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: 01671/2010
Datum: 05.04.2010

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

Kandidat: **MIHA RAVŠELJ**

Naslov: **GASTONVO – RAČUNALNIŠKA PODPORA ZA PRODAJO VGRADNIH OMAR**

GASTONVO – COMPUTER SUPPORT FOR MARKETING OF BUILT-IN WARDROBES

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Predmet diplomske naloge je programska oprema, ki podpira prodajo in opis vgradnih omar z drsnimi vrati, ki predstavljajo parametrično opisan sestav z spremenljivo geometrijo. Med problemi, ki nastopajo v procesu prodaje in proizvodnje, sta tudi prenos podatkov iz pohištvenega salona proizvajalcu in priprava podatkov za proizvodnjo.
Opišite glavne principe delovanja programa, ki omogoča prodajo vgradnih omar v pohištvenih salonih. Program temelji na preprostem uporabniškem grafičnem vmesniku, ki je zgrajen na podlagi kompleksne XML podatkovne strukture in preddefinirane knjižnice osnovnih elementov oziroma sestavnih delov omare. Opišite tudi možno rešitev za pripravo proizvodnih podatkov v razvojnem CAD okolju SolidWorks. Z izdelavo vnaprej definirane knjižnice in vtičnika za program SolidWorks naj bi bilo mogoče izdelati 3D sestav vgradne omare, ki je bila posredovana kot projektna datoteka programa GastonVO .

Mentor:

prof. dr. Saša Divjak

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Franc Solina

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Miha Ravšelj

GASTONVO – RAČUNALNIŠKA PODPORA ZA PRODAJO VGRADNIH OMAR

DIPLOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Saša Divjak

Ljubljana, 2010

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

diplomskega dela

Spodaj podpisani/-a Miha Ravšelj,

z vpisno številko 63010121,

sem avtor/-ica diplomskega dela z naslovom:

GASTONVO – RAČUNALNIŠKA PODPORA ZA PRODAJO VGRADNIH OMAR

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

  prof. dr. Saša Divjak

  in somentorstvom (naziv, ime in priimek)

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 13.06.2010 Podpis avtorja/-ice:

Zahvala

Hvala družini in Klari za vso podporo in spodbudo med študijem.

Kazalo

Povzetek 1
Abstract 3
1 Uvod 5
2 *GastonVO* – računalniška podpora za prodajo vgradnih omar 9
2.1 Splošno o aplikaciji *GastonVO* 9
2.1.1 Tehnologija in programski jezik 9
2.1.2 Funkcionalnosti z vidika udeležencev v procesu prodaje vgradnih omar 9
2.1.3 Organizacija programa z vidika programerja 10
2.2 Uporabniški grafični vmesnik 11
2.2.1 Delovanje programa z vidika uporabnika 11
2.2.2 Risanje sestava vgradne omare 11
2.3 Jedro 13
2.3.1 XML sestav 13
2.3.2 Podatkovna baza 24
2.4 Uporaba XML strukture 25
2.5 Knjižnica 28
3 GastonVO vtičnik za *SolidWorks* 30
3.1 Kratka predstavitev *SolidWorks* okolja 30
3.1.1 O programu *SolidWorks* 30
3.1.2 *SolidWorks* API arhitektura 30
3.1.3 Osnovno delovanje vtičnika *GastonVO* 30
3.2 Knjižnica osnovnih sestavov 31
3.2.1 Analogija z knjižnico *GastonVO* 31
3.2.2 Priprava *SolidWorks* knjižnice osnovnih sestavov 31
3.3 *GastonVO* vtičnik 33
4 Sklepne ugotovitve 36
Literatura 39

Seznam uporabljenih kratic in pojmov

CAD – *Computer Aided Design*: računalniško podprto konstruiranje

PDM sistem – *Product Data Management*: informacijski sistem za upravljanje s podatki o izdelkih

CAM – *Computer Aided Manufacturing*: računalniško podprta proizvodnja

GUI – *Graphical User Interface*: uporabniški grafični vmesnik

SINGLETON – pri objektno usmerjenem programiranju se uporabi tak način kadar želimo samo eno instanco posameznega razreda znotraj aplikacije

XSD – *XML schema definition*: XML shema opisuje strukturo XML dokumenta

Povzetek

Podjetja, ki proizvajajo notranjo opremo, navadno prodajajo svoje produkte v pohištvenih salonih. Številna slovenska pohištvena podjetja že vrsto let uporabljajo program *Gaston* podjetja *ib-CADdy d.o.o*. *Gaston* zagotavlja postavljanje notranjih elementov v prostoru (kuhinje, kopalnice, spalnice ipd) na podlagi preddefinirane knjižnice posameznega proizvajalca. Vsi *Gastonovi* elementi imajo preddefinirane 2D simbole in 3D geometrijo, kar pomeni, da sam program ne omogoča geometrijskih sprememb na elementih. V zadnjem času se je na trgu pojavila potreba po programski opremi, ki bi podpirala prodajo in opis vgradnih omar z drsnimi vrati, ki predstavljajo parametrično opisan sestav z spremenljivo geometrijo. Med problemi, ki nastopajo v procesu prodaje in proizvodnje, sta tudi prenos podatkov iz pohištvenega salona proizvajalcu in priprava podatkov za proizvodnjo.

V diplomski nalogi sem v prvem delu opisal glavne principe delovanja programa *GastonVO*, ki omogoča prodajo vgradnih omar v pohištvenih salonih. Program temelji na preprostem uporabniškem grafičnem vmesniku, ki je zgrajen na podlagi kompleksne XML podatkovne strukture in preddefinirane knjižnice osnovnih elementov oziroma sestavnih delov omare.

Drugi del opisuje eno izmed možnih rešitev za pripravo proizvodnih podatkov v razvojnem CAD okolju *SolidWorks*. Z izdelavo preddefinirane knjižnice in vtičnika za program *SolidWorks* je mogoče izdelati 3D sestav vgradne omare, ki je bila posredovana kot projektna datoteka programa *GastonVO* .

Ključne besede: *Gaston*, *GastonVO, SolidWorks*, parametrični sestavi, XML shema, vgradna omara

Abstract

Companies manufacturing furniture usually sell their products in their furniture stores. Many Slovene furniture companies have been using application *Gaston* by *ib-CADdy*. With this application based on predefined library it is possible to place various furniture elements inside kitchen, bathroom, living room… *Gaston* elements are predefined by 2D symbols and 3D mesh which means that the application itself does not support any geometrical modifications. Software supporting selling built-in wardrobes with sliding doors is therefore one of the latest market needs. A built-in wardrobe is a parametrically described assembly with varying geometry. In the process of selling and manufacturing companies are faced with two major difficulties: data transfer from the store to the manufacturer and the preparation of the received data for manufacturing.

In the first part of this diploma work the basic principles of *GastonVO* (software supporting selling built-in wardrobes in furniture stores) are presented. This application is based on common graphical user interface built on the top of the complex XML data structure and predefined library of basic assemblies/parts.

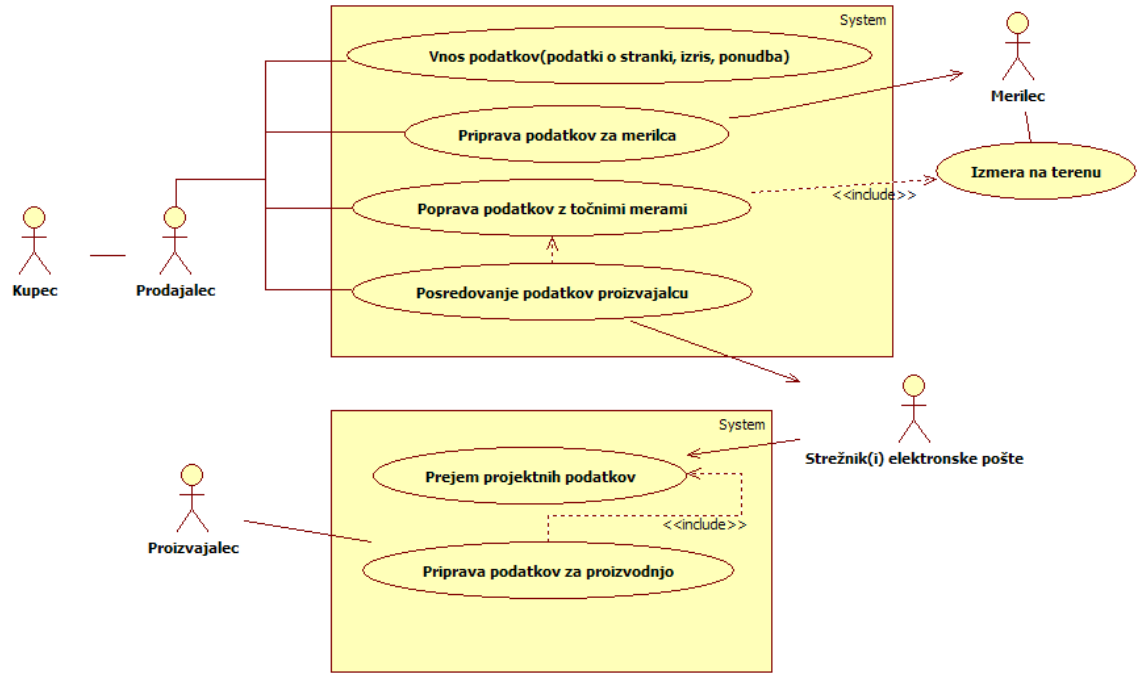
In the second part of this diploma one possible solution for manufacturing data preparations is presented within *SolidWorks* CAD environment. The predefined library and addin for *SolidWorks* enable us to generate 3D built-in wardrobe assembly considering the received *GastonVO* project file.

