UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Roman Žurga

IZMENJAVA PODATKOV MED IZVAJALCI ZDRAVSTVENIH STORITEV IN INFORMACIJSKIM SISTEMOM ZAVODA ZA ZDRAVSTVENO ZAVAROVANJE

DIPLOMSKO DELO NA
UNIVERZITETNEM ŠTUDIJU

Mentor:
prof.dr. Viljan Mahnič

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: 01690/2010
Datum: 01.09.2010

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

Kandidat: **ROMAN ŽURGA**

Naslov: **IZMENJAVA PODATKOV MED IZVAJALCI ZDRAVSTVENIH STORITEV IN INFORMACIJSKIM SISTEMOM ZAVODA ZA ZDRAVSTVENO ZAVAROVANJE**

DATA INTERCHANGE BETWEEN HEALTH CARE PROVIDERS AND THE HEALTH INSURANCE INSTITUTE OF SLOVENIA

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Proučite zahteve Zavoda za zdravstveno zavarovanje Slovenije (ZZZS), ki se nanašajo na izmenjavo podatkov med informacijskim sistemom Zavoda in informacijskimi sistemi izvajalcev zdravstvenih storitev. Na tej osnovi realizirajte programsko opremo, ki omogoča dobavitelju medicinskih pripomočkov za vid zajem in posredovanje podatkov v informacijski sistem Zavoda. Uporabnik naj s pomočjo izdelane programske opreme pridobiva podatke o zavarovancih ZZZS, veljavnosti njihovih zavarovanj in predpisanih pripomočkih. V sistem pa naj zapisuje izdaje pripomočkov. Rešitev naj podpira in avtomatizira vse poslovne procese, povezane s poslovanjem z ZZZS.

Mentor:

prof. dr. Viljan Mahnič

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

Zahvala

Zahvaljujem se izr. prof. dr. Viljanu Mahniču za mentorstvo, nasvete in pripombe pri izdelavi diplomskega dela.

Fotooptiki Rio, Barbara Lušič-Knez s.p. iz Izole in posebej gospodu Robertu Knezu za pomoč pri načrtovanju in testiranju rešitve.

Samu Šlosarju, uni.dipl.org, za lektoriranje diplomske naloge.

Posebej se zahvaljujem vsem, ki so mi pomagali in spodbujali v času študija.

Kazalo

Kazalo slik

Kazalo uporabljenih kratic

kratica	*opis*
API	Application Programming Interface
DOM	Document Object Model
DTD	Document Type Definitions
FTP	File Transfer Protocol
GML	Generalized Markup Language
HTML	HyperText Markup Language
HTTP	Hyper Transfer Protocol
ISO	International Organization for Standardization
KZZ	Kartica zdravstvenega zavarovanja
MTP	Medicinsko tehnični pripomoček
OLZZ	On-line zdravstveno zavarovanje
OZZ	Osnovno zdravstveno zavarovanje
PK	Profesionalna kartica zdravstvenega zavarovanja
PZZ	Prostovoljno zdravstveno zavarovanje
RPC	Remote Procedure Call
RUP	Rational Unified Process
SGML	Standard Generalized Markup Language
SMTP	Simple Mail Transfer Protocol
UDDI	Universal Distribution, Discovery, and Interoperability
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WSDL	Web Services Description Language
XML	eXtensible Markup Language
XPath	XML Path Language
XPointer	XML Pointer Language
XSLT	eXtensible Stylesheet Language
ZPZZ	Zavarovalnica za prostovoljna zdravstvena zavarovanja
ZZZS	Zavod za zdravstveno zavarovanje Slovenije

Povzetek

Naloga diplomskega dela je razvoj programske opreme, ki omogoča dobavitelju medicinskih pripomočkov za vid (optika) zajem in posredovanje podatkov v sistem "On-line zdravstveno zavarovanje" Zavoda za zdravstveno zavarovanje Slovenije (ZZZS). Uporabnik s pomočjo izdelane programske opreme pridobiva podatke o zavarovancih ZZZS, veljavnosti njihovih zavarovanj in predpisanih pripomočkih. V sistem zapisuje izdaje pripomočkov. Rešitev podpira in avtomatizira vse poslovne procese, povezane s poslovanjem z ZZZS.

V uvodu je podan opis problema, namen in cilji naloge. Pri izdelavi rešitve so uporabljeni XML jezik (ang. eXtensible Markup Language) in sorodne tehnologije, ki so opisane v drugem poglavju. Naslednje poglavje opisuje sistem "On-line zdravstveno zavarovanje" in način izmenjevanja podatkov med sistemoma ZZZS in dobavitelja. Temu sledi opis razvoja.

Pri razvoju je uporabljena metodologija Rational Unified Process (RUP). Razvoj je opisan po postopkih RUP-a: poslovno modeliranje, zajem zahtev, analiza in načrtovanje, implementacija, testiranje in postavitev. Zaradi velike kompleksnosti poslovnih procesov, katere podpira izdelana rešitev posebej podrobno, sta opisana postopka poslovno modeliranje in zajem zahtev. Poglavje se konča z opisom glavnih funkcionalnosti izdelane programske opreme. Opise lahko uporabljamo kot uporabniška navodila.

V sklepnih ugotovitvah so opisane težave pri razvoju, uspešnost doseganja zastavljenih ciljev, oceno o opravljenem delu in ideje za nadaljnji razvoj.

Ključni pojmi

optika, programska oprema, Zavod za zdravstveno zavarovanje Slovenije, XML, RUP

Abstract

The aim of this diploma thesis is a development of the software which enables acquiring and submitting data to On-line Health Insurance System of Health Insurance Institute of Slovenia (ZZZS) by the supplier of medical technical vision aids (optician).
The customer with developed software acquires data of insured persons, validity of insurances and prescribed aids and submits issuance of aids. The solution supports and automates all business processes in business with ZZZS.

The introduction presents the description of the problem, purpose and aims of the diploma thesis. The solution is based on XML (eXtensible Markup Language) and related technologies, which are described in the second chapter. The following chapter describes the On-line Health Insurance System and exchanging data between the ZZZS and supplier systems. This is followed by a description of the development process.

The methodology of Rational Unified Process (RUP) has been used during the development process. The development is presented according to the RUP processes: business modeling, requirements, analysis and design, implementation, test and deployment. The processes of business modeling and requirement are specifically described in detail due to the high complexity of business processes, which are supported by the developed solution. The chapter is ended with a description of main software functionality, which may be used as the user manuals.

The conclusions are describing development difficulties, success of achieving goals, evaluation of work and ideas for further development.

Keywords

optician, software, Health Insurance Institute of Slovenia, XML, RUP

1. Uvod

1.1 Opis problema

Zavod za zdravstveno zavarovanje Slovenije (ZZZS) je leta 2006 začel razvoj novega sistema "On-line zdravstveno zavarovanje", ki je nastal kot nadgradnja nepovezanega (angl. off-line) sistema "Sistem kartice zdravstvenega zavarovanja". Dosedanji sistem se je od leta 2000 uporabljal za izmenjevanje podatkov med izvajalci zdravstvenih storitev in ZZZS. Izmenjava podatkov je potekala tako, da so uporabniki sistema brali in zapisovali podatke na kartico zdravstvenega zavarovanja. Novi sistem omogoča neposredno, takojšnje izmenjevanje podatkov med: izvajalci zdravstvenih storitev, ZZZS in zavarovalnicami za prostovoljna zdravstvena zavarovanja. Izvajalci med drugim lahko iz sistema pridobijo podatke o zavarovani osebi, zdravstvenem zavarovanju, predpisanih in izdanih zdravilih ter predpisanih in izdanih medicinsko tehničnih pripomočkih (MTP). V sistem zapisujejo podatke o zavarovani osebi, opravljenih storitvah in dobavljenih zdravilih ter pripomočkih.
Z uvedbo naj bi se omogočilo: enostavnejše uresničevanje pravic iz zdravstvenega zavarovanja, večja kakovost storitev in manjši stroški poslovanja vseh udeležencev.

Vsi izvajalci zdravstvenih storitev vključno z dobavitelji MTP, ki želijo poslovati z ZZZS, so bili obvezni do marca 2010. nadgraditi svoje informacijske sisteme tako, da bodo ti omogočali izmenjevanje podatkov s sistemom "On-line zdravstveno zavarovanje" po predpisanih poslovnih pravilih.

Po večletnem uspešnem sodelovanju s Fotooptiko Rio, Barbara Lušič Knez s.p. iz Izole smo se odločili, da bom razvil programsko opremo, ki bo naročniku omogočila uspešno poslovanje z ZZZS.

1.2 Namen in cilji naloge

Namen diplomske naloge je razvoj programske opreme, ki bo omogočala naročniku, dobavitelju medicinsko tehničnih pripomočkov za vid (optik), izmenjevanje podatkov s sistemom "On-line zdravstveno zavarovanje" in obračunavanje opravljenih storitev ter dobavljenih pripomočkih po predpisanih poslovnih pravilih s strani ZZZS.

Danes na trgu obstaja večje število rešitev, ki rešujejo zastavljeni problem. Konkurenčne rešitve nisem analiziral, ker je njihov razvoj potekal istočasno z razvojem moje rešitve.

Cilji naloge so:

- pripraviti sistem, ki bo naročniku reševal problem in hkrati omogočal enostavno uporabo, zvišal kakovost storitev, zmanjšal stroške poslovanja in povečal zadovoljstvo njegovih kupcev,
- izdelati programsko opremo z robustno arhitekturo, ki bo omogočala: dolgoletno uporabo in enostavno ter hitro nadgrajevanje v primerih spremembe poslovnih pravil s strani ZZZS,
- spoznati in uporabiti postopek razvoja po objektni metodologiji RUP (ang. Rational Unified Process)[12, 13],
- podrobno proučiti XML jezik in sorodne tehnologije,
- seznaniti se z načinom izmenjevanja podatkov s sistemom "On-line zdravstveno zavarovanje" in poslovnimi pravili pri poslovanju z ZZZS.

2. XML in sorodne tehnologije

Kot je razvidno iz namena in ciljev diplomske naloge, bo glavni in ključni del rešitve izmenjevanje podatkov s sistemom "On-line zdravstveno zavarovanje". Sistem OLZZ je zasnovan v XML jeziku in v sorodnih tehnologijah. Za uspešno razumevanje delovanja sistema in rešitev problema izmenjevanja podatkov, je potrebno najprej podrobno spoznati te tehnologije in šele po tem začeti s proučevanjem sistema OLZZ in planiranjem projekta.

V prvih podpoglavjih so predstavljeni XML jezik (angl. eXtensible Markup Language) in Document Type Definitions (DTD). S temi se določa slovnica za razred XML dokumentov. Temu sledi opis XML Scheme, jezika za definiranje in opis razreda XML dokumentov, ki danes skoraj popolnoma nadomešča DTD. V sistemu OLZZ se uporablja za definiranje in preverjanje veljavnosti XML sporočil, ki se izmenjujejo med sistemoma izvajalca in OLZZ.

Poglavje se nadaljuje s predstavitvijo jezika za naslavljanje delov XML dokumenta, XPath (XML Path Language) in abstraktnim programskim vmesnikom (API) Document Object Model (DOM), ki omogoča dostop in upravljanje z XML dokumenti.

Ob koncu poglavja so na kratko predstavljene spletne storitve (angl. Web Service), ki omogočajo zajem in posredovanje podatkov.

2.1 XML

XML je kratica za razširljivi označevalni jezik (angl. eXtensible Markup Language), ki je enostaven fleksibilen tekstovni format za predstavitev strukturiranih informacij, podmnožica SGML (Standard Generalized Markup Language) (ISO 8879) [1].

World Wide Web Consortium (W3C) v priporočilu Extensible Markup Language (XML) 1.1 (Second Edition) [2] XML definira kot jezik, ki opisuje razred podatkovnih objektov, imenovanih XML dokumenti, in delno obnašanje računalniških programov, ki jih obdelujejo.

XML dokumenti so zgrajeni iz enot shranjevanja (angl. storage units), imenovanih entitete (angl. entities). Entiteta vsebuje podatek, ki ga XML analizator bodisi razčleni (angl. parse) bodisi ne razčleni**.** Podatek, ki se razčleni, je predstavljen z znaki, ki tvorijo označevanje in znakovni podatek (angl. character data). Z označevanjem določimo obliko shranjenega dokumenta in logično strukturo. XML vsiljuje omejitve pri obliki shranjenega dokumenta in logični strukturi.

Osnovne lastnosti jezika so vidne že iz samega naziva:

- označevalni (angl. Markup)
  - dodajanje pomena podatku katerega označuje.

- razširljivi (angl. eXtensible)
  - dovoljuje dodajanje novih označb.
  - je meta-jezik, omogoča definiranje drugih označevalnih jezikov.

Namenjen je strukturiranju, shranjevanju in prenosu podatkov. XML dokument je zapisan kot tekstovna datoteka, kar mu omogoča, da je neodvisen od računalniške platforme in operacijskih sistemov.

XML je odprti standard konzorcija W3C z javno in vsem dostopno specifikacijo.

Danes je XML zelo priljubljen in razširjen med uporabniki in razvijalci programske opreme. Ker je dokument zapisan kot tekstovna datoteka, ga lahko pregledujemo in spreminjamo v navadnem urejevalniku besedila. Dokument je pregleden in na osnovi značke hitro ugotovimo kaj pomeni kateri podatek. Obstaja veliko število različnih orodij in programskih knjižnic za delo z XML. Izdelava programske opreme, ki uporablja XML, je zelo preprosta in hitra.
Z nadgradnjo XML-zasnovanega formata zapisa podatkov, pri katerem se ohrani združljivost za nazaj, ni nujna sprememba obstoječe programske opreme. Večina razvijalcev se odloča uporabljati XML za izvoz in uvoz podatkov, ker ta omogoča integracijo aplikacij. Podatki se (skoraj) samodejno prenašajo iz ene aplikacije v drugo, namesto da bi se večkrat ročno vnašali.

Nekaj sto jezikov je zasnovano na osnovi XML vključno s SOAP, RSS, Atom, XHTML, ebXML, XQuery, XLink, SVG, SMIL. Najbolj razširjeni pisarniški paketi Microsoft Office (Office Open XML), OpenOffice.org (OpenDocument) in Apple's iWork uporabljajo XML-zasnovan format zapisa podatkov [3].

XML zaradi redundantne sintakse ni primeren za shranjevanje zvočnega in video zapisa.

2.1.1 Nastanek in razvoj

Razvoj označevalnih jezikov (predhodnikov XML-a), se je začel leta 1969 z jezikom Generalized Markup Language (GML). GML je bil namenjen integraciji odvetniških pisarn in je omogočal urejanje besedila, formatiranje, iskanje in deljenje dokumentov.

Ključen mejnik pri razvoju označevalnih jezikov je osnutek standarda SGML (Standardized Generalized Markup Language), ki je bil objavljen leta 1980 in razglašen kot standard 8879 [4] leta 1986. Uporabljali so ga vojska, državna uprava, vesoljska agencija in izdelovalci letal, ki so potrebovali sistem za upravljanje obsežne tehnične dokumentacije. Z razliko od njegovih slabosti obsežnosti, zapletenosti in redundantnosti, njegov največji uspeh predstavlja izpeljanka HTML (HyperText Markup Language) in podmnožica XML.

XML so leta 1996 razvili Jon Bosak in sodelavci znotraj XML Working Group pod okriljem World Wide Web Consortium (W3C), ki ga je standardiziral februarja 1998. Vsebuje glavne značilnosti in pomembnejše zmogljivosti SGML-ja.

Cilji pri oblikovanju XML so bili:

- XML naj bo enostaven za uporabo preko interneta;
- XML naj podpira velika množica različnih programskih aplikacij;
- XML naj bo kompatibilen s SGML;
- izdelava programov, ki obdelujejo XML dokumente, naj bo enostavna;
- število neobveznih funkcij (angl. optional features) v XML naj bo minimalno, idealno nič;
- XML dokumenti naj bodo človeško-berljivi in precej jasni;
- oblikovanje XML je potrebno hitro dokončati;
- specifikacija XML naj bo formalna in jedrnata;
- XML dokumenti naj se dajo preprosto ustvariti;
- jedrnatost v XML označevanju ima najmanjšo pomembnost [2].

2.1.2 Sintaksa

XML dokument ima logično in fizično strukturo. Fizična strukturo dokumenta predstavljajo enote, imenovane entitete. Dokument se začne z entiteto dokumenta (angl. document entity). Če pogledamo na dokument kot na drevo, entiteta dokumenta predstavlja koren drevesa. Logična struktura je sestavljena iz deklaracij, elementov, opomb, referenc znakov in ukazov za obdelavo.

2.1.2.1 Dobro oblikovan XML dokument

V priporočilu združenja W3C, "Extensible Markup Language (XML) 1.1" [2] najdemo naslednjo definicijo, ki opisuje, kdaj je XML dokument dobro oblikovan.

Tekstualni dokument je *dobro-oblikovan XML* dokument, če:

1) kot celota predstavlja dokument, označen s produkcijami (angl. production labeled document).
   To pogojuje naslednje:
   - Dokument vsebuje enega ali več elementov.
   - Obstaja natanko en element, imenovan koren (angl. root) ali element dokumenta. Niti en del korena se ne nahaja znotraj kateregakoli drugega elementa.
   - Za vse ostale elemente velja, če je začetna značka (angl. start-tab) vsebovana v nekem elementu, potem je končna značka (angl. end-tab) vsebovana v tem elementu. Elementi so pravilno ugnezdeni.

2) upošteva vse omejitve dobre oblikovanosti, ki jih predpisuje specifikacija.

3) vsaka razčlenjena entiteta, ki je vključena neposredno ali posredno v dokument je dobro oblikovana.

Dokument je sestavljen iz treh delov:

- Uvoda (angl. prolog) – v katerem se nahaja: deklaracija XML, s katero podamo navodila XML analizatorju o uporabljeni verziji, uporabljenem kodiranju Unicode znakov ter samostojnosti dokumenta; ukazi za obdelavo (PI) in komentarji.
- Telesa – predstavlja vsebino dokumenta. Začne se z začetno in konča s končno značko entitete dokumenta.
- Zaključka (angl. epilog) – vsebuje zaključne komentarje in ukaze za obdelavo.

2.1.2.2 Znaki

Osnovna enota teksta je znak. V XML dokumentu so dovoljeni Unicode in ISO/IEC 10646 znaki, ki so lahko predstavljeni z UTF-8 ali UTF-16.

Pri XML dokumentih, ki so shranjeni v datoteke, je vsebina, zaradi boljše preglednosti pri urejanju in pregledovanju besedila, razdeljena na vrstice. Vsaka vrstica se konča z znakom (*carriage return* (CR), *next line* (NL), #x2028) ali zaporedjem znakov (CR+LF, CR+NL), ki predstavljajo oznako za konec vrstice.

Da bi dosegli boljšo berljivost dokumenta, znotraj in zunaj označevanja uporabljamo presledek. Presledek je definiran kot znak ali poljubno zaporedje znakov: " " (#20), horizontalni tabulator (HT), CR ali line feed (LF).

2.1.2.3 Označevanje in znakovni podatki

Označevanje (angl. Markup) naredimo z uporabo naslednjih oznak:

- *začetna značka (angl. start-tag)*
  Začetek vsakega ne-praznega XML elementa je označen z začetno značko.

  Vsaka značka vsebuje ime elementa, ki je predstavljen z znakovnim nizom črk, številk, posebnih znakov ("_",".", "-",":") in presledkov. Pri določanju imena elementa obstajata samo dve omejitvi. Ime se ne sme začeti z:
  - znakovnim nizom (('X'|'x')('M'|'m')('L'|'l')), ker so takšna imena rezervirana za standardizacijo v specifikacijah XML,
  - številkami [0-9], "." (piko), "-", .

  Uporaba znaka ":" (dvopičje) je dovoljena na začetku in znotraj imena, ampak se ne priporoča razen, če gre za *ime* iz imenskega prostora.

  Pri določanju imena moramo upoštevati dejstvo, da XML razlikuje velike in majhne črke. Dolžina imena ni omejena.

  Primer: *<zavarovanec zzzstst="0015"> ...*

- *končna značka (angl. end-tag)*
  Na koncu vsakega elementa se mora nahajati končna značka, ki se začne z znakovnim nizom "</" , kateremu sledi ime elementa in znak ">".

  Primer: *... </zavarovanec>*

- *značka praznega elementa (angl. empty element-tag)*
  Element, ki nima vsebine, se imenuje prazni element.

  Primer: *<zavarovanec zzzst="0015"/>*

- *referenca entitete (angl. entity reference)*
  Sklicujejo se na vsebino entitete z imenom, ki je navedena v referenci.
  Na referenco entitete lahko gledamo kot na zamenjavo, analizator pri analizi XML dokumenta referenco zamenja z vsebino entitete.

- *referenca znaka (angl. charachter reference)*
  Referenco znaka je smiselno uporabljati pri znakih, katerih ne moremo vnesti z napravami za vnos podatkov (npr. tipkovnica) ali jih ("&", "<", ">", (') , (")) ni dovoljeno zapisati kot črko na nekaterih mestih (npr. "<" zapišemo kot &#x3C ali &lt).

  Primer: *znak "<" zapišemo kot <*

  Reference znaka se ne smejo nahajati znotraj imen elementov ali atributov. Dovoljena je uporaba znotraj vrednosti atributa.

- *komentar (angl. comment)*
  Komentarji se lahko nahajajo kjerkoli znotraj dokumenta, razen znotraj drugih označevanj in pred deklaracijo XML.
  Primer: <!-- opomba -->

- *znakovna sekcija (angl. CDATA section)*
  Znotraj CDATA sekcije se znakovni niz ne razčlenjuje. Uporablja se na tistih mestih v dokumentu, kjer bi analizator znakovni niz zaznal kot označevanje. Gnezdenje CDATA sekcij ni dovoljeno.
  Primer: <!CDATA[<ime> ana </ime>]]>

- *deklaracija tipov dokumenta (angl. document type declaration) (DTD)*
  Vsebuje ali kaže na mesto, v katerem se nahaja definicija gramatike za razred dokumenta.

- *ukaz za obdelavo (angl. processing instruction) (PI)*
  Ti omogočajo, da dokument vsebuje ukaze za aplikacije.
  Ukazi za obdelavo se lahko nahajajo kjerkoli znotraj dokumenta, vendar ne znotraj drugih označevanj in pred deklaracijo XML.

- *deklaracija XML (angl. XML declaration)*
  Uvod (angl. prolog) XML dokumenta se začne z deklaracijo XML, v kateri določimo verzijo, naziv kodiranja Unicode znakov in samostojnost dokumenta. Priporočilo za XML verzijo 1.1 predpisuje, da se mora dokument za razliko od verzije 1.0 obvezno začeti z deklaracijo XML.
  Primer: <?xml version=1.1 encoding='UTF-8' standalone="no"?>

  Atributi deklaracije XML-a:
  - *version* - Iz vrednosti atributa *version* XML analizator pridobi informacijo o XML verziji, uporabljeni v dokumentu. Če analizator ne podpira verzije, navedene v deklaraciji, ne nadaljuje z delom in vrne napako.
  - *encoding* - Vrednost predstavlja naziv kodiranja, s katerim je kodirana množica Unicode znakov. XML analizator na osnovi prvih dveh bitov določi, katero kodiranje je uporabljeno; UTF-8 ali UTF-16. Če je v dokumentu uporabljeno kakšno drugo kodiranje znakov, moramo imeti v deklaraciji obvezno atribut encoding z vrednostjo uporabljenega kodiranja. Ni obvezen atribut v deklaraciji.
  - *standalone*
    "yes" - Dokument nima definicije v zunanji podmnožici DTD ali parametrični entiteti (notranji ali zunanji).
    "no" - Dokument ima lahko definicijo v zunanji podmnožici DTD ali parametrični entiteti (notranji ali zunanji). (Privzeta vrednost).

- *deklaracija besedila (angl. text declaration)*
  Entitete se lahko nahajajo zunaj dokumenta (angl. external parsed entity) v drugem objektu za shranjevanje XML dokumentov oz. datoteki.
  Deklaracija besedila je obvezna na začetku objekta oz. datoteke, če znaki niso predstavljeni z UTF-8 ali UTF-16.
  Primer: <?xml encoding='ISO-8859-1' ?>

- *presledek (angl. white space)*

2.1.2.4 Element

Vsak XML dokument ima najmanj en element. Element omejujeta začetna in končna značka ali pri praznem elementu značka za prazni element. Tekst med značkami predstavlja vsebino (angl. content) elementa.

Vsebina ni obvezna, obstajajo elementi brez vsebine, prazni elementi. Prazni elementi imajo končno značko takoj za začetno značko ali samo značko za prazen element.
Element (razen pri praznih elementih) lahko poleg znakovnih podatkov, vsebuje druge elemente, reference, znakovne sekcije, ukaze za obdelavo in komentarje.

Primer:

```
<zavarovanec>Anton Wolf<zavarovanec/>
<opomba />
<zdravnik>
        <ime>Janez<ime/>
        <priimek>Štimec<priimek/>
        <naslov>Mirtoviči 5<naslov/>
        <mesto>Osilnica<mesto/>
<zdravnik/>
```

2.1.2.5 Atribut

Razen za shranjevanje podatkov v elementu, atribute uporabljamo za opis lastnosti elementa oz. dosežemo razlikovanje med elementi istega tipa (z istim imenom). Element, ki je opisan z atributom, v dokumentu poiščemo z enim ukazom, namesto da preberemo vse elemente dokumenta in na osnovi vsebine elementa najdemo iskani element.

Vsak element ima lahko neomejeno število atributov. Edina omejitev je, da morajo atributi znotraj elementa imeti različna imena.
Vrednost atributa je znakovni niz, referenca ali poljubno zaporedje teh.

Primer:

V dokumentu imamo pogostokrat večje število elementov z istim imenom. Tako v dokumentu naprimer shranjujemo seznam uporabnikov. Za vsakega uporabnika shranjujemo ime, primek, datum rojstva in poklic. Elemente razlikujemo na osnovi atributov *ime* in *priimek*.

```
<uporabnik ime='Ivan' priimek='Novak'>
        <rojen>1954-06-09<rojen/>
        <poklic>trgovec<poklic/>
<uporabnik/>
<uporabnik ime='Ana' priimek='Javornik'>
        <rojen>1980-01-12<rojen/>
        <poklic>študent<poklic/>
<uporabnik/>
<uporabnik ime='Sandra' priimek='Javornik'>
        <rojen>1971-06-17<rojen/>
        <poklic>optik<poklic/>
<uporabnik/>
```

2.1.2.6 Entiteta

Entiteto definiramo v DTD (Document Type Definitions). Vsaka entiteta ima vrednost, katero predstavlja besedilo. Znotraj dokumenta lahko postavimo referenco entitete in s tem dosežemo, da XML analizator vsako referenco entitete zamenja z vrednostjo entitete oz. besedilom.
Entitete delimo na splošne in parametrične, notranje in zunanje ter razčlenjene in nerazčlenjene.
Entite in njihova uporaba sta podrobno opisana v podpoglavju 2.2.3.

Primer: *<!ENTITY naslov "Ljubljanska 15, 1000 Ljubljana">*

2.1.3 Imenski prostor (angl. Namespace)

Imenski prostori imajo v XML dva namena: [5]

1. Razlikovanje med elementi in atributi iz različnih slovarjev (jezikov), ki imajo isto ime in različen pomen.
2. Združevanje vseh (povezanih) elementov in atributov iz ene XML aplikacije.

Imenski prostor je določen z referenco URI (Uniform Resource Identifier). Ni obvezno, da referenca URI imenskega prostora dejansko kaže na dokument ali stran. Elementi in atributi, ki so pridruženi isti referenci URI, pripadajo istemu imenskemu prostoru.
Razširjeno ime (angl. expanded name) je sestavljeno od imena imenskega prostora določenega z URI in lokalnega imena. Referenca URI ni primerna za uporabo v XML dokumentih, ker je ta običajno dolga in vsebujejo znake, kateri niso dovoljeni v XML dokumentih. Namesto razširjenih imen uporabljamo kvalificirano ime (angl. qualified name).

V kvalificiranem imenu (*prefiks:lokalno_ime)* mora biti prefiks imenskega prostora pridružen referenci URI imenskega prostora. Lokalno ime določa posamezni element ali atribut znotraj imenskega prostora.

