumentacije in spremembe pri zahtevah funkcionalnosti so nas prisilile, da smo veliko komunicirali s preostalimi člani projektne skupine. Kot zanimivost naj povemo, da so pri dogovarjanju o najprimernejši XML strukturi sporočil nastale štiri različne verzije.

Pristop k problemu je bil naslednji:

- določanje zadolžitev posameznikov v skupnem projektu,
- razumevanje osnovnih zahtev iz dokumentacije,
- seznanjanje z vhodnimi podatki (XML sporočila),
- izbira komunikacije s komunikacijskim strežnikom BizTalk,
- cikli razvoja in testiranja.

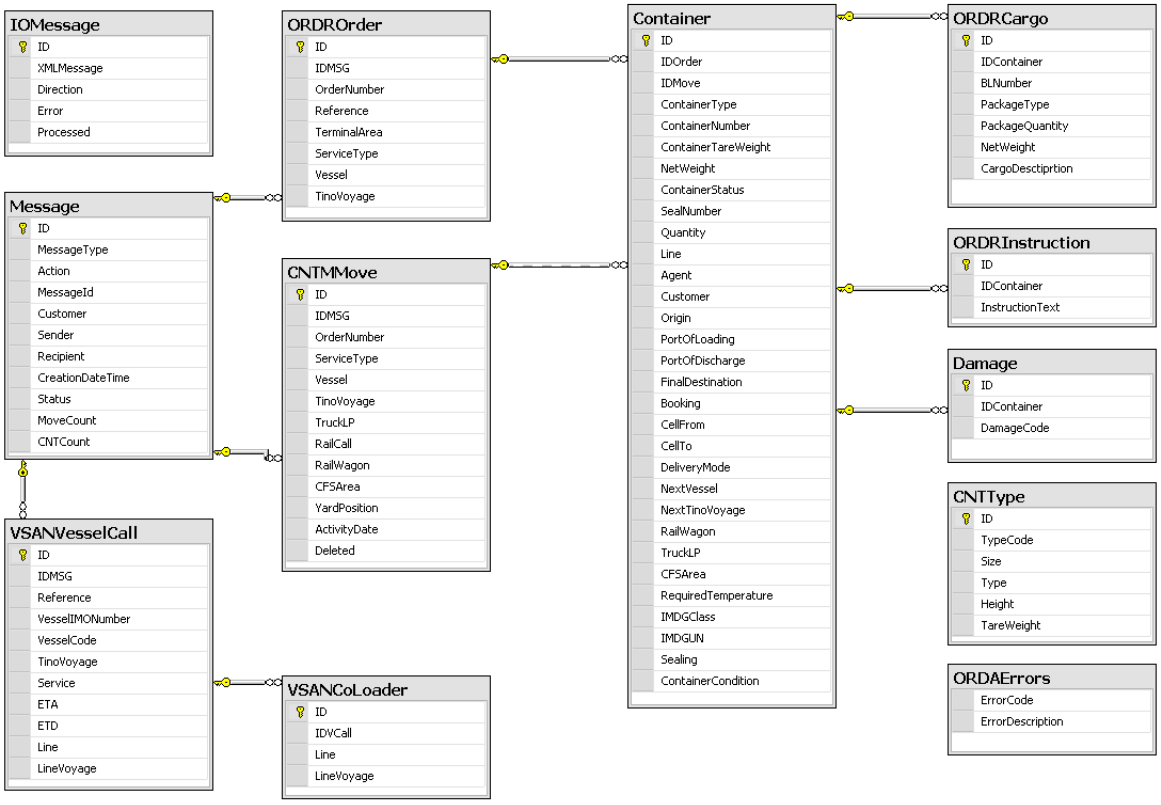
Razvoj je potekal v iteracijah. Vsako funkcionalnost, ki smo jo končali, smo testirali, če deluje v optimalnih okoliščinah, odpravili smo napake in se nato lotili naslednje funkcionalnosti. Izdaje programske rešitve so si sledile glede na zmožnost interpretacije vhodnih XML dokumentov. V osnovi so bile štiri, a so tudi že razviti deli aplikacije v določenih primerih dobili popravke, ker so se navezovali na trenutni razvoj. Da bo razumljivo, pri razvoju funkcionalnosti nad kontejnerskimi zapisi je bilo potrebno popraviti tudi pregled dispozicij.

3.2.1 Opis

Potrebno je preveriti ali imamo v podatkovni bazi nova sporočila. Ta se hranijo v tabeli, preko katere aplikacija komunicira s komunikacijskim strežnikom BizTalk. Ob zagonu aplikacije se izvede preverjanje, če obstajajo nova sporočila. Aplikacija vsa nova sporočila razbere in podatke vpiše v pripadajoče tabele v podatkovni bazi ter prikaže vsa že obdelana sporočila na grafičnem vmesniku.

Obdelani podatki se hranijo v podatkovni bazi, kot prikazuje slika 10.

Na sliki sta vidni tudi tabeli `CNTType` in `ORDAErrors`, ki vsebujeta šifrante in sta bili dodani v kasnejših fazah razvoja, uporabljata pa se v aplikaciji.



Slika 10: Podatkovna baza za testno orodje

3.2.2 Povezava s komunikacijskim strežnikom

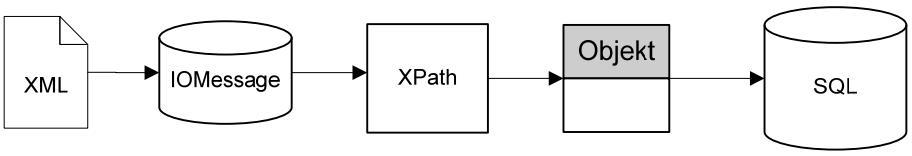
Pri povezovanju s komunikacijskim strežnikom smo imeli na voljo res velik nabor protokolov, a smo se v izogib zapletom odločili za zelo enostavnega. Ker ne potrebujemo dodatnih varnostnih ukrepov, saj se testiranje izvaja v zaprtem okolju, smo se odločili, da uporabimo kar protokol preko SQL vrat in zapis podatkov v tabelo podatkovne baze.