Keywords: *Gaston, GastonVO, SolidWorks,* parametrical assembly, XML schema, built-in wardrobe

1 Uvod

Danes si težko predstavljamo kakršno koli trgovino, ki bi lahko delovala brez ustreznega informacijskega sistema, ki podpira prodajo. Če se odpravimo v najbližjo trgovino, lahko takoj opazimo, da skoraj nikjer ni več nalepk s ceno, ampak so cene del informacijskega sistema. V navadnih prodajalnah, ki prodajajo končane izdelke (TV, PC, zvočniki, ...), je cena fiksno določena, ker stranka ne more vplivati na že narejen izdelek. Stranka nima na voljo možnosti, da bi vstavila v neko televizijo še DVD predvajalnik, ampak mora kupiti drugačen tip televizije, ki tak predvajalnik že vsebuje. V pohištveni industriji se prav tako pojavljajo izdelki na katere stranka ne more neposredno vplivati. Takšni elementi so fiksne omare različnih mer (stol, miza, postelja). Stranke lahko sicer izbirajo barvo, oziroma material enega ali več delov posameznega elementa, ne morejo pa spremeniti strukture elementa, na primer dodati v konkretno omaro še eno polico (lahko izberejo drugačno omaro, ki ima polico v kolikor le-ta obstaja). V zadnjem času se na tržišču v večjem obsegu pojavljajo vgradne omare, ki pa jih ni možno prodajati na zgoraj opisan način.

Vgradno omaro določa veliko različnih parametrov (širina prostora, višina prostora, globina prostora, število vratnih kril, število notranjih korpusov,…). Možnih izvedb vgradne omare je mnogo preveč, da bi jih lahko proizvajalci delali na zalogo, vendar to še zdaleč ni edini problem, ki se pri vgradnih omarah pojavi. Stranke želijo po svojih željah in potrebah določiti, kako naj bo omara sestavljena. Pri klasični vgradni omari (notranji del omare + drsna vrata) se lahko določi, koliko korpusov naj vsebuje notranji del vgradne omare, kakšni elementi naj nastopajo v prvem korpusu (police, obešalna palica, izvlečno obešalo), drugem korpusu (notranji element s predali) itn. Prav tako lahko izberejo ali želijo imeti npr. 2 ali 3 krilna vrata, določajo lahko material polnil posameznemu vratnemu krilu. V pohištvenih salonih, ki prodajajo takšne omare, je potrebno stranki predstaviti možnosti, ki jih posamezni proizvajalci ponujajo, poleg tega pa stranka vedno želi zvedeti tudi ceno. Nekatera predvsem manjša podjetja so vgradne omare poskusila prodajati »na list papirja« tako, da je trgovec na list papirja narisal skico in s pomočjo preddefinirane tipske karte izračunal ceno vgradne omare. V primeru nakupa je potem prodajalec po faksu poslal podatke proizvajalcu. V času računalnikov in programske opreme lahko pri takšnem načinu najdemo samo slabosti (nepotrebna komunikacija, človeški faktor, nesporazumi, nepoznavanje vgradnih omar proizvajalca, šolanje trgovcev itn). Za uspešno prodajanje takšnih omar je torej nujna ustrezna programska oprema, ki omogoča prodajo. Za ilustracijo si oglejmo enega izmed možnih procesov prodaje vgradne omare in posredovanje podatkov proizvajalcu na spodnjem diagramu primera uporabe.



Slika 1: Proces prodaje vgradne omare[1]

Na kratko opišimo proces prodaje in izdelave vgradne omare. Kupec pride v pohištveni salon in želi videti, kakšne vgradne omare salon ponuja. Prodajalec mu potem s pomočjo programa za risanje vgradnih omar izriše želeno omaro, kupec dobi ponudbo in plača predujem. Prodajalec pošlje podatke o projektu merilcu, ki opravi izmero na terenu. Ko merilec dostavi točne mere prodajalcu, jih ta ustrezno popravi v projektu. V kolikor se cena vgradne omare spremeni, prodajalec s tem seznani kupca, ki mora spremembo potrditi. Potrjen projekt se posreduje proizvajalcu preko elektronske pošte. Proizvajalec te podatke z ustreznim sistemom zajema in jih vključi v proizvodni proces. Kako ima posamezno podjetje organizirano proizvodnjo, je poglavje zase in presega temo te diplomske naloge. V okviru proizvodnega procesa bom opisal pripravo CAD podatkov, glede na vhodne podatke iz pohištvenih salonov in vnaprej pripravljene knjižnice osnovnih gradnikov vgradne omare.

V takšen procesu se pojavijo različne želje udeležencev, ki jim je potrebno ugoditi z vidika načrtovanja aplikacije. Naštejmo nekaj teh želja glede na udeleženca v procesu:

➢ Želje kupca

- Sprotni izpis cene - potencialnega kupca v vsakem trenutku najbolj zanima, koliko stane trenutna vgradna omara.
- Predogled vgradne omare - slika pove več kot tisoč besed.

[1] Skica procesa predstavlja zgolj eno izmed variant

- Možnost sodelovanja pri risanju - stranka ne ve ničesar o delovanju programa, vendar lahko intuitivno razume delovanje, če je uporabniški vmesnik pravilno zasnovan.

➢ Želje prodajalca

- Enostavnost programa - prodajalci naj se v najkrajšem času spoznajo z aplikacijo in njeno funkcionalnostjo. Večje kot je število parametrov, ki nastopajo v aplikaciji, več časa je potrebno, da se prodajalec nauči upravljanja z njo. Najboljša aplikacija za trgovca je tista, ki jo razume skorajda brez šolanja.

➢ Želje proizvajalca

- Podatki o projektu naj vsebujejo vso informacijo, ki jo proizvajalec potrebuje, da lahko naredi omaro, ki jo je prodajalec (glede na želje kupca) izrisal.
- Podatki naj se posredujejo iz salona v elektronski obliki, tako da ponovno vnašanje podatkov v nek drug sistem (človeški faktor) ni potrebno.
- »Aplikacija naj opravi vse, mi bomo samo proizvajali.« - Najboljša rešitev za proizvajalca je takšna, da dobi iz salona podatke pripravljene za proizvodnjo.
- Možnost dodajanja elementov.

Pri snovanju takšne aplikacije je potrebno upoštevati zgornje želje vseh udeležencev v procesu. Lahko si zamislimo primer preveč kompleksnega programa, ki sicer zmore veliko oziroma praktično vse, kar proizvajalec želi, vendar je njegov uporabniški vmesnik prezapleten za prodajalca, kaj šele kupca. Če je program preveč enostaven, je sicer pisan na kožo prodajalcem in kupcem, vendar pa ne omogoča vsega, kar proizvajalec želi oziroma zahteva. Načrtovalec aplikacije mora imeti v mislih želje vseh udeležencev v procesu in poskusiti kar najbolj ugoditi večini teh želja.