Imena atributov za deklaracijo imenskih prostorov ustvarimo z družino rezerviranih atributov[6]:

- *xmlsIns:prefiks* – Vsi elementi in atributi znotraj elementa, v katerem je deklariran imenski prostor z atributom *xmlsIns:prefiks* in imajo v kvalificiranem imenu *prefiks*, pripadajo deklariranemu imenskemu prostoru. Atributu *xmlsIns:*prefiks je pridružena referenca URI imenskega prostora.

- *xmlsIns* – Atribut se uporablja za deklariranje privzetega imenskega prostora. Vsi elementi (otroci) znotraj elementa, v katerem je definiran in nimajo prefiksa nekega drugega imenskega prostora, pripadajo privzetemu imenskemu prostoru.

Primer:

```
<rio:ocala xmlns:rio="http://www.rio.si/ocala">
        <rio:sifra>999871</rio:sifra>
        <rio:kolicina>10</rio:kolicina>
</rio:ocala>
```

2.2 DTD (Document Type Definitions)

Slovnico za razred XML dokumentov določimo z Document Type Definitions (DTD), s katerim povemo, katere lastnosti imajo posamezne značke in v kakšnih so lahko medsebojnih odnosih.

Če je XML dokument beseda, ki pripada jeziku, definiranemu z DTD ($w \in L(DTD)$), in je dokument "dobro oblikovan", potem je dokument veljaven. Analizatorji veljavnosti (angl. validity parser) preverjajo veljavnost dokumenta glede na DTD. Nekateri analizatorji, če XML dokument ni veljaven, dokument zavrnejo, drugi zavrnejo samo del, ki ni veljaven.

DTD določimo znotraj značke DOCTYPE. Sama definicija se lahko nahaja: znotraj DOCTYPE (notranja podmnožica DTD), zunaj dokumenta (zunanja podmnožica DTD) ali se prvi del nahaja v zunanji in drugi del v notranji podmnožici.

2.2.1 Definicija elementa

V veljavnem dokumentu mora biti definiran vsak element. Z definicijo določimo ime in specifikacijo vsebine elementa. Vsebina se definira z znakovnimi podatki in otroci (elementi), njihovim številom in zaporedjem. Element ne sme imeti več kot eno definicijo.

Specifikacija vsebine elementa je lahko definirana kot:

- EMPTY - Element je prazen element.
- ANY - Vsebina elementa ni določena.
- *Mixed* - Vsebina elementa je poljubno zaporedje znakovnih podatkov in/ali elementov, katerih zaporedje in njihovo število ni določeno.
- *children* - Element vsebuje enega ali več elementov, v poljubnem ali določenem zaporedju. Znakovni podatki niso dovoljeni.

Primeri:

```
<!ELEMENT tel (#PCDATA)>        ; element vsebuje samo znakovne podatke
<!ELEMENT ime (tel)>            ; element ima obvezno enega otroka tel
```

<!ELEMENT zaposlenec (ime, tel|gsm)> ; obvezna sta otroka *ime* in bodisi *tel* bodisi *gsm*. Zaporedje pojavitve otrokov, v elementu je določeno.

```
<zaposlenec>                ali    <zaposlenec>
    <ime></ime>                        <ime></ime>
    <tel></tel>                        <gsm></gsm>
</zaposlenec>                      </zaposlenec>
```

<!ELEMENT zaposlenec (ime, tel)>* ; element lahko vsebuje poleg enega elementa *ime* neomejeno število elementov *tel*.

2.2.2 Definicija atributa

V veljavnem dokumentu moramo za vsak element definirati vse atribute. Atribute definiramo z ATTLIST. ATTLIST definira naziv, tip in privzeto vrednost atributa. Z eno definicijo ATTLIST lahko definiramo več atributov enega elementa.

Vrednost atributa lahko predstavlja poljuben niz znakov v oklepajih. "Dobra oblikovanost" XML dokumenta vsiljuje samo omejitev, da v tem nizu znakov moramo vsako pojavitev znakov '<' , '&' in narekovajih (') ali (") (samo tiste s katerimi je vrednost atributa postavljena v narekovaje) zamenjati z ustrezno referenco znaka ali nadomestnim znakovnim nizom. Definicija atributa postavlja dodatno omejitev, tip atributa.

Tipi atributa:

- CDATA – vrednost atributa je niz znakov.
- ID – za elemente, ki imajo definiran argument tega tipa, velja omejitev, da v dokumentu ne moreta obstajati dva elementa, katerih vrednost definiranega argumenta je enaka. Dovoljene so vrednosti atributa, katere ustrezajo omejitvam za XML ime (*Name*). Element ima lahko največ en argument tega tipa.
- IDREF – vrednost atributa je enaka vrednosti nekega atributa tipa ID. Na atribut tega tipa lahko gledamo kot na kazalec, ki kaže na neki drug element.

  Primer:

  ```
  <zavarovanec ime_priimek="Janez Kovač" enotazzzs="Z15099007"/>
  <zavarovanec ime_priimek="Anton Benčič" enotazzzs="Z15099008"/>
  <zavarovanec ime_priimek="Mirko Žvab" enotazzzs="Z15099007"/>
  <enota_zzzs st="Z15099007">
      <mesto>Koper</mesto>
  </enota_zzzs st>
  <enota_zzzs st="Z15099008">
      <mesto>Izola</mesto>
  </enota_zzzs st>
  ```

- IDREFS – podobna je IDREF z razliko, da ta tip predstavlja zaporedje več ID vrednosti, ločenih s presledkom.

  Primer:

  ```
  <območna_enota='Koper" enotazzzs="Z15099007 Z15099008"/>
  ```

- ENTITY – vrednost atributa je ime zunanje entitete definirane znotraj DTD, katera se ne razčlenjuje (angl. external unparased entity).
- ENTITIES – zaporedje ENTITY ločeno s presledki.
- NMTOKEN – vrednost atributa je niz znakov, kateri je sestavljen iz znakov, dovoljenih v XML imenu za začetnim znakom.
- NMTOKENS – zaporedje NMTOKEN ločeno s presledki.
- notacija *(NotationType)* – vrednost atributa je ime notacije definirane v DTD.
  Element sme imeti največ en atribut tipa notacije. Prazen element ne more imeti atributa tega tipa.
- naštevni tip (*EnumeratedType*) – dovoljene vrednost atributa, se določijo z naštevanjem vseh vrednosti.

  Primer:

  ```
  <! ATTLIST zavarovanec kategorija ( 1 | 2 | 13 | 14| 17 | 21) #REQUIRED>
  ```

  Dovoljene vrednosti atributa morajo ustrezati definiciji NMTOKEN in se ne smejo podvajati.

V definiciji atributa lahko definiramo ali je atribut obvezen in njegovo privzeto vrednost.

- #REQUIRED – atribut je obvezen pri vseh elementih.
- #IMPLIED – atribut ni obvezen.
- #FIXED – atribut ni obvezen. Vrednost atributa je nespremenljiva. Če je v elementu podana vrednost atributa, mora biti enaka definirani prevzeti vrednosti. Če v elementu ni podane vrednosti atributa, ima ta prevzeto vrednost.

2.2.3 Definicija entitete

V definiciji entitete definiramo njeno ime in vrednost. Vrednost entitete lahko vsebuje besedilo, označevanje in reference na druge entitete. Če vsebuje označevanje, mora biti vrednost "dobro oblikovana". Znotraj vrednosti entitete se ne smejo nahajati narekovaji znotraj katerih je vrednost postavljena v definiciji.

Glede na to, kje se nahaja vrednost entitete, te delimo na:

- *notranje entitete* (angl. internal entity) - Vrednost entitete se nahaja v definiciji entitete.
- *zunanje entitete* (angl. external entity) - Vrednost entitete se nahaja v fizično ločenem objektu za shranjevanje (npr. zunanji datoteki).

Glede na namen razlikujemo:

- *splošno entiteto* (angl. general entity)

Splošno entiteto oz. referenco entitete uporabljamo zunaj definicij (DTD). Edina izjema je definicija privzete vrednosti atributa. Pri entitetah, ki se razčlenjujejo, XML analizator zamenja referenco entitete z vrednostjo entitete (znakovni niz). Pri zunanjih entitetah zamenja referenco z vsebino, ki se nahaja v viru URI. Poseben namen imajo zunanje entitete, ki se ne razčlenjujejo. Njihov namen je sporočiti aplikaciji, kje so podatki in kakšnega tipa so.

Primer: Zamenjava znakovnega niza v vsebini elementa.

```
<!ENTITY postopek "Standard">
<okvir>Okvir-&postopek;</okvir>   ; vsebina =  'Okvir-Standard'
```

- *parametrično entiteto* (angl. parameter entity)

Parametrična entiteta se razčlenjujejo ne glede na to ali so zunanje ali notranje. Reference parametrične entitete se uporabljajo izključno v definicijah (DTD), ko želimo v notranjo definicijo vključiti definicije shranjene zunaj dokumenta (zunanja podmnožica) ali uporabiti vrednost entitete znotraj večjega števila definicij.

Primer:

```
<!ENTITY % naslov_stranke "naslov,mesto,posta">
<!ELEMENT (ime, priimek, %naslov;)>
```

2.3 XML Schema

Namen XML Scheme je definiranje in opis razreda XML dokumentov z uporabo komponent sheme, ki omejujejo in opisujejo pomen, uporabo in razmerja njihovih sestavnih delov: podatkovnih tipov, elementov in njihove vsebine ter atributov in njihove vrednosti. Omogoča določanje privzete vsebine elementov ter privzete vrednosti atributov [7].
Z XML Schemo definiramo slovar za razred dokumentov, na osnovi katerega se določa, če beseda iz primerka dokumenta pripada jeziku, definiranemu s shemo oz. veljavnost primerka XML dokumenta.

World Wide Web Consortium (W3C), ki skrbi za razvoj XML Scheme, je leta 2001 odobril prvo priporočilo za XML Schemo. Drugo, popravljeno in razširjeno priporočilo je objavljeno leta 2004.

2.3.1 Razlike med DTD in XML Schemo

XML Schema je alternativa, ki nadomešča DTD (*Data Type Definition*).

Prednosti XML Scheme nad DTD so:

- *napisana v XML*. Za razliko od DTD, ki ni pisana v standardni sintaksi XML, XML Schema je zapisna v XML sintaksi in za XML predstavitev komponent sheme uporablja slovar, definiran v imenskem prostoru *http://www.w3.org/2001/XMLSchema*. Kot prefiks imenskega prostora se običajno uporablja *xsd:* .

- *razširljivost.* Iz samega dejstva, da so napisane v XML jeziku, izhaja enostavna razširljivost z novimi funkcionalnostmi. Ponovna uporaba sheme znotraj drugih shem. Uporaba več shem znotraj enega dokumenta.

- *podpira podatkovne tipe*. Razen osnovnih podatkovnih tipov (datum, znakovni niz, celo število, realna števila, ...) omogoča definiranje enostavnih in kompleksnih tipov.
  Podpira dedovanje oz. izpeljavo novih tipov z omejitvijo in razširitvijo.

- *boljša funkcionalnost*. Vsebine elementov in vrednosti atributov omejimo, razen s tipom, tudi z eksplicitno določeno zalogo vrednosti. Določanje števila in zaporedja pojavitve otrokov znotraj elementa.

- *podpira imenske prostore*. Ker je DTD nastal pred uvedbo imenskih prostorov, ne podpira direktno imenskih prostorov. Pri definiciji elementa ali atributa v DTD moramo podati kvalificirano ime in ne samo lokalno ime. S tem je DTD omejen na uporabo samo znotraj enega imenskega prostora.
  Na shemo lahko gledamo kot na zbirko definicij tipov in elementov katerih ime pripada posameznemu imenskemu prostoru, ki ga imenujemo *ciljni imenski prostor* (angl. target namespace). Ciljni imenski prostor omogoča razlikovanje med definicijami iz različnih imenskih prostorov. Če ciljni imenski prostor ni deklariran, shemo lahko uporabimo za definicijo v več različnih imenskih prostorih ali jo vključimo v druge sheme.

Edina prednost DTD je možnost definiranja entitete in uporaba reference entitete.

2.3.2 Struktura dokumenta XML Schema

Abstraktni podatkovni model XML Scheme je zgrajen iz naslednjih komponent, razdeljenih v tri skupine [7]:

- *primarne komponente*
  - definicije enostavnih tipov (angl. Simple type)
  - definicije kompleksnih tipov (angl. Complex type)
  - definicije elementov
  - definicije atributov
- *sekundarne komponente*
  - definicije skupinskega modela (angl. Model group definitions)
  - definicije skupine atributov (angl. Attribute group definitions)
  - definicije notacij (angl. Notation definitions)
  - definicije omejitev identitet (angl. Identity-constraint definitions)
- *pomožne komponente* – niso samostojne komponente, uporabljajo se znotraj primarnih in sekundarnih komponent.
  - pripombe (angl. Annotations)
  - skupinski modeli (angl. Model groups)
  - omejitve števila pojavitev (angl. Particles)
  - nadomestniki (angl. Wildcards)
  - uporabe atributov (angl. Attribute Uses)

Shema je lahko napisana v enem ali več ločenih dokumentih, ki vsebujejo eno ali več <*schema*> značk. Korenski element vsake sheme je značka <schema>.

2.3.2.1 Veljavnost primerka dokumenta

Da bi se preverjala veljavnost nekega elementa in njegovih potomcev, moramo elementu pridružiti shemo. Elementu v primerku dokumenta pridružimo shemo tako, da v element postavimo atribut *xsi:SchemaLocation* ali *xsi:noNamespaceSchemaLocation*.
Vrednost atributa *xsi:SchemaLocation* vsebuje enega ali več parov referenc URI. Prva referenca v paru predstavlja ciljni imenski prostor sheme. V drugi se nahaja lokacija sheme (dokumenta) za ta imenski prostor. Pri shemah, ki nimajo določenega ciljnega imenskega prostora, se uporablja se atribut *xsi:noNamespaceSchemaLocation*. Ta kot vrednost ima lokacijo dokumenta sheme.

Primer:

```
<ocala   xmlns="http://www.rio.si/ocala"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.rio.si/ocala http://www.rio.si/ocala.xsd">
    ....
</ocala>
```

2.3.3 Definicija tipa

V XML Schemi obstajata dva tipa:

- enostavni tip (angl. Simple Type)
- kompleksni tip (angl. Complex Type)

Osnovna razlika je v tem, da za razliko od enostavnega tipa, kompleksni tip lahko vsebuje elemente in atribute.
Tipe definiramo kot *imenovane* ali *neimenovane*. Imenovani tipi imajo svoje ime. Uporabljajo se v definiciji elementa in atributa za določanje tipa elementa ali atributa. Iz njih lahko izpeljujemo nove tipe. Neimenovani tipi so tipi brez imena. Definirajo se in uporabljajo znotraj definicije: elementa, atributa ali drugega tipa za definiranje tipa le tega.

2.3.3.1 Enostavni tip

Enostavni tip izpeljemo iz kateregakoli drugega vgrajenega (angl. built-in) ali izpeljanega enostavnega tipa z uporabo omejitvije (angl. restriction). Tip, iz katerega je izpeljan novi tip, imenujemo *osnovni tip*. Izpeljani tip je podmnožica osnovnega tipa. Vsi enostavni tipi so podmnožice tipa *anySimpleType*.

Primer:

```
<xsd:simpleType name="myInt">
    <xsd:restriction base="xsd:integer">
        <xsd:minInclusive value="100">
        <xsd:maxInclusive value="900">
    </xsd:restriction>
</xsd:simpleType name>
```

Enostavne tipe delimo na:

- *atome* (angl. atomic types) – Nedeljiv tip, ne moremo ga razdeliti tako, da bi posamezni del ali deli imeli neki pomen. (primer: NMTOKEN vrednost SI je oznaka za Slovenijo, posamezne črke S in I nimajo same zase nobenega pomena.)

- *sezname* (angl. list type) – Predstavlja zaporedje vrednosti nekega tipa, tipa atom. XML Shema ima tri vgrajene tipe seznamov: NMTOKENS, IDREFS in ENTITIES. Elementi seznama so lahko tipa, podanega v atributu *itemType* ali se definirajo znotraj elementa sheme *list*.
  Omejitve (angl. facets): *length*, *minLength*, *maxLength*, *pattern* in *enumeration* lahko uporabimo pri definiranju novega tipa, izvedenega iz tipa seznam.

  Primer:

  ```
  <xsd:simpleType name="myIntListType">
      <xsd:list ItemType="xsd:integer">
  </xsd:simpleType name>

  <myIntListType>7 13 17 21</myIntListType>
  ```

- *unije* (angl. union types) – Tip unija je tip, katerega je vsaki primerek hkrati tudi primerek enega od tipov iz množice tipov navedene v definiciji unije. Množica tipov unije lahko vsebuje tipe atom in seznam. Na tipu unija so dovoljene omejitve *pattern* in *enumeration*.

Pri definiciji enostavnega tipa z atributom *fixed* lahko določimo, da se posamezna omejitev ne sme spreminjati pri izpeljavah v katerih tip nastopa kot osnovni tip.

2.3.3.2 Kompleksni tip

Kompleksni tip deklariramo z XML Schema elementom *complexType*. Vsebuje definicije elementov in atributov. Kompleksne tipe glede na vsebino delimo: na tipe z enostavno in tipe s kompleksno vsebino.

Izpeljujemo jih iz nekega osnovnega tipa:

- z omejevanjem (angl. restriction) – Izpeljani tip je podmnožica osnovnega tipa.
- z razširitvijo (angl. extension) – Osnovnega tipa je podmnožica izpeljanega tipa.

Pri kompleksnih tipih s kompleksno vsebino lahko določimo skupinski model oz. zaporedje podelementov v vsebini elementa.

Skupinski model zgradimo iz enega od naslednjih gradnikov:

- *all* – konjunkcija. Vsebuje samo elemente. V primerku dokumenta se vsak element iz skupine pojavi kvečjemu enkrat. Zaporedje elementov je poljubno. Komponenta znotraj katere je definirana konjunkcija, ne sme imeti niti enega drugega otroka.
- *choice* – disjunkcija. Dovoljen natanko en podelement (otrok) gradnika *choice*.
- *sequence* – obvezno zaporedje. V primerku dokumenta elementi si morajo slediti v istem vrstnem redu, kot so definirani.

2.3.4 Definicija elementa

Elementi, ki imajo atribute ali vsebujejo podelemente (otroke) so kompleksnega tipa, vsi drugi so enostavnega tipa. Element je definiran kot globalni element, če je definiran kot otrok elementa *schema*.

Vsebina elementa je lahko:

- *mešana* (angl. mixed) – Pri mešani vsebini se besedilo lahko nahaja med podelementi.
- *prazna* (angl. empty) – Prazen element nima vsebine.
- tipa *anyType* – Če elementu ni eksplicitno definiran tip, potem je tipa *anyType*. Elementa tipa *anyType* nima nobene omejitve glede vsebine.

2.3.5 Definicija atributa

Atributi so vedno samo enostavnega tipa (*simpleType)*. V XML Schemi lahko definiramo lokalne in globalne atribute. Atribut je definiran kot globalni atribut, če je definiran kot otrok elementa *schema*. Lokalni atributi se definirajo znotraj definicije kompleksnega tipa in definicije skupine atributov.

V definiciji atributa se lahko določi ali mora imeti element v primerku dokumenta eksplicitno podan atribut (*required, optional, prohibited)* ter prevzeta (*default)* ali nespremenljiva (*fixed)* vrednost atributa.

- *required* – Atribut je obvezen.
- *optional* – Neobvezen atribut. Privzeta vrednost atributa *use*.
- *prohibited* – Ne sme se uporabljati definirani atribut.

2.4 XPATH

XPath (XML Path Language) je jezik za naslavljanje delov XML dokumenta [8]. Ni XML jezik. Uporablja se v celoti ali delno tudi znotraj XSLT (EXtensible Stylesheet Language), XPointer (XML Pointer Language), XML Schema in drugih jezikov. XPath je povpraševalni jezik.
Osnovni gradnik XPath-a je izraz. Povpraševanje po izrazu iz XPath-a vrne kot rezultat objekt, ki je osnovnega XPath tipa. Obstajajo štiri osnovni tipi: množica vozlišč, logični tip (angl. boolean), števila predstavljena s plavajočo vejico in znakovni niz. Podpira uporabo imenskih prostorov.

World Wide Web Consortium (W3C) je leta 1999 odobril prvo priporočilo za XPath 1.0. Leta 2007 je bilo odobreno priporočilo za verzijo XPath 2.0. Verzija 2.0 se precej razlikuje od prejšnje verzije in ni popolnoma združljiva z njo. Spremenjen je bil podatkovni model. Od tedaj osnovno vrednost predstavlja sekvenca. Vozlišče je sekvenca z dolžino ena. Dodanih je večje število tipov, funkcij in operatorjev. XPath 2.0 je v bistvu podmnožica jezika XQuery 1.0.
V nadaljevanju so predstavljene osnove jezika XPath s poudarkom na verziji 1.0, ki je v širši uporabi kot verzija 2.0.

2.4.1 XPath drevo

XPath si XML dokument predstavlja kot drevo vozlišč. Obstajata dve ureditvi vozlišč; dokumentna ureditev (angl. document order) in obratna dokumentna ureditev (angl. reverse document order). Dokumentna ureditev je odvisna od zaporedja pojavljanja elementov v XML dokumentu. Prvo vozlišče je korensko vozlišče. Vozlišča elementov se nahajajo pred njihovim otrokom. Vozlišča, v katerih so shranjeni atributi in imenski prostori elementa, se nahajajo pred njegovim otrokom. Vozlišča atributov se nahajajo pred vozlišči imenskih prostorov. Ureditev vozlišč atributov in imenskih prostorov znotraj njih samih je odvisna od implementacije.
Za vsako vozlišče se določi *string-value.* Odvisno od tipa vozlišča *string-value* je del vozlišča ali se izračuna iz *string-value* potomcev. Nekatera vozlišča imajo razširjeno ime poimenovano *expanded-name*.

Drevo ima lahko sedem različnih tipov vozlišč:

- *korensko vozlišče (angl. root node)* – V drevesu obstaja samo eno. Korenski element XML dokumenta je shranjen v vozlišču tipa vozlišče elementa, ki je otrok korenskega vozlišča. Ostali otroci korenskega vozlišča so ukazi za obdelavo in komentarji, ki se nahajajo v prologu ali za korenskim elementom XML dokumenta. *String-value* je seštevek vseh vozlišč besedila, ki so potomci korenskega vozlišča v dokumentni ureditvi.

- *vozlišče elementa (angl. element node)* – Za vsak element iz XML dokumenta obstaja eno vozlišče elementa. Razširjeno ime (*expanded-name*) vozlišča je razširjeno ime elementa iz dokumenta. Otroci tega vozlišča so lahko vozlišča elementa, komentarjev, ukazov za obdelavo in besedila. V vozliščih besedila je shranjeno besedilo vsebine elementa. Če ima element atribut tipa ID, potem ima vozlišče element edinstveno oznako elementa ID. *String-value* je seštevek vseh vozlišč besedila, ki so potomci elementa v dokumentni ureditvi.

- *vozlišče besedila (angl. text node)* – Znakovni podatki elementa se shranjujejo v vozlišča besedila. *String-value* je znakovni podatek shranjen v vozlišče.

- *vozlišče atributa (angl. attribute node)* – Vsako vozlišče elementa je lahko oče enega ali več vozlišč atributa. Paziti je potrebno, da vozlišče atributa ni otrok svojega očeta. V XML DOM-u element ni oče svojih atributov. Ime vozlišča je razširjeno ime atributa. V množici vozlišč atributov nekega elementa se nahajajo samo vozlišča atributov, ki so eksplicitno podana v dokumentu ali so definirana v DTD kot atributi s privzeto vrednostjo. *String-value* je normalizirana vrednost atributa.

- *vozlišče imenskega prostora (angl. namespace node)* – Vsak element ima množico vozlišč imenskega prostora. Množica vsebuje vozlišče privzetega imenskega prostora (*xmls*) in vozlišča imenskih prostorov s prefiksom (*xmls:*) deklariranih v elementu ali njegovih prednikih. Element je oče vozlišč imenskega prostora. Vozlišča niso otroci svojega očeta. *String-value* je URI imenskega prostora. Razširjeno ime (*expanded-name*) vsebuje prefiks imenskega prostora.

- *vozlišče ukaza za obdelavo (angl. processing instruction nodes)* – Vsak ukaz, razen ukazov, vsebovanih v deklaraciji tipov dokumenta ima svoje vozlišče. Razširjeno ime vsebuje cilj ukaza. *String-value* je del ukaza, ki sledi cilju ukaza.

- *vozlišče komentarja (angl. comment node)* – Za vsak komentar, ki ni vsebovan v deklaraciji tipov dokumenta, obstaja eno vozlišče komentarja. *String-value* predstavlja vsebino komentarja.

2.4.2 Pot do lokacije (angl. Location path)

Osnovna ideja iz katere izhaja tudi ime jezika je, da naslavljanje opišemo kot zaporedje korakov z uporabo zapisa poti (angl. path notation). Zapis poti uporabljamo na primer takrat, ko želimo podati lokacijo neke datoteke v datotečnem sistemu.

XPath izrazi so odvisni od konteksta. Kontekst sestavljajo kontekstno vozlišče, kontekstna pozicija, kontekstna velikost, množica pridruženih spremenljivk, knjižnica funkcij in množica deklaracij imenskih prostorov. Pot do lokacije je najpomembnejša vrsta XPath izrazov. Z njo se glede na kontekstno vozlišče izbere množica vozlišč. Sestavljena je iz korakov poti, ki so znotraj poti razmejeni z znakom '/'. (*/doc/chapter[2]/section[1]*)
Poti delimo na absolutne in relativne. Relativna pot predstavlja zaporedje korakov poti. Vsak korak poti glede na kontekstno vozlišče določi množico vozlišč. V naslednjem koraku je vsako vozlišče iz množice kontekstno vozlišče. Vse množice vozlišč določene s posameznimi kontekstnimi vozlišči, ki so pridobljene v enem koraku, se združijo v eno množico. Ta množica postane množica kontekstnih vozlišč v naslednjem koraku.
Absolutna pot se vedno začne pri korenskem vozlišču in se od njega naprej nadaljuje kot relativna pot.

Vsak korak (*axis::nodetest[predicate1]...[predicateN])* je sestavljen iz treh delov:

- *osi (angl. axis)* – Določa razmerje med kontekstnim vozliščem in množico vozlišč določeno z njim. Ta del koraka določi množico vozlišč na osnovi kontekstnega vozlišča in

osi. Vsaka os ima lastnost, imenovano osnovni tip vozlišč (angl. principal node type). Ta pogojuje, da so vsa vozlišča iz množice vozlišč tipa, ki je enak osnovnemu tipu vozlišč.
V XPath-u obstaja trinajst osi [8]: *ancestor, ancestor-or-self, attribute, child, descendant, descendant-or-self, following, following-sibling, namespace, parent, preceding, preceding-sibling, self*. Privzeta je os *child*, s katero določimo, da množica vozlišč vsebuje samo otroke kontekstnega vozlišča. Osi so glede na ureditev vozlišč usmerjene; naprej v smeri dokumentne ureditve ali nazaj.

- *testiranja vozlišča (angl. node test)* – S testiranjem preverjamo, če lastnost vozlišča ustreza pogoju testiranja. Pogoj (*nodetest*) je lahko ime vozlišča ali tip vozlišča. Ko preverjamo ime vozlišča, se razširjeno ime iz pogoja primerja z razširjenim imenom (*expanded-name*) vozlišča. Pri preverjanju tipa vozlišča pogoj zapišemo kot: *text(), node(), processing-instruction(), comment() ali * (dovoljen katerikoli tip)*. V tem delu se iz množice vozlišč izločijo vsa vozlišča, ki ne ustrezajo pogoju testiranja.

- *enega ali več predikatov* – Na koncu se množica vozlišč, določena in urejena glede na os, katere vozlišča ustrezajo pogoju testiranja, filtrira s predikatom ali z zaporedjem predikatov.
  Vsako vozlišče ima kontekstno pozicijo, ki predstavlja pozicijo vozlišča znotraj urejene množice. Prva pozicija ima vrednost 1. Predikat je funkcija podana z XPath izrazom *(predicate)*. Najprej se izračuna izraz glede na kontekstno vozlišče, kontekstno pozicijo in kontekstno velikost. Če je rezultat logična vrednost (*boolean*), potem je to tudi rezultat predikata.
  V primeru da rezultat izračuna izraza numerično število in je to število enako kontekstni poziciji vozlišča, potem je rezultat predikata resnica *(true)*, drugače je neresnica (false).
  Vozlišče se vključi v novo množico vozlišč, ki je rezultat koraka, če imajo za to vozlišče vsi predikati kot rezultat resnico *(true)*.