Tabela `IOMessage` je zelo enostavna, saj vsebuje le polja:

- `ID` – služi kot ključ, pod katerim je shranjeno sporočilo.
- `XMLMessage` – XML sporočila v izvirni obliki.
- `Direction` – služi kot oznaka smeri, v katero potuje sporočilo; 0 za vhodno, 1 za izhodno.
- `Error` – služi kot indikacija ali je prišlo do napake.
- `Processed` – nova sporočila imajo vrednost 0 že obdelano.

Ko uporabnik požene aplikacijo, se izvede preverjanje, ali obstajajo nova sporočila v tabeli `IOMessage`. Med delovanjem aplikacije uporabnik lahko preveri ali obstajajo novi zapisi, in sicer s pritiskom na gumb `Osveži`. Če obstajajo nova sporočila, se jih razbere s pomočjo jezika XPath. Branje poteka od korenskega vozlišča (ang. root node) po segmentih do

zadnjega otroka vozlišča (ang. child node). Podatki se prepisujejo v podatkovno strukturo (objekt), ta pa se zapiše na podatkovno bazo (slika 11).



Slika 11: Proces pisanja podatkov na PB

Strukturo smo zasnovali z dedovanjem. Naredili sem razred z osnovnimi objekti, ki imajo presek skupnih lastnosti vseh sporočil. Tako smo se izognili tudi nepotrebnemu pisanju, ob morebitnih zahtevah po spremembah pa ne bo nepotrebnega iskanja po kodi na več mestih.

Osnovna definicija podatkovne strukture sporočila za segment `message`:

```
[Serializable]
public class BaseDoc
{
    public string MessageType = "";
    public string Action = "";
    public string MessageId = "";
    public string Sender = "";
    public string Recipient = "";
    public DateTime CreationDate = DateTime.MinValue;
}
```

`Message` segment v dispoziciji, z dedovanjem iz osnovne podatkovne strukture:

```
[Serializable]
public class ORDRMessage : BaseDoc //1
{
    public string Customer = "";
    public string Status = "";
    public int ContainerCount = 0;
    public int MoveCount = 0;

    public ORDROrder order;

    public ORDRMessage()
    {
        order = new ORDROrder();
    }
}
```

3.2.3 Shranjevanje sporočil v podatkovno bazo

Pri zapisu sporočil v več tabel hkrati, kjer se dedujejo novo nastali ključi, je najboljše, če se izvedejo vsi zapisi ali nobeden. Drugače se v podatkovni bazi pričnejo kopičiti nekonsistentni podatki, ki nimajo pripadajočih zapisov in tvorijo neuporaben balast. Največ težav smo zasledili, ko je povpraševalni izraz vračal `null` vrednost in je to vrednost aplikacija poskusila zapisati v naslednjem SQL izrazu v tabelo. Da bi se izognili takim zapisom, je potrebno podatke zapisati »hkrati«. Mehanizem za tako zapisovanje se imenuje transakcija.

Transakcija je sestavljena iz več SQL izrazov, ki se zaporedno izvajajo. Ob neuspešnem zapisu samo enega od izrazov celotno transakcijo storniramo in tako ohranjamo celovitost podatkov. Ta mehanizem nam je prišel še posebej prav, ko je bilo potrebno vpisovati nova sporočila. Vsak nov zapis namreč dobi novo ustvarjen ključ (`ID`), ki je lahko vezni člen na vrstico v drugi tabeli. Zato je pomembno, da se vsi zapisi izvedejo brez napak.

Iz primera zapisa najave ladje (VSAN) v podatkovno bazo je razvidna uporaba zapisovanja v transakciji. Parametri za povezavo na podatkovno bazo, skupaj z uporabniškim imenom in geslom (t.i. ConnectionString), so shranjeni v ločeni datoteki. Ravno tako so ločeno shranjeni tudi vsi SQL izrazi. Opazna je uporaba parametriziranega načina zapisa vrednosti.

Parametriziran SQL izraz, uporabljen v spodnjem primeru:

```
public static string saveVSANMessage =
"INSERT INTO Message (MessageType, Action,MessageId, Customer, Sender, "+
"Recipient, CreationDateTime, Status) " +
"VALUES (@MessageType, @Action, @MessageId, @Customer, @Sender, @Recipient,
"+ "@CreationDateTime, @Status);" +
"SELECT CAST(scope_identity() AS int);"
```

Izraz na koncu vrne tudi ključ pravkar zapisane vrstice. Dobljeni podatek se vpiše kakor tuj ključ v naslednjem zapisu, torej pri izvajanju transakcije.