Ko prodajalec pošlje dokončne podatke proizvajalcu, je svoje delo opravil in sedaj mora proizvajalec na podlagi takšnih podatkov vgradno omaro narediti. Katere podatke proizvajalec dobi iz salona, bo podrobno opisano v nadaljevanju. Vsekakor pa ni potrebno, da bi aplikacija v salonu vsebovala vse tehnične podatke o vgradni omari. Program v salonu mora z vidika prodajalca vsebovati natanko toliko informacije, da se lahko izračuna cena in nič več. Z vidika proizvajalca pa je potrebno zagotoviti dovolj podatkov, da je omaro možno proizvesti. Podatki v programu za prodajanje navadno niso neposredno primerni za delavca v proizvodnji, ampak jih je potrebno še pretvoriti v ustrezen format. Kakšen je ta format, je odvisno od proizvajalca. Za manjšega proizvajalca je verjetno dovolj, da natisne potrebne podatke v tekstovni obliki na list papirja in ta list potem posreduje delavcu. Za večja podjetja pa takšna rešitev odpade, saj potrebujejo CAD modele, na podlagi katerih se lahko pripravi tehnična dokumentacija za proizvodnjo. Takšni podatki so lahko del širšega PDM procesa (npr. *SolidWorks Enterprise PDM*), ki skrbi za hranjenje in organizacijo teh podatkov. Na koncu proizvodnega procesa so navadno delavci, delovni stroji, linije, ki izdelujejo potrebne elemente, da se takšna vgradna omara lahko sestavi. Za takšne stroje je potrebno izdelati

strojno kodo, ki jo lahko pripravi bodisi človek bodisi proizvodni sistem na podlagi CAD podatkov in ustrezne aplikacije kot je na primer *CAMWorks za SolidWorks*.

V podjetju *ib-CADdy d.o.o*. smo razvili aplikacijo poimenovano *GastonVO*, ki je primarno namenjena prodaji vgradnih omar. Koncept in funkcionalnost aplikacije bom predstavil v naslednjem poglavju. V tretjem poglavju bo predstavil konkretno rešitev, kako generirati *SolidWorks* CAD sestav vgradne omare na podlagi podatkov iz salona in preddefinirane *SolidWorks* knjižnice osnovnih sestavov/sestavnih delov vgradne omare.

2 *GastonVO* – računalniška podpora za prodajo vgradnih omar

2.1 Splošno o aplikaciji *GastonVO*

2.1.1 Tehnologija in programski jezik

Aplikacija je napisan v .NET okolju, in sicer v jezikih MC++ in C#. Pri izdelavi programa smo uporabili več zunanjih knjižnic napisanih za .NET okolje:

- .NET Framework 3.5 [1],
- FLEE [2]- Motor za preračunavanje izrazov,
- DotNetZip [3]– knjižnica za upravljanje z zip datotekami,
- Xsd2Code [4] - generator razredov za programski jezik C#,VB in MC++,
- Crystal Reports 2008 [5]– orodje za pripravo in prikaz poročil.

2.1.2 Funkcionalnosti z vidika udeležencev v procesu prodaje vgradnih omar

Glede na želje uporabnikov v sistemu (kot je bilo že opisano v uvodu), mora aplikacija izpolnjevati določene zahteve, da bi dobili kar najbolj učinkovit in uporabniku prijazen program. Program, ki ne izpolni zahtev vseh treh udeležencev v procesu (kupec, prodajalec in proizvajalec), ni dober, zato *GastonVO* podpira naslednje osnovne funkcionalnosti:

- enostaven vmesnik za sestavljanje vgradne omare

Temelji na principu povleci-spusti. Uporabnik vizualno dodaja in odstranjuje elemente vgradne omare v/iz sestav/a. Program prepoznava tipe osnovnih sestavov in ponudi referenčne točke, kamor se določen element lahko vstavi. S tako funkcionalnostjo izpolnimo zahtevo po preprostem uporabniškem vmesniku. Uporabnik mora sicer nekaj malega vedeti o vgradnih omarah, vendar mu, tudi če ne ve ničesar, program ne dovoli sestaviti napačnega sestava.

- 3D parametrični opis osnovnih sestavov/sestavnih delov

Podatki o osnovnih sestavih vgradne omare se nahajajo v XML datotekah. Tak osnovni opis mora vsebovati dovolj informacije, da je mogoče umestiti sleherni element v prostoru. Kot bomo spoznali v nadaljevanju, je ta struktura tudi jedro aplikacije *GastonVO*. Takšna struktura prav tako omogoča dovolj informacije za generiranje SolidWorks CAD modela, kar bom opisal v poglavju 3. Struktura predstavlja tudi osnovo za uporabniški grafični vmesnik, ki mora biti kar najbolj enostaven.

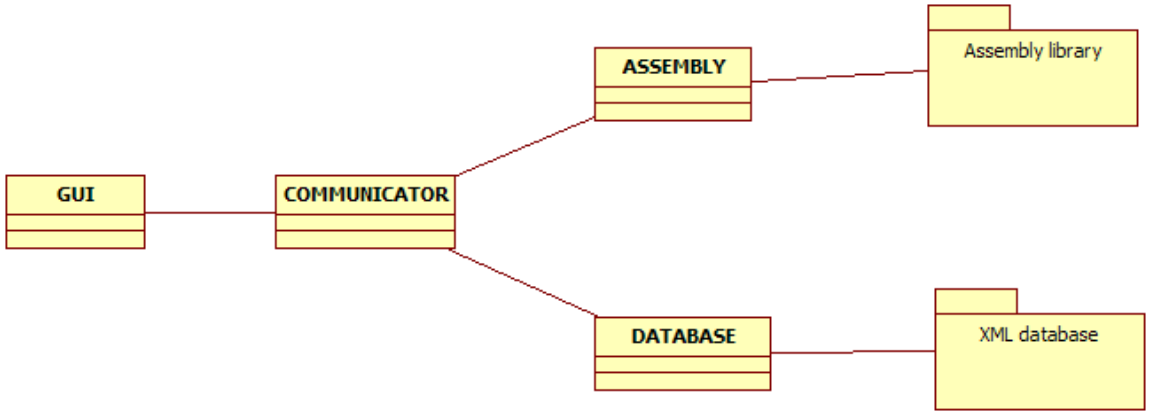
- sprotni izpis cene omogoča uporabniku, da ob vsaki spremembi programa vidi tudi vpliv spremembe na končno ceno vgradne omare

- vsaka aplikacija uporablja svojo bazo in zato ne potrebuje dodatnega strežnika za bazo podatkov, saj so vsi potrebni podatki shranjeni v XML datoteki

- zaradi enostavne baze podatkov je možno enostavno arhivirati celotno bazo
- aplikacija je prenosljiva (poenostavljena distribucija namestitev in popravkov)
- uvoz/izvoz projektov iz/v prenosljivo projektno datoteko

2.1.3 Organizacija programa z vidika programerja

Osnovna organizacija programa je predstavljena na spodnji shemi:



Slika 2: Organizacija aplikacije *GastonVO*

- Grafični uporabniški vmesnik (GUI) - zagotavlja uporabniške kontrole, dialoge za upravljanje z aplikacijo. Ta segment je z vidika končnega uporabnika najbolj pomemben.
- Upravljavec sestava (ASSEMBLY) črpa podatke iz knjižnice osnovnih sestavov vgradne omare in generira uporabniški vmesnik za določen sestav, ki ga potem preko povezovalnega modula prikaže uporabniški vmesnik.
- Upravljavec baze podatkov (DATABASE) opravlja vprašanja nad XML podatkovno bazo, ki vsebuje podatke o strankah, projektih in cenah posameznih osnovnih sestavov omare.
- Povezovalni modul (COMMUNICATOR) je organiziran kot razred *singleton* in skrbi za komuniciranje med ostalimi tremi moduli.

Takšna organizacija omogoča skrivanje podatkov uporabniškemu vmesniku. Programiranje uporabniškega vmesnika je omejeno na API nivo, ki ga izpostavlja povezovalni modul. Preko povezovalnega modula je prav tako zagotovljena tudi komunikacija med bazo podatkov in modulom za upravljanje z sestavi.

2.2 Uporabniški grafični vmesnik

2.2.1 Delovanje programa z vidika uporabnika

Z vidika končnega uporabnika (prodajalec v salonu) poteka delovanje programa v naslednjih korakih:

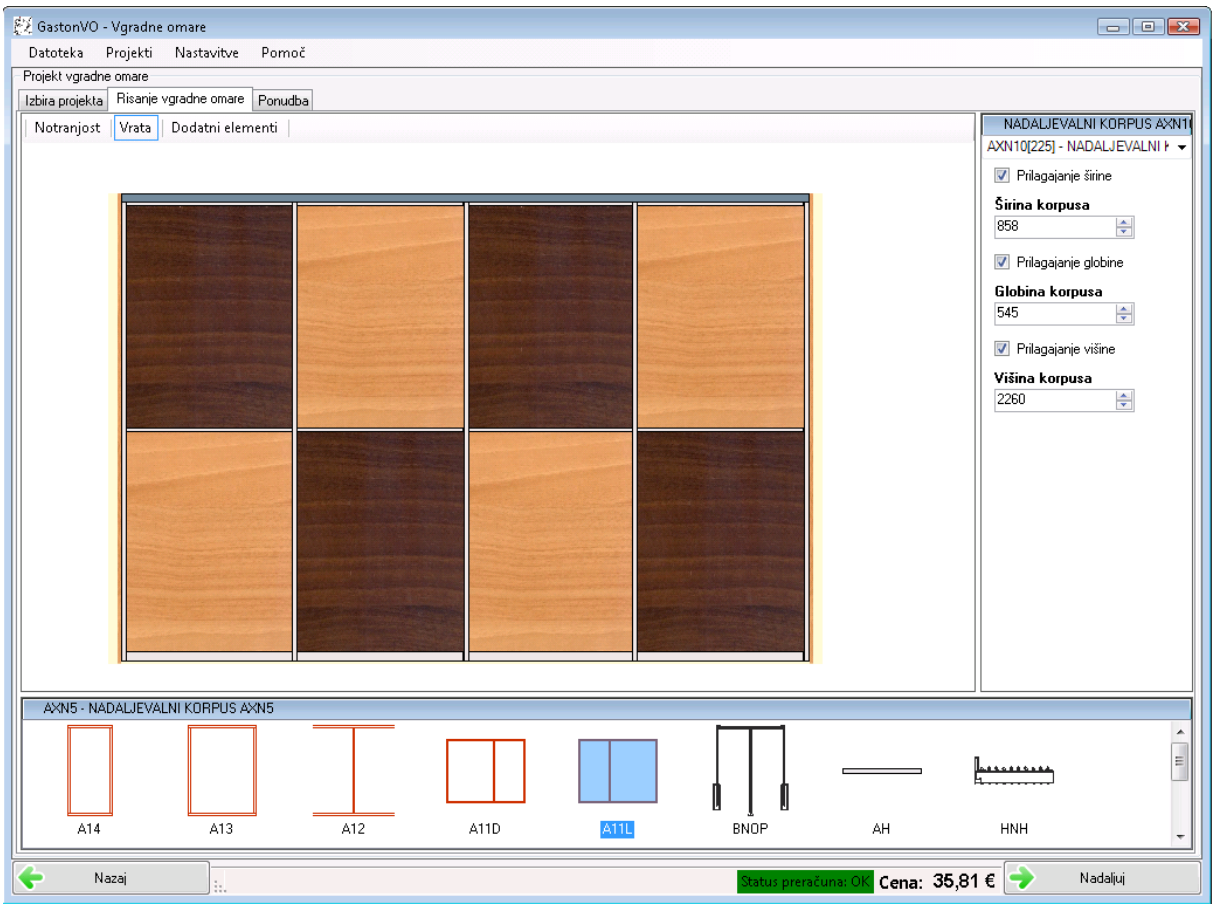
- odpiranje novega projekta,
- izbira oziroma vnos nove stranke,
- risanje sestava vgradne omare,
- prikaz in tiskanje poročil (ponudba, obrazec za izmero, ...).

2.2.2 Risanje sestava vgradne omare

Za preprost opis vgradne omare je potreben preprost uporabniški vmesnik. Organizacija grafičnih uporabniških kontrol je razdeljena na 3 segmente, kot je prikazano na sliki na strani 12. Največji del je namenjen narisu vgradne omare, desno od narisa je prostor za določanje vhodnih parametrov, pod narisom pa je prostor za prikaz sestavov, ki jih trenutni sestav lahko sprejme. Ko uporabnik klikne na določen sestav znotraj narisa vgradne omare, se osvežita oba pogleda: urejevalnik vhodnih parametrov prikaže vhodne parametre trenutnega sestava, v prostoru za prikaz sestavov pa se pojavijo sestavi, ki jih izbrani sestav lahko sprejme. Groba delitev uporabniškega vmesnika za risanje vgradnih omar je postavljena v času razvoja, medtem ko se vsebina teh kontrol spreminja v času izvajanja glede na obstoječo XML knjižnico osnovnih sestavov.

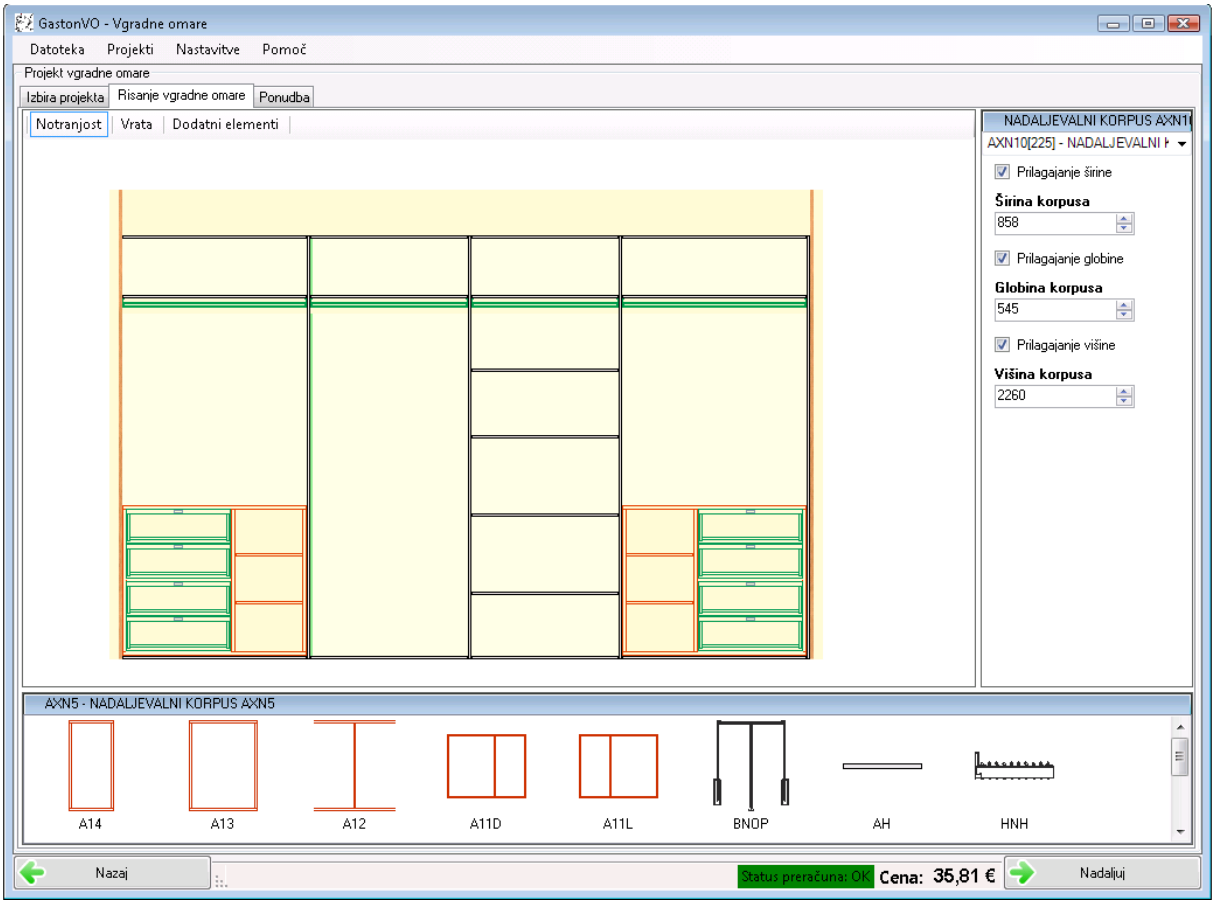
Ko uporabnik odpre nov projekt, se na zavihku risanje vgradne omare pojavi prazen vgradni prostor, ki se mu lahko v prostoru za parametre določi osnovne gabarite prostora: širino, višino ter globino. V vgradni prostor lahko sprejme več različnih tipov vgradnih omar, glede na tip vgradnje. Velikost omare se ustrezno prilagodi vnesenim osnovnim gabaritom. Klasična vgradna omara je sestavljena iz dveh pod-sestavov, in sicer vrata in notranji del omare oziroma prostor za korpuse.

Vrata so sestavljena iz dveh ali več vratnih kril za katere velja, da imajo vedno enako širino. Posamezno vratno krilo je sestavljeno iz levega in desnega vertikalnega profila, delitvenih profilov in polnil vratnih kril. Uporabnik določi razporeditev vratnih kril, ki je podrobno opisana v nadaljevanju, posameznim polnilom pa določi material na način, kot je prikazano na spodnji sliki.



Slika 3: Določanje polnil vratnih kril

Notranji del omare je sestavljen iz več korpusov. Vsak korpus lahko vsebuje različne elemente vgradne omare, kot so police, predali, izvlečne police, garderobno dvigalo, obešalnik za hlače, kravate itn. Korpus lahko vsebuje tudi notranji element, v katerega lahko vstavimo zopet nove elemente. Možnosti za razporeditev elementov vgradne omare je praktično neskončno.