2.4.3 Skrajšana sintaksa

XPath omogoča, da pot do lokacije zapišemo tudi s skrajšano sintakso. Tako dobimo preglednejši in krajši zapis poti. Najbolj pomembna okrajšava je, da se v koraku poti kot privzeta upošteva os *child*. To pomeni, da izpustimo *child:: ,* če je v koraku os *child.*

Ostale štiri okrajšave so:

@	attribute::
//	/descadent-or-self::node()/
.	self::node()
..	parent::node()

2.4.4 Funkcije

Vsaka implementacija XPath-a mora zagotoviti naslednje osnovne funkcije:

- funkcije množice vozlišč: *number* last*(), number* position*(), number* count*(node-set), node-set* id*(object), string* local-name*(node-set?), string* namespace-uri*(node-set?), string* name*(node-set?),*
- funkcije znakovnih nizov: *string* string*(object?), boolean* starts-with*(string, string), string* substring-after*(string, string), string* substring*(string, number, number?), number* string-length*(string?), string* normalize-space*(string?), string* translate*(string, string, string),*

- boolove funkcije: *boolean* not*(boolean), boolean* true*(), boolean* false*(), boolean* lang*(string),*
- numerične funkcije: *number* number*(object?), number sum(node-set), number* floor*(number), number* ceiling*(number), number* round*(number).*

2.4.5 Operatorji

Operatorje v XPath-u delimo na:
- *operatorje na množici vozlišč (* | [*expr*] / //)
- *primerjalne operatorje (= != <= > >=)* – Operandi so lahko različnega tipa, rezultat je vedno boolovo število.
- *logične operatorje (or and)* – Operandi se pretvorijo v boolov tip s funkcijo *boolean.*
- *aritmetične operatorje (-expr * + - div mod)* – Operatorji operande pretvorijo v numerični tip s funkcijo *number*. Rezultat operacije je numeričnega tipa.

2.5 XML DOM

Document Object Model (DOM) je abstraktni programski vmesnik (API) za veljavne HTML in dobro oblikovane XML dokumente. Definira logično strukturo dokumenta in način dostopanja ter upravljanja z dokumenti.[9] Omogoča: izdelavo dokumentov, premikanje po strukturi, dodajanje, brisanje in spreminjanje elementov.
Glavni cilj DOM-a je, da omogoči standardizirani programski vmesnik neodvisen od programskega jezika.

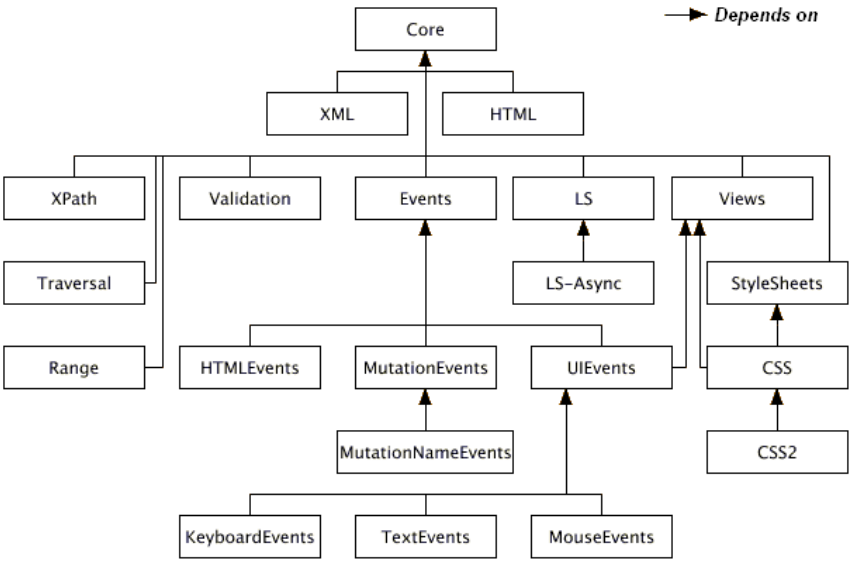
Razvoj DOM-a so začeli izdelovalci spletnih brskalnikov sredi 90-tih let. Leta 1998 je W3C izdal prvo priporočilo za DOM "Level 1". Trenutno priporočilo je "Level 3".

DOM je objektni model (angl. object model) in kot tak določa:
- vmesnike in objekte, ki se uporabljajo za predstavitev in upravljanje z dokumentom,
- semantiko vmesnikov in objektov vključno z lastnostmi in obnašanjem,
- medsebojne odnose in sodelovanje vmesnikov in objektov.

Dokumenti so zgrajeni iz objektov. Logična struktura dokumentov je podobna drevesu. Vozlišča drevesa so objekti, ki vsebujejo podatke in funkcije.

DOM mora prebrati in razčleniti celoten dokument, preden lahko začnemo z delom na dokumentu. Ker ima ta operacija veliko časovno in prostorsko kompleksnost, ni primerna za delo z velikimi dokumenti in v primerih, kjer so potrebne hitre operacije na majhnem delu dokumenta.

DOM je sestavljen iz šestnajstih modulov (Slika 1). Osnovni modul je jedro (*Core module*), ki vsebuje osnovne vmesnike. Od njega so odvisni vsi ostali moduli. Vsaka DOM implementacija mora imeti implementirane vse vmesnike jedra. XML modul (*XML*) vsebuje vmesnike za delo z XML dokumenti.
XML DOM definira standardni način za dostopanje in upravljanje z XML dokumenti.
Vsaka XML DOM implementacija mora imeti implementirane vse vmesnike iz modulov jedro in XML.



Vir: W3C Recommendation "Document Object Model (DOM) Level 3 Core Specification", W3C, 2004

Slika 1: DOM - Arhitektura

2.5.1 DOM vmesniki

V tem podpoglavju so opisani samo vmesniki, potrebni pri implementaciji XML DOM. Vmesniki so vsebovani v dveh modulih: jedru in XML modulu.

Modul jedro vsebuje naslednje vmesnike:

- *DOMException* – Omogoča dostop do podatkov o izjemi. Operacije sprožijo izjemo, ko se ne morejo uspešno dokončati ali če niso podani pravilni vhodni podatki.
- *ExceptionCode* – Vsebuje celoštevilčno vrednost, ki predstavlja podatek o vrsti napake.
- *DOMStringList* – Predstavlja seznam DOMString elementov. Omogoča samo dostopanje do posameznih elementov. Nima metod za brisanje ali dodajanje elementov v seznam.
- *NameList* – Seznam katerega elementi vsebujejo ime in URI imenskega prostora.
- *DOMImplementationList* – Seznam elementov *DOMImplementation.*
- *DOMImplementationSource* – Omogoča, da za zahtevano funkcionalnost in verzijo pridobimo DOMImplementation objekt.
- *DOMImplementation* – Vmesnik omogoča izvajanje operacij, ki so neodvisne od primerka dokumenta. Z metodo vmesnika *hasFeature* preverimo, če v implementaciji obstaja določena funkcionalnost.
- *DocumentFragment* – Uporablja se za shranjevanje dela drevesa dokumenta in izdelavo novega dela dokumenta.
- *Document* – Vmesnik predstavlja celotni HTML ali XML dokument. Objekt *Document* je koren drevesa, s katerim je predstavljen primerek dokumenta. Omogoča dostop do drevesa in njegovih objektov.
- *Node* – Osnovni vmesnik v DOM-u. Z njim dostopamo do vseh vozlišč drevesa.
- *NodeList* – Seznam elementov *Node.* Zaporedje elementov v seznamu je urejeno glede na drevo oz. dokumentno ureditev.
- *NamedNodeMap* – Seznam elementov *Node* pri katerem do elementov dostopamo z imenom elementa (Node). Do elementov seznama je mogoče dostopiti tudi z indeksom. Zaporedje elementov v seznamu ni urejeno.

- *CharacterData* – Podpira delo z znakovnimi nizi. Vmesnik je izpeljan iz *Node*.
- *Attr* – Predstavlja atribut elementa. Atributi elementa so shranjeni zunaj elementa, v vozliščih, katerih oče je element, ampak niso hkrati njegovi otroci. Deduje vmesnik *Node*.
- *Element* – Vmesnik predstavlja element v HTML ali XML dokumentu.
- *Text* – Omogoča delo z objektom *Text*, v katerega se shranjuje vsebina elementa, ali znakovna predstavitev vrednosti atributa.
- *Comment* – Objekti *Comment* so vozlišča, v katerih je shranjena vsebina komentarjev. Vmesnik se uporablja za delo z objekti *Comment*. Izpeljan je iz vmesnika *CharacterData*.
- *TypeInfo* – Vmesnik s katerim dostopamo do podatkov o tipu elementa in atributa. Podatek o tipu se pridobi iz sheme dokumenta.
- *UserDataHandler* – Omogoča pridružitev uporabniških podatkov posameznemu vozlišču.
- *DOMError* – Z vmesnikom dostopamo do opisa napake.
- *DOMErrorHandler* – Pokliče se, ko DOM implementacija sporoči napako.
- *DOMLocator* – Opisuje podatke o lokaciji v samem dokumentu (datoteki).
- *DOMConfiguration* – Omogoča nastavitev parametrov, od katerih je odvisno razčlenjevanje dokumenta.

Vmesniki XML modula:

- *CDATASection* – Dostop in delo z vozlišči, ki predstavljajo znakovno sekcijo. Izpeljan je iz vmestnika *Text*.
- *DocumentType* – Omogoča dostop do podatkov, vsebovanih v DTD-ju dokumenta: notacij, entitet, notranjih in zunanjih podmnožicah, shranjenih v DTD dokumentu. Objekt *Document* ima kvečjemu eno vozlišče oz. objekt *DocumentType*.
- *Notation* – Vmesnik predstavlja notacijo, deklarirano v DTD. Objekt *DocumentType* vsebuje seznam vseh notacij v DTD dokumentu.
- *Entity* – Entiteta definirana v DTD dokumentu.
- *EntityReference* – Referenca splošnih entitet.
- *ProcessingInstruction* – Z vmesnikom dostopamo do ukaza za obdelavo.

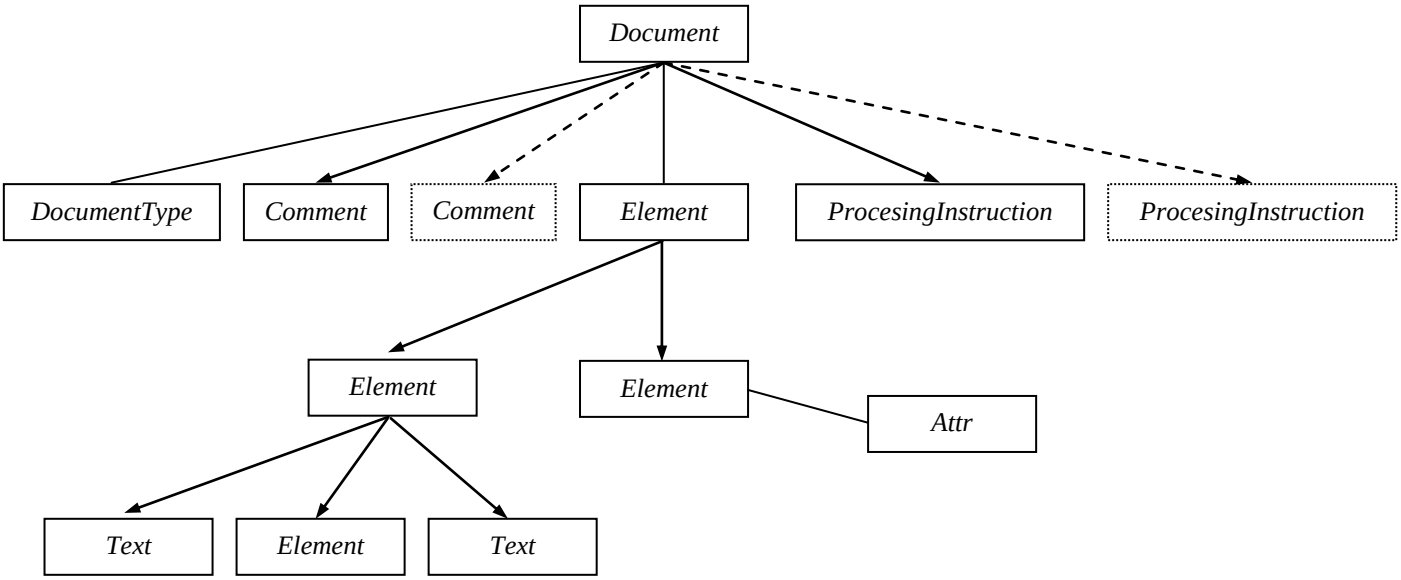
2.5.2 Drevo

XML dokument je predstavljen v DOM-u kot drevo zgrajeno iz objektov (Slika 2). Objekti, ki so hkrati tudi vozlišča drevesa, so tipa *Node* ali tipa izpeljanega iz njega. Vsebujejo: povezave do očeta, prvega in zadnjega otroka, levega in desnega brata ter seznam s povezavami do vseh otrokov.

Vozlišča so lahko enega od naslednjih tipov: *Document*, *DocumentFragment*, *DocumentType* *EntityReference*, *Element*, *Attr*, *ProcessingInstruction*, *Comment*, *Text*, *CDATASection*, *Entity* in *Notation*.

Koren drevesa je objekt *Document*, ki omogoča dostop do vseh vozlišč drevesa. Otroci tega objekta so tipa:

- *Element* – Vsak *Dokument* mora imeti obvezno en objekt *Element*, ki predstavlja korenski element v primerku XML dokumenta.
- *ProcesingInstruction* – Eden ali več objektov tega tipa predstavljajo ukaze za obdelavo, ki se nahajajo v prologu in za korenskim elementom v primerku XML dokumenta.
- *Comment* – Eden ali več objektov tega tipa predstavljajo komentarje, ki se nahajajo v prologu in za korenskim elementom v primerku XML dokumenta.
- *DocumentType* – Objekt vsebuje podatke o definicijah znotraj DTD.



Slika 2: DOM - Drevo objektov

Kot otroci objekta *Element* se lahko pojavijo: drugi objekti *Element*, *ProcessingInstruction*, *Comment*, *Text*, *CDATASection* in *EntityReference*.
Vsebina elementa ni shranjena v objektu Element, ampak v enem ali več otrocih tipa: *Text*, *CDATASection* in *EntityReference*.
Potrebno je biti pozoren na to kako se tvorijo otroci elementa kompleksnega tipa. Npr. če nima element vsebine in ima dva podelementa, potem ima objekt *Element* v drevesu pet otrok. Tri otroci so tipa *Text*, ki imajo prazno vsebino. Eno *Text* vozlišče se nahaja pred vsakim podelementom in eno za zadnjim. Pri elementu z mešano vsebino, se shrani znakovna vsebina, odvisno od tega kje se nahaja znotraj elementa, v enem ali več *Text* vozlišč.

Atributi elementa so shranjeni v objektih *Attr*, ki niso "pravi" otroci objekta *Element.* Vozlišča atributov niso otroci elementa, čeprav je ta njihov oče. Vsebina atributa se shranjuje v objektih *Text* ali *EntityReference*, ki so otroci atributa.

2.5.3 DOM Objekti

2.5.3.1 Element

Objekt *Element* predstavlja element iz primerka XML dokumenta. Vmesnik nam omogoča dostop do podatkov elementa, njegovih otrok in atributov.

Atributi:
- *attributes* – Seznam atributov elementa (*NamedNodeMap*).
- *baseURI* – URI dokumenta, ki vsebuje element *(DOMString*).
- *childNodes* – Urejen seznam vseh otrok elementa (*NodeList*).
- *firstChild* – Prvi otrok elementa (*Node*).
- *lastChild* – Zadnji otrok elementa (*Node*).
- *localName* – Lokalno ime elementa *(DOMString*).
- *namespaceURI* – URI imenskega prostora *(DOMString*).
- *nextSibling* – Desni brat elementa (*Node*).

- *nodeName* – Ime vozlišča *(DOMString*). Ima isto vrednost kot atribut *tagName*.
- *nodeType* – Tip vozlišča *(unsigned short).* V tem primeru je tipa *Element.*
- *ownerDocument* – Koren drevesa objektov *(Document).*
- *parentNode* – Oče elementa (*Node*).
- *prefix* – Prefiks imenskega prostora *(DOMString*).
- *previousSibling* – Levi brat elementa (*Node*).
- *schemaTypeInfo* – Podatki o tipu elementa, pridobljeni iz sheme *(TypeInfo*).
- *tagName* – Ime elementa *(DOMString*).
- *textContent* – Vsebina (besedilo) elementa in vseh njegovih naslednikov *(DOMString*).

Atributi dodani vmesniku v implementaciji DOM-a pri Internet Explorer-ju, ki niso definirani v priporočilu:

- *text* – Vsebina (besedilo) elementa in vseh njegovih naslednikov *(DOMString*).
- *xml* – Besedilo *(DOMString*), ki predstavlja zapis elementa in njegovih naslednikov v primerku XML dokumenta *(DOMString*).

Metode:

- *appendChild* – Doda novega otroka (vozlišče). Novo ustvarjeno vozlišče je najbolj desno vozlišče med vsemi otroki elementa.
- *cloneNode* – Naredi in vrne novo vozlišče, ki je enako obstoječem. Odvisno od vhodnega argumenta *deep* lahko naredi novo vozlišče elementa ter vozlišča vseh njegovih naslednikov.
- *compareDocumentPosition* – Primerja vozlišče elementa, v katerem je metoda poklicana z vozliščem, ki je podano kot argument.
- *getAttribute, getAttributeNS, getAttributeNode, getAttributeNodeNS* – Vrnejo vrednost atributa. Metode se med seboj razlikujejo samo po vhodnih argumentih, s katerimi se določi atribut.
- *getAttributeNode, getAttributeNodeNS* – Vrnejo vozlišče atributa. Metode se medseboj razlikujejo samo po vhodnih argumentih, s katerimi se določi atribut.
- *getElementsByTagName* – Vrne urejeni seznam vozlišč z vsemi nasledniki elementa, ki imajo ime enako imenu, podanemu v vhodnem argumentu.
- *getElementsByTagNameNS* – Vrne urejeni seznam vozlišč z vsemi nasledniki elementa, ki ustrezajo vhodnim argumentom: lokalnem imenu in URI imenskega prostora.
- *getFeature* – Za zahtevano funkcionalnost in verzijo vrne objekt *DomObject*.
- *getUserData* – Vrne podatek, katerega je uporabnik pridružil elementu s *setUserData.*
- *hasAttribute* – Rezultat metode je *true*, če ima element atribut (vozlišče *Attr*) določen z imenom.
- *hasAttributeNS* – Podobna metoda kot *hasAttribute()* z razliko, da je atribut določen z lokalnim imenom in URI imenskega prostora.
- *hasAttributes* – Vrne rezultat *true*, če ima element vsaj en atribut.
- *hasChildNodes* – Vrne rezultat *true*, če ima element vsaj enega otroka.
- *insertBefore* – Vrine novo vozlišče pred vozliščem podanim v vhodnem argumentu.
- *isDefaultNamespace* – Preveri, če je URI imenskega prostora, privzeti imenski prostor v elementu.
- *isEqualNode* – Preveri, če sta podani vozlišči enaki. Dve vozlišči sta enaki, ko sta istega tipa, imata enake argumente in otroke.
- *isSameNode* – Preveri, če sta obadva podana vozlišča, isto vozlišče.
- *isSupported* – Preveri, če implementacija podpira neko funkcionalnost.
- *lookupNamespaceURI* – Poišče URI imenskega prostora za podani prefiks.
- *lookupPrefix* – Poišče prefix podanega URI imenskega prostora.

- *normalize* – Normalizira: vsa podrejena vozlišča elementa in njegovih atributov, ki so tipa *Text.*
- *removeAttribute, removeAttributeNS in removeAttributeNode* – Odstranijo atribut. Po odstranitvi obstoječega vozlišča DOM doda novo vozlišče tega atributa z njegovo privzeto vrednostjo, če je atribut definiran kot atribut s privzeto vrednostjo. Metode se med seboj razlikujejo samo po vhodnih argumentih, s katerimi se določi atribut.
- *removeChild* – Odstrani vozlišče, ki je otrok elementa.
- *replaceChild* – Obstoječe vozliče, ki je otrok elementa, zamenja z novim.
- *setUserData* – Elementu pridruži uporabnikove podatke.
- *setAttribute, setAttributeNS, setAttributeNode, setAttributeNodeNS* – Postavijo vrednost atributa v atribut. Če ne obstaja vozlišče za vhodne argumente, se doda novo vozlišče. Metode se med seboj razlikujejo samo po vhodnih argumentih, s katerimi se določi atribut.
- *setIdAttribute, setIdAttributeNS, setIdAttributeNode* – Postavi atributu lastnost ID (identiteta). Potem je element enolično določen s tem atributom.

2.5.3.2 Attr

V objektih *Attr* so shranjeni atributi. Vmesniki tipa *Attr* so izpeljani iz tipa *Node*. Vozlišča, v katerih so shranjeni atributi nekega elementa, niso njegovi otroci. Atributi niso v bistvu del drevesa dokumenta, ampak se hranijo zunaj drevesa. Tako, da nimajo "pravega" očeta (*parentNode*) ni bratov (*previousSibling*, *nextSibling*). Z atributom objekta *ownerElement* so atributi povezani z elementom.

Atributi (lastnosti):
- *baseURI* – URI dokumenta, ki vsebuje atribut *(DOMString*).
- *isID* – Vrednost *true* pove, da je atribut tipa ID. Atribut predstavlja identiteto elementa (*boolean*).
- *localName* – Lokalno ime atributa *(DOMString*).
- *name* – Ime atributa *(DOMString*). Če je *localName* različno od *NULL*, *Name* je kvalificirano ime atributa.
- *namespaceURI* – URI imenskega prostora *(DOMString*).
- *nodeName* – Ime vozlišča *(DOMString*). Ima isto vrednost kot *name*.
- *nodeType* – Tip vozlišča *(unsigned short).* V tem primeru je tipa *Attr*.
- *nodeValue* – Vrednost vozlišča. Ima isto vrednost kot atribut *value*.
- *ownerDocument* – Koren drevesa objektov *(Document).*
- *ownerElement* – Vozlišče elementa kateremu pripada atribut (*Element*).
- *prefix* – Prefiks imenskega prostora *(DOMString*).
- *schemaTypeInfo* – Podatki o tipu atributa, pridobljeni iz sheme *(TypeInfo*).
- *value* – Normalizirana vrednost atributa *(DOMString*).
- *textContent* – Vrednost vsebine (besedilo) atributa *(DOMString*).
- *specified* – Ima vrednost *true*, če je vrednost atributa eksplicitno podana v primerku XML dokumenta (*boolean*). Ko aplikacija spremeni vrednost atributa, se vrednost *specified* spremeni v *true*.

Atributi dodani vmesniku v implementaciji DOM-a pri Internet Explorer-ju, ki niso definirani v priporočilu:
- *text* – Vrednost vsebine (besedilo) atributa *(DOMString*).
- *xml* – Besedilo *(DOMString*), ki predstavlja zapis atributa v primerku XML dokumenta *(DOMString*).

2.5.3.3 Text

Vmesniki Text so izpeljani iz vmesnika *CharacterData*. V objektih Text se shranjuje vsebina (besedilo) elementov in atributov.

Atributi:
- *data* – Besedilo vozlišča *(DOMString*).
- *isElementContentWhitespace* – Vrednost atributa je *true*, če je vsebina vozlišča presledek (angl. whitespace) (*boolean*).
- *length* – Dolžina besedila oz. število znakov (*unsigned long*).
- *wholeText* – Besedilo vsebovano v vozlišču in vseh bratih tipa *Text* tega vozlišča*(DOMString*).

Metode:
- *appendData* – Doda novo besedilo na konec obstoječega.
- *deleteData* – Pobriše določeno število (*count*) znakov v besedilu. Brisanje se začne na poziciji *offset*.
- *insertData* – Vstavi novo besedilo v obstoječe na poziciji *offset*.
- *replaceData* – Zamenja določeno število znakov od pozicije *offset* z znaki iz vhodnega argumenta *arg*.
- *replaceWholeText* – Zamenja obstoječe besedilo v vozlišču in vseh bratih tega vozlišča z novim besedilom.
- *splitTextSplits* – Razdeli besedilo na poziciji *offset* v dva dela. Znaki, ki se nahajajo za znakom na poziciji *offset*, se shranijo v novem vozlišču. Novo vozlišče postane brat obstoječega.
- *substringData* – Vrne del besedila.

2.5.3.4 NamedNodeMap

Objekt *NamedNodeMap* je seznam vozlišč, ki ni urejen glede na dokumentno ureditev vozlišč. Omogoča iskanje vozlišča v seznamu na osnovi njegovega imena.

Atribut:
- *length* – Število vozlišč (*unsigned long*).

Metode:
- *getNamedItem* – Vrne vozlišče za argument: ime.
- *getNamedItemNS* – Vrne vozlišče za argumente: lokalno ime in URI imenskega prostora.
- *item* – Vrne vozlišče, ki se nahaja na poziciji *index*.
- *removeNamedItem* – Odstrani vozlišče za argument: ime.
- *removeNamedItemNS* – Odstrani vozlišče za argumente: lokalno ime in URI imenskega prostora.
- *setNamedItem* – Doda novo vozlišče ali zamenja vozlišče, če obstaja vozlišče z istim imenom.
- *setNamedItemNS* – Doda novo vozlišče ali zamenja obstoječe vozlišče, če obstaja vozlišče z istim lokalnim imenom in URI imenskega prostora.

2.5.4 Dostop do vozlišča

Naslednji primer opisuje, kako dostopimo do korenskega elementa XML dokumenta in njegovih otrok (elementov).

Najprej preberemo dokument v objekt *Document*.

```
xmlDokument=loadXMLDoc("test.xml");
```

Vozlišče na katerega kaže atribut *documentElement* je korenski element dokumenta XML.
V urejenem seznamu *childNodes* tega vozlišča se nahajajo povezave do njegovih otrok.

```
seznamOtrokov=xmlDokument.documentElement.childNodes;
```

Da bi izpisali imena vseh otrok, iz seznama preberemo povezavo do posameznega vozlišča.

```
for (i=0;i< seznamOtrokov.length;i++)
{ if (seznamOtrokov[i].nodeType==1) // izpiši samo vozlišča, ki vsebujejo elemente
  document.write(seznamOtrokov[i].nodeName); }
```

2.5.5 Dodajanje novega vozlišča v drevo

Najprej preberemo dokument v objekt *Document*.

```
xmlDokument=loadXMLDoc("test.xml");
```

Naredimo novi element.

```
noviElement= xmlDokument.createElement("tel");
```

Poiščemo prvi element kupec v dokumentu.

```
oceElementa=xmlDokument.getElementsByTagName("kupec")[0];
```

Elementu kupec dodamo novega otroka, ki je prej ustvarjeni element.

```
oceElementa.appendChild(noviElement);
```

2.6 SPLETNE STORITVE

Spletna storitev (angl. Web Service) je vsaka storitev dostopna preko interneta, ki uporablja standardizirani XML sporočilni sistem in ni omejena na določen operacijski sistem ali programski jezik [10]. Ta naj bi bila samoopisna (angl. self-describing). Obstajala naj bi možnost registracije in odkrivanja storitve. Kot sporočilni sistem se uporabljajo: XML-RPC (Extensible Markup Language-Remote Procedure Call), SAOP, HTTP (Hypertext Transfer Protocol) POST/GET.
Arhitekturo spletnih storitev lahko opišemo na dva načina; z vlogami udeležencev ali kot protokolni sklad.

Vloge udeležencev:

- *ponudnik storitve (angl. Service provider)* – Implementira in omogoča storitev.
- *povpraševalec storitve (angl. Service requestor)* – Vsak uporabnik, ki pošlje zahtevo.
- *registracijska storitev (angl. Service registry)* – Centralni imenik storitev, ki omogoča objavo novih storitev in iskanje obstoječih.

Protokolni sklad:

- *transportne storitve (angl. Service transport)* – Sloj, ki prenaša sporočila med aplikacijami. (HTTP (Hypertext Transfer Protocol), SMTP (Simple Mail Transfer Protocol), FTP (File Transfer Protocol))
- *XML sporočanje (angl. XML messaging)* – Sloj odgovoren za kodiranje sporočila v XML formatu, ki bo razumljiv vsem udeležencem. (XML-RPC, SAOP)
- *opisovanje (angl. Service description)* – Opisuje storitve. (Web Services Description Language (WSDL))
- *odkrivanje (angl. Service discovery)* – Centralni imenik storitev, ki omogoča objavo novih storitev in iskanje obstoječih.