Primer zapisa najave ladje (VSAN) v podatkovno bazo, jezik C#:

```
public static bool SaveNewVSAN(VSANMessage message)
{
 SqlTransaction transaction;
 SqlConnection conn = new SqlConnection(Settings.ConnString);
 SqlCommand cmd = new SqlCommand();
 cmd = conn.CreateCommand();
 cmd.Connection.Open();
 transaction = conn.BeginTransaction();
 cmd.Connection = conn;
 cmd.Transaction = transaction;
 try
 {
     int msgId;
     int vcId;
     cmd.CommandText = TosterSQL.saveVSANMessage;
     cmd.Parameters.AddWithValue("@MessageType", message.MessageType);
     cmd.Parameters.AddWithValue("@Action", message.Action);
     cmd.Parameters.AddWithValue("@MessageId", message.MessageId);
     cmd.Parameters.AddWithValue("@Customer", message.Customer);
     cmd.Parameters.AddWithValue("@Sender",message.Sender);
     cmd.Parameters.AddWithValue("@Recipient", message.Recipient);
     cmd.Parameters.AddWithValue("@CreationDateTime", message.CreationDate);
     if (message.Status == "") cmd.Parameters.AddWithValue("@Status", "Nov");
     else cmd.Parameters.AddWithValue("@Status", message.Status);
     msgId = (int)cmd.ExecuteScalar();
     cmd.Parameters.Clear();
     cmd.CommandText = TosterSQL.saveVSANVesselCall;
     cmd.Parameters.AddWithValue("@IDMSG", msgId);
     cmd.Parameters.AddWithValue("@Reference", message.vesselCall.Reference);
     cmd.Parameters.AddWithValue("@VesselIMONo",message.vesselCall.VesselIMONo);
     cmd.Parameters.AddWithValue("@VesselCode", message.vesselCall.VesselCode);
     cmd.Parameters.AddWithValue("@TinoVoyage", message.vesselCall.TinoVoyage);
     cmd.Parameters.AddWithValue("@Service", message.vesselCall.Service);
     cmd.Parameters.AddWithValue("@ETA", message.vesselCall.ETA);
     cmd.Parameters.AddWithValue("@ETD", message.vesselCall.ETD);
     cmd.Parameters.AddWithValue("@Line", message.vesselCall.Line);
     cmd.Parameters.AddWithValue("@LineVoyage", message.vesselCall.LineVoyage);
     vcId = (int)cmd.ExecuteScalar();
     foreach (VSANCoLoader vcCL in message.vesselCall.RowsCoLoader)
     {
         cmd.Parameters.Clear();
         cmd.CommandText = TosterSQL.saveVSANCoLoader;
         cmd.Parameters.AddWithValue("@IDVCall", vcId);
         cmd.Parameters.AddWithValue("@Line", vcCL.Line);
         cmd.Parameters.AddWithValue("@LineVoyage", vcCL.LineVoyage);
         cmd.ExecuteNonQuery();
      }
     transaction.Commit();
     return true;
 }
 catch (Exception)
 {
     transaction.Rollback();
     return false;
 }
}
```

3.2.4 Tipi vhodnih sporočil

Zagotavljanje nemotene komunikacije med sklopi je ena glavnih skrbi sistema. Zato se veliko pozornosti polaga v pravilno oblikovanje in pretvarjanje sporočil. TOS sicer sam po sebi ne pretvarja sporočil, saj to zanj počne BizTalk strežnik, jih pa tvori kot odgovore operaterja v XML obliki.

Imamo dve vrsti vhodnih sporočil. Najava ladje, ki, kot že samo ime pove, najavlja prihod ladje v pristanišče. Dispozicija je dokument, ki nosi s sabo zahteve po manipulaciji z blagom. Ker gre v našem primeru za kontejnerski terminal, so to blago kontejnerji. Razlikujeta se že po vpisu v korenskem elementu.

Najava ladje VSAN (ang. Vessel Call Announcement):

```
<VSAN xmlns="http://luka-koper.si/VSAN">
  <message>
    …
    <vessel_call>
      …
    </vessel_call>
  </message>
</VSAN>
```

Primer dispozicije ORDR (ang. Workorder):

```
<ORDR xmlns="http://luka-koper.si/ORDR">
  <message>
    …
    <order>
      …
      <container>
      …
        <cargo>
          …
        </cargo>
      </container>
    </order>
  </message>
</ORDR>
```

3.2.5 Tipi izhodnih sporočil

Za sporočanje operaterja iz kontejnerskega terminala proti stranki se uporabljajo tri vrste sporočil ali odgovorov. Normalen potek dela zahteva pozitivno potrditev v naslednjem vrstnem redu:

- sprejem najave ladje,
- sprejem dispozicije,
- premikanje kontejnerjev.

3.2.5.1 Odgovor najave ladje VSAA

Odgovori na sporočila so zelo nezahtevni XML dokumenti. Odgovor na najavo ladje se imenuje VSAA (ang. Vessel Call Announcement Acknowledgement) in se pošilja ob pozitivnem ali negativnem odgovoru. Za nadaljevanje operacij z dispozicijami, ki se navezujejo na to ladjo, mora bit odgovor ugoden.

Odgovora se razlikujeta v segmentu `vessel call` in v vsebini elementa `error_flag`.

Izsek strukture pozitivnega odgovora VSAA:

```
<VSAA xmlns="http://luka-koper.si/VSAA">
  <message>
    …
    <vessel_call>
      …
      <error_flag>OK</error_flag>
    </vessel_call>
  </message>
</VSAA>
```

Negativni odgovor ima v XML dokumentu še enega ali več segmentov »error«, ki opisujejo napake in razloge zavrnitve.

Izsek strukture negativnega odgovora VSAA:

```
<VSAA xmlns="http://luka-koper.si/VSAA">
  <message>
    …
    <vessel_call>
      …
      <error_flag>ER</error_flag>
      <error>
        <error_code>ER0</error_code>
        <error_text>WRONG_VESSEL_NAME</error_text>
        <error_variable>0</error_variable>
      </error>
    </vessel_call>
  </message>
</VSAA>
```

3.2.5.2 Odgovor na prejeto dispozicijo ORDA

Preden začnemo s premiki po kontejnerskem terminalu, je potrebno sprejeti prejeto dispozicijo. Imenuje se `ORDA` (ang. Workorder Acknowledgement) in vsebuje osnovne identifikacijske podatke. Sporočilo je zelo podobno pozitivnemu odgovoru najave ladje.

Če se dispozicijo zavrača, je pred pošiljanjem potrebno izpolniti obrazec za kakšno vrsto napake gre, možno je popraviti številko napake in dodati svoj opis le-te. Osnovni opis napak je dogovorjen, zato se okvirni šifrant nahaja v podatkovni bazi, v tabeli `ORDAErrors`. Pri tvorjenju XML sporočila se doda toliko novih `error` segmentov z opisi napak, kolikor jih predhodno zapišemo.