Slika 4: Postavljanje notranjosti vgradne omare[2]

Uporabniku je omogočeno vizualno dodajanje in odstranjevanje posameznih elementov. Program sam prepoznava, ali določen element lahko vstavimo v določeno komponento sestava. V kolikor je to potrebno, program tudi testira prekrivanje komponent v prostoru. *GastonVO* dejansko pretvori 3D problem vgradne omare na 2D problem z vnosom minimalne informacije. Vse ostalo pa zagotavlja sistem. Uporabnik ves čas vidi prednji pogled omare (vrata, notranjost[3]) in se ne zaveda, da dejansko postavlja elemente v prostoru.

Uporabniku je v vsakem trenutku na razpolago tudi informacija o trenutni ceni vgradne omare, kar je pogosto tudi ključnega pomena za nakup.[4]

2.3 Jedro

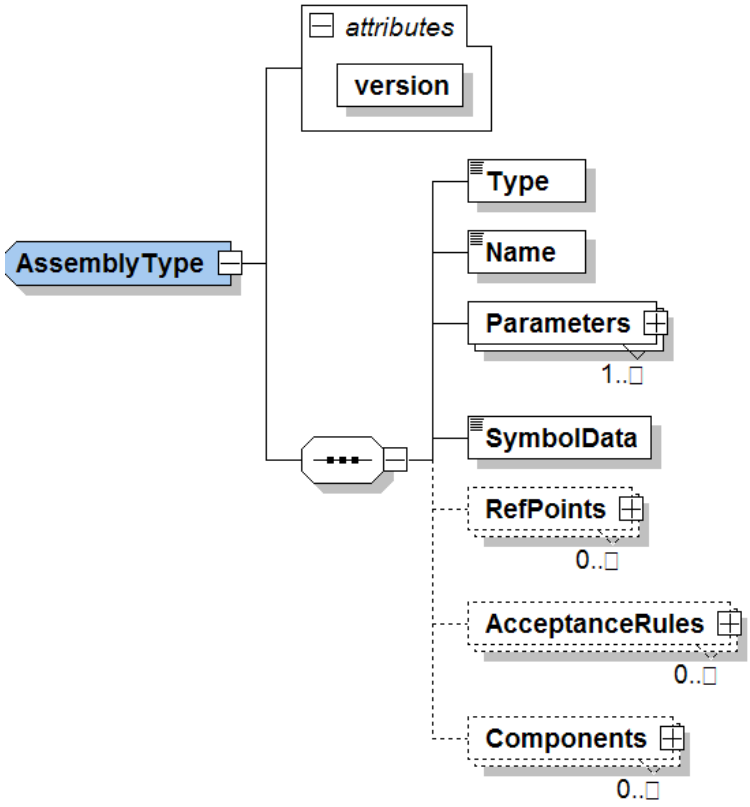
2.3.1 XML sestav

Kot sem že omenil, je program zasnovan na XML shemi(XSD [6]), ki predstavlja osnovni sestav v aplikaciji. Razvojno okolje projekta Microsoft .NET vsebuje veliko podporo za

[2] Sestavi prikazani na sliki so del prvega proizvajalca, ki uporablja program GastonVO (Alples d.d.).
[3] Program omogoča preklop med pogledi, ki so prikazani na obeh zgornjih slikah. (Pogled Dodatni element predstavlja specifičen pogled za proizvajalca in nima neposredne zveze s principom delovanja programa.)
[4] Cena na slikah je simbolična.

upravljanje z XML datotekami. Program *GastonVO* uporablja predvsem XML serializacijo [7], ki omogoča neposredno branje XML datoteke v primerek(*instance*) razreda tipa sestav (*Assembly*) in obratno zapis določenega primerka sestava v XML format. Na tem mestu se tudi uporabi odprtokodna knjižnica Xsd2Code [2], ki omogoča neposredno pretvorbo XML sheme v jezik C# znotraj .NET okolja.

V nadaljevanju razdelka bomo podrobneje pogledali osnovno idejo XML sestava. Na spodnji sliki je prikazana zunanja struktura XML sestava.

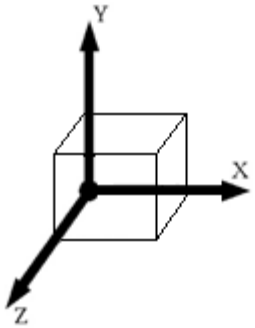


Slika 5: Shema XML sestava[5]

Splošne lastnosti XML sestava:

- Zasnovan je kot vozlišče v drevesni strukturi.
- Ne vsebuje 3D geometrije (mreže modela).
- Vsebuje 3D koordinatni sistem, kot je prikazano na spodnji sliki.

[5] Zaradi preglednosti ni prikazana celotna XML shema, ampak samo bistveni elementi za razumevanje njenega delovanja.



Slika 6: Koordinatni sistem sestava

Za vsak sestav velja, da ima njegov obsegajoči kvader izhodišče *levo, spodaj, zadaj* kot je prikazano na sliki levo.

Dolžina kvadra vzdolž X-osi je ŠIRINA sestava.
Dolžina kvadra vzdolž Y-osi je VIŠINA sestava.
Dolžina kvadra vzdolž Z-osi je GLOBINA sestava.

Koordinatni sistem je implicitno vključen v shemo in se ga upošteva pri vnosu parametrov in referenčnih točk

- Sestav je določen z množico parametrov, ki opisujejo njegove lastnosti in ga dokončno definirajo.
- Sestav je tipiziran kar pomeni, da je vsak tip predstavlja določene lastnosti, ki sestav enolično določajo. Sestavi istega tipa se morajo ujemati v številu in tipih parametrov, razlikujejo pa se lahko v vrednostih parametrov.
- Sestav ima množico referenčnih točk.
- Sestav ima množico pravil sprejemanja.
- Sestav A lahko sprejeme sestav B, če obstaja pravilo, ki sprejema tip B, in sicer tako, da postavi referenčno točko RB tipa B na referenčno točko RA tipa A.

Obrazložitev delovanja sheme po posameznih elementih v shemi:

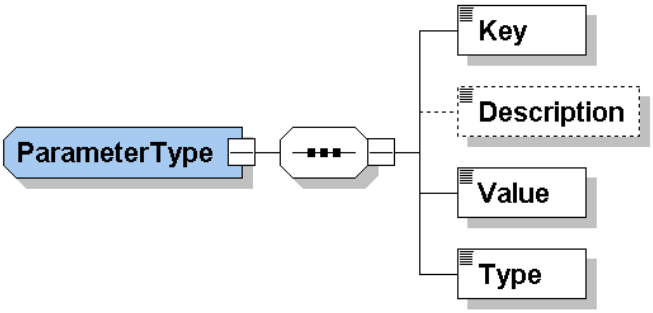
- tip sestava (*type*)

Polje vsebuje znakovni niz, ki predstavlja tip sestava (npr. *POLICA*). Kot bo razvidno v razdelku □, je tip ključen za definiranje pravil sprejemanja. Množica razpoložljivih tipov je globalna (definirana na nivoju sheme) in vidna slehernemu sestavu. Na tak način se minimizira napaka pri vpisovanju knjižnice, saj preverjanje veljavnosti s strani XML urejevalnikov javi napako, ko je vpisan tip sestava, ki ni definiran v shemi.

- ime sestava (*name*)

Unikatno ime posameznega vozlišča v sestavu. Kot sem že omenil predstavlja ena XML datoteka samo vozlišče v drevesni strukturi. Za vsako takšno vozlišče je potrebno zagotoviti, da ima unikatno ime.

- parametri (*parameters*)



Slika 7: Abstraktni tip *ParameterType*

V element *Key* se vpiše ključ, ki unikatno opisuje parameter znotraj sestava (W-širina, H-višina, …). Element *Description* je opcijski in je namenjen uporabniku za lažje razumevanje pomena parametrov, če je to potrebno. Elementa *Value* in *Type* pa opisujeta vrednost in tip vrednosti parametra. Z vidika sheme lahko vpišemo v polje *Value* poljubno vrednost, ki jo zna .NET tip s pomočjo pretvornika tipa [8], pretvoriti v ustrezen primerek razreda *Type*. Omenjena elementa potrebujeta nekoliko bolj podrobno razlago, kar bom ponazoril s konkretnima primeroma.

Primer 1: Preprost tip parametra

Parameter predstavlja nek osnovni tip (celo število, znakovni niz, znak, …), kar v XML zapišemo na sledeč način:

```
…
  <Value>5</Value>
  <Type> System.Int32</Type>
…
```

Vprašanje, ki se postavi na tem mestu, je, kako program ve, da mora niz 5 predstaviti kot celoštevilski tip. Rešitev je omogočena preko delnih razredov (*partial class*) in get/set metode znotraj lastnosti (*property*) razreda v jeziku C#. Delni razred pomeni, da je mogoče posamezni razred definirati v več datotekah. V našem primeru je takšna ureditev nujna, saj se del razreda avtomatsko generira iz XML sheme, drugi del pa predstavlja kodo, ki XML podatke ustrezno procesira.