2.6.1 WSDL (Web Services Description Language)

WSDL je na XML-u zasnovan jezik za opisovanje spletnih storitev. Neodvisen je od operacijskega sistema in programskega jezika. Jezik opisuje: vmesnik, ki vsebuje javno dostopne funkcije; podatkovne tipe, ki se uporabljajo v sporočilih; transportni protokol uporabljen za prenos sporočil; lokacijo, kjer se storitev nahaja.

Uporabnik na osnovi WSDL storitve pridobi vse informacije, potrebne za uporabo opisane spletne storitve. Opisi storitev se običajno nahajajo v centralnih imenikih storitev.

Na osnovi WSDL opisa specializirana orodja izdelajo skoraj vso potrebno kodo in obratno, za izdelano storitev, izdelajo WSDL opis storitve.

WSDL ima šest glavnih elementov:

- *definitons* – Korenski element WSDL dokumenta. V elementu deklariramo imenske prostore in ciljni imenski prostor WSDL dokumenta.
- *type* – Element vsebuje definicije vseh potrebnih tipov podatkov za opisovanje izmenjave sporočil, ki niso definirani v drugih shemah.
- *message* – Z elementom definiramo posamezno sporočilo. Vrednost atributa *name* je ime sporočila, ki mora biti edinstveno znotraj celotnega opisa. Element vsebuje podelemente *part*, ki predstavljajo vsebino sporočila. Pri sporočilu namenjenemu uporabi v RPC, elementi *part* predstavljajo parametre procedure.
- *portType* – Element vsebuje množico operacij, definiranih v podelementih *operation.* Vsaka operacija lahko vsebuje naslednja sporočila: vhodno, izhodno in sporočilo o napaki.
- *binding* – Za vsako posamezno operacijo definira format sporočila in protokol po katerem se sporočila prenašajo.
- *service* – Vsebuje množico podelementov *port.* Vsak element *port* definira lokacijo, na kateri se storitev nahaja oz. lokacijo končnega vozlišča.

WSDL ima tudi dva pomožna elementa:

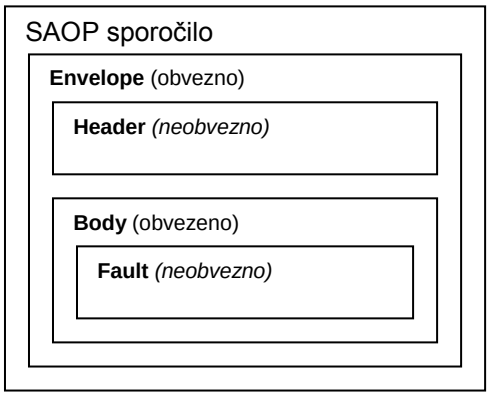
- *document* – V element se shranjuje dokumentacija, namenjena uporabniku (človeku).
- *import* – WSDL elementi so lahko shranjeni v neodvisnih dokumentih. Z elementom *import* uvozimo WSDL elemente iz zunanjih dokumentov v osnovni dokument. Omogoča ponovno uporabo dela kode. Iz skupnih elementov se lahko naredi več različnih WSDL opisov.

2.6.2 SOAP

SOAP je na XML-u zasnovan, enostavni protokol za izmenjavo strukturiranih informacij v decentraliziranih in distribuiranih okoljih. Neodvisen je od operacijskega sistema in programskega jezika. Omogoča, da se aplikacija na strani odjemalca poveže z oddaljeno storitvijo in uporablja njene metode. Med odjemalcem in strežnikom poteka komunikacija tako, da si med seboj pošiljata SAOP sporočila.
Prenos SAOP sporočil ni omejen na določeni komunikacijski protokol. Uporabimo lahko HTTP (Hypertext Transfer Protocol), SMTP (Simple Mail Transfer Protocol), FTP (File Transfer Protocol) ali katerikoli drugi. Najbolj je razširjena uporaba HTTP, ker požarni zidovi ne blokirajo njegov prenos.

2.6.2.1 Sporočilo (angl. Message)

SAOP sporočilo je osnovna enota v komunikaciji med SAOP vozlišči (Slika 3). Uporablja se za enosmerni prenos sporočila med pošiljateljem in prejemnikom. Naprimer za zahtevo, katero odjemalec pošlje strežniku ali za odgovor, katerega strežnik pošlje odjemalcu.



Slika 3: SAOP - Sporočilo

Ovojnica (Envelope) – Vsako SAOP sporočilo je ovito v ovojnico *Envelope*, ki je korenski element XML dokumenta s sporočilom. Elementi SAOP sporočila morajo biti kvalificirani in se nahajajo v imenskem prostoru z URI *"http://schemas.xmlsaop.org.saop/envelope/"*, ki se obvezno deklarira v elementu *Envelope*. Verzija 1.2 ima imenski prostor z URI *"http://www.w3.org/2001/09/envelope/"*.

```
<SAOP-ENV:Envelope xmlns:SAOP-ENV="http://schemas.xmlsaop.org.saop/envelope/"
                                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                xmlns:xsd="http://www.w3.org/2001/XMLSchema">
...      </SAOP-ENV:Envelope>
```

Glava (Header) – Ni obvezen element sporočila. V njem se lahko določi pot do končnega SAOP vozlišča, če sporočilo potuje po vmesnih vozliščih. Shranjujejo se dodatne zahteve npr. digitalni podpis in uporabniško ime.

Telo (Body) – Obvezen element pri vseh sporočilih. Vsebuje naziv sporočila in njegove parametre.

Napaka (Fault) – V primeru, ko se zgodi napaka pri obdelavi zahteve, strežnik vrne v elementu *Fault* podatke o napaki. Vsebuje podelemente: *faultCode* (koda napake), *faultString*

(opis napake), *faultActor* (podatek o vozlišču, ki je vrnilo sporočilo z napako, če to ni končno vozlišče) in *detail* (podrobno obvestilo o napaki. Če ne obstaja, potem napaka ni nastala pri obdelavi vsebine telesa *(Body)* sporočila).

2.6.3 UDDI (Universal Distribution, Discovery, and Interoperability)

UDDI je tehnična specifikacija, ki definira množico storitev za registriranje in odkrivanje podjetji, organizacij in drugih ponudnikih spletnih storitev ter njihovih spletnih storitev in vmesnike za uporabo UDDI storitev.
Zasnovana je na splošno uveljavljenih standardih HTTP, XML, XML Schema in SOAP.
Omogoča izdelavo javnih in zasebnih UDDI registrov. Do registrov lahko dostopamo s spletno (Web) ali SOAP zasnovanimi vmesniki.

Podatkovni model (angl. Data Model) je sestavljen iz štirih osnovnih elementov:

- *businessEntity* – Za vsako podjetje, ki je registriralo svojo storitev, obstaja en element. Vsebuje splošne podatke o podjetju: ime, naslov, opis podjetja in kontaktne podatke. Vsak *businessEntity* je enolično določen s ključem v podelementu *businessKey*. Ključ se določi pri registraciji. Omogoča povezavo s storitvami, katerih podatki so shranjeni v elementu *businessService*. Podelement *identifierBag* vsebuje eno ali več identifikacijskih oznak podjetja. Podjetja se v registru klasificirajo glede na podelement *categoryBag*, ki vsebuje standardizirane oznake panoge, blaga ali storitve.
- *businessService* – Vsebuje podatke o vsaki posamezni storitvi ali skupini storitev. Podelement *businessKey* omogoča povezavo z elementom *businessEntity* oz. povezuje storitev s podjetjem. Vsaka storitev ima enolično določen ključ *serviceKey*.
- *bindingTemplate* – Element vsebuje podatke o komunikacijskem protokolu in naslovu na katerem se storitev nahaja. Vsaka storitev se lahko implementira na več naslovov in z več protokolov.
- *tModel* – Uporablja se kot kazalec na zunanjo tehnično specifikacijo. Oblika tehnične specifikacije ni določena. Pri protokolu SAOP lahko kaže na WSDL datoteko.

Do storitev UDDI registra dostopamo s programski vmesnikom zasnovanem na SAOP protokolu. Vmesnik je razdeljen na dva dela: na del za povpraševanje (*angl. Inquary API*) in na del za objavo (registracijo) storitev (*angl. Publisher API*).

Vmesnik za povpraševanje (*angl. Inquary API*) – Omogoča iskanje in pridobivanje podatkov iz registra. Iskanje omogočajo funkcije: *find_binding, find_business, find_service, find_tModel.* Funkcije za iskanje kot rezultat vrnejo enega ali več ključev, ki enolično določajo iskane elemente. V naslednjem koraku podatke o iskanem elementu pridobimo z ustreznimi funkcijami (*get_bindingDetail, get_businessDetail, get_serviceDetail, get_tModelDetail)*, katerih argument je ključ elementa.

Vmesnik za objavo storitev (*angl. Publisher API*) vsebuje funkcije za:

- avtentikacijo uporabnika (*get_authToken, discard_authToken*),
- dodajanje novih storitev in spreminjanje podatkov o že objavljeni storitvi (*save_binding, save_business, save_service, save_tModel),*
- brisanje objavljenih storitev (*delete_binding, delete_business, delete_service, delete_tModel).*

UDDI zahteva, da je pri vseh funkcijah obvezna avtentikacija uporabnika in njihov prenos po sloju varnih vtičnic (angl. secure socket layer, krat. SSL).

3. Sistem "On-line zdravstveno zavarovanje"

V tem poglavju je podan opis on-line sistema, ki je ključen za razumevanje načrtovanja, izdelave in delovanje programske opreme, razvite v okviru tega diplomskega dela.

3.1 Splošno o sistemu

"Sistem omogoča neposredno, takojšnje izmenjevanje podatkov med informacijskim sistemom izvajalca zdravstvenih storitev in informacijskimi sistemi Zavoda za zdravstveno zavarovanje Slovenije (ZZZS) in zavarovalnic za prostovoljna zdravstvena zavarovanja (ZPZZ)." [11]

Sistem je nastal kot nadgradnja nepovezanega (angl. off-line) sistema "Sistem kartice zdravstvenega zavarovanja", ki se je od leta 2000 uporabljal za izmenjevanje podatkov. Kartica zdravstvenega zavarovanja (KZZ) se je v sistemu, razen za identifikacijo zavarovanca, uporabljala kot medij za prenos podatkov. Izmenjava je potekala tako, da so uporabniki sistema brali in zapisovali podatke na kartico posameznega zavarovanca.

Razvoj novega sistema se je začel leta 2006. ZZZS je razvil in postopno uvedel novi verziji kartice zdravstvenega zavarovanja in profesionalne kartice zdravstvenega zavarovanja (PK). Zgrajen je nov sistem, ki deluje na novi infrastrukturi in omogoča neposredni dostop do podatkov.

Pričakovane pridobitve z uvedbo novega sistema so za:
- *zavarovance* – Enostavnejše uresničevanje pravic iz zdravstvenega zavarovanja, predvsem z opustitvijo obveznega potrjevanja kartice na samopostrežnih terminalih.
- *izvajalce zdravstvenih storitev* – Izvajalcem omogoča izdelavo in uvedbo novih informacijskih sistemov, ki bodo posledično privedli do večje kakovosti storitev in manjših stroškov poslovanja.
- *zavarovalnice* – Zmanjšujejo se stroški poslovanja, ker ni več zlorabe pravic iz naslova zavarovanja. Zavarovalnice bodo iz pridobljenih podatkov lahko načrtovale svoje finančne obveznosti do izvajalcev storitev in ocenile tveganje pri zavarovanju posameznega zavarovanca.

Pilotna uvedba se je začela leta 2008 na območju Nove Gorice. Nacionalna uvedba se je zaključila, z izjemo Univerzitetnega kliničnega centra Ljubljana, marca 2010.

Sistem poleg ZZZS in zavarovalnic za prostovoljna zdravstvena zavarovanja uporabljajo še: bolnišnice, zdravstveni domovi, domovi za starejše, zdravniki, zobozdravniki, lekarne, dobavitelji medicinsko tehničnih pripomočkov (MTP), dobavitelji MTP za vid (v nadaljevanju: OPTIKI) in ostali izvajalci zdravstvenih storitev v breme zdravstvenih zavarovanj.

Izvajalci lahko iz sistema pridobijo naslednje podatke o zavarovani osebi:
- osnovni osebni podatki,
- podatki o obveznem zdravstvenem zavarovanju,
- podatki o dopolnilnih prostovoljnih zdravstvenih zavarovanjih,
- podatki o nadstandardnih prostovoljnih zdravstvenih zavarovanjih,

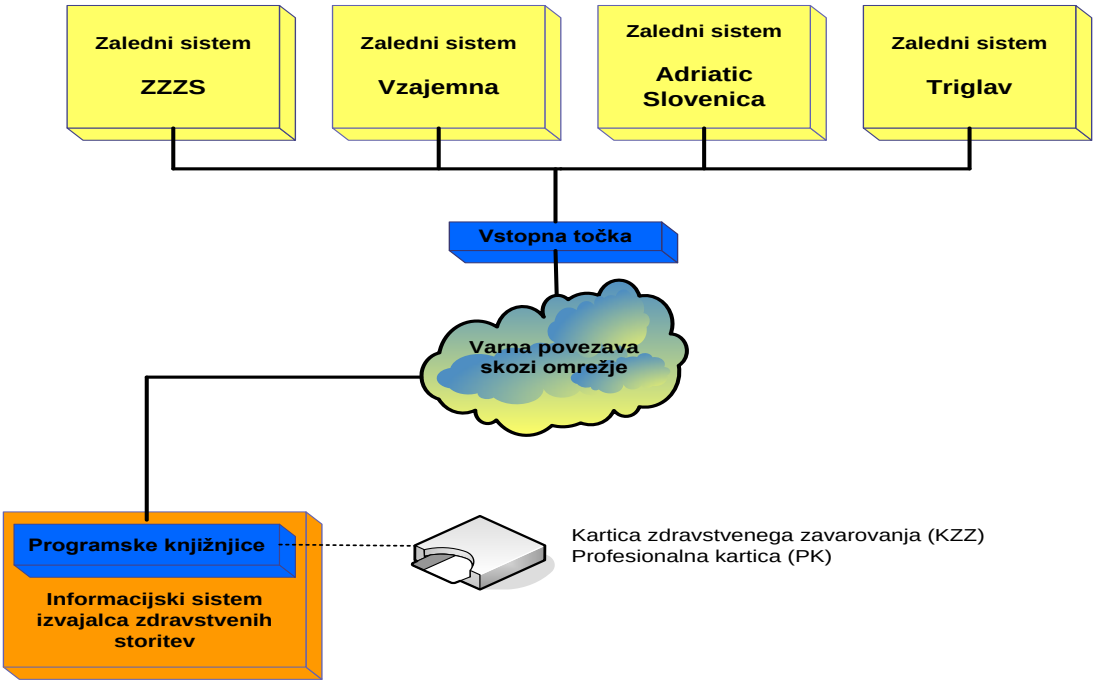
- podatki o izbranih osebnih zdravnikih,
- podatki o predpisanih medicinsko tehničnih pripomočkih,
- podatki o izdanih medicinsko tehničnih pripomočkih,
- podatki o izdanih zdravilih,
- podatki o izjavi osebe za darovanje organov,
- podatki o nosečnosti,
- podatki o OBMP (oploditve z biomedicinsko pomočjo).

Izvajalci, odvisno od tega katere zdravstvene storitve opravljajo, v sistem obvezno pošiljajo naslednje podatke:

- podatki o novi izbiri osebnega zdravnika,
- podatki o izdaji zdravila,
- podatki o izdaji naročilnice za medicinsko tehnični pripomoček (MTP),
- podatki o izdaji, izposoji, vračilu, servisiranju, popravilu MTP,
- podatki o nosečnosti,
- podatki o opravljeni OBMP.

3.2 Tehnične značilnosti sistema

Sistem je zasnovan kot spletna storitev. S tem se omogoča enostavna in varna povezava med informacijskimi sistemi izvajalcev in sistemom. Dostop je mogoč vsem informacijskim sistemom izvajalcev ne glede na njihov operacijski sistem ali programski jezik, v katerem so izdelane aplikacije.
Izvajalci v svoj sistem vključijo programske knjižnice (*IHISxxx.ddl*), ki vsebujejo vmesnik za dostop in izmenjevanje podatkov s sistemom. Vmesnik in vstopna točka sistema izmenjujeta sporočila po komunikacijskem protokolu Hypertext Transfer Protocol Secure (HTTPS), ki omogoča, da se komunikacija izvaja po šifrirani povezavi.
Vmesnik in vstopna točka sistema izmenjujeta strukturirane informacije po SAOP protokolu.



Slika 4: Arhitektura sistema "On-line zdravstveno zavarovanje"

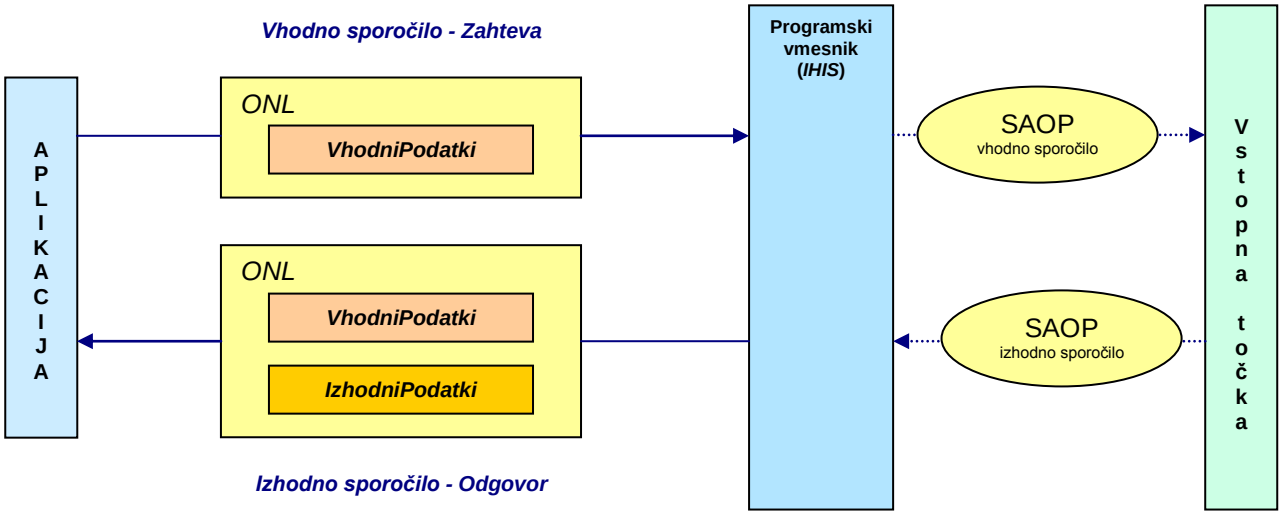
Glavne komponente sistema (Slika 4) so:

- *programske knjižnice* – Programski vmesnik za dostop do on-line sistema.
- *vstopna točka (angl. entry point)* – Predstavlja osrednjo komponento sistema. Preko programskega vmesnika vzpostavimo povezavo z vstopno točko, ki nato izvede identifikacijo uporabnika, preveri istovetnost identitete (angl. authentication) in pooblastila uporabnika (angl. authorisation). Vsa sporočila se pošiljajo v vstopno točko. Preverjena in dovoljena sporočila vstopna točka usmeri v ustrezni zaledni sistem. Zaledni sistemi posredujejo odgovore vstopni točki, ki jih nato po varni povezavi pošlje programskemu vmesniku.
- *zaledni sistemi (angl. back-end systems) ZZZS in zavarovalnic za PZZ* – Zaledni sistem: shranjuje poslane podatke, zagotavlja zahtevane podatke in omogoča vodenje sledi dostopov do podatkov. Vsak zaledni sistem skrbi za obseg podatkov iz svoje dejavnosti.
- *kartica zavarovane osebe (KZZ)* – Identifikacijska listina zavarovane osebe. Vsebuje nekvalificirano digitalno potrdilo (ZZZS-CA), ki ni zaščiteno z osebno identifikacijsko številko (angl. Personal Identification Number, krat. PIN) za dostop do osebnih podatkov o zavarovancu.
- *profesionalna kartica (PK)* – Identifikacijska listina zaposlenega pri izvajalcu zdravstvenih storitev. Omogoča identifikacijo in elektronsko podpisovanje. PK vsebuje digitalno potrdilo. Samo pri izvajalcih, ki predpisujejo zdravila, je digitalno potrdilo kvalificirano (POŠTA®CA). Vsak imetnik PK ima glede na svojo vlogo natančno definirana pooblastila v sistemu. Pooblastilo določa vrsto dostopa in obseg dostopnih podatkov. Kartica je zaščitena s 4-mestnim PIN-om.

Poleg produkcijskega on-line sistema je ZZZS zgradil in vzdržuje testni on-line sistem, ki omogoča razvijalcem razvoj in testiranje aplikacij na vnaprej pripravljenih podatkih. Za uporabo testnega sistema potrebujemo testne KZZ in PZ.

3.3 Struktura XML ukazov in rezultatov

Aplikacija informacijskega sistema izvajalca zdravstvenih storitev komunicira z on-line sistemom tako, da preko programskega vmesnika izmenjuje XML sporočila (Slika 5).



Slika 5: Prenos sporočil preko programskega vmesnika IHIS

Vse operacije, ki jih izvajajo spletne storitve on-line sistema, so operacije tipa zahteva-odgovor (angl. Request-response). Operacije tega tipa so sestavljene iz dveh sporočil: vhodnega in izhodnega. Vhodno sporočilo vsebuje ukaz, za katerega spletna storitev posreduje rezultat, vsebovan v izhodnem sporočilu.

Operacija se prične tako, da spletni storitvi pošljemo zahtevo z vhodnim sporočilom. Storitev na zahtevo odgovori z izhodnim sporočilom.

Vsako XML sporočilo mora biti veljavno glede na XML Schemo *ONL.xsd.*

Korenski element XML sporočila je element *ONL*. V elementu se deklarira XML Schema, v kateri so definirani elementi in podatkovni tipi, ki se uporabljajo v sporočilih. XML Schema nima določenega ciljnega imenskega prostora. Elementi, definirani v tej shemi, ne pripadajo nekemu določenemu imenskemu prostoru oz. imena elementov in atributov ne vsebujejo prefiks imenskega prostora.

```
<ONL xsi:noNamespaceSchemaLocation="..\..\..\..\Projects\On-lineZZ\Documents\XMLsheme\0001\ISI\ONL.xsd" ..>
    <VhodniPodatki> ... </VhodniPodatki>
    <IzhodniPodatki> ... </IzhodniPodatki>
</ONL>
```

Element *ONL* ima kvečjemu dva otroka, elementa: *VhodniPodatki* in *IzhodniPodatki*. Element *VhodniPodatki* je obvezen pri vhodnem in izhodnem sporočilu. Element *IzhodniPodatki* vsebujejo samo izhodna sporočila.

Element *VhodniPodatki* vsebuje ukaz in parametre ukaza. Pri vhodnem in izhodnem sporočilu ima element *VhodniPodatki* enako vsebino oz. izhodno sporočilo ima element, ki je kopija elementa v vhodnem sporočilu.
V elementu *IzhodniPodatki* spletna storitev vrne rezultat za posredovani ukaz.

3.3.1 Vhodni podatki

V elementu *VhodniPodatki* (Slika 6) podamo splošne podatke o vhodnem sporočilu in ukaz, katerega želimo poslati on-line sistemu.

Vsebuje samo en obvezen element *VsebinskiPodatki*, ki ima dva otroka:

- *Glavo* – Vsebuje podatke o sporočilu, pošiljatelju in zavarovancu za katerega se pošilja ukaz on-line sistemu.
- *Telo* – Vsebuje podelemente, ki predstavljajo ukaz in parametre ukaza.

Slika 6: Struktura XML ukazov in rezultatov – Vhodni podatki

3.3.1.1 VsebinskiPodatki\Glava

V atributih in potomcih elementa *Glava* (Slika 7) se nahajajo podatki, ki so obvezni pri vseh ukazih.

Atributi opisujejo samo sporočilo:

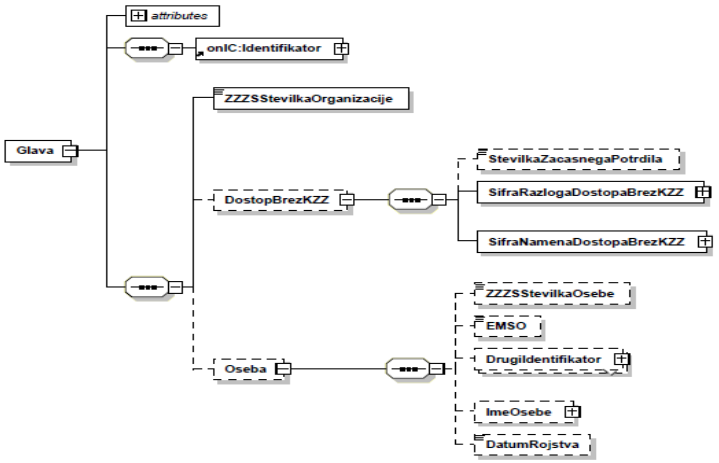
- verzijo sporočila oz. verzijo sheme po kateri je sporočilo oblikovano *(verzijaPosiljke)*,
- datum in čas pošiljanja sporočila (*datumInCasPosiljke)*,
- status izmenjave, ki določa, ali je sporočilo namenjeno produkcijskemu ali testnemu okolju *(statusIzmenjave)*.
- identifikator sporočila, ki enolično določa sporočilo na nivoju celotnega on-line sistema *(onlC:Identifikator*).

Vsako vhodno sporočilo vsebuje element *ZZZSStevilkaOrganizacije*, katerega vsebina predstavlja šifro organizacije (zdravstveni dom, podjetje,..) izvajalca zdravstvene dejavnosti.

Vsako vhodno sporočilo z ukazom za branje ali zapisovanje podatkov mora vsebovati podatek, ki enolično določa zavarovanca, za katerega se izvaja operacija. Način kako zapišemo ta podatek, je odvisen od delovanja on-line sistema, delovanja lokalnega sistema ter prisotnosti in delovanja kartice KZZ v trenutku obravnave zavarovanca.
Standardni način uporabljamo v primeru, ko so vsi pogoji izpolnjeni; on-line sistem in lokalni sistem delujeta v trenutku obravnave in zavarovanec ima KZZ, ki deluje. Zavarovanca določimo tako, da ZZZS številko zavarovanca zapišemo v podelement *ZZZSStevilkaOsebe* elementa *Oseba.*

Vse ostale primere ZZZS obravnava kot izredne. V izrednih primerih imajo vsi ukazi za branje podatkov dodatne parametre, shranjene v elementu *DostopBrezKZZ*. ZZZS preverja razlog (*SifraRazlogaDostopaBrezKZZ*) in namen (*SifraNamenaDostopaBrezKZZ*) dostopa oz. posredovanja sporočila, pri katerem niso izpolnjeni vsi pogoji.
V izreden primer sodi tudi obravnava zavarovanca, kateremu je ZZZS izdal listino z začasnim potrdilom, ki nadomešča KZZ. V tem primeru je številka začasnega potrdila (*StevilkaZacasnegaPotrdila*) obvezen element v elementu *DostopBrezKZZ.*

Ostali podelementi elementa *Oseba* predstavljajo parametre ukaza za branje osnovnih osebnih podatkov (*OsnovniOsebniPodatki*), ki so skupaj z ukazom opisani v podpoglavju 3.3.3.