Izgled zavrnitve dispozicije:

```
<ORDA xmlns="http://luka-koper.si/ORDA">
  <message>
    <message_type>ORDA</message_type>
    <action>CRT</action>
    <message_id>000000000000011</message_id>
    <sender>CTTW</sender>
    <recipient>LUKA</recipient>
    <creation_datetime>2011-06-14T22:36:19</creation_datetime>
    <order>
      <order_number>2116450</order_number>
      <reference>21164500000</reference>
      <error_flag>ER</error_flag>
      <error>
        <error_code>302</error_code>
        <error_text>ERROR_ON_CONTAINER_DATA</error_text>
        <error_variable></error_variable>
      </error>
      <error>
        <error_code>702</error_code>
        <error_text>ERROR_ON_INSTRUCTION_DATA</error_text>
        <error_variable></error_variable>
      </error>
    </order>
  </message>
</ORDA>
```

3.2.5.3 Opisi premikov kontejnerjev

Kontejnerska gibanja ali XML sporočila `CNTM` (Container Move - Confirmation) so sporočila, ki so posledica gibanja/premikov kontejnerja iz ladje do lokacije na kontejnerskem terminalu. Za vsak kontejner, ki se razklada, se pošlje sporočilo v sistem. Zato je največje možno število premikov enako številu kontejnerjev na dispoziciji, kar se tudi zabeleži in sešteje ob poslanem sporočilu. Ker gre za testno okolje, se morebitno ponovno pošiljanje ne prišteva.

Primer gibanja s poškodovanim kontejnerjem:

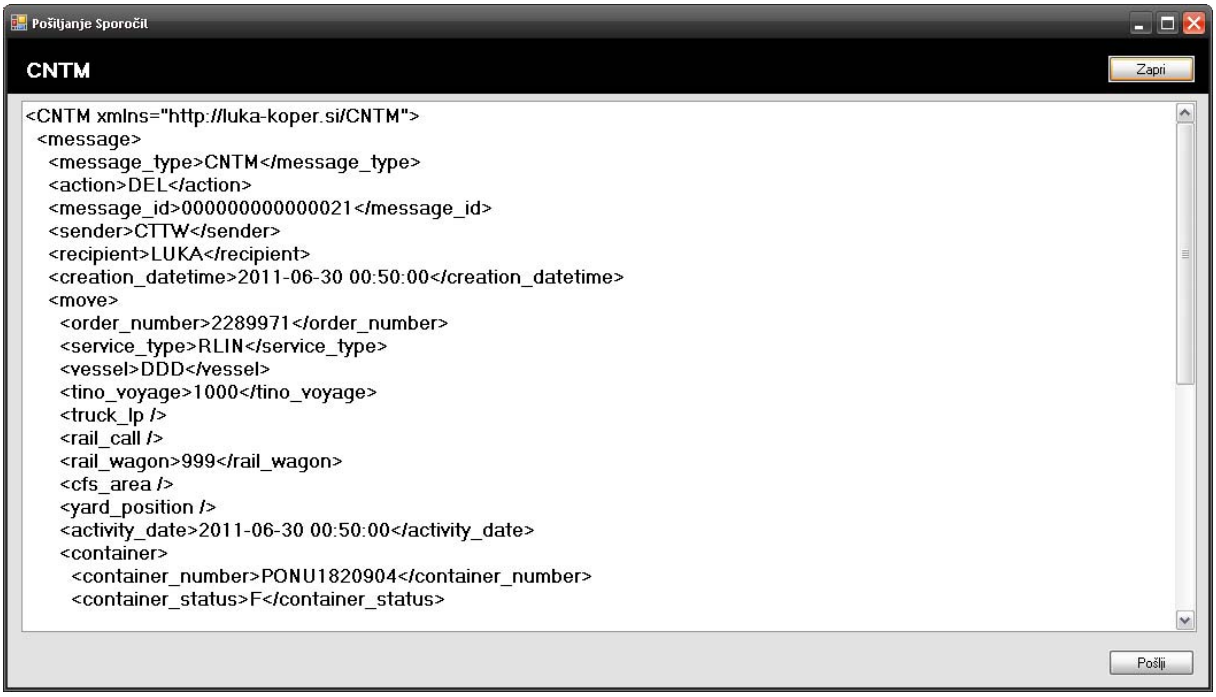
```
<CNTM xmlns="http://luka-koper.si/CNTM">
  <message>
    <message_type>CNTM</message_type>
    <action>CRT</action>
    <message_id>000000000000016</message_id>
    <sender>CTTW</sender>
    <recipient>LUKA</recipient>
    <creation_datetime>2011-06-29 23:12:10</creation_datetime>
    <move>
      <order_number>2116471</order_number>
      <service_type>VSIN</service_type>
      <vessel>HANA</vessel>
      <tino_voyage>46331</tino_voyage>
      <truck_lp />
      <rail_call />
      <rail_wagon />
      <cfs_area />
      <yard_position />
      <activity_date>2011-06-29 23:12:10</activity_date>
      <container>
        <container_number>TRLU3105424</container_number>
        <container_status>F</container_status>
        <container_type>20DV</container_type>
        <tare_weight>2100</tare_weight>
        <net_weight />
        <line>HAN</line>
        <agent>SIGU</agent>
        <customer />
        <booking />
        <seal_number>2406</seal_number>
        <container_condition>OS</container_condition>
        <damage>
          <damage_code>RF03</damage_code>
        </damage>
        <damage>
          <damage_code>LR06</damage_code>
        </damage>
        <damage>
          <damage_code>ROOF</damage_code>
        </damage>
      </container>
    </move>
  </message>
</CNTM>
```

Kot je razvidno iz primera, dokument nosi s seboj vse podatke o:

- lokaciji premika,
- prevoznem sredstvu,
- lokaciji na terminalu,
- aktivnosti,
- carinjenju,
- lastnosti kontejnerja,

- lastništvu kontejnerja,
- prevozniku,
- morebitnih poškodbah kontejnerja.

Implementirana je tudi možnost brisanja gibanja, v primeru, da bi prišlo do napačno posredovane informacije s strani operaterja na terminalu. Ob brisanju se seštevek gibanj zmanjša, v sistem pa pošlje enak dokument kakor je potrditev gibanja, razlikuje se le v akciji, ki ima vrednost `DEL` (ang. delete) (slika 12), kar pomeni briši.