Razred *ParameterType* vsebuje še pomožno uporabniško definirano lastnost *ValueRuntime*, ki lahko sprejme katerikoli tip. Program potem neposredno operira samo z lastnostjo *ValueRuntime*, dejanski podatki so vedno v spremenljivki *Value*, ki je tipa *System.String* in se shranjuje v XML datoteko. Program pa mora operirati z dejanskim tipom parametra, kar pomeni, da je potrebno znakovno predstavitev pretvoriti v primerek tipa, ki je zapisan v članu razreda (*class member*) *Type*.

Branje lastnosti *ValueRuntime* (get metoda):

- Če je vrednost pomožnega privatnega člana razreda *_valueRuntime* enaka NULL, se vrednost v spremenljivki *Value* s pomočjo .NET mehanizma pretvorbe tipov, pretvori v primerek tipa *Type*. Primerek tipa se shrani v član razreda *_valueRuntime*, s čimer se prihrani na procesiranju, saj se pretvorba opravi samo pri prvem branju.
- Vsak naslednji dostop vrne samo člana razreda *_valueRuntime*.

Pisanje v spremenljivko *ValueRuntime*(set metoda):

- Ob vsakem pisanju se vrednost zapiše v član razreda *_valueRuntime*.
- Ob vsakem pisanju pa se opravi tudi pretvorba v nasprotno smer (iz primerka v znakovno predstavitev), tako je v članu razreda *Value* vedno shranjena pravilna vrednost.

Primer 2: Uporabniško definiran tip vrednosti parametra

Pri razvoju XML sestav-a se največkrat uporablja osnovne tipe (*integer, string, boolean*). Lahko pa se pojavi tudi potreba po parametru, ki predstavlja bolj kompleksni tip parametra. V nadaljevanju si oglejmo konkreten parameter, ki se uporablja za določanje razporeditve tlorisa. Vratno vgradne omare imajo sprednje in zadnje vodilo na zgornji in spodnji tirnici. Razporeditev vratnih kril lahko prestavimo z nizom znakov *zszs* (zadaj, spredaj, zadaj,spredaj), kar pomeni, da je prvo vratno krilo nameščeno na zadnjem vodilu na tirnici, drugo na prednjem itn. Očitno za takšen tip ne moremo uporabiti nobenega osnovnega tipa, ampak je potrebno vpeljati uporabniško definiran razred, ki sem ga poimenoval *DoorLayout*.

XML zapis zgornjega razreda bi zapisali na naslednji način:

```
...
  <Value>zszs</Value>
  <Type>GastonAssembly.DoorLayout</Type>
...
```

Razred (predstavljena je samo osnovna zgradba), predstavljen v nadaljevanju, se uporablja za predstavitev razporeditve vratnih kril v programu. Prikazana je tudi uporaba .NET tehnologije pretvornika tipa. Vsak tip potrebuje svoj pretvornik tipa, ki opravi pretvorbo, ko je to potrebno. Funkcija *ConvertFrom* opravi pretvorbo iz XML tipa *Value*, ki je tipa *System.String* v tip *DoorLayout*, ki se nato uporablja v aplikaciji.

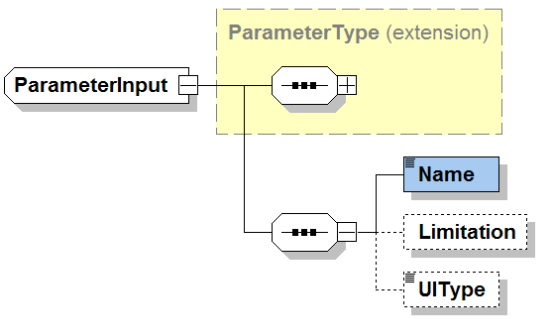
```
//Tipu DoorLayout se določi atribut pretvornik tipa
[TypeConverter(typeof(DoorLayoutTypeConverter))]
public class DoorLayout : ICloneable
{
    string _layout;

    public string Layout
    {
        get { return _layout; }
        set { _layout = value; }
    }
```

```
        public DoorLayout(string layout, int min, int max) { ... }

        public override string ToString() {...}

        //pretvornih tip je privaten razred razreda DoorLayout
        private class DoorLayoutTypeConverter : StringConverter
        {
            public override bool CanConvertFrom(ITypeDescriptorContext
context, Type sourceType) {...}

            public override object ConvertFrom(
                ITypeDescriptorContext context,
                System.Globalization.CultureInfo culture,
                object value) {...}
        }
    }
```

Naj ob takšni organizaciji XML strukture za parametre omenim, da je število bajtov veliko manjše, kot če bi razred opisali na klasičen način (definicija razreda v XSD shemi, implementacija v XML datoteki), kar posledično prinese tudi hitrejše branje xml datoteke in bolj kompakten, pregleden in razumljiv zapis.

Kot sem že omenil, je razred *ParameterType* abstrakten kar pomeni, da je v praksi vedno uporabimo njegove izpeljane tipe.

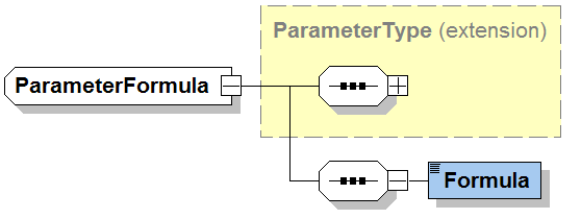
- **Vhodni parameter** (*ParameterInput*) – tip parametra sporoča, da vrednost parametera priskrbi uporabnik aplikacije, zato vhodni parameter obvezno potrebuje ime (polje *Name*), s katerim se predstavi uporabniku. Če gre za bolj zahteven tip parametra (npr. zgoraj omenjeni *DoorLayout*), potrebuje tak parameter še ustrezen tip za uporabniški vmesnik, ki je narejen posebej za določen parameter. Ker vrednost parametra določi uporabnik, je potrebno zagotoviti ustrezen vnos parametra, za kar poskrbi vnesena omejitev, ki uporabnika ustrezno omeji pri vnosu vrednosti. Polje *Limitation* je abstraktnega tipa *ConstraintType*, ki je organiziran podobno kot tip *ParameterType* tako, da imamo več različnih omejitev (omejitev števila se določi z minimalnim, maksimalnim številom in korakom, omejitev niza se določi z regularnim izrazom in dolžino, itn).



Slika 8: Tip *ParameterInput*

- **Konstantni parameter** (*ParameterFixed*) – predstavlja parameter s konstantno vrednostjo (npr. debelina spodnje tirnice je predstavljena kot konstantni parameter).

- **Parameter formula** (*ParameterFormula*) – v polju *Formula* je zapisan še izraz, ki se izračuna ob vsakem preračunu sistema sestava. Program *GastonVO* uporablja za preračun izrazov knjižnico FLEE [2]. V polje vrednost lahko zapišemo poljuben izraz, ki je sestavljen iz predhodno definiranih parametrov. Vzemimo za primer uporabe takšnega parametra pozicijo police od vrha notranjega korpusa vgradne omare. Najprej potrebujemo 2 parametra: vhodni parameter Višina korpusa (VK) in konstantni parameter Odmik police (OP). Pozicijo police (PP) določimo z uporabo formule VK-OP.



Slika 9: Tip *ParameterFormula*

- **Zunanji parameter** (*ParameterExternal*) – je poseben tip parametra, s katerim s sklicujemo na druge že obstoječe parametre, ki so definirani v drugih vozliščih sestava bodisi v vozliščih staršev bodisi v vozlišču enega izmed otrok. Ta parameter dejansko predstavlja kazalec na parameter, pri čemer je ključ vrednost kazalca. Zunanji parameter lahko podamo na več načinov, ki so prikazani v spodnji preglednici:

Način sklicevanja	**Sintaksa ključa**	**Uporaba**	**Primer**
Parameter starša	*ime_parametra*#...# (n-krat)	Sklic na starševski parameter n-nivojev gor, glede na trenutni sestav.	W## (parameter W, ki je 2 nivoja višje od parametra).
Parameter otroka	*ime_parametra*@ ime1@ ime2@ …@ imeN	Sklic na parameter otroka poljubno globoko v drevesni strukturi. *Ime1*…*ImeN* so imena sestavov, pri čemer je *ImeN* pri otrok trenutnega sestava *Ime1* pa je sestav, ki vsebuje parameter *ime_parametra*.	W@Polica-10@Korpusi-6 (parameter W se nahaja v sestavu Polica-10, ki je otrok sestava Korpus-6, ki je otrok sestava, ki vsebuje zunanji parameter).
Parameter istega nivoja	ime_parametra	Ko želimo uporabiti parameter znotraj istega nivoja, definiramo zunanji parameter z enakim	

		imenom ključa.	