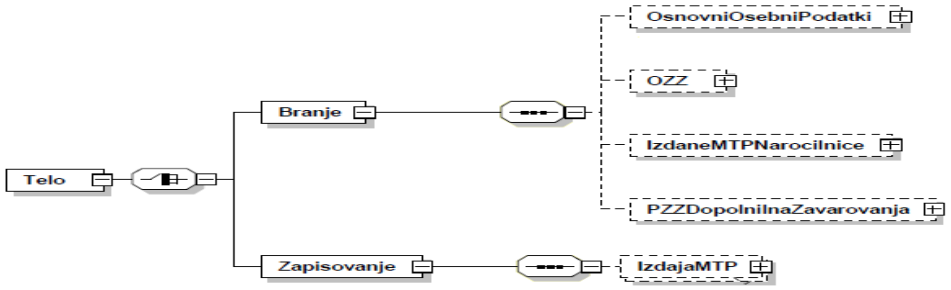
Slika 7: Struktura XML ukazov in rezultatov – Vhodni podatki/VsebinskiPodatki/Glava

3.3.1.2 VsebinskiPodatki\Telo

Vsako vhodno sporočilo ima element *Telo* z enim otrokom (Slika 8), ki predstavlja ukaz ali skupino
ukazov.
Ukazi so razdeljeni v tri skupine:

- *samostojni ukazi* – Vsebujejo ukaze za branje in zapisovanje podatkov na KZZ ter za prijavo in preverjanje delovanja on-line sistema. Elementi, ki predstavljajo te ukaze, so otroci elementa *Telo*.
  Ukaze prijava v on-line sistem (Prijava), preverjanje delovanja sistema (*StanjeSistema*) in branje identifikacijskih podatkov iz KZZ (*OsnovniIdentifikacijskiPodatki*) ne uporabljamo neposredno. Programske knjižnice vsebujejo funkcije, ki nam vrnejo rezultate za te ukaze in pri tem same poskrbijo za oblikovanje in pošiljanje vhodnega sporočila.
- *ukazi za pridobivanje (Branje) podatkov iz on-line sistema* – Ukazi iz te skupine omogočajo pridobivanje: osnovnih osebnih podatkov (*OsnovniOsebniPodatki*) zavarovanca, podatkov o osnovnem zdravstvenem zavarovanju (*OZZ*), izdanih naročilnicah za medicinsko tehnične pripomočke (*IzdaneMTPNaročilnice*) in podatkov o dopolnilnem prostovoljnem zdravstvenem zavarovanju (*PZZDopolnilnaZavarovanja*) zavarovanca.
- *ukazi za zapisovanje (Zapisovanje) podatkov v on-line sistem* – Z ukazi *IzdajaMTP* zapišemo v sistem pripomočke, predpisane v naročilnici, katere dobavitelji (optiki) izdajo oz. prodajo zavarovancu.

V naslednjih podpoglavjih bodo opisani samo ukazi uporabljeni pri izdelavi diplomske naloge oz. ukazi, potrebni za izmenjevanje podatkov med on-line sistemom in informacijskim sistemom optika, dobavitelja MTP pripomočkov za vid. Vsi ostali ukazi so predstavljeni v tehnični dokumentaciji [11].



Slika 8: Struktura XML ukazov in rezultatov – Vhodni podatki/VsebinskiPodatki/Telo

3.3.2 Izhodni podatki

V elementu *IzhodniPodatki* (Slika 9) spletna storitev poda splošne podatke o izhodnem sporočilu in rezultat ukaza, posredovanega v zahtevi.

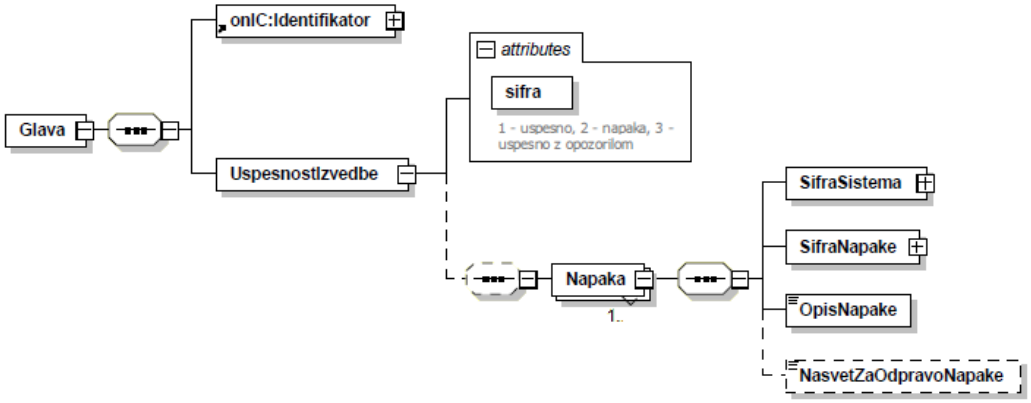
Vsebuje samo en obvezen element *VsebinskiPodatki*, ki ima dva otroka:

- *Glava* – Vsebuje identifikacijsko oznako sporočila in podatke o uspešnosti sprejema in obdelave vhodnega sporočila (zahteve).
- *Telo* – Vsebuje podelemente, ki predstavljajo rezultat ukaza.

Slika 9: Struktura XML ukazov in rezultatov – Izhodni podatki

3.3.2.1 VsebinskiPodatki\Glava

Element *Glava* ima dva podelementa *onlC:Identifikator* in *UspesnostIzvedbe* (Slika 10).
Identifikator enolično določa sporočilo na nivoju on-line sistema.
Element *UspesnostIzvedbe* vsebuje informacijo o uspešnosti obdelave samega sporočila in je neodvisen od same izvedbe ukaza, vsebovanega v sporočilu. On-line sistem po prejemu vhodnega sporočila preveri veljavnost sporočila glede na XML Schemo in pravice pošiljatelja do pošiljanja sporočil. Če sistem ugotovi napako do koraka, v katerem so ukaz in vsi parametri ukaza iz sporočila pripravljeni za obdelavo, se v element *UspesnostIzvedbe* zapišejo podatki o napaki, komponenti sistema, v kateri je nastopila napaka, in nasvet za odpravo napake.



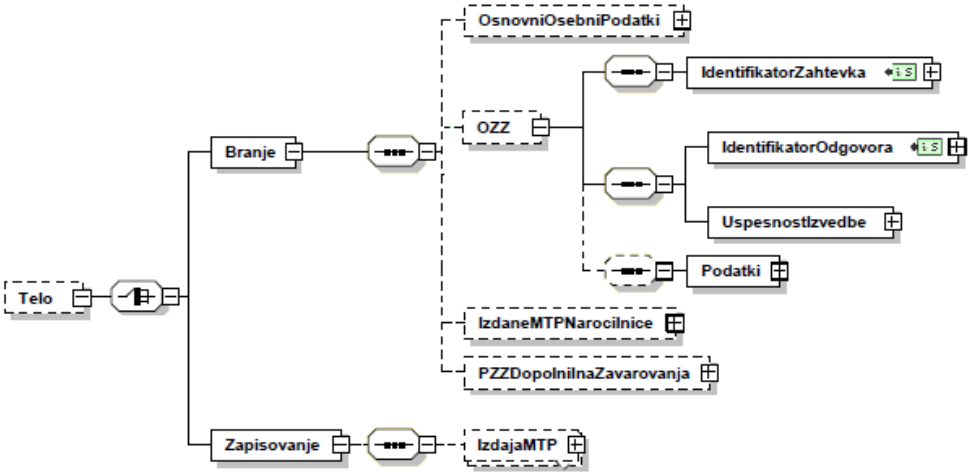
Slika 10: Struktura XML ukazov in rezultatov – Vhodni podatki/VsebinskiPodatki/Glava

3.3.2.2 VsebinskiPodatki\Telo

Element vsebuje v svojih potomcih rezultat za posredovani ukaz ali rezultate v primeru, ko vhodno sporočilo vsebuje več ukazov. Poddrevesi elementov *Telo* (Slika 11) v vhodnem in izhodnem sporočilu sta enaki do nivoja, na katerem se nahajajo vozlišča z ukazi. V primeru, da imamo v vhodnem sporočilu ukaz *OZZ* in je sporočilo uspešno obdelano (*UspesnostIzvedbe)*, potem ukaz *OZZ* obstaja tudi v elementu *Telo* izhodnega sporočila.

Vsak element, ki vsebuje rezultat ukaza (npr. *OZZ*), poleg rezultata vsebuje še podatke o odgovoru, shranjene v elementih:

- *IdentifikatorZahtevka* – Se posreduje znotraj vhodnega sporočila in enolično določa ukaz.
- *IdentifikatorOdgovora* – Enolično določa posamezni rezultat.
- *UspesnostIzvedbe* – Vsebuje podatke o uspešni izvedbi ukaza in v primeru neuspešne izvedbe podatke o napaki, komponenti sistema, v kateri je nastopila, in nasvet za odpravo.



Slika 11: Struktura XML ukazov in rezultatov – Vhodni podatki/VsebinskiPodatki/Telo

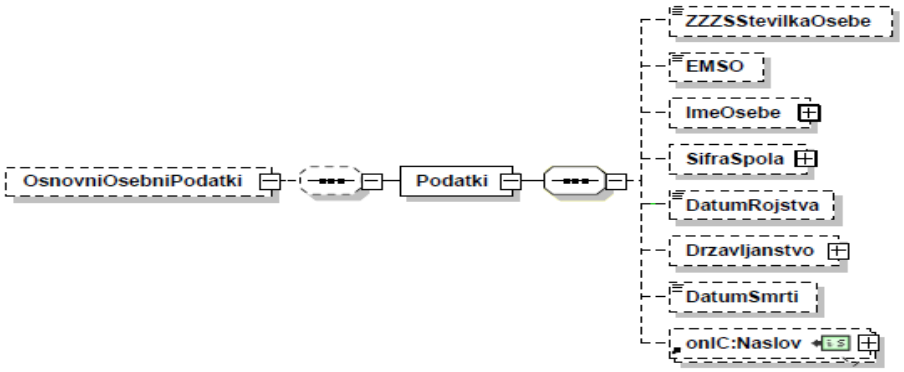
3.3.3 Branje osnovnih osebnih podatkov

Ukaz *OsnovniOsebniPodatki* (Slika 12) je namenjen pridobivanju osnovnih podatkov o zavarovani osebi. Osnovni podatki so: ZZZS številka, EMŠO, ime in priimek osebe, datum rojstva, državljanstvo, datum smrti, naslov stalnega in začasnega prebivališča.
Ta ukaz se od ostalih razlikuje v tem, da so nekateri parametri ukaza shranjeni v elementu *Telo* vhodnega sporočila.

Parametri ukaza *OsnovniOsebniPodatki* so shranjeni v podelementih elementa *Oseba*, ki je vsebovan v elementu *Glava* vhodnega sporočila.
Zavarovano osebo, za katero ukaz prebere osnovne osebne podatke, določimo z:

- ZZZS številko *(ZZZSStevilkaOsebe)*,
- številko začasnega potrdila *(StevilkaZacasnegaPotrdila)*,
- enotno matično številko občana *(EMŠO)*,
- imenom in priimkom *(ImeOsebe)* ter datumom rojstva *(DatumRojstva)*.

Glavni namen ukaza je pridobivanje ZZZS številke v primeru, ko zavarovanec nima KZZ.

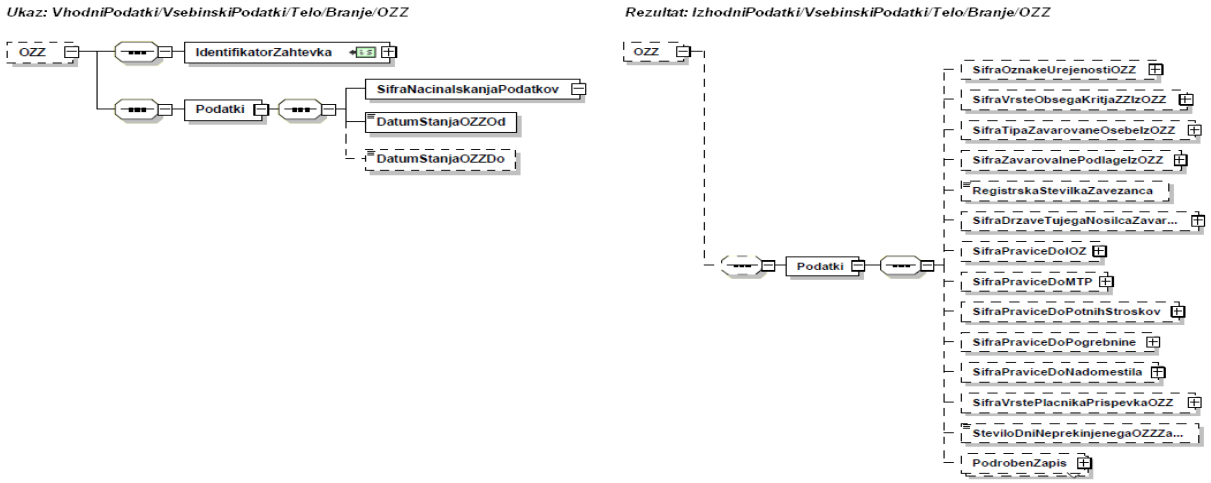


Slika 12: Struktura XML ukazov in rezultatov – Rezultat ukaza OsnovniOsebniPodatki

3.3.4 Branje podatkov o obveznem zdravstvenem zavarovanju

Preden izvajalec opravi neko storitev ali izda MTP mora preveriti osnovno zavarovanje obravnavane (OZZ*)* osebe. V nasprotnem primeru vse stroške, ne glede na obstoj zavarovanja, krije sam izvajalec.
Obstoj in kritje se preverita z ukazom *OZZ* (Slika 13), ki ima kot parametre šifro načina iskanja (*SifraNacinaIskanjaPodatkov*) in obdobje, za katero preverjamo zavarovanje (*DatumStanjaOZZod*, *DatumStanjaOZZDo*).

Obstajajo štiri načini iskanja: za datum (tekoči, pretekli dan), za preteklo obdobje, za datum smrti zavarovane osebe in za prihodnje obdobje.



Slika 13: Struktura XML ukazov in rezultatov – Ukaz za branje podatkov o OZZ

Kot rezultat ukaza dobimo vse podatke, ki so pomembni pri zaračunavanju storitev zavarovancu in ZZZS-ju. V aplikacijah, ki podpirajo poslovanje optikov in drugih dobaviteljev MTP, se uporabljajo samo naslednji podatki:

- *SifraOznakeUrejenostiOZZ* – Informacija o rednem plačevanju prispevka.
- *SifraVrsteObsegaKritjaZZIzOZZ* – Šifra določa obseg kritja stroškov. Stroški storitev in pripomočkov se lahko krijejo: v celoti, v celoti samo pri nujnih posegih ali se krije samo del.
- *SifraTipaZavarovalneOsebeIzKZZ* – Tip zavarovane osebe deli zavarovance na skupine, katerih zavarovanci imajo določen obseg kritja stroškov iz naslova OZZ (otroci, dijaki, priporniki, socialno ogroženi, ..).
- *SifraZavarovalnePodlageIzOZZ* – Šifra podlage za zavarovanje (kmet, družinski član, upokojenec, delavec, ..).
- *RegistrskaStevilkaZavarovanca* – Iz registrske številke aplikacija pridobi podatek kateri enoti ZZZS pripada zavarovanec. Podatek je pomemben za zaračunavanje storitev.
- *SifraPraviceDoMTP* – Šifra določa pravico do prejema MTP v breme zavarovanja.
- *PodrobenZapis* – Podatki o začetku in koncu obdobja v katerem je obstajalo ali obstaja veljavno zavarovanje (*DatumZacetkaVeljavnostiOZZ, DatumKoncaVeljavnostiOZZ).*

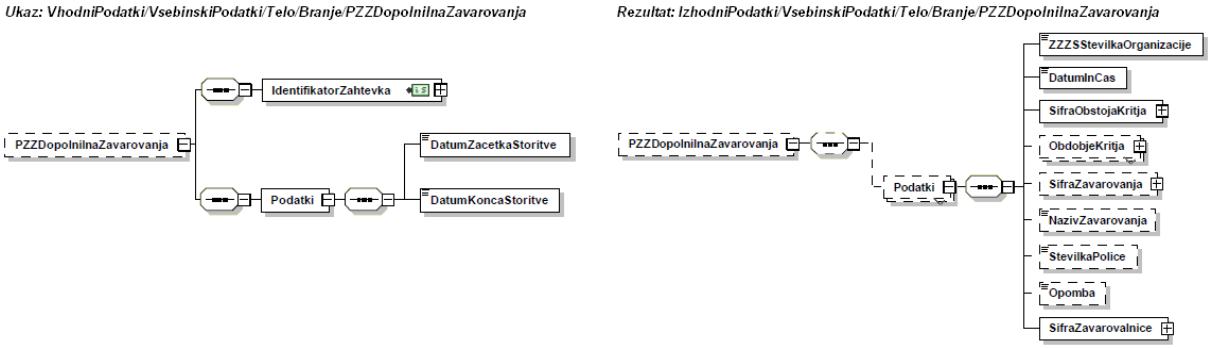
3.3.5 Branje podatkov o dopolnilnem zdravstvenem zavarovanju

Zavarovanci, katerim osnovno zavarovanje ne krije celoten strošek storitve ali pripomočka, lahko sklenejo prostovoljno dopolnilno zavarovanje ali doplačajo razliko do polne cene.
Z ukazom za branje podatkov *PZZDopolnilnaZavarovanja* (Slika 14) preverimo, če ima zavarovanec veljavno dopolnilno zavarovanje in pridobimo podatke o zavarovanju in zavarovalnici.

Ukaz vrne podatke o sklenjenem zavarovanju v obdobju. Obdobje se določi v parametrih *DatumZacetkaStoritve* in *DatumKoncaStoritve*.

Zavarovanec ima v nekem obdobju sklenjenih eno ali več zavarovalnih pogodb. Rezultat za vsako zavarovalno pogodbo (zavarovanje) vsebuje v elementih *Podatki* naslednje podatke:

- *ZZZSStevilkaOrganizacije* – Šifra organizacije v on-line sistemu.
- *SifraObstojaKritja* – Šifra kritja (Da, Ne, Karenca). Informacija, ki pove ali zavarovalnica v nekem obdobju krije stroške storitev in pripomočkov. Ko zavarovanec sklene zavarovalno pogodbo z zavarovalnico, se začne prehodno obdobje oz. karenca, v katerem zavarovalnica ne krije stroškov do preteka obdobja.
- *ObdobjeKritja* – Obdobje zavarovanja, ki opisujejo ostali podatki.
- *SifraZavarovanja* – Šifra zavarovanja.
- *NazivZavarovanja* – Naziv zavarovanja.
- *StevilkaPolice* – Številka police zavarovanja.
- *SifraZavarovalnice* – Šifra zavarovalnice (*0-AdriaticaSlovenica, 1-Vzajamna, 2-Triglav*).



Slika 14: Struktura XML ukazov in rezultatov – Ukaz za branje podatkov o PZZ

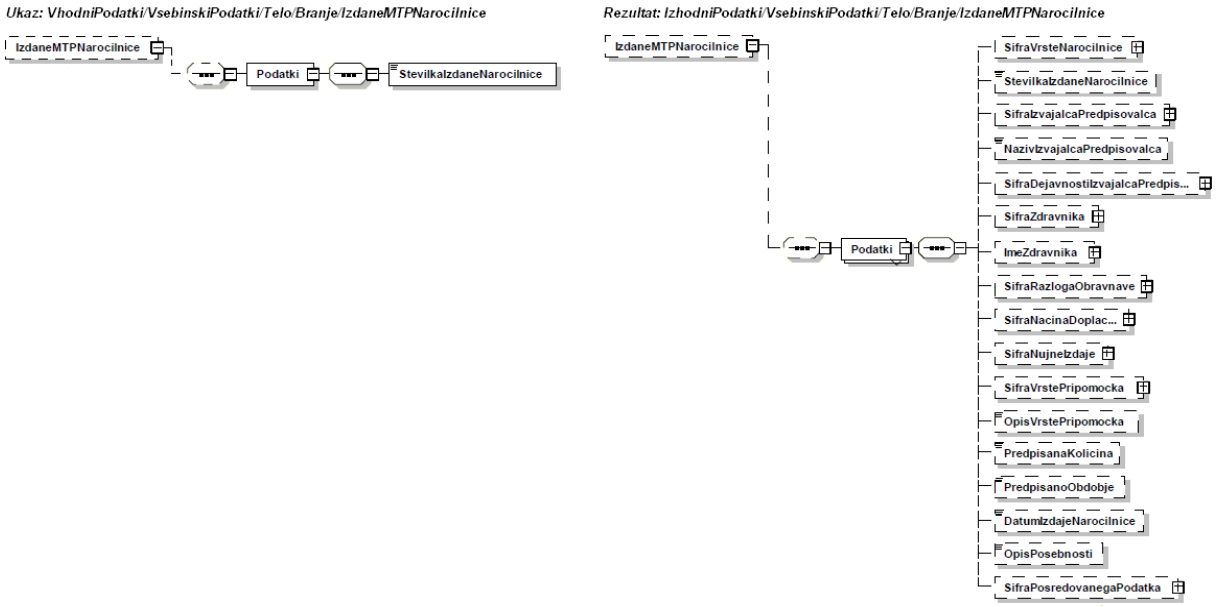
3.3.6 Branje podatkov o izdanih naročilnicah za MTP

Zdravniki specialisti predpišejo zavarovancu pripomoček tako, da v on-line sistem zapišejo naročilnico za MTP in hkrati izpolnijo predpisani papirni obrazec za naročilnico.
Da bi dobavitelj lahko prebral naročilnico, shranjeno v on-line sistemu, potrebuje izpolnjeni obrazec naročilnico, ki vsebuje številko naročilnice. Številko ustvari on-line sistem ob zapisu naročilnice in je glavni parameter (*StevilkaIzdaneNarocilnice*) ukaza *IzdaneMTPNaročilnice* (Slika 15).

Rezultat ukaza je sestavljen iz enega ali več elementov Podatki. Vsak element vsebuje podatke o enem pripomočku, predpisanem v naročilnici. Pri dobaviteljih MTP za vid so pomembne naslednje skupine podatkov iz naročilnice:

- *podatki o naročilnici*: šifra vrste (*SifraVrsteNarocilnice*), šifra vrste opravljenega posla *(SifraPosredovanegaPodatka = {Izdaja, Popravilo, Vzdrževanje})*, številka naročilnice (*StevilkaIzdaneNarocilnice*), datum izdaje (*DatumIzdajeNarocilnice*), nujnost opravljanja storitve predpisane z naročilnico (*SifraNujneIzdaje*), razlog (bolezen, poklicna bolezen, poškodba pri delu,..) obravnave (*SifraRazlogaObravnave*) in šifra načina doplačila (*SifraNacinaDoplacila= {oproščen, samoplačnik, zavarovalnica, proračun})*,
- *podatki o predpisovalcu*: ime in priimek zdravnika (*ImeZdravnika*), šifra zdravnika (*SifraZdravnika*), šifra izvajalca oz. šifra organizacije pri kateri je zdravnik zaposlen (*SifraIzvajalcaPredpisovalca*), naziv izvajalca (*NazivIzvajalcaPredpisovalca*), šifra dejavnosti izvajalca (*SifraDejavnostiIzvajalcaPredpisovalca*),
- *podatki o pripomočku:* šifra vrste pripomočka (*SifraVrstePripomocka*), opis vrste pripomočka (*OpisVrstePripomocka*), predpisana količina (*PredpisanaKolicina*), predpisano obdobje uporabe pripomočka (*PredpisanoObdobje*) in opis posebnosti (*OpisPosebnosti*).

Podatke, pridobljene s tem ukazom, uporabljamo pri zapisovanju izdaje MTP v on-line sistem ter pri zaračunavanju opravljenih storitev in dobavljenih pripomočkov.



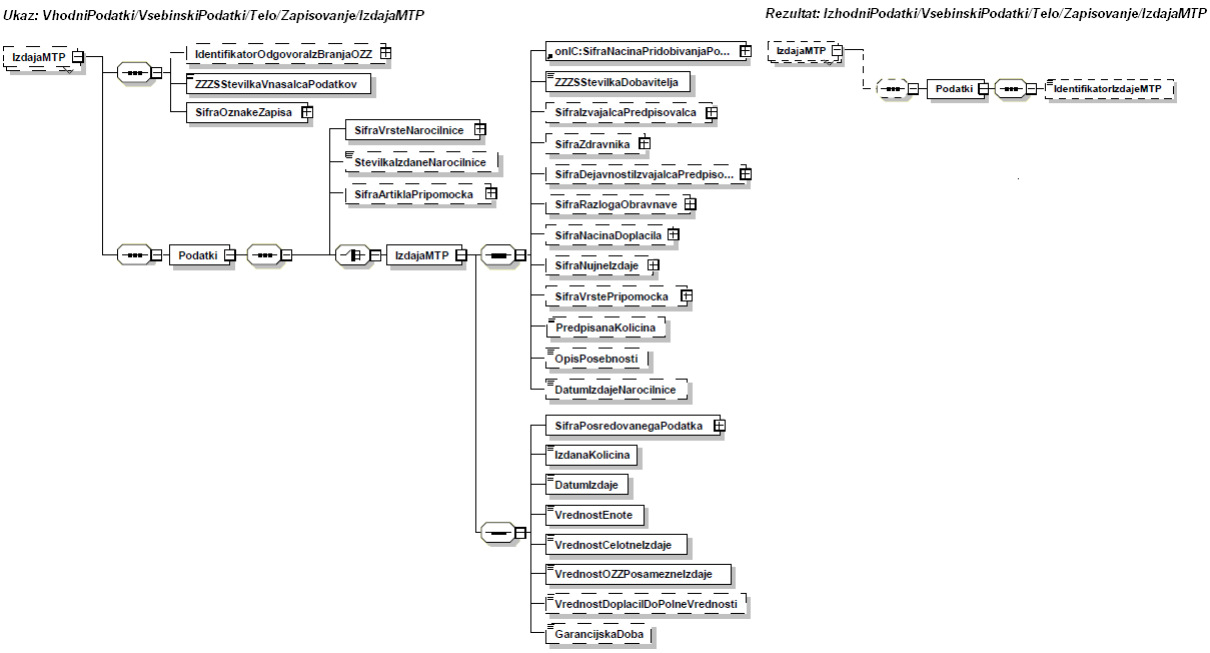
Slika 15: Struktura XML ukazov in rezultatov – Ukaz za branje podatkov o izdanih naročilnicah za MTP

3.3.7 Zapisovanje podatkov o izdanih MTP

Dobavitelj MTP za vid (optik) mora, pri izdaji pripomočka predpisanega, z naročilnico, izdajo zapisati v on-line sistem. Z ukazom *IzdajaMTP* zapišemo izdajo posameznega pripomočka. Če naročilnica vsebuje več pripomočkov, potrebujemo za vsak pripomoček en ukaz.

V ukazu *IzdajaMTP* so zapisani naslednji podatki (parametri ukaza):
- dobaviteljeva *ZZZS* številka organizacije (*ZZZSStevilkaDobavitelja*),
- identifikator pridobljen pri preverjanju obveznega zdravstvenega zavarovanja (*IdentifikatorOdgovoraBranjaOZZ*),
- šifra načina preverjanja veljavnosti OZZ (*SifraNacinaPridobivanjaPodatkovOZZ*),

- vnašalec podatkov (oseba, ki naredi zapis izdaje MTP) (*ZZZSStevilkaVnasalcaPodatkov*),
- šifra oznake zapisa, ki določa, ali se zapisuje nova izdaja MTP, ali se stornira že zapisana izdaja (*SifraOznakeZapisa*),
- številka naročilnica in vrsta naročilnice, ki označuje vrsto MTP in njihovo število v naročilnici,
- nemedicinski (poslovni) del vsebine naročilnice v primeru, da naročilnica ni zapisana v on-line sistem in ima zavarovanec samo naročilnico v papirni obliki; zapisujejo se elementi v XML Schemi (Slika 16) od *SifraIzvajalcaPredpisovalca* do *DatumIzdajeNarocilnice*; podatki so enaki podatkom pri ukazu za branje podatkov o izdanih naročilnicah za MTP,
- šifra artikla pripomočka (*SifraArtiklaPripomocka*), če pripomoček ni izdelek dobavitelja, ki je v tem primeru tudi proizvajalec,
- šifra posredovanega podatka (*SifraPosredovanegaPodatka*), ki je za izdajo MTP 1,
- datum izdaje pripomočka (*DatimIzdaje*),
- količina izdanih pripomočkov (*IzdanaKolicina*),
- cena pripomočka na enoto (*VrednostEnote*),
- vrednost izdanih pripomočkov (*VrednostCelotneIzdaje*),
- znesek oz. del vrednosti pripomočkov, kateri plača OZZ (*VrednostOZZPosamezneIzdaje*),
- znesek oz. del vrednosti pripomočkov (*VrednostDoplacilDoPolneVrednosti*), kateri se za socialno ogrožene osebe in pripornike plača iz proračuna RS;
  v primeru, da ima zavarovanec pravico do zneska v breme proračuna, potem velja *VrednostDoplacilDoPolneVrednost=VrednostCelotneIzdaje-VrednostOZZPosamezneIzdaje*,
- garancijska doba (*GarancijskaDoba*).