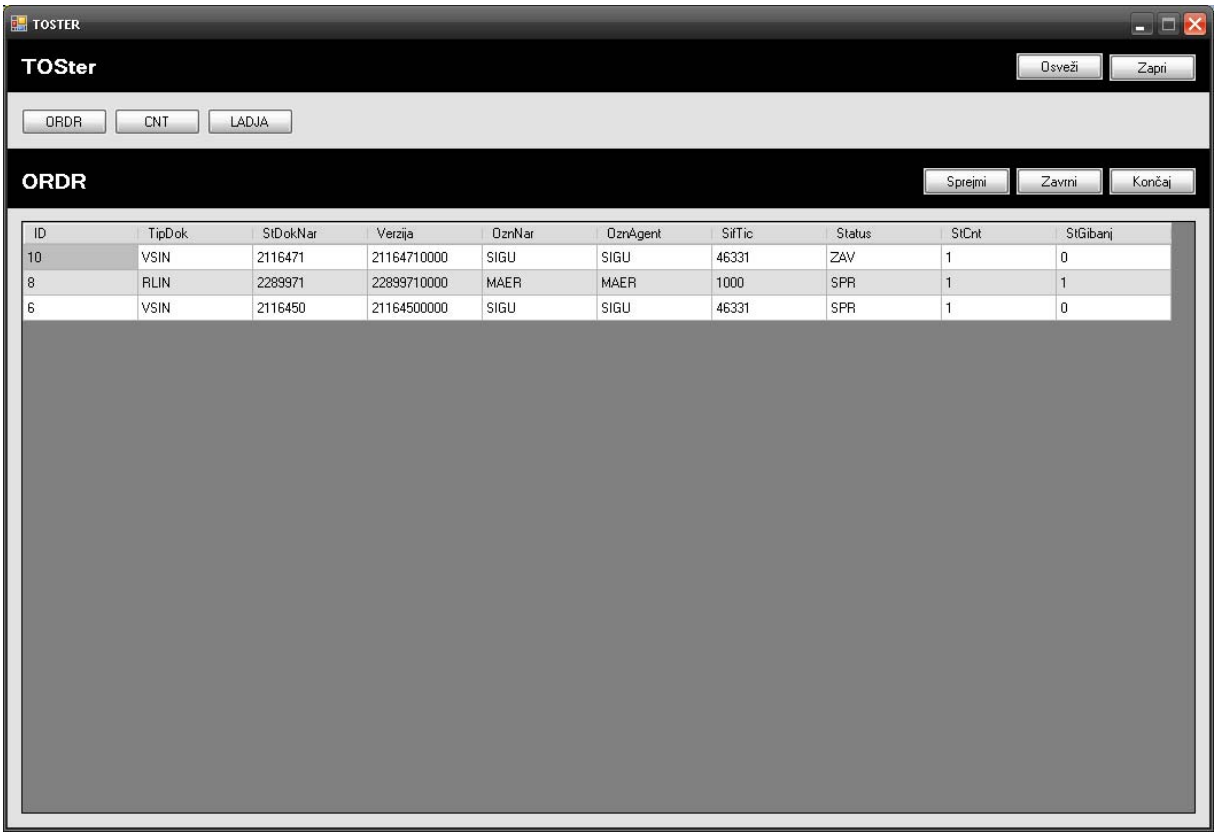


Slika 12: Izpis CNTM XML dokumenta za brisanje gibanja

Na podatkovni bazi se dejanska vrstica ne izbriše iz tabele, ampak se ji spremeni samo zastavica `Deleted`, ki je tipa bit, torej na 1. Na tak način imamo shranjeno vso zgodovino.

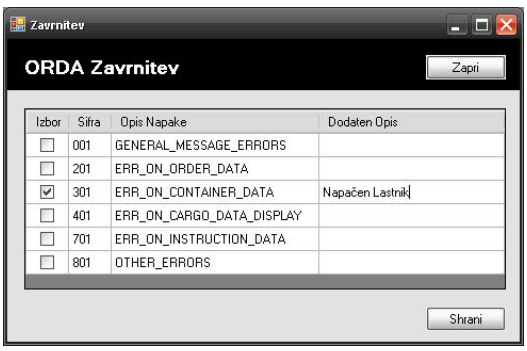
4 Izgled in uporaba

Izgled samega testnega orodja je zelo enostaven, ker sta hitra dostopnost in funkcionalnost pomembnejša kakor estetika. Grafični vmesnik ima tri poglavitne poglede. Ob zagonu aplikacije se izvede branje novih sporočil iz podatkovne baze in prikaže tabela z dispozicijami (ORDR), kot prikazuje slika 13. Preko gumbov lahko dispozicijo in najavo ladij uporabnik zavrne ali sprejme oz. pri kontejnerju uporabnik pošlje ali briše gibanje. Pri vseh treh se odpre okno s predogledom XML dokumenta (slika 16). Posebnost je zavrnitev dispozicije.



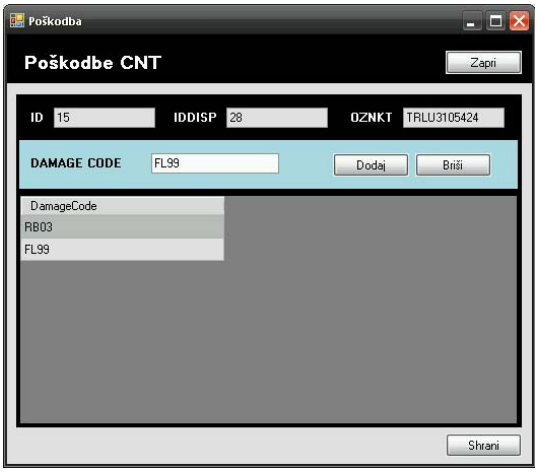
Slika 13: Grafični vmesnik našega testnega orodja

Če uporabnik zavrne dispozicijo, se odpre obrazec s šifrantom napak (slika 14). Ko uporabnik izbere napake, lahko po želji doda še opis in pošlje podatke v tvorbo XML dokumenta.