Tabela 1: Načini uporabe zunanjega parametra

Za primer zunanjega parametra vzemimo vrata vgradne omare, ki vsebujejo dva ali več vratnih kril. Vrata vgradne omare so določena s parametri: višina, širina, razporeditev vratnih kril. Vzemimo, da imajo vrata širino 2500 mm, višino 2200 mm in razporeditev *zsz* (tip razporeditev vrat) in so predstavljena v sestavu vrat. Na podlagi teh podatkov lahko na nivoju vrat izračunamo tudi višino posameznega vratnega krila, ki je spravljena kot formula v parametru višina vratnega krila (VVK). Sedaj želimo definirati sestav vratno krilo. Višina posameznega vratnega krila je neposredno odvisna od višine celotnih vrat. Ker se višina vratnega krila izračuna na nivoju sestava vrata, je potrebno v sestavu vratnega krila uporabiti samo zunanji parameter. Na ta način dosežemo, da ob spremembi originalnega parametra na vseh mestih v programu vedno dobimo pravo vrednost parametra.

Na tem mestu velja omeniti še problem, ki izhaja iz okolja .NET, ki pozna dve vrsti tipov glede na obravnavo v pomnilniku: vrednostni in referenčni tipi. Problem, ki se je pojavil pri razvoju zunanjega parametra je bil zapis kazalca na vrednostne tipe, ki se največkrat uporabijo za opis parametrov. Vrednostni tipi se v .NET okolju pri prirejanju vedno kopirajo, pri referenčnih tipih pa se ob prirejanju vedno kopira samo referenco (kazalec).

Kako implementirati kazalec na vrednostni tip? Odgovor je v preprostem razredu, ki ga vpeljemo s pomočjo generikov v jeziku C#. Razred kazalec je implementiran kot:

```
public class Ptr<T>
{
    Func<T> getter;
    Action<T> setter;

    public Ptr(Func<T> g, Action<T> s)
    {
        getter = g;
        setter = s;
    }

    public T Deref
    {
        get { return getter(); }
        set { setter(value); }
    }
}
```

Primer kazalca na tip *System.String* in njegova uporaba sta predstavljena s spodnjo kodo:

```
int i = 5;
Ptr<int> pi = new Ptr<int>(() => i, val => i = val);
int j = pi.Deref;
pi.Deref = i;
```

Navzven se mora tip zunanjega parametra ujemati z originalnim tipom parametra. Z vidika programa pa je v pomožnem polju dejansko spravljen kazalec tipa *Ptr<T>*. Kot sem že

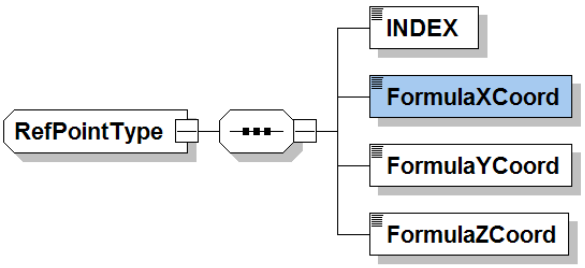
omenil se do vrednosti parametrov vedno dostopa prek lastnosti *ValueRuntime*, ki poskrbi da se vselej vrne prava vrednost (dereferenciacija vrednosti kazalca).

- Simbol sestava (*SymbolData*)

Polje lahko vsebuje poljubno simbolično predstavitev sestava. Polje je tipa znakovni niz (*string)*, ki lahko predstavlja relativno pot na zunanjo sliko (*jpg, png, emf, wmf*) npr. *./slike/slika.jpg*. Lahko pa vsebuje tudi *base64* znakovni niz, ki ga potem program med branjem pretvori v enega izmed podprtih slikovnih formatov, ki jih zna prikazati. Program *GastonVO* uporablja *emf* in *base64* zapis.

- Referenčne točke (RefPoints)

Točke predstavljajo neposredno razpoložljiva mesta, kamor je možno dodajati določene tipov sestavov, ki so zapisani v pravilih sprejemanja, kar bo razvidno v poglavju □. Posamezna referenčna točka je definirana z unikatnim indeksom in formulami, v katerih lahko kot spremenljivke nastopajo parametri določeni v tabeli parametrov (poglavje □). Vsaka točka v 3D prostoru je določena s tremi koordinatami v X, Y in Z smeri. Za vsako koordinatno os program dopušča svojo formulo, ki pa je lahko seveda tudi konstantna točka, pri čemer so vpisane vrednosti konstante. Program *GastonVO* za preračun koordinat posamezne točke uporablja že omenjeno knjižnico za računanje izrazov.



Slika 10: Referenčna točka

Ob tem velja omeniti, da se lahko ena referenčna točka glede na formulo razvije v n realnih točk v 3D prostoru, zato si oglejmo konkreten primer, ki predstavi funkcionalnost takšne točke.

Primer 3: Referenčna točka za polico osnovnega korpusa

Notranji del vgradne omare je pogosto zgrajen iz korpusov, ki so sestavljeni iz dna, stropa, leve in desne stranice ter včasih tudi hrbtišča. Predstavljajo nekakšen osnovni gradnik za sestavljanje vgradne omare, ki ga je potrebno še dopolniti z dodatnimi elementi kot so police,

izvlečne police, predali, dvigalo, razna obešala itn. Oglejmo si torej, kako opišemo v XML referenčno točko za polico korpusa. Predpostavim, da je korpus zgrajen iz plošč debeline 18 mm. Leva in desna stranica imajo že izvrtane luknje, kamor se lahko pritrdijo police. Odmik prve luknje od dna in zadnje luknje od stropa korpusa je 40 mm. Luknje so izvrtane s korakom 32 mm. Višina korpusa je predstavljena z parametrom H, globina pa z D. Primer XML zapisa referenčne točke je predstavljen s spodnjo XML kodo:

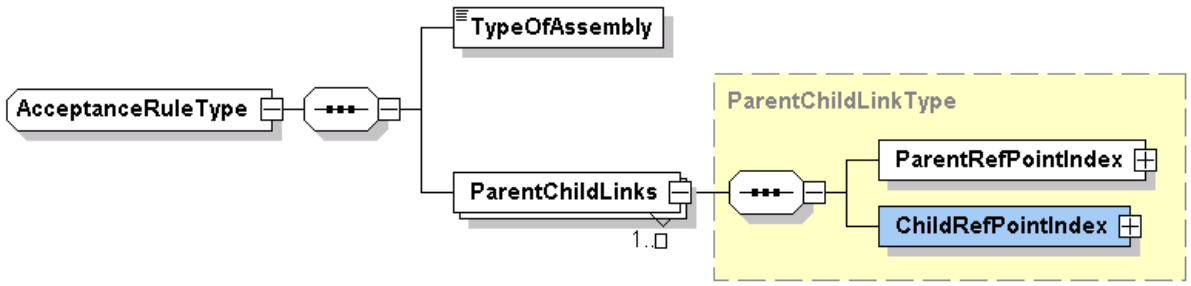
```
…
<RefPoints>
   <INDEX>0</INDEX>
   <FormulaXCoord>18</FormulaXCoord>
   <FormulaYCoord>Interval(40,H-40,32)</FormulaYCoord>
   <FormulaZCoord>D</FormulaZCoord>
</RefPoints>
…
```

Posebnost zgornjega opisa je v tem, da se referenčna točka z indeksom 0 razvije v n-referenčnih točk. Interval predstavlja statično uporabniško definirano funkcijo, ki jo povežemo s knjižnico FLEE. Takšen postopek vpisovanja omogoča veliko lažji razvoj knjižnice in zmanjša možnost pri vnosu podatkov s strani uporabnika. Lahko si predstavljamo, koliko vpisov bi bilo potrebno pri korpusu višine 2500 mm oziroma koliko popravljanja bi bilo potrebno, če bi kar naenkrat spremenili npr. odmik prve luknje s 40 na 50 mm. Preko zgornjega vpisa je potrebna samo ena sprememba.