Slika 16: Struktura XML ukazov in rezultatov – Ukaz za zapisovanje podatkov o izdanih MTP

Ukaz pri uspešnem zapisu izdaje vrne kot rezultat identifikator izdaje (*IdentifikatorIzdajeMTP*), ki zagotavlja plačilo storitve ali pripomočka s strani zavarovalnic za OZZ in PZZ.

Storniranje zapisa izdaje se naredi z enakim ukazom in parametri z razliko, da ima pri storniranju parameter šifra oznake zapisa (*SifraOznakeZapisa*) vrednost 2.

Ko storniramo eno izdajo pripomočka za neko naročilnico, on-line sistem avtomatično stornira vse izdaje pripomočkov za to naročilnico.

3.4 Komuniciranje aplikacije z on-line sistemom

Programske knjižnice IHIS (*ihis2.dll*) vsebujejo vmesnik za dostop do on-line sistema, branje in zapisovanje podatkov v on-line sistem ter branje in zapisovanje podatkov na kartico (KZZ), ki se nahaja v čitalcu kartic, priklopljenemu na lokalni računalnik. Vmesnik skrbi za samo komunikacijo in oblikovanje SAOP sporočil, tako da je naloga razvijalca samo priprava in pošiljanje ustreznih ukazov ter razčlenjevanje odgovora.

Predpogoji za uporabo knjižnice na lokalnem računalniku so:
- nameščen eden izmed operacijskih sistemov MS Windows (2000, XP, Vista, Windows 7 in vsi strežniški OS),
- nameščen NET framework 2.0 ali višji,
- čitalnik kartic: Gemplus GCR700 (gonilnik 1.0.0.3), Xiring Prium 3S ali Gemalto GCR 5500-D,
- Gemalto classic client 5.1.7,
- nameščeno korensko potrdilo POŠTA®CA.

Knjižnico lahko uporabimo v vseh programskih jezikih, ki omogočajo nalaganje (*LoadLibrary*) dinamično deljene knjižnice v času izvajanja ali statično povezavo (angl. static link) pri izdelavi izvršljive kode programa. (Microsoft Visual Studio (C#, C++, Basic, FoxPro), Embarcadero Delphi, Elipse Java, ..)
Pri svojem delovanju knjižnica uporablja lastno konfiguracijsko XML datoteko IHIS2config.xml, ki se mora nahajati v istem direktoriju kot datoteka z izvajalno kodo programa.

3.4.1 Funkcije knjižnice IHIS

Vse operacije, ki jih izvajajo spletne storitve on-line sistema, so operacije tipa zahteva-odgovor (angl. Request-response). Pri vsaki operaciji odjemalec pošlje vhodno sporočilo (zahtevo) strežniku, za katerega ta vrne izhodno sporočilo (odgovor).
Na podoben način uporabljamo tudi funkcije iz programskih knjižnic. Pri vsaki posamezni operaciji ustrezno funkcijo pokličemo dvakrat zapored. Pri prvem klicu funkcije v parametru *xmlData* pošljemo vhodno sporočilo (zahtevo). Drugi klic funkcije vrne izhodno sporočilo (odgovor) v parametru *xmlData.*

Programske knjižnice vsebujejo naslednje funkcije:
- *int* Login*([in] string* servername*, [in] int* port*, [out] string xmlData, [out] int* size*)*
  Kreira sejo s PK kartico (zahteva vnos PIN-a) in TLS povezavo z vstopno točko. V vstopni točki se za uporabnika s PK kartico preveri istovetnost identitete (angl. authentication) in določijo pooblastila uporabnika (angl. authorisation). Pred uporabo ostalih funkcij programske knjižnice v on-line sistemu, je obvezna uspešna izvedba funkcije *Login.* Nekatere funkcije delujejo tudi *lokalno*: lokalni računalnik, čitalnik kartic in KZZ. Pri lokalni uporabi ni obvezna prijava pred klicem funkcije.

- *int* Logout *([in] int* handle*, [out] string* xmlData*, [out] int* size*)*
  Prekine TLS sejo in deaktivira profesionalno kartico.

- *int* GetData *([in] int* handle*, [in][out] string xmlData, [out] int* size*)*
  Omogoča pridobitev podatkov iz on-line sistema. Uporablja se pri pošiljanju zahtev z ukazi, s katerimi povprašujemo po določenih podatkih. Pri klicu funkcije v parametru *xmlData* podamo XML sporočilo, ki predstavlja zahtevo. Funkcija po izvedbi vrne v parametru *xmlData* odgovor ali sporočilo o napaki.

- *int* SetData *([in] int* handle*, [in][out] string* xmlData*, [out] int* size*)*
  Zapis podatkov v on-line sistem. Uporablja se pri zahtevah z ukazi za katere on-line sistem zapiše posredovane podatke. Pri klicu funkcije v parametru *xmlData* podamo XML sporočilo, ki predstavlja zahtevo za zapisovanje podatkov. Funkcija po izvedbi vrne v parametru *xmlData* odgovor o uspešnem zapisovanju ali sporočilo o napaki.

- *int* GetKZZ *([in] int* handle*, [out] string* xmlData*, [out] int* size*)*
  Branje osnovnih podatkov o zavarovancu iz KZZ kartice. Po vzpostavitvi on-line sistema se uporablja samo za branje ZZZS številke zavarovanca. Pri lokalni uporabi profesionalna kartica (PK) in povezava z vhodno točko nista obvezni.

- *int* SetMTP *([in] int* handle*, [in][out] string* xmlData*, [in][out] int* size*)*
  Zapiše podatke o medicinsko tehničnih pripomočkih (MTP) na KZZ kartico.
  Z vzpostavitvijo on-line sistema se funkcija več ne uporablja, ker kartica ne služi več kot medij za prenos podatkov.

- *int* GetMTP *([in] int* handle*,[out] string xmlData, [out] int* size*)*
  Branje podatkov o MTP iz KZZ kartice. Z vzpostavitvijo on-line sistema se funkcija več ne uporablja, ker kartica ne služi več kot medij za prenos podatkov.

- *int* VerifySystem *([in] int* verifycationType*, [out]* xmlData*, [out] int* size*)*
  Preveri delovanje lokalnih in centralnih komponent sistema (vstopna točka in zaledni sistemi).

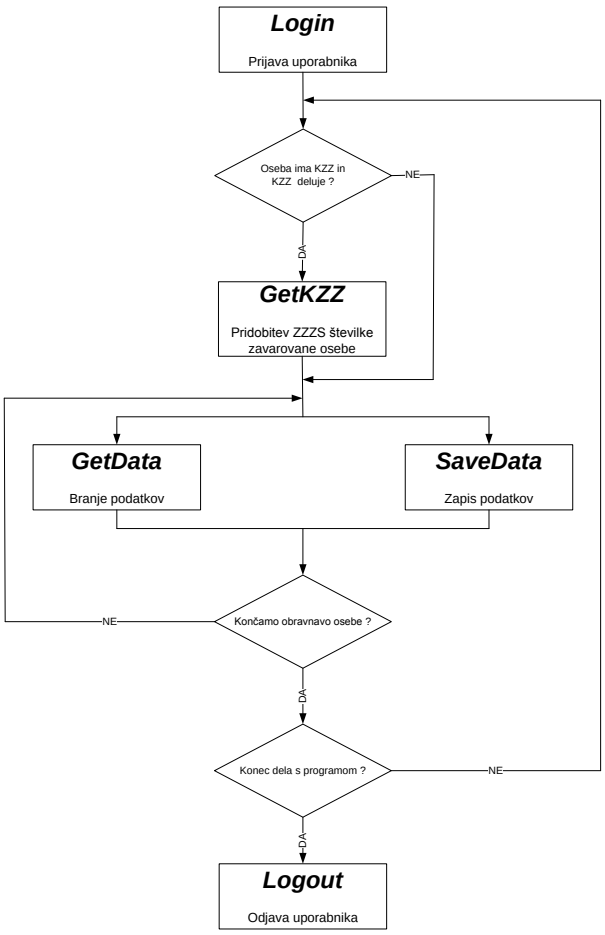
Parametri funkcij:

- *servername* – URL naslov vstopne točke. Če *servername* ni določen (NULL), pridobi privzeto vrednost iz konfiguracijske datoteke (*IHIS2config.xml*).
- *port* – Številka vrat za komunikacijo z vstopno točko. Če se postavi vrednost –1, funkcija prebere prevzeto vrednost iz konfiguracijske datoteke.
- *xmlData*
  Vhod [in]: XML sporočilo z vhodnimi podatki (ukazom).
  Izhod [out]: XML sporočilo z odgovorom (rezultatom) za posredovano zahtevo ali sporočilo o napaki in navodila za odpravo napak.
- *size* – Število Unicode znakov v parametru *xmlData*. Pri klicu funkcije podamo število znakov v vhodnem parametru *xmlData*. Funkcija po izvedbi v tem parametru vrne število znakov v odgovoru, shranjenem v parametru *xmlData.*
- *verifycationType* – Tip preverjanja.
  0 Preveri se samo lokalni sistem (čitalnik, gonilniki, …).
  1 Preveri povezljivost do vstopne točke in delovanje bistvenih komponent strežnika.
  2 Preveri lokalni sistem in strežnik.
  3 Preveri informacijo o delovanju sistema na spletnem strežniku ZZZS.
  4 Preveri lokalni sistem in informacijo o delovanju sistema na spletnem strežniku ZZZS.

Vse funkcije imajo naslednje vrednosti, ki so odvisne od uspešnosti izvedbe funkcije:
> 0 – Funkcija se je uspešno izvedla.
= 0 – Neuspešno izvedena funkcija. Funkcija vrne v parametru *xmlData* podatke o napaki.

3.4.2 Zaporedje klicanja funkcij pri komunikaciji z on-line sistemom

Na začetku (Slika 17) se prijavimo v on-line sistem s klicem funkcije *Login*.
V naslednjem koraku moramo pridobiti ZZZS številko zavarovanca, ki je shranjena na kartici zdravstvenega zavarovanja. To storimo s funkcijo *GetKZZ*. Alternativni način za pridobivanje ZZZS številke v primerih, ko zavarovanec nima KZZ ali ta ne deluje, je posredovanje ukaza za branje osnovnih osebnih podatkov v on-line sistem. Ukaz posredujemo tako, da ga podamo v parametru *xmlData* pri klicanju funkcije *GetData*. Ukazi in rezultati (odgovori on-line sistema) so podrobno opisani v podpoglavju 3.3.
Ko smo pridobili ZZZS številko, nadaljujemo z zanko, v kateri kličemo funkcije *GetData* in *SetData*, odvisno od tega ali želimo podatke pridobiti iz on-line sistema ali jih zapisati v on-line sistem.
Če želimo obravnavati novega zavarovanca, se moramo vrniti na branje ZZZS številke zavarovanca iz KZZ.
Preden končamo z delom, se odjavimo s funkcijo *LogOut*.



Slika 17: Zaporedje klicanja funkcij programskih knjižnic IHIS

3.4.3 Uporaba knjižnic v programskem jeziku C++

V tem podpoglavju je opisana uporaba knjižnic v programskem jeziku Microsoft Visual C++. Primer opisuje prijavljanje v on-line sistem, pridobivanje ZZZS številke zavarovanca shranjene na KZZ in pridobivanje osebnih podatkov o zavarovancu iz on-line sistema.
Iz programske kode so zaradi večje preglednosti izpuščeni vsi ukazi, ki niso pomembni za sam opis uporabe.

Knjižnico naložimo v času izvajanja z ukazom *LoadLibrary* ali jo pri kodiranju uvozimo z direktivo *#import* in jo statično povežemo pri izdelavi izvršljive kode programa.

```
HINSTANCE hDLL;
CString lsDDLPath=lsCurrentDir+_T("\\DLL2.6.7\\ihis2.dll");
hDLL = LoadLibrary(lsDDLPath);
if (hDLL == NULL)      //Napaka. Nalaganje knjižnice neuspešno.
        return false;
```

Pridobimo naslov funkcije *Login* iz knjižnice (*ihis2.dll*) in ga shranimo v kazalec na funkcijo.

```
typedef int (*LPFNDLL_Login)(LPTSTR* servername,int port ,LPTSTR xmlData ,int &size );
LPFNDLL_Login pLogin;
pLogin = (LPFNDLL_Login)GetProcAddress(hDLL,"Login");
if (pLogin==NULL)      //Napaka
        return false;
```

Pokličemo funkcijo *Login*.

```
BOOL lbTryError=false;
iHandleToZZZS_ON_Line = (pLogin)(NULL, -1, NULL, liCharCount);
```

Če se prijava neuspešno izvede, funkcija vrne rezultat 0. Da bi pridobili informacijo o napaki, moramo še enkrat poklicati funkcijo *Login*. Zaporedje znakov na katere kaže kazalec *lpPathBuf*, predstavlja XML sporočilo s podatki o napaki in navodili za odpravo napak.

```
if (iHandleToZZZS_ON_Line<=0)
{
        LPTSTR lpPathBuf=(LPTSTR) calloc((2*liCharCount)+1,sizeof(TCHAR));
         (pLogin)(NULL, -1, lpPathBuf, liCharCount);
        ....
        // Izpišemo obvestilo o napaki
        free(lpPathBuf);
        return false
}
```

Pokličemo funkcijo GetKZZ, s katero pridobimo osebne podatke o zavarovancu, shranjene na KZZ. Funkcija vrne vrednost, večjo od 0, če so uspešno pridobljeni podatki.

```
liRezultat=(pGetKZZ)(iHandleToZZZS_ON_Line, NULL, liCharCount);
```

Ne glede na rezultat funkcije to še enkrat pokličemo, da dobimo podatke o osebi ali podatke o napaki.

```
LPTSTR lpPathBuf=(LPTSTR) calloc((2*liCharCount)+1,sizeof(TCHAR));
(pGetKZZ)(iHandleToZZZS_ON_Line, lpPathBuf, liCharCount);
free(lpPathBuf);
```

Če je bil prvi klic funkcije uspešen, pridobimo iz odgovora (*lpPathBuf*) v obliki XML sporočila ZZZS številko zavarovanca.
V naslednjem koraku pošljemo s klicem funkcije *GetData* zahtevo, za katero nam on-line sistem vrne osebne podatke o zavarovani osebi. Zahteva je XML sporočilo (*lptCommand*), ki je oblikovano glede na veljavno XML Schemo.

```
liRezultat=(pGetData)(iHandleToZZZS_ON_Line,lptCommand, liCharCount);
```

Ne glede na rezultat funkcije to še enkrat pokličemo, da dobimo zahtevane podatke ali podatke o napaki.

```
(pGetData)(iHandleToZZZS_ON_Line, lpPathBuf, liCharCount);
```

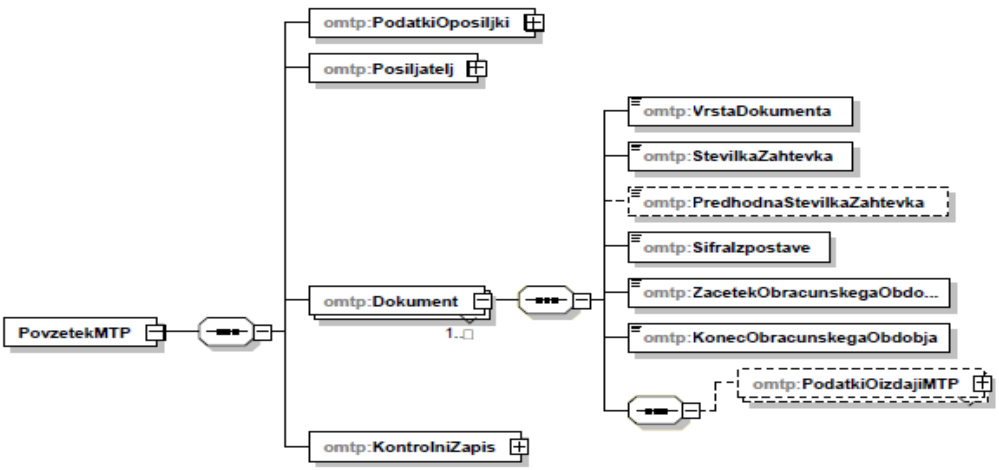
Če je bil prvi klic funkcije uspešen, pridobimo osebne podatke o zavarovani osebi iz razčlenjenega XML sporočila odgovora (*lpPathBuf*).

Preden zaključimo uporabo on-line sistema obvezno pokličemo funkcijo *Logout* in s tem prekinemo TLS sejo do vstopne točke in deaktiviramo PK.

```
liRezultat= (pLogOut)(iHandleToZZZS_ON_Line, NULL, liCharCount);
```

3.5 Računalniško izmenjevanje podatkov

Izvajalec na koncu obračunskega obdobja izstavi zavarovalnicam enega ali več zahtevkov za plačilo. Zahtevek je dokument, katerega namen je, da se z njim zahteva plačilo celotne ali dela vrednosti opravljenih storitev in dobavljenih pripomočkov. Vsebuje vse obvezne sestavine računa.



Slika 18: Povzetek o izdanih medicinsko tehničnih pripomočkov

Izvajalec po elektronski pošti kot prilogo zahtevkom pošlje XML dokument "Povzetek o izdanih medicinsko tehničnih pripomočkih".

Pri vsaki izdaji MTP se vrednost in podatki o pripomočkih zapišejo v enega ali več zahtevkov. Odvisno je od tipa zavarovanca in obstoja dopolnilnega zavarovanja. En zahtevek vsebuje podatke o vseh izdajah MTP v nekem obdobju, katerega vrednost ali del vrednosti plača zavarovalnica.

Povzetek o izdanih medicinsko tehničnih pripomočkih je XML dokument (*PovzetekMTP*), ki vsebuje podatke o pošiljki, pošiljatelju, dokumente in kontrolni zapis (Slika 18).
V elemente *Dokument* so zapisani podatki o posameznem zahtevku in podatki o vseh izdanih MTP, za katere zahtevamo plačilo z zahtevkom.

4. Programska oprema za zajem in posredovanje podatkov v sistem "On-line zdravstveno zavarovanje"

V tem poglavju je opisan potek razvoja in glavne funkcionalnosti izdelane programske opreme. Opis razvoja je razdeljen po postopkih metodologije RUP: poslovno modeliranje, zajem zahtev, analiza in načrtovanje, implementacija, testiranje in postavitev. Za razvoj po metodologiji RUP sem se odločil, ker omogoča iterativen in inkrementalen razvoj, s katerim čim prej pridemo do začetne delne rešitve. V teku razvoja delne rešitve dopolnjujemo, dokler ne pridemo do končne rešitve. Delne rešitve so omogočile boljšo komunikacijo z naročnikom in čimprejšnje ugotavljanje rizikov in napak, ki so se pojavljale v teku razvoja.
Kot rezultat razvoja po tej metodologiji sem pridobil delujočo programsko opremo in dokumentacijo, ki bo omogočala nadaljnji razvoj in vzdrževanje.

4.1 Poslovno modeliranje

Cilji poslovnega modeliranja so: seznanitev z organizacijo naročnika, razumevanje trenutnih problemov in odkrivanje možnih izboljšav. Rezultati tega postopka olajšajo zajem zahtev in analizo.

4.1.1 Opis problemskega področja

Dobavitelj medicinsko tehničnih pripomočkov za vid (OPTIK) mora pri svojem poslovanju z ZZZS izvajati predpisana poslovna pravila, ki vključujejo zajem in posredovanje podatkov v "On-line zdravstveno zavarovanje". Pri obravnavi vsakega zavarovanca (kupca), kateremu se izdajo pripomočki za vid, ki jih v celoti ali delno plača ZZZS in zavarovalnice za prostovoljno zavarovanje, se mora izvesti vrsta zelo zahtevnih postopkov. Najprej se morajo pridobiti splošni podatki o zavarovancu. Potem se na osnovi pridobljenih podatkov o tipu zavarovane osebe preverja veljavnost sklenjenega zavarovanja. V naslednjem koraku se iz sistema pridobiva naročilnica, ki vsebuje predpisane pripomočke. Obravnava kupca se konča, ko se izdane pripomočke, skupaj z njihovo ceno, zapiše v sistem.
Po preteku obračunskega obdobja se vse izdaje obračunajo in pošljejo zahtevki za plačilo izpostavam ZZZS in zavarovalnicam za prostovoljna zdravstvena zavarovanja. Dobaviteljevo delo se konča s pripravo in posredovanjem XML datoteke, ki vsebuje vse izdane pripomočke.

Brez ustrezne rešitve, ki avtomatizira vse poslovne procese, naročnik ne more uspešno poslovati, ker:

- so stroški izvajanja samih poslovnih pravil večji od vrednosti izdanih pripomočkov,
- ne dobi plačila za izdajo napačnih pripomočkov,
- ne dobi plačila za izdajo pripomočkov zavarovancu, ki nima urejenega zavarovanja,
- v primeru, da na koncu obdobja posreduje napačne podatke o izdaji pripomočkov ali pripravi napačne zahtevke za plačilo, pride do podaljšanja roka plačila,
- zaposleni ne morejo ročno izvajati zelo zahtevnih postopkov brez napak.

4.1.2 Slovar

a) **Dobavitelj MTP za vid**
   Optik, ki ima sklenjeno pogodbo z ZZZS.

b) **Konvencija**
Meddržavni sporazum o socialnem zavarovanju (konvencija).
c) **NAR-2**
Listina Naročilnica za pripomoček za vid.
d) **Obračunsko obdobje**
Za vsako obdobje dobavitelj MTP za vid naredi obračun izdanih pripomočkov v tem obdobju. Obračunsko obdobje se začne prvi dan v mesecu in konča zadnji dan meseca.
e) **Potrdilo**
Potrdilo, ki nadomešča KZZ (papirni dokument, ki ga izda ZZZS).

4.1.3 Vizija

Naročnikova vizija projekta je izdelava rešitve, ki bo omogočila zajem in posredovanje podatkov v sistem "On-line zdravstveno zavarovanje" in avtomatizacijo vseh poslovnih procesov pri poslovanju z ZZZS in zavarovalnicami za prostovoljna zavarovanja.

4.1.4 Cilji

Glavni cilji so:
- pravilno izvajanje predpisanih poslovnih pravil pri poslovanju z ZZZS,
- zmanjšanje stroškov poslovanja,
- zmanjšanje napak pri poslovanju s kupci in ZZZS,
- povečanje kakovosti storitev,
- povečanje zadovoljstva kupcev.

4.1.5 Pravila

Poslovni procesi morajo upoštevati naslednja pravila:
- Kupec (zavarovanec) mora pri obravnavi imeti veljavno kartico zdravstvenega zavarovanja ali potrdilo in naročilnico s predpisanimi pripomočki.
- Veljavnost naročilnice je 30 dni.
- V primeru, da naročilnica ni vpisana v sistem "On-line zdravstveno zavarovanje" mora dobavitelj zapisati naročilnico v sistem. Kot številka naročilnice se uporabi številka, natisnjena na listini NAR-2.
- V primeru nedelovanja KZZ ali uporabe potrdila sistemu "On-line zdravstveno zavarovanje" se mora sporočiti namen in razlog dostopa brez KZZ.
- Podatki o zavarovancu in zavarovanju so veljavni od trenutka, ko so prebrani, do konca dneva (do 24 h).
- Zavarovana oseba ima lahko osnovno in prostovoljno zdravstveno zavarovanje.
- Pri zapisovanju izdaje pripomočkov v sistem "On-line zdravstveno zavarovanje" se obvezno zapisujejo identifikatorji odgovora, pridobljeni pri zajemu podatkov o zavarovanjih zavarovane osebe.
- ZZZS s pogodbo določi ceno posameznega pripomočka standardne kakovosti. Znesek izdanih pripomočkov se deli na dva dela, katerih razmerje določi ZZZS. Prvi del plača ZZZS. Zavarovalnica za prostovoljna zavarovanja plača drugi del zneska pod pogojem, da ima zavarovanec veljavno zavarovanje.
  Pravilo ne velja v primeru:
  - Če je oseba: otrok, učenec, dijak, študent, duševno ali telesno prizadeta oseba. Tedaj ZZZS plača celoten znesek pripomočka standardne kakovosti.

- Če je oseba: socialno ogrožena, pripornik ali obsojenec. V tem primeru drugi del plača ZZZS v breme proračuna Republike Slovenije.

Če dobavitelj izda na naročnikovo zahtevo nadstandardni pripomoček, mora naročnik doplačati razliko v ceni.

- Za izdane pripomočke mora dobavitelj MTP za vid pripraviti in poslati zahtevke za plačilo ZZZS in zavarovalnicam za prostovoljno zavarovanje. Zahtevki se morajo poslati do 15. v mesecu za prejšnje obračunsko obdobje.
- Zahtevek vsebuje zneske po kategorijah zavarovanja in skupni znesek za plačilo. Vsaki zavarovalnici se izstavi zahtevek za plačevanje ustreznega dela vrednosti pripomočkov, ki so izdani njenim zavarovancem.

  Zahtevki, ki se izstavljajo ZZZS in Vzajemni d.d. (zavarovalnica za prostovoljno zavarovanja) morajo biti izstavljeni tako, da vsak posamezni zahtevek vsebuje samo obveznosti ene izpostave ZZZS.

  Eni izpostavi ZZZS se lahko izstavi tudi več zahtevkov za eno obračunsko obdobje, ker se morajo izdani pripomočki obračunati ločeno glede na tip zavarovanja:
  - socialno ogroženi, priporniki in obsojenci (kritje razlik iz proračuna RS),
  - osebe zavarovane po konvenciji (za vsako konvencijo svoj zahtevek),
  - osebe, ki imajo tip zavarovanja, različen od prvih dveh tipov (ostalo).
- Kopije zahtevkov je treba poslati računovodskemu servisu do 20. v mesecu za prejšnje obračunsko obdobje.
- Dobavitelj MTP za vid mora poslati na ZZZS povzetek o izdanih medicinsko tehničnih pripomočkih.

4.1.6 Akterji

a) Kupec – zavarovanec

Oseba, kateri se na osnovi naročilnice izdajo predpisani pripomočki.

b) ZZZS

Zavod za zdravstveno zavarovanje Slovenije v sistem posreduje:

- šifre novih pripomočkov in njihove cene,
- spremembe cen pripomočkov,
- razmerje kritja stroškov izdanih pripomočkov med ZZZS in zavarovalnicami za prostovoljno zavarovanje.

Iz sistema mora pridobiti:

- zahtevke za plačilo,
- XML datoteko, ki vsebuje povzetek o izdanih medicinsko tehničnih pripomočkih.

c) ZPZZ

Zavarovalnice za prostovoljno zdravstveno zavarovanje iz sistema pridobivajo zahtevke za plačilo.

d) On-line sistem

Sistem "On-line zdravstveno zavarovanje" je storitev, ki omogoča:

- pridobivanje podatkov o zavarovancu in zavarovanju,
- pridobivanje podatkov o naročilnicah s predpisanimi pripomočki,
- shranjevanje podatkov o izdanih pripomočkih.

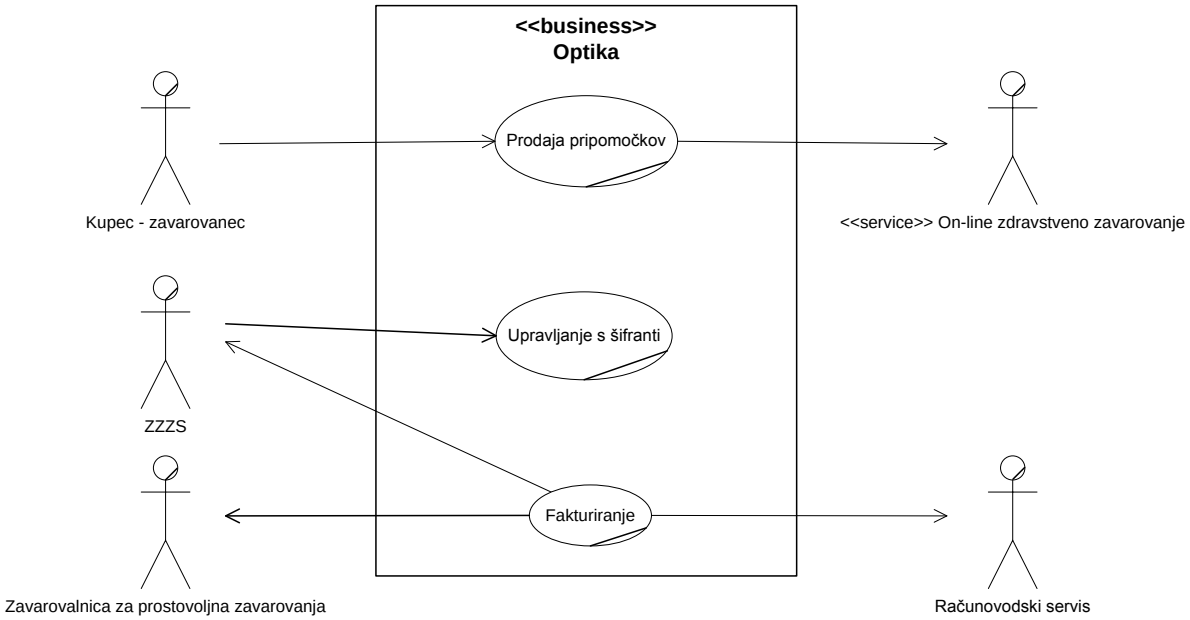
e) Računovodski servis
Zunanjemu računovodskemu servisu se iz sistema posredujejo zahtevki za plačilo.