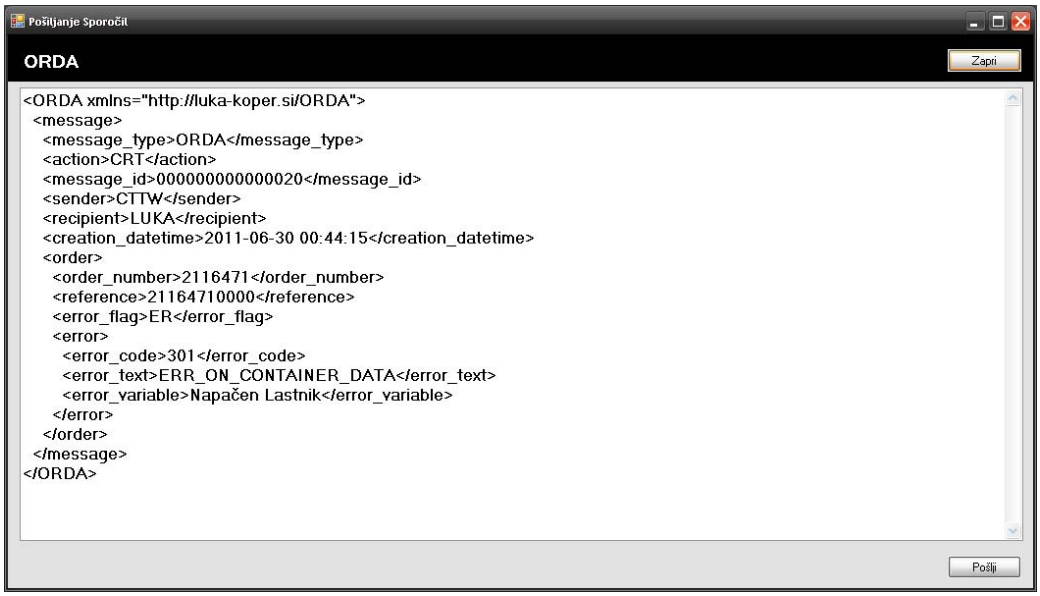
Slika 14: Okno za izbiro napak pri zavrnitvi dispozicije

Ob izbiri dispozicije in pritisku na gumb CNT se nam prikaže seznam vseh kontejnerjev v sprejeti dispoziciji. Vsak kontejner, ki pride v pristanišče, naj bi pregledali, da ni poškodovan, morebitne poškodbe pa mora inšpektor zabeležiti. Zaradi zavarovalniškega kritja je pomembno, ali so bile poškodbe povzročene med prevozom do pristanišča ali na terminalu. Poškodbe na grafičnem vmesniku vpiše uporabnik. S pritiskom na gumb Poškodbe se odpre okno (slika 15), v katerega vpiše standardizirane šifre poškodb. Ko uporabnik poškodbe shrani, se te shranijo na podatkovno bazo in njihov seštevek se prikaže tudi v vrstici kontejnerja.



Slika 15: Okno za vpis nastalih poškodb na kontejnerju

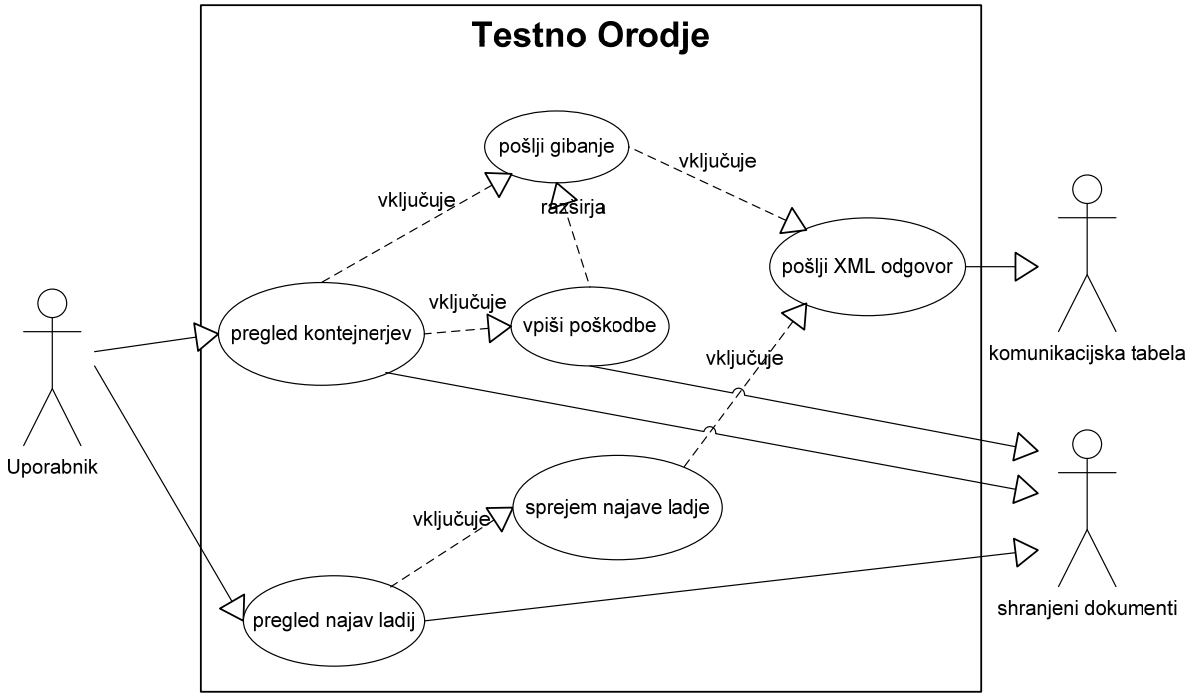
Med reševanjem problema se je izkazala dobra stran objektnega programiranja. S polimorfizmom smo dosegli to, da se za pregled in pošiljanje vseh vrst XML dokumentov na podatkovno bazo uporablja en sam vmesnik (slika 16). Razlikujejo se le v klicih metod, ki grafični vmesnik prikličejo in napolnijo s sporočilom. Tako imamo šest vrst sporočil, pet metod in en vmesnik.