- Pravila sprejemanja (*AcceptanceRules*)

Vsak tip sestava je definiran s številom in tipi parametrov. Vsak parameter pa lahko sprejema tudi druge tipe sestavov. V zadnjem primeru smo spoznali, da lahko korpus vgradne omare sprejme različne elemente. Katere elemente posamezen sestav lahko sprejeme, je določeno v tabeli pravila sprejemanja. Vsak zapis vsebuje najprej tip sestava in enega ali več povezav starš-otrok, na katere se določeni tip sestava lahko poveže. Povezava starš-otrok je definirana z referenco na konkretno 3D točko starševskega sestava (npr. korpus vgradne omare) in otrokovega sestava (npr. polica). Kot sem opisal v prejšnjem poglavju, lahko posamezen zapis referenčne točke predstavlja več konkretni točk v 3D prostoru zato ni dovolj, da bi povezali samo referenčne točke starša in otroka v drevesni strukturi, ampak potrebujemo še dodatno specializacijo referenčne točke oziroma indeks določene rešitve referenčne točke, ki pove, katera izmed rešitev znotraj referenčne točke se povezuje. Konkretna točka je potem opisana

kot par: indeks referenčne točke in indeks določene rešitve referenčne točke.[6]



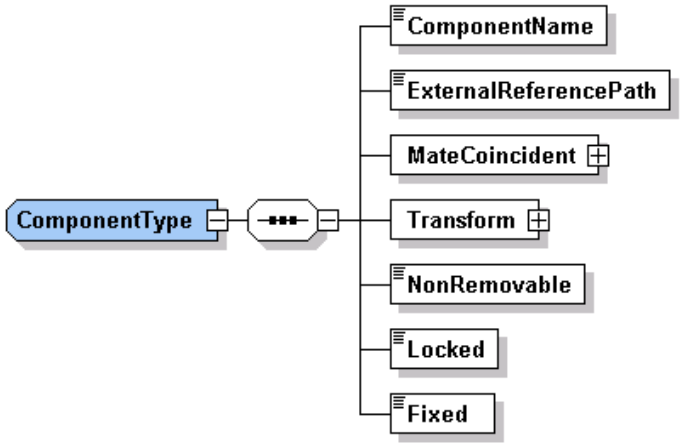
Slika 11: Pravilo sprejemanja

- Komponente sestava (*Components*)

Zadnja bistvena lastnost XSD sheme so komponente. Komponente predstavljajo most med vozliščem starša in otroka. Vsako komponento določa:

- unikatno ime znotraj sestava (*ComponentName*) (npr. *Korpus-10*),
- relativna pot do referenčnega sestava (*ExternalReferencePath*), ki ga komponenta predstavlja (npr. *./korpusi/korpus1.xml*),
- povezava starš-otrok (*MateCoincident*), ki mora biti veljavna povezava v sestavu,
- transformacijska matrika (*Transform*), ki vsebuje trenutno relativno pozicijo glede na sestav v 3D prostoru. Matrika je z vidika programa *GastonVO* samo pisalna. Vrednosti v matriki lahko preberejo drugi programi med generiranjem 3D pogleda.
- neodstranljiva komponenta – komponente ni mogoče izbrisati iz sestava. S kontrolno vrednostjo preprečimo uporabniku, da bi izbrisal komponento iz sestava.
- zaklenjena komponenta – interakcija uporabnika s komponento je onemogočena. Vrednost pove programu, da komponenta ne reagira na nobeno uporabniško interakcijo s programom. Zaklenjene komponente so v glavnem vključene samo zaradi pravilne vizualizacije znotraj programa
- fiksirana komponenta – komponente uporabnik ne more premakniti, lahko pa jo izbriše is sestava.

[6] *ParentRefPointIndex* in *ChildRefPointIndex* na Slika 11: Pravilo sprejemanja nista prikazana v celoti.



Slika 12: Komponenta

Po končani deserializaciji XML datoteke se začne po-procesiranje prebranih podatkov, kjer se na podlagi XML podatkov izpolnijo ustrezne dinamične strukture, ki so definirane v že omenjenih delnih razredih. Po-procesiranje naloži podatke o sestavih za vsako komponento, kar pomeni, da se rekurzivno ponovi identičen postopek (serializacija + po-procesiranje) za sestav, ki ga določa relativna referenčna pot znotraj komponente. V pomnilniku ima vsaka komponenta referenco na ustrezni sestav, vsak sestav (razen korena) pa ima referenco na ustrezno komponento, kar nam omogoča, da se izvajajo različne operacije na celotnem drevesu. Dobimo torej drevesno strukturo, kjer so sestavi, ki imajo komponente, vozlišča. Sestavi brez komponent pa so list v drevesu.

2.3.2 Podatkovna baza

Podatkovna baza prav tako uporablja .NET XML tehnologijo (serializacija/deserializacija objektov v/iz XML datoteke). Podatkovna baza je razdeljena na 3 segmente, ki so podrobneje opisani v nadaljevanju. Z vidika aplikacije je baza podatkov vidna kot 3 datoteke, in sicer *GVO.xml.Database*, *GVOCommon.xml.Database* in *GVOUser.XML.Database*. Omenjene datoteke so navadne arhivirane (*zip*) datoteke, ki pa uporabljajo AES šifriranje z 256 bitnim ključem. Na ta način se uporabniku onemogoči neposreden vpogled v podatke, spreminjanje podatkov in omogoči ščitenje podatkov pred nepooblaščenim vpogledom tretje osebe. V spodnjih razdelkih so podrobneje opisane funkcionalnosti posamezne podatkovne baze.

- GVO

Predstavlja samo bralno XML podatkovno bazo, ki vsebuje parametričen sistem šifer/cen glede na knjižnico osnovnih sestavov. Pomembna lastnost baze je povezava s knjižnico osnovnih sestavov vgradne omare. Potrebno je namreč zagotoviti cene za sestave, ki so cenovno ovrednoteni in jih povezati s knjižnico osnovnih sestavov. Cena se nato računa kot vsota vseh osnovnih sestavov s ceno.

- GVOCommon

Bralno pisalna XML podatkovna baza ima več funkcij. Uporablja se za shranjevanje začetnih nastavitev programa (izbiro cenika, trgovca, podatki o salonu, …). Poleg nastavitev pa se jo uporablja tudi v povezavi s Crystal Reports 2008 [5] tehnologijo za prikaz ponudbe, slik, obrazca za izmero.

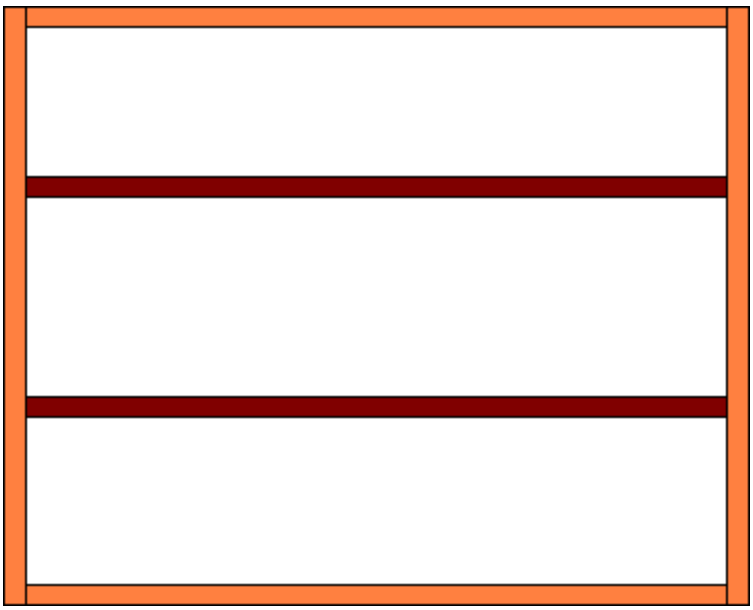
- GVOUser

Bralno pisalna XML podatkovna baza je preprosta baza za shranjevanje tekočih projektov. Dejansko je sestavljena iz dveh delov, in sicer zapisa v bazi in referenčne projektne datoteke, kjer je shranjen sestav vgradne omare.

Omenimo, da program omogoča unikatno generiranje serijskih številk projekta, če proizvajalec tako želi. Serijska številka je sestavljena iz številke salona, številke delovne postaje v salonu, številke podatkovne baze in zaporedne številke projekta.

2.4 Uporaba XML strukture

Sedaj prehajam k opisu konkretnega sestava, ki bi se vstavil v notranji korpus vgradne omare, kot je prikazano na Slika 4. V posamezen korpus vgradne omare lahko dodajamo različne sestave/sestavne dele, kot so dodatni notranji elementi, police, predali itn. Za prikaz delovanja XML strukture poimenujmo sestav *Notranji Element* in njegov tip naj bo TNE. Element tipa TNE naj bo konstanten v globini, višini in širini. Sestavljen je iz leve in desne stranice, stropa in dna, ki pa jih ni možno spreminjati. Debelina materiala elementa TNE je 18 mm. Simbol za prikaz elementa je spravljen v EMF datoteki *TNE.emf*. V sestav tipa TNE lahko vstavimo police tipa TP. Polica tipa TP se prilagodi razpoložljivemu prostoru elementa tipa TNE. Debelina police je vedno enaka debelini