4.1.7 Primeri uporabe

a) Prodaja pripomočkov
Poslovni proces, v katerem se kupcu-zavarovancu prodajajo pripomočki za vid. Iz OLZZ sistema se pridobijo vsi potrebni podatki za obravnavo kupca in obračun izdanih pripomočkov po predpisanih pravilih ZZZS (Slika 19).

b) Upravljanje s šifranti
Poslovni proces omogoča shranjevanje novih in spremembo obstoječih podatkov v šifrantih, ki so potrebni pri izvajanju vseh ostalih poslovnih procesov.
Skrbi za naslednje šifrante:
- Izpostave ZZZS,
- Območne enote,
- Zavarovalnice,
- Pripomočke,
- Razmerje OZZ/DOPZ.

c) Fakturiranje
Poslovni proces zajema obračunavanje izdanih pripomočkov oz. izdelavo zahtevkov za plačilo in povzetka o izdanih medicinsko tehničnih pripomočkih.



Slika 19: Diagram poslovnih primerov uporabe

4.1.8 Entitete

a) Naročilnica

Entiteta vsebuje splošne podatke o kupcu, predpisane pripomočke za vid in podatke o izdanih pripomočkih.

Naročilnica je generalizacija entitet:

- *Naročilnica-Nar2*
  Listina Naročilnica za pripomoček za vid.
- *Naročilnica-OL*
  Naročilnica shranjena v sistemu OLZZ.
- *Naročilnica-Int*
  Entiteta nastane znotraj sistema. Vsebuje podatke, pridobljene iz Nar2, in podatke pridobljene pri izdaji pripomočka.

b) Zahtevek

Zahtevek je dokument, katerega namen je, da se z njim zahteva plačilo celotne ali dela vrednosti opravljenih storitev in dobavljenih pripomočkov. Vsebuje vse obvezne sestavine računa.

c) Šifrant pripomočkov

Šifrant vsebuje splošne podatke o pripomočkih, vključno s predpisano ceno za pripomoček standardne kakovosti.

d) Razmerje OZZ/DOPZ

Entiteta vsebuje podatke, ki določajo razmerje obveznosti ZZZS in zavarovalnic za prostovoljna zavarovanja pri plačevanju vrednosti izdanega pripomočka.

4.1.9 Delavci

a) Prodajalec

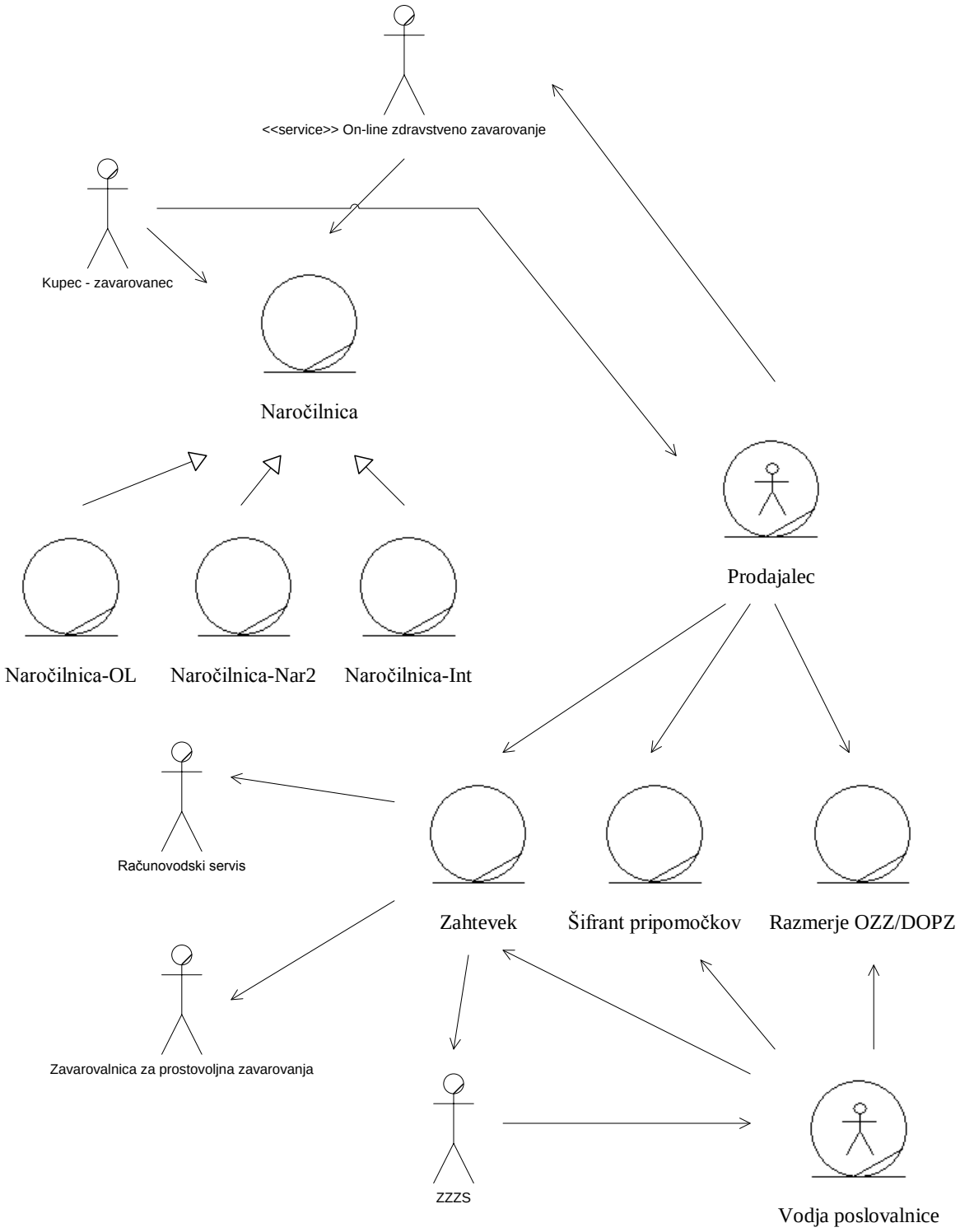
Oseba skrbi za: obravnavo kupcev, prodajo pripomočkov in evidentiranje prodanih pripomočkov.

b) Vodja poslovalnice

Naloge vodje poslovalnice so: obračunavanje izdanih pripomočkov (fakturiranje), skrb za podatke, potrebne pri izvajanju poslovnih procesov (upravljanje s šifranti) in pravočasno posredovanje zahtevkov zavarovalnicam in računovodskemu servisu.

4.1.10 Analitični model

Razredni diagram poslovnega analitičnega modela (Slika 20) opisuje razmerje med akterjimi, entitetami in delavcem.



Slika 20: Razredni diagram poslovnega analitičnega modela

4.2 Zajem zahtev

V tem postopku metodologije RUP moramo določiti funkcionalne in nefunkcionalne zahteve, primere uporabe in model primerov uporabe sistema, ki ga želimo zgraditi.

4.2.1 Funkcionalne zahteve

Sistem mora biti izdelan tako, da omogoča naslednje zahteve:

- vnos, pridobivanje iz sistema OLZZ, shranjevanje in iskanje podatkov o kupcu, vključno s podatki o njegovem zavarovanju,
- vnos, pridobivanje iz sistema OLZZ in posredovanje podatkov v sistem OLZZ, shranjevanje in iskanje naročilnic,
- izdelava in tiskanje zahtevkov,
- priprava XML dokumenta s povzetkom o izdanih MTP;
- zaključevanje in izpis vseh zahtevkov za obračunsko obdobje;
- upravljanje s šifranti (zavarovalnice, območne enote ZZZS, izpostave ZZZS, pripomočki, razmerje OZZ/DOPZ).

4.2.2 Nefunkcionalne zahteve

Poleg funkcionalnih sistem mora izpolnjevati tudi naslednje nefunkcionalne zahteve:

- omejevanje dostopa uporabnikom do posamezne funkcionalnosti sistema,
- avtomatska posodobitev programske opreme,
- shranjevanje sprejetih in poslanih sporočil v sistem "On-line zdravstveno zavarovanje",
- zanesljivost in varnost sistema,
- večuporabniško delo.

4.2.3 Model primerov uporabe sistema

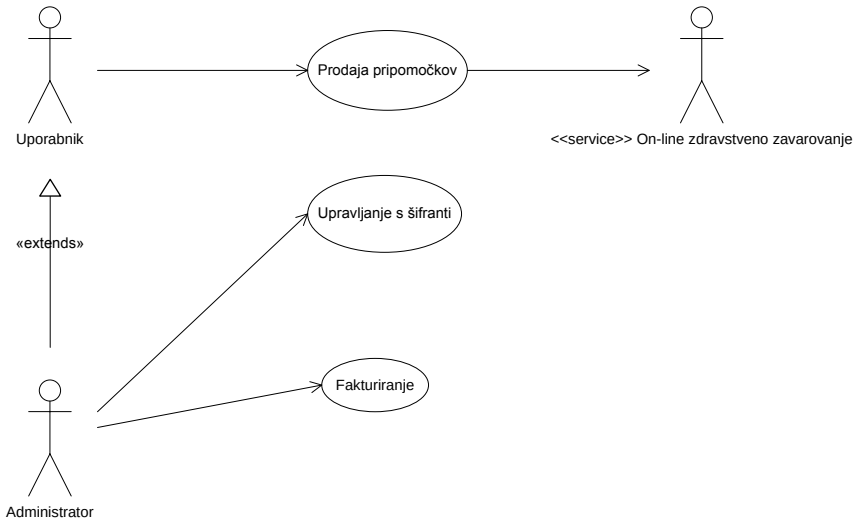
4.2.3.1 Akterji

a) Uporabnik

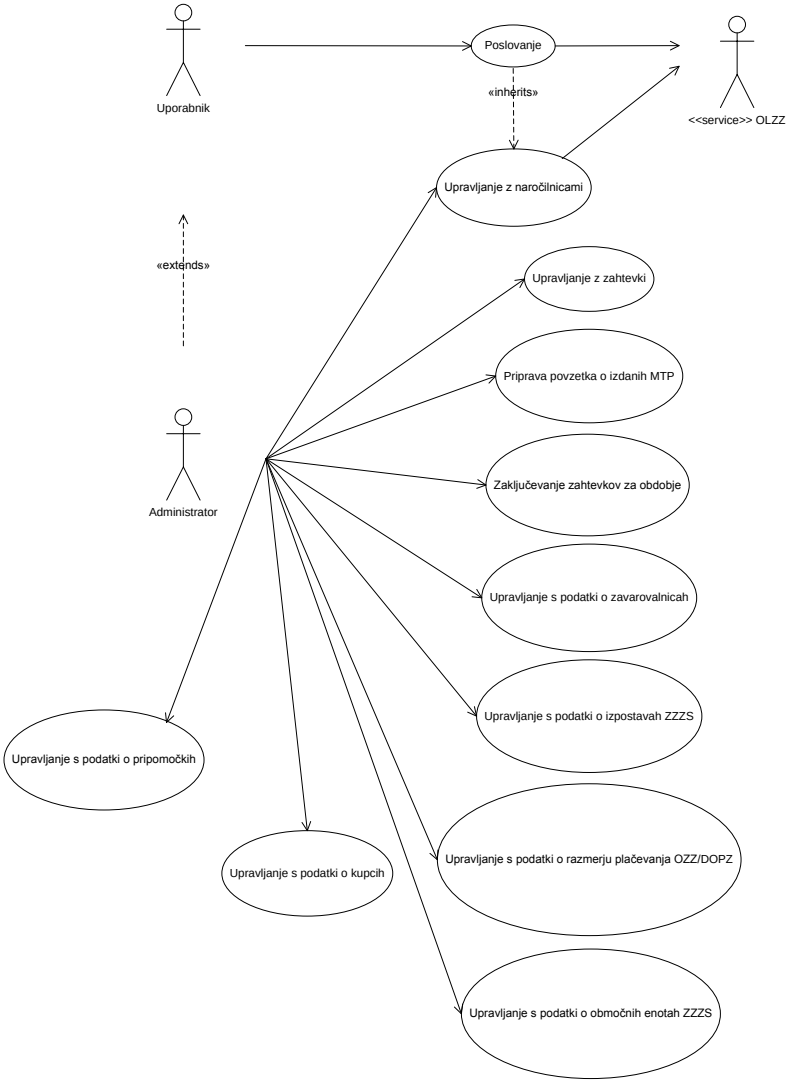
Uporabnik je oseba opredeljena v poslovnem modeliranju kot poslovni delavec: Prodajalec. Uporabnik kupcem prodaja pripomočke (poslovanje) in skrbi za pravilno pravilen vnos in zapisovanje naročilnic v sistem "On-line zdravstveno zavarovanje" (upravljanje z naročilnicami).

b) Administrator

Administrator je oseba opredeljena v poslovnem modeliranju kot poslovni delavec: Vodja poslovalnice. Ima pravico do uporabe vseh funkcij sistema.



Slika 21: Skupni diagram primerov uporabe



Slika 22: Razčlenjeni diagram primerov uporabe

4.2.4 Opis primerov uporabe sistema

V tem podpoglavju so predstavljeni vsi primeri uporabe sistema (Slika 21 in 22). Podrobno sta opisana samo dva ključna primera uporabe; *Poslovanje* in *Upravljenje z naročilnicami*, ki imata za razliko od drugih zelo obsežen glavni tok dogodkov in vsebujeta vsaj en alternativni tok dogodkov.

a) Poslovanje

Najbolj pomemben primer uporabe v sistemu. Omogoča uporabniku pridobivanje podatkov o kupcu, sklenjenih zavarovanjih, predpisanih naročilnicah in zapisovanje izdaje pripomočkov.

Glavni tok dogodkov:

Branje podatkov o zavarovancu in zavarovanju z delujočo KZZ

A1. Uporabnik vstavi KZZ v čitalnik.
A2. Sistem prebere šifro zavarovanca iz KZZ.
A3. Sistem prebere iz sistema OLZZ podatke o kupcu in njegovih zavarovanjih.
A4. Sistem izpiše na zaslonu podatke o kupcu in njegovih zavarovanjih.

Branje podatkov o zavarovancu in zavarovanju brez uporabe KZZ

B1. Uporabnik določi namen in razlog dostopa brez KZZ.
B2. Uporabnik določi način iskanja kupca v sistemu OLZZ in vpiše potrebne podatke.
B3. Sistem pošlje sistemu OLZZ zahtevo za iskanje kupca za izbrani pogoj.
B4. Sistem izpiše na zaslonu podatke o kupcu in njegovih zavarovanjih.

Branje podatkov o zavarovancu iz lokalne baze

C1. Uporabnik začne z vnosom imena v vnosno polje kupec (po prvi vneseni črki sistem avtomatično prikaže seznam kupcev) ali zahteva prikaz seznama vseh kupcev.
C2. Uporabnik izbere kupca iz seznama.
C3. Sistem izpiše na zaslonu podatke o kupcu in njegovih zavarovanjih.

Izdaja (prodaja) pripomočkov

Pogoj: Pridobljeni podatki o kupcu in zavarovanju iz sistema OLZZ.
D1. Sistem prikaže okno za vnos številke naročilnice.
D2. Uporabnik vnese številko naročilnice.
D3. Sistem prebere podatke o naročilnici iz sistema OLZZ.
D4. Sistem naredi novo naročilnico.
D5. VKLJUČITEV: Prične se z izvajanjem primera uporabe: Upravljanje z naročilnicami.

Alternativni tok dogodkov:

Pogoj, ki sproži izvajanje alternativnega toka dogodkov: Napaka pri branju podatkov iz sistema OLZZ ali iz KZZ in v primeru, ko sistem sporoči, da zahtevani podatki ne obstajajo.
E1. Sistem izpiše obvestilo o napaki.
E2. Tok dogodkov se zaključi.

b) Upravljanje z naročilnicami

Administrator s tem primerom uporabe vnaša, pregleduje in zapisuje izdajo pripomočkov po pravilih ZZZS.

Glavni tok dogodkov:

Izdaja pripomočkov in zapisovanje izdaje v sistem OLZZ

A1. Uporabnik preveri obstoječe podatke v naročilnici.
A2. Uporabnik preveri obstoj in veljavnost podatkov o zavarovanjih.
A3. Uporabnik zahteva od sistema zapisovanje podatkov o izdaji v sistem OLZZ.
A4. Sistem izpiše na zaslonu pri vseh pripomočkih identifikator izdaje pripomočka, pridobljen iz sistema OLZZ in novi status "Izdana".
A5. Sistem spremeni status Naročilnice v status "Izdana".
A6. Sistem izpiše obvestilo o uspešnosti izvedbe zapisovanja izdaje pripomočka v sistem OLZZ.

Alternativni tok dogodkov:

Pogoj, ki sproži izvajanje alternativnega toka dogodkov: Naročilnica ne vsebuje podatkov (o NAR-2), prebranih iz sistema "On-line zdravstveno zavarovanje".

B1. Uporabnik vnese obvezne splošne podatke iz naročilnice NAR-2.
B2. Uporabnik v postavkah naročilnice vnese šifre predpisanih pripomočkov.
B3. Uporabnik zahteva od sistema, da prebere podatke o osnovnem zavarovanju iz sistema OLZZ. Če sistem ne pridobi podatkov, se tok zaključi.
B4. Uporabnik zahteva od sistema, da prebere podatke o prostovoljnem zavarovanju iz sistema OLZZ. Če sistem ne pridobi podatkov, se tok zaključi.
B5. Vrnemo se na korak A3. v glavnem toku.

Pogoj, ki sproži izvajanje alternativnega toka dogodkov: Podatki o zavarovanjih niso prebrani, ali niso več veljavni.

C1. Uporabnik zahteva od sistema, da prebere podatke o osnovnem zavarovanju iz sistema OLZZ. Če sistem ne pridobi podatkov, se tok zaključi.
C2. Uporabnik zahteva od sistema, da prebere podatke o prostovoljnem zavarovanju iz sistema OLZZ. Če sistem ne pridobi podatkov, se tok zaključi.
C3. Vrnemo se na korak A3. v glavnem toku.

Pogoj, ki sproži izvajanje alternativnega toka dogodkov: Pri zapisovanju izdaje pripomočka v sistem OLZZ je prišlo do napake tako, da niso zapisane vse izdaje.

D1. Sistem prikaže okno z opisom napak.
D2. Sistem izpiše na zaslonu pri vsakem pripomočku, katerega izdaja je uspešno zapisana v sistem OZZZ, identifikator izdaje pripomočka, pridobljen iz sistema OLZZ in novi status "Delno izdana" v postavki naročilnice.
D3. Vrnemo se na korak A6. v glavnem toku.

Pogoj, ki sproži izvajanje alternativnega toka dogodkov: Pri zapisovanju izdaje pripomočka v sistem OLZZ je prišlo do napake. Niti ena izdaja pripomočka ni zapisana.

E1. Sistem prikaže okno z opisom napak.
E2. Vrnemo se na korak A6. v glavnem toku.

c) Upravljanje z zahtevki
Primer uporabe administratorju omogoča pregled in tiskanje zahtevkov, ki se avtomatično ustvarijo pri zapisovanju izdaje pripomočkov v primeru uporabe: Upravljanje z naročilnicami.

d) Priprava povzetka o izdanih MTP
Omogoča administratorju izdelavo XML dokumenta, ki vsebuje povzetek o izdanih MTP.

e) Zaključevanje zahtevkov za obdobje
Namen primera uporabe je pohitriti delo pri zaključevanju zahtevkov ob koncu obdobja. Administrator z uporabo tega primera v enem koraku zaključi in izpiše vse zahtevke. V nasprotnem bi moral ročno iskati zahtevke v sistemu in jih ročno zaključevati.

f) Upravljanje s podatki o kupcih
Primer uporabe administratorju omogoča vnos in shranjevanje splošnih podatkov o kupcu in podatkov o zavarovanju, pridobljenih iz sistema "On-line zdravstveno zavarovanje", pregled in iskanje po imenu in številki zdravstvenega zavarovanja.

g) Upravljanje s podatki o zavarovalnicah
Uporablja ga administrator za vnos, spremembo in shranjevanje podatkov o ZZZS in zavarovalnicah za prostovoljna zavarovanja (naziv, naslov, davčna številka, tel, fax, e-mail, kontaktna oseba)

h) Upravljanje s podatki o območnih enotah ZZZS
Primer uporabe administratorju omogoča vnos, shranjevanje, in pregled podatkov ter iskanje po nazivu območne enote.

i) Upravljanje s podatki o izpostavah ZZZS
Administratorju omogoča vnos in shranjevanje podatkov o izpostavah ZZZS, pregled in iskanje po: nazivu izpostave in območni enoti.

j) Upravljanje s podatki o pripomočkih
Omogoča administratorju vnos in shranjevanje podatkov o pripomočkih (naziv, šifra, cena), pregled in iskanje po nazivu in šifri pripomočka.

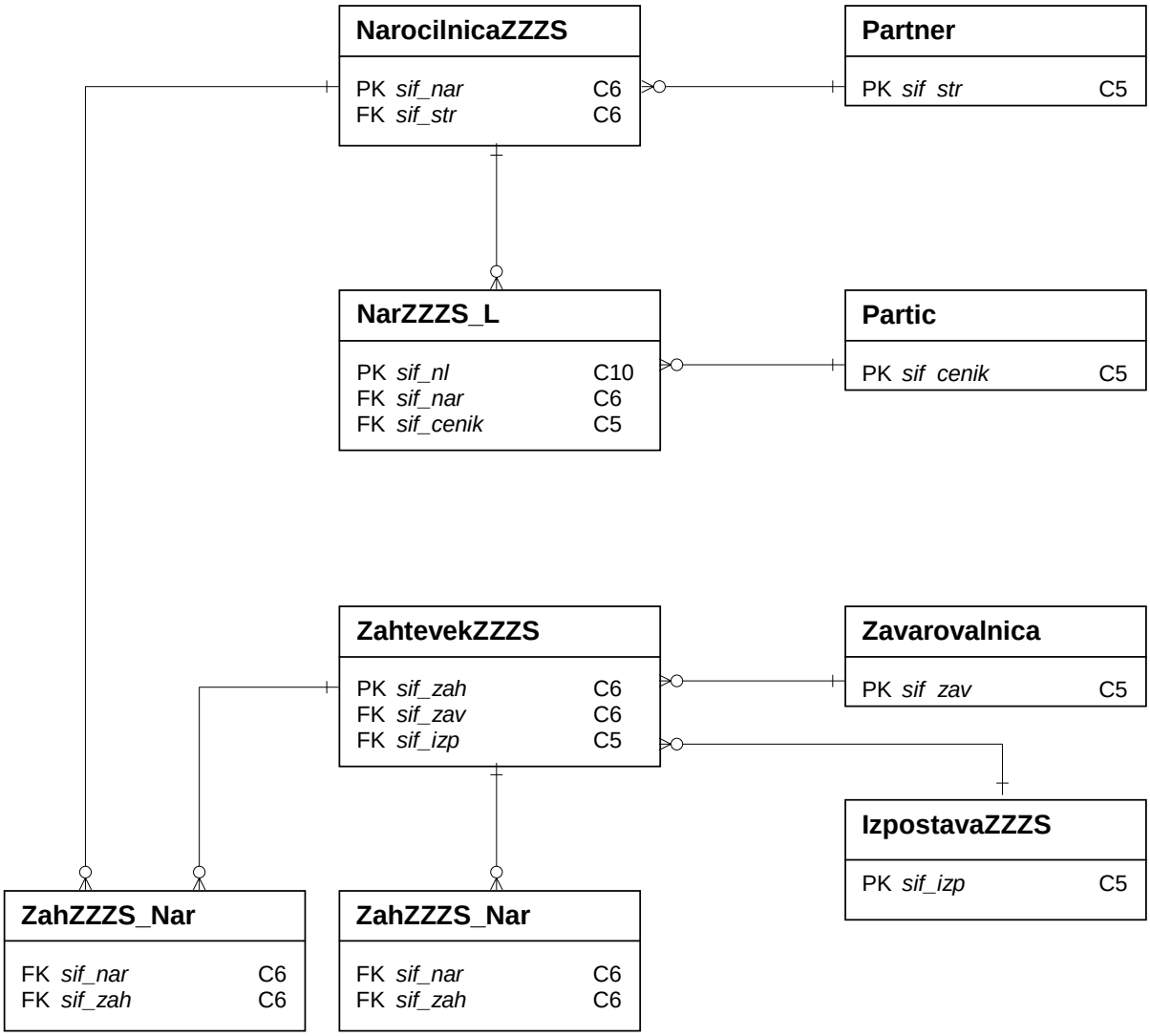
k) Upravljanje s podatki o razmerju plačevanja OZZ/DOPZ
Ta primer uporabe omogoča administratorju shranjevanje in spreminjanje podatkov o razmerju plačevanja vrednosti izdanega pripomočka med ZZZS in Zavarovalnicami za prostovoljno zavarovanje.

4.3 Analiza in načrtovanje

V poglavju analiza in načrtovanje so predstavljeni del konceptualnega modela, razredni diagram in najbolj pomemben diagram zaporedja. V postopku načrtovanja je izdelan tudi diagram postavitve, ki je predstavljen v podpoglavju 4.6 (Postavitev).

4.3.1 Konceptualni model

Konceptualni model (Slika 23) vsebuje veliko število entitet, zato so prikazane in opisane samo najbolj pomembne entitete in atributi.