Slika 16: Vmesnik za pregled in pošiljanje XML sporočil

Uporabnikom je aplikacija na pogled všeč in je odzivna kot so pričakovali. Uporaba samega testnega orodja je pogojena s poznavanjem pravil in standardov v pristanišču. Dobro je tudi, če uporabnik pozna celo logistično verigo in sam proces dela na pristaniškem terminalu, saj je v uporabi veliko šifrantov, ki jih laik ne bo razumel.

Diagram UML prikazuje primere uporabe (slika 17) testnega orodja.



Slika 17: Prikaz primerov uporabe s pomočjo jezika UML

Slika 17 prikazuje branje shranjenih zapisov in pošiljanje XML odgovorov preko enotnega vmesnika. Akterja »komunikacijska tabela« in »shranjeni dokumenti« sta del istega podatkovnega strežnika, vendar sta ločeni enoti. Pregled kontejnerjev je vmesnik, ki omogoča pregled kontejnerjev, pošiljanje gibanj ter vpis poškodb. Gibanja lahko vključujejo tudi poškodbe.

Preko vmesnika za najave ladij omogoča pregled ladij. Pošiljanje odgovorov poteka preko istega vmesnika, ki se uporablja za pošiljanje kontejnerskih gibanj.

Če bi hoteli s primeri uporabe prikazati več funkcionalnosti, bi postal diagram nepregleden. Ne glede na to nudi diagram UML bralcu občutek o kompleksnosti testnega orodja.

5 Problemi in rešitve

Razvoj aplikacije se je zaradi odsotnosti članov ekipe zavlekel za kak teden, vendar smo težave vseeno reševali preko elektronskih komunikacijskih poti. Med samim programiranjem testnega orodja smo naleteli na težave, ki smo jih s pomočjo izkušenejših sodelavcev in raziskovanja po spletu rešili.

V začetnih fazah testiranja branja XML dokumentov smo naleteli na težavo. Izkazalo se je, da `XPath` metoda ni bila najboljša izbira, a ker je bil del kode za branje že v celoti spisan, ni bilo pravega razloga, da bi začeli znova.

Vsa vhodna sporočila, ki se zapišejo v `IOMessage` tabelo, imajo v korenskem elementu vpisan atribut `namespace`, ki se uporablja za zagotavljanje edinstvenega imena elementov in atributov v XML dokumentu. XPath je imel neizmerne probleme z branjem vsebine pravilno naloženega dokumenta. Kljub pravilni navedbi iskanega vozlišča, je vračal ničelno vrednost. Zato smo dobili nasvet in uporabili `RegEx` razred, ki z metodo regularnih izrazov (ang. regular expression) zamenja iskane znake v spremenljivki.

Uporabljena je bila na naslednji način:

```
xmlString = Regex.Replace(xmlString, "<VSAN xmlns=\".*?\">", "<VSAN>",
RegexOptions.IgnoreCase | RegexOptions.Singleline);
xml.LoadXml(xmlString);
XmlNode xnMsg = xml.SelectSingleNode("/VSAN/message");
```

V XML dokumentu poišče znakovno polje, ki se začne z "`<VSAN xmlns=`" in vsak znak, tudi ponavljajoč, vključno s presledki, do niza "`>`" zamenja z nizom "`<VSAN >`". Med operacijo ignorira posebne znake in velikost črk, in sicer s pomočjo zadnje poševnice.

Razlaga regularnih izrazov:

`.` (pika) – ustreza vsak znak, razen ubežni znaki (npr. `\n`, `\r`), velja samo za eno vrstico

`*?` – poskusi ponoviti predhodni znak 0 ali se ponovi več kot enkrat

`.*?` – izbere vse znake

`\` – zadnja poševnica izloči posebne znake iz izraza, da ne bi vplivali na regularni izraz

`IgnoreCase` – ignorira pomembnost velikih in malih črk

`SingleLine` – spremeni pomen .(pike), tako da upošteva vsak znak, tudi ubežnega `\n`

Ko smo programirali izpis vseh kontejnerjev iz dispozicije v svojo preglednico, objekt tipa `DataGridView` (tabelarični prikaz), smo naleteli na nevšečnost. Med zahtevami je bila želja po prikazu stolpcev, ki jih v podatkovni bazi ni, ter da bi bil izgled celic v stolpcu `PoslanoGibanje` predstavljen kakor razred `CheckBox` (polje za odkljukat). Problem je rešljiv, a ne s sprotnim ustvarjanjem in polnjenjem tabele. Zato je potrebno ustvariti posebno vrsto razreda, tipizirani `DataSet` (slika 18). Tak razred obdrži svoje lastnosti in metode, ki omogočajo dostop do vsebine tabele na tipsko varen način (type-safe). Struktura tipiziranega `DataSeta` se predstavlja z XML shemo, XSD, kakršno ima končnico tudi datoteka, v kateri je shranjen.

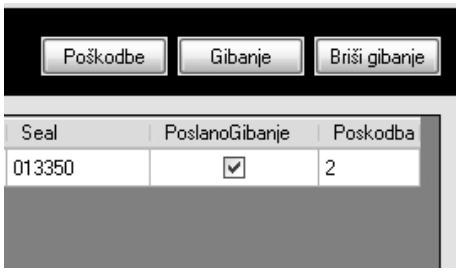
Slika 18: Primer tipiziranega razreda `DataSet`

Taka vrsta razreda je posebna, ker ji programer lahko določi imena polj in tipe podatkov, ne da bi slednji sploh obstajali, na samem prikazu tabele pa se bodo prikazali. Programer lahko določi še dimenzije celice, poljubno ime in interpretacijo vsebine.

V našem testnem orodju smo tak razred tabele najprej uporabili pri prikazovanju seznama kontejnerjev. Zahteve uporabnika so bile, da se v stolpiču `PoslanoGibanje` prikaže vsebina kot razred `CheckBox`. `CheckBox` je razred, ki vrača logično vrednost. Kljukica označuje poslano, prazno polje pa neposlano gibanje (slika 19). Rešitev je bila poizvedba, ki je vračala za to celico celoštevilski rezultat. Ker ima vsak kontejner lahko samo eno gibanje, smo interpretirali `integer` (celoštevilski) rezultat kot `boolean` (logično vrednost).