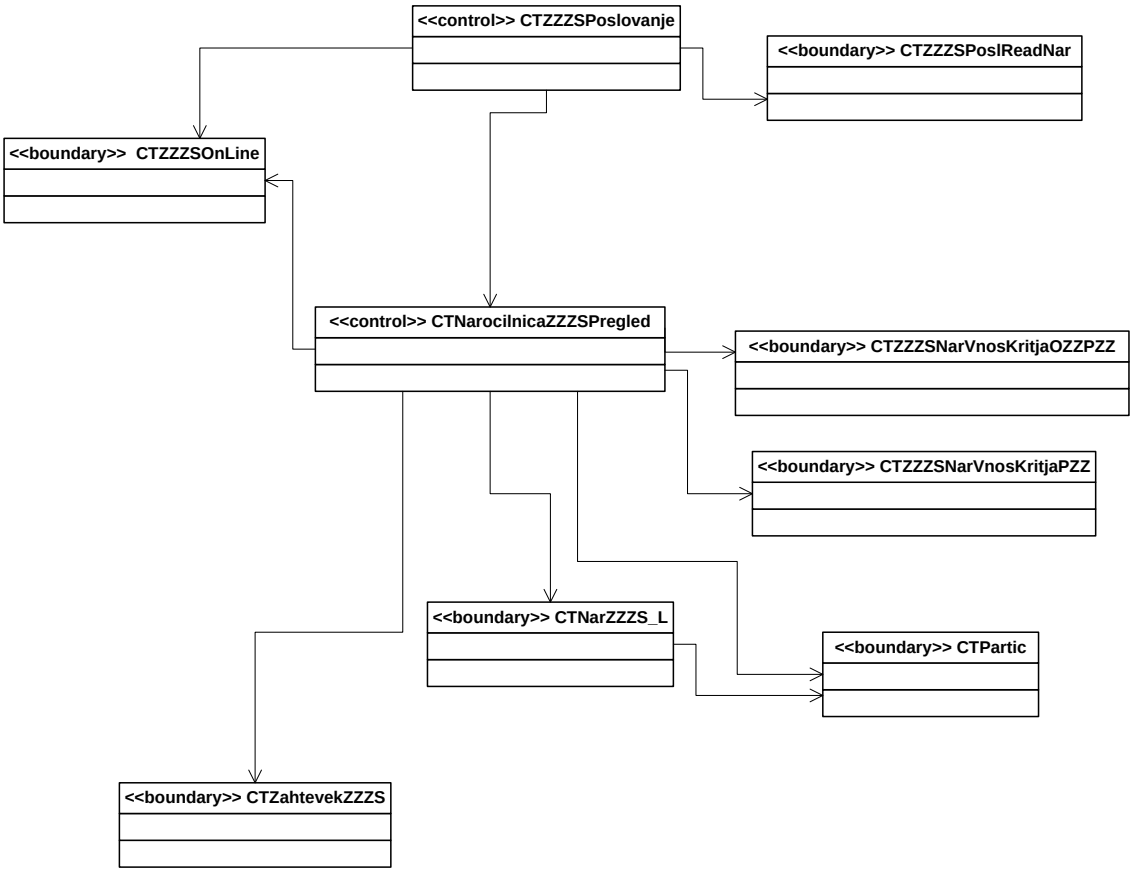
Slika 23: Konceptualni model

Na sliki so naslednje entitete:

- *NarocilnicaZZZS* (vsi podatki naročilnice, razen predpisanih pripomočkov),
- *NarZZZS_L* (podatki o posameznem pripomočku naročilnice),
- *Partner* (šifrant, ki vsebuje splošne podatke o kupcu (zavarovancu) in njegovih zavarovanjih),
- *Partic* (šifrant pripomočkov),
- *ZahtevekZZZS* (splošni podatki o zahtevku),
- *ZahZZZS_L* (postavke zahtevka),
- *ZahZZZS_Nar* (povezuje zahtevek z naročilnicami),
- *Zavarovalnica* (šifrant s podatki o ZZZS in zavarovalnicah za prostovoljna zavarovanja),
- *IzpostavaZZZS* (šifrant s podatki o izpostavah ZZZS).

4.3.2 Razredni diagram

V tem podpoglavju so opisani samo najpomembnejši razredi, ki realizirajo primer uporabe *Poslovanje*. Zaradi obsežnosti so izpuščeni atributi in operacije razredov.



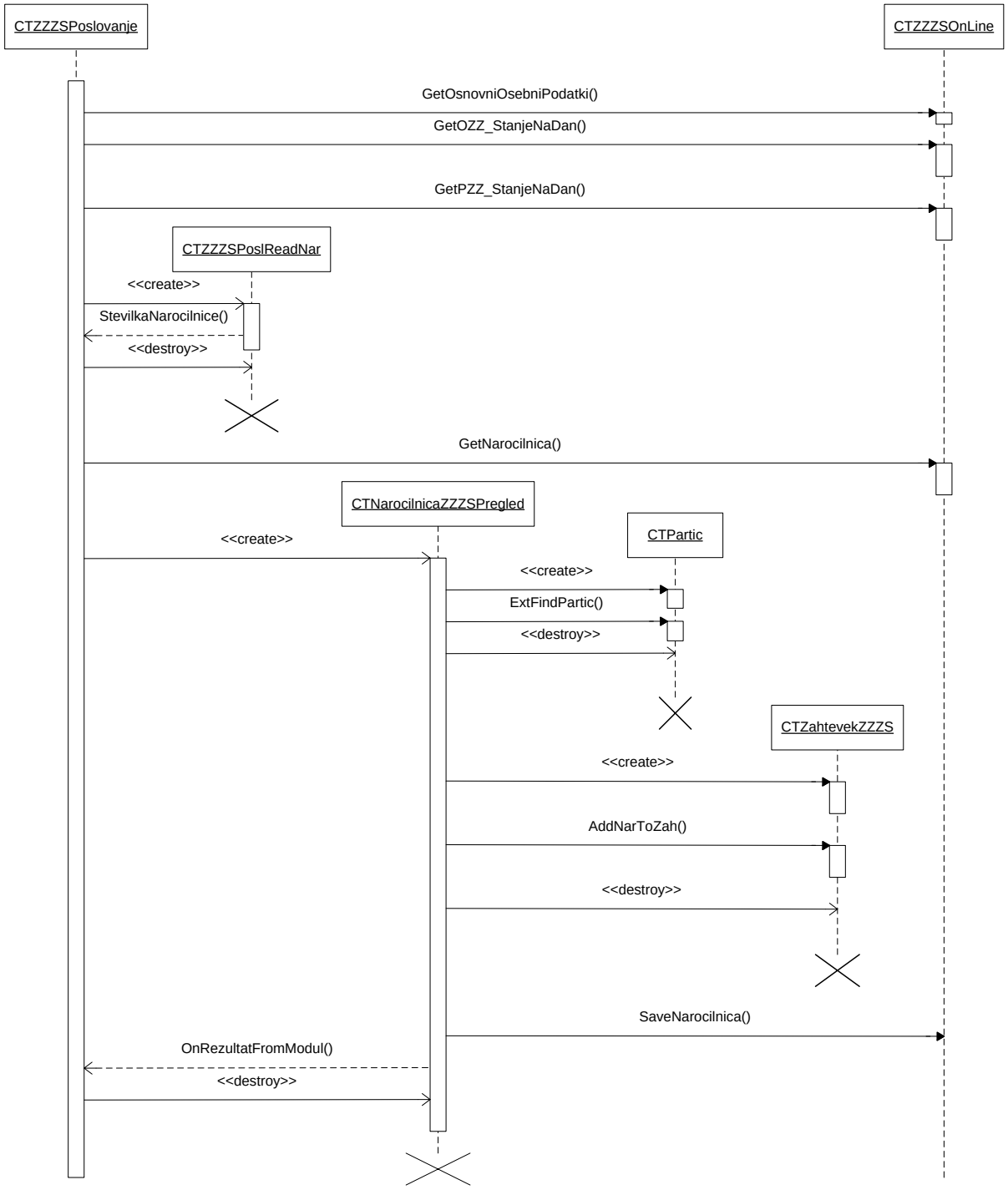
Slika 24: Razredni diagram

Razredi v razrednem diagramu (Slika 24) so:

- *CTZZZSPoslovanje* (realizira primer uporabe *Poslovanje*),
- *CTZZZSPoslReadNar* (omogoča vnos številke naročilnice),
- *CTZZZSOnLine* (skrbi za pošiljanje in sprejem sporočil v OLZZ),
- *CTNarocilnicaZZZSPregled* (vnos naročilnice, zapisovanje izdaje pripomočkov),
- *CTNarZZZS_L* (pregled in vnos pripomočkov, predpisanih v naročilnici),
- *CTNarVnosKritjaOZZPZZ* (omogoča izbor načina kritja stroškov izdanih pripomočkov),
- *CTNarVnosKritjaPZZ* (omogoča določitev, če zavarovalnic za prostovoljna zavarovanja krije del stroškov izdanih pripomočkov) ,
- *CTPartic* (prikaže seznam vseh pripomočkov (šifrant)
- *CTZahtevekZZZS* (realizira primer uporabe *Upravljanje z zahtevki).*

4.3.3 Diagram zaporedja

Diagram (Slika 25) opisuje zaporedje operacij, ki se izvajajo pri izdajanju MTP. Zaradi boljše preglednosti so izpuščene operacije, s katerimi se izpisujejo podatki na zaslon.



Slika 25: Diagram zaporedja

4.4 Implementacija

V začetni fazi razvoja sem ocenil, da največje tveganje predstavlja izmenjevanje sporočil s sistemom "On-line zdravstveno zavarovanje" (OLZZ). Da bi se zmanjšala nevarnost napačnega zajema zahtev, načrtovanja in posledično izdelava neuporabne rešitve, sem začel implementacijo že v začetni fazi.

Zaradi večletnih izkušenj sem se odločil za izdelavo programske opreme v programskem jeziku C++. Pri izdelavi sem uporabljal orodje Microsoft Visual Studio.
Za izdelavo funkcij za delo z XML dokumenti sem uporabljal programski vmesnik (API) Document Object Model (DOM). Pri delu s podatkovno bazo sem uporabljal programski vmesnik ActiveX Data Objects (ADO).

Kot ogrodje programa sem uporabil razrede, izdelane v prejšnjih projektih. V naslednjem koraku sem izdelal podatkovno bazo s tabelami, ki so prenesene iz prejšnjih projektov in tako dobil prvo delujočo verzijo.

Naslednji cilj je bil izdelava razredov, ki omogočajo izmenjevanje sporočil s sistemom OLZZ. Zaradi nepopolne dokumentacije in nedokončanega razvoja samega sistema OLZZ je bilo treba izdelati ključne funkcije, da bi ugotovil kako se pravilno pošiljajo sporočila in kakšne odgovore vrne sistem za posamezno sporočilo. Šele na osnovi teh ugotovitev sem lahko začel z bolj podrobnim poslovnim modeliranjem in zajemom zahtev.

V prvih dveh iteracijah faze konstrukcija sem izdelal uporabniške vmesnike za primere uporabe in ostale tabele podatkovne baze. Z uporabo gradnikov iz lastne knjižnice, razvite za potrebe prejšnjih projektov, sem si olajšal, izdelavo uporabniškega vmesnika,

Najbolj obsežen problem, izdelava XML sporočil in analiza pridobljenih odgovorov iz OLZZ, je potekal v naslednjih dveh iteracijah in je zahteval 80 % porabljenega časa v postopku implementacije.

Da bi se še olajšalo delo uporabniku oz. da bi dobili večjo preglednost podatkov na zaslonu, so bile v fazi prevzema potrebne samo manjše spremembe uporabniškega vmesnika.

Lahko ocenim, da je bila implementacija uspešna, ker izdelana programska oprema deluje stabilno in brez napak.

4.5 Testiranje

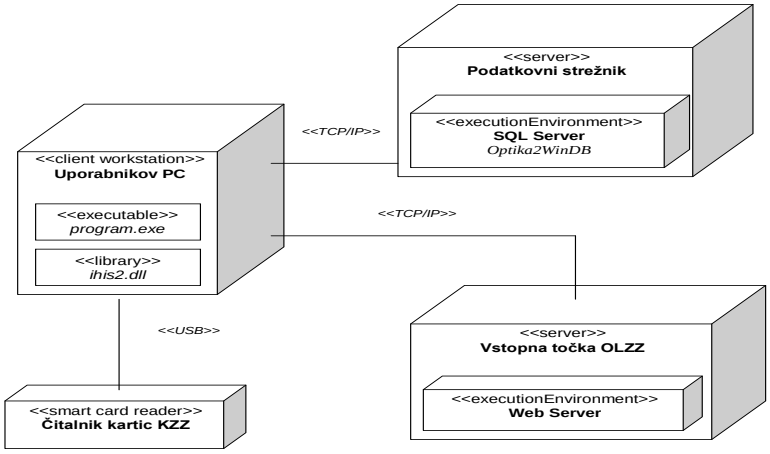
Potek testiranja lahko razdelim na dva dela; na testiranje v postopku izdelave in preverjanje rezultatov v testnem obdobju delovanja (RUP faza: prevzem).

V postopku implementacije sem kodo preverjal po diagramih aktivnosti in sproti odpravljal napake. Po zaključku implementacije vsakega primera uporabe sem opravil teste na testnih primerih, ki so bili pripravljeni v postopku analize in načrtovanja. Tako sem imel na koncu vsake iteracije preverjeno in delujočo verzijo.

Izdelana programska oprema je v testnem obdobju zapisovala podatke o vseh poslanih in prejetih sporočilih v sistem OLZZ ter napake, ki so se pojavljale pri komunikaciji. Za vsako naročilnico (listina Nar-2) se je preverjalo, ali so podatki pravilno pridobljeni iz sistema in če je izdaja pripomočka pravilno zapisana v sistem. Preverjala se je tudi pravilnost zahtevkov in povzetka o izdanih MTP (XML dokument).

4.6 Postavitev

Za opis postavitve si pomagamo z diagramom postavitve (Slika 26), ki je bil izdelan v postopku načrtovanja.



Slika 26: Diagram postavitve

Za delovanja sistema morajo biti prisotne naslednje komponente:

- ***Uporabnikov PC***
  - *Postopek postavitve:*
    Na uporabnikov računalnik naložimo datoteke: *program.exe* in *ihis2.dll*.
  - *Sistemske zahteve:*
    Windows platforma (2000, XP, Vista, Windows 7 in vsi strežniški OS),
    Microsoft .NET framework 2.0 ali višji,
    Gemalto Classic Client 5.1.7 for ZZZS,
    in korensko potrdilo POŠTA®CA.

- ***Čitalnik pametnih kartic***
  - *Sistemske zahteve:*
    PC/SC združljiv čitalnik (npr. Gemplus GCR700, Gemalto GCR 5500-D).

- ***Podatkovni strežnik***
  - *Postopek postavitve:*
    Na podatkovnem strežniku naredimo podatkovno bazo *Optika2Win* in v njo uvozimo, tabele shranjene v datoteki *iOptika2Win.bak*.
  - *Sistemske zahteve:*
    Microsoft SQL strežnik ver. 6.0 ali višji, Microsoft SQL ServerExpress Edition ali MSDE.

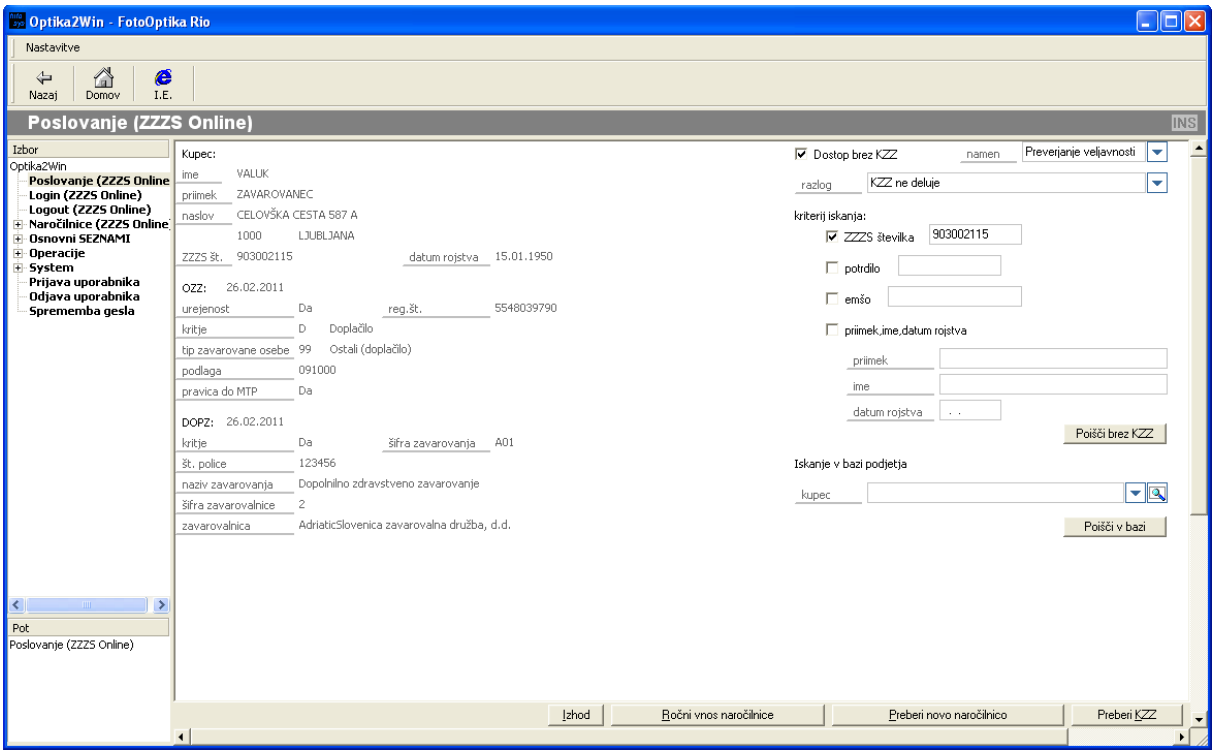
- ***Vstopna točka*** sistema "On-line zdravstveno zavarovanje"

4.7 Opis funkcionalnosti izdelane programske opreme

V tem podpoglavju se opisujeta samo dve funkcionalnosti, ki se najpogosteje uporabljata: Poslovanje in Naročilnica.

4.7.1 Poslovanje (z zavarovanci)

Namen funkcionalnosti *Poslovanje* (Slika 27) je prodaja pripomočkov (MTP za vid) kupcem, ki so zavarovanci ZZZS in jim je zdravnik-oftalmolog izdal naročilnico NAR-2 oz. predpisal pripomočke.



Slika 27: Poslovanje (z zavarovanci)

Če se omejimo samo na začetno okno, potem se funkcionalnost zoži na pridobivanje splošnih podatkov o kupcu in sklenjenih zavarovanj. Iskanje podatkov o kupcih delimo lahko glede na vir, od kod pridobivamo podatke na; podatke pridobljene iz lokalne baze in podatke pridobljene iz sistema OLZZ.

Podatke iz lokalne baze pridobimo tako, da začnemo z vnosom imena (samo prva črka) v vnosnem polju kupec ali če kliknemo z miško na prvo ikono na desni strani vnosnega polja. Program nam nato prikaže seznam kupcev iz katerega, izberemo želenega. Ko smo izbrali kupca, program v vnosnem polju izpiše njegovo ime. Podatki o kupcu se izpišejo, ko to uporabnik zahteva s pritiskom na gumb *"Poišči v bazi"*.

Podatke iz sistema OLZZ pridobimo na dva načina: z uporabo KZZ in brez uporabe KZZ.

Pri uporabi KZZ vstavimo kartico v režo čitalnika in pritisnemo na gumb *"Preberi KZZ"*. Program pridobi podatke o kupcu iz sistema OLZZ in jih izpiše.

Postopek pridobivanja podatkov iz sistema OLZZ brez uporabe KZZ je naslednji:

- Najprej določimo namen in razlog dostopa do podatkov brez uporabe KZZ.
- Kot iskalni kriterij kupca lahko določimo:
  - *ZZZS številko,*
  - *Potrdilo,*
  - *Emšo,*
  - *Priimek, ime in datum rojstva.*
- Vnesemo potrebne podatke v ustrezna vnosna polja in pritisnemo gumb "Poišči brez KZZ".
- V primeru, da v sistemu OLZZ obstaja kupec (zavarovanec), ki ustreza iskalnemu kriteriju program, izpiše podatke na zaslonu.

Ko smo pridobili podatke o kupcu, lahko začnemo z vnosom naročilnice.
Standardni postopek se začne tako, da pritisnemo na gumb *"Preberi novo naročilnico"*. Na zaslonu se odpre novo okno, v katero vnesemo številko naročilnice. Program poskuša prebrati podatke o naročilnici iz sistema OLZZ in če mu uspe, nadaljujemo delo v funkcionalnosti *Naročilnice*, ki je opisana v naslednjem podpoglavju.

V primeru, da je stranka prinesla naročilnico (listina NAR-2), ki ni vpisana v sistem OLZZ, pritisnemo na gumb *"Ročni vnos naročilnice"*. Program odpre funkcionalnost *Naročilnice* in nadaljujemo vnos podatkov iz listine NAR-2 v naročilnico.

4.7.2 Naročilnica

Omogoča nam vnos naročilnice NAR-2 in zapisovanje izdaje pripomočkov v sistem OLZZ.

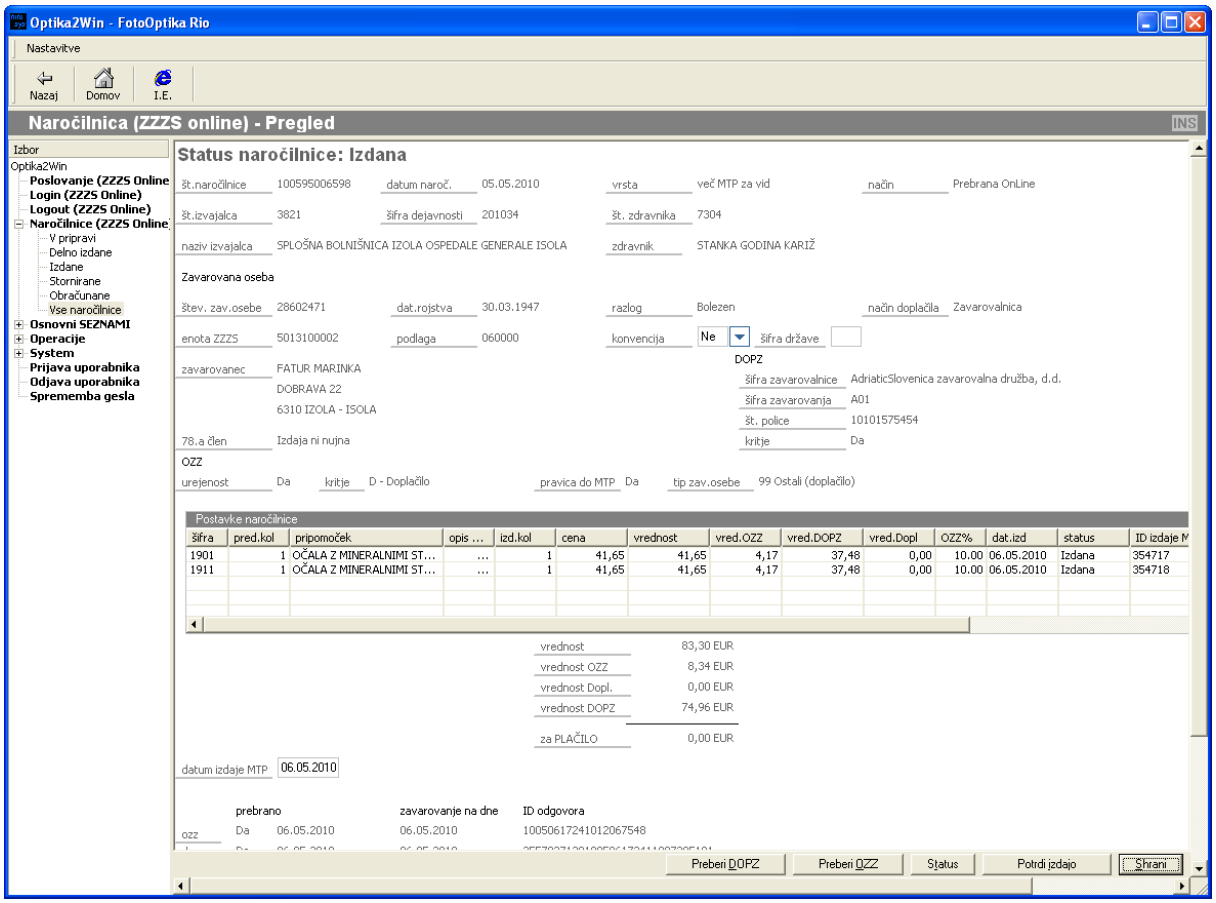
V primeru ročnega vnosa, uporabnik najprej vnese podatke iz NAR-2 vključno s predpisanimi pripomočki (Slika 28).

Če imamo naročilnico, ki je nismo vnesli na današnji dan, ali nimamo današnjih podatkov o veljavnosti zavarovanj, potem moramo najprej prebrati nove podatke o veljavnih zavarovanjih. To storimo tako, da pritisnemo gumba: *"Preberi OZZ"* in *"Preberi DOPZ"*. Program pridobi iz sistema nove podatke o obveznem in prostovoljnem (dopolnilnem) zavarovanju.

Izdajo pripomočkov zapišemo v sistem OLZZ tako, da pritisnemo gumb *"Potrdi izdajo"*. Program po uspešni izvedbi operacije izpiše nove statuse (*"Izdana"*) in identifikatorje izdaje (*"ID izdaja MTP"*). S tem je postopek izdaje pripomočka MTP končan.

Če ugotovimo, da je prišlo do napačnega vnosa podatkov, ko je naročilnica v statusu *"Izdana"*, moramo najprej stornirati izdajo. Pritisnemo gumb *"Status"* in v meniju izberemo *"Storniraj naročilnico"*. Program stornira naročilnico in stornira izdajo v sistemu OLZZ.

V primeru, da želimo podatke v naročilnici popraviti in ponovno zapisati izdajo MTP, pritisnemo gumb *"Status"* in v meniju izberemo *"Postavi naročilnico V pripravo"*.



Slika 28: Naročilnica

5. Sklepne ugotovitve

V nalogi je opisan razvoj programske opreme, ki omogoča povezovanje informacijskega sistema dobavitelja MTP za vid in sistema "On-line zdravstveno zavarovanje" ter obračunavanje opravljenih storitev in dobavljenih pripomočkih po predpisanih poslovnih.

Pri izdelavi programske opreme sem uporabljal preverjeno in stabilno tehnologijo, ki jo uporabljam že več let, tako da ta ni predstavljala večjih težav ali zavirala razvoj.
Največje težave so se pojavljale v začetni fazi (ang. inception) in v fazi zbiranja informacij (ang. elaboration). Vse težave so izhajale iz dejstva, da sem razvijal rešitev, ki mora izmenjevati sporočila z novim sistemom na strani ZZZS, ki je še sam v neki zaključni fazi razvoja. Dokumentacija posredovana s strani ZZZS, ki opisuje izmenjevanje podatkov s sistemom "On-line zdravstveno zavarovanje" in poslovna pravila pri poslovanju z ZZZS so zelo obsežna, razdeljena na več ločenih dokumentov, dvoumna in hkrati pomanjkljiva. Skoraj polovico časa, porabljenega za razvoj, sem potreboval za analizo, kako se uporabljajo funkcije vmesnika za dostop do sistema in kakšna mora biti vsebina sporočil, ki se izmenjujejo. Posebne težave so nastopile v fazi konstrukcije (ang. construction), ko so na strani ZZZS spremenili vsebino ključnih sporočil in pravila o izvajanju posameznih operacij, kar je zahtevalo ponovno izvajanje procesa analize in načrtovanja rešitve.

Z izdelavo diplomske naloge sem pridobil dragocene izkušnje pri integraciji informacijskih sistemov in uporabi metodologije RUP.

Izdelana rešitev predstavlja samo začetni korak pri prenovi informacijskega sistema naročnika. V prihodnosti se načrtuje nadaljevanje razvoja, s katerim se bodo podprli tudi drugi poslovni procesi naročnika.

Izdelana programska oprema se uporablja že skoraj leto dni in v tem času naročnik ni prijavil nobene napake pri delovanju ali težave pri izvajanju poslovnih procesov, podprtih s to rešitvijo. Iz tega sklepam, da so bili uspešno doseženi vsi cilji diplomske nalog.

6. Viri

[1] W3C. XML Essentials. Dostopno na: http://www.w3.org/standards/xml/core

[2] W3C (2006). W3C Recommendation: Extensible Markup Language (XML) 1.1 (Second Edition). Dostopno na: http://www.w3.org/TR/2006/REC-xml11-20060816

[3] XML. Wikipedija. Dostopno na: http://en.wikipedia.org/wiki/XML

[4] SGML Users' Group History. International SGML Users' Group. Dostopno na: http://xml.coverpages.org/sgmlhist0.html

[5] Elliotte Rusty Harold, W. Scott Means, XML in a Nutshell, 3th ed. Sebastopol, CA: O'Reilly Media, 2004.

[6] W3C (2009). W3C Recommendation: Namespaces in XML 1.0 (Third Edition). Dostopno na: http://www.w3.org/TR/REC-xml-names/

[7] W3C (2004). W3C Recommendation "XML Schema Part 1: Structures Second Edition". Dostopno na: http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/structures.html#Model_Groups

[8] W3C (1999). W3C Recommendation: XML Path Language (XPath) Version 1.0. Dostopno na: http://www.w3.org/TR/xpath

[9] W3C (2004). W3C Recommendation: Document Object Model (DOM) Level 3 Core Specification. Dostopno na: http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/Overview.html

[10] Ethan Cerami, Web Services Essentials, Sebastopol, CA: O'Reilly Media, 2002.

[11] Tehnična navodila za uporabo sistema on-line zdravstvenega zavarovanja, Zavod za zdravstveno zavarovanje Slovenije, Ljubljana, 2009.

[12] Ivar Jacobson, Grady Booch, James Rumbaugh, The Unified Software Development Process, MA: Addison Wesley Longman, 1999.

[13] Eric J. Naiburg, Robert A. Maksimchuk, UML for Database Design, MA: Addison-Wesley, 2001.