Izsek iz poizvedbe:

```
(SELECT COUNT(ID) FROM CNTMMove WHERE IDMSG = @ID AND Deleted = 0) as
PoslanoGibanje
```



Slika 19: Celica `PoslanoGibanje`, celoštevilski rezultat predstavljen kot logična vrednost

Enak tip razreda `DataSet` smo uporabil tudi za implementacijo obrazca napak pri zavrnitvi dispozicije in beleženju poškodb kontejnerjev.

Zadnja težava, s katero smo se ukvarjali in je vzela kar nekaj časa, je bil problem drevesnega prikaza v večvrstičnem tekstovnem polju. V `textbox` komponenti, z aktivirano lastnostjo o večvrstičnem prikazu teksta, je nastal problem kako prikazati sporočilo XML v drevesni obliki. Privzeto prikaže sporočilo v eni neprekinjeni vrstici. Po več samostojnih poskusih dodajanja praznih znakov smo na spletu poiskali rešitev [14], ki nam je olajšala delo. Z nekaj popravki smo jo uporabili kot novonastalo metodo `FormatXML4Reading`, ki vrača pravilno urejen znakovni niz.

Rešitev temelji na instanci DOM bralnika, ki iz XML dokumenta sam razbere drevesno strukturo. Nato s pomočjo instance `XmlTextWriter` razreda določimo format izpisa. V našem primeru je to v zamaknjenem načinu (ang. idented mode). Nato trenutni spomin shrani v spominsko datoteko. To datoteko v toku (ang. stream) izpiše tekst v spomin in ga vrne kot tekst z zamiki, kot je prikazano na sliki 20.

Slika 20: Drevesni izpis XML sporočila v textbox komponenti

Tako prikazan zapis smo brez spreminjanja in izločanja praznih ali ubežnih znakov shranili v XML polje podatkovne baze. Zapisal se je pravilen XML dokument, saj pri pisanju sam strežnik izloči vse odvečne znake.

6 Sklepne ugotovitve

V okviru diplomske naloge smo razvili testno orodje za sistem upravljanja pristaniških terminalov, aplikacija pa se trenutno obnaša po naših željah. Zaradi omejenosti s časom, sredstvi in predhodnim poznavanjem programskega jezika, smo izbrali agilen razvoj. Izbira samega programskega jezika je bila v skladu s politiko podjetja, vendar to ne vpliva bistveno na rezultat, saj jezik pri današnjih zmogljivostih strojne opreme niti ni tako pomemben.

Uporabniki so zadovoljni z odzivnostjo aplikacije. Všeč jim je možnost urejanja XML dokumentov pred pošiljanjem, potrebno pa je povedati, da trenutno pošiljajo sporočila v obdelavo uporabniki sami, zato se jih ne pojavlja veliko naenkrat. Pravo oceno bomo lahko podali šele, ko bo sistem samostojno pošiljal večje število sporočil. Že v naprej je jasno, da bodo operacije nad večjim številom vhodnih sporočil časovno najbolj zahtevne.

V nadaljnjih korakih razvoja sledi še dokončna implementacija dnevnika napak, ki se lahko pojavijo med branjem vhodnih sporočil, in možnost popravljanja ter shranjevanje podrobnosti posameznega kontejnerja. Med podrobnosti kontejnerja spada tudi podatek o tipu kontejnerja. Ta podatek je zelo občutljive narave, saj je povezan s šifrantom, le-tega pa ne smemo spreminjati, ker vpliva na vse kontejnerje enakega tipa. Zato je še pod vprašajem, ali bo aplikacija omogočala neposredno urejanje šifranta.

Možno je, da bo potrebna nadgraditev z novim tipom sporočil RLPL (ang. Rail Plan). Gre za dokument, s pomočjo katerega uporabniki načrtujejo natovarjanje vlakovnih kompozicij. Implementacija je bila sicer v načrtih že na začetku projekta, vendar so jo uporabniki kasneje umaknili iz zahtev. Aplikacijo bomo v prihodnih tednih razvili do končne podobe in funkcionalnosti.

Tema diplomske naloge je vsekakor zanimiva, ker je aktualna in se dotika realnega problema Luke Koper. Ob razvoju aplikacije smo se seznanili s procesi na kontejnerskem terminalu, pridobili smo nova znanja s področja agilnih metodologij, naučili pa smo se tudi nekaj novih prijemov v objektnem programiranju.

Seznam slik

Viri

[1] Interna dokumentacija združbe Actual-IT d.d.

[2] Microsoft BizTalk 2010 Technical Overview, dostopno na:
http://msdn.microsoft.com/en-us/library/aa547058(v=BTS.70).aspx

[3] Projekt TinO na spletnih straneh združbe Actual IT:
http://www.actual.si/poslovne_resitve/po_dejavnosti/logistika/studija_primera

[4] Cosmos TOS, dostopno na: http://www.cosmosworldwide.com

[5] TideWorks TOS, dostopno na: http://www.tideworks.com

[6] C# Microsoft MSDN Library, dostopno na:
http://msdn.microsoft.com/en-us/library/z1zx9t92.aspx

[7] Mono Project, odprtokodni projekt C#, dostopen na:
http://www.mono-project.com/Main_Page

[8] XML, informacije, dostopne na:
http://sl.wikipedia.org/wiki/XML

[9] XSD, informacije, dostopne na:
http://gunston.gmu.edu/healthscience/720/XSDschema.asp

[10] XPath, informacije, dostopne na: http://en.wikipedia.org/wiki/XPath

[11] SQL, informacije, dostopne na: http://sl.wikipedia.org/wiki/SQL

[12] (2001) Prof. dr. Marko Bajc, Agilne metodologije razvoja, članek, 2001, IS:
http://bajecm.fri.uni-lj.si/CRP2001/Clanki/Agilne Metodologije Razvoja IS.pdf

[13] (2009) Prof. dr. Marko Bajc, Razvoj Informacijskih Sistemov, prosojnice

[14] XML drevesni prikaz v `textbox` komponenti, rešitev:
http://www.knowdotnet.com/articles/indentxml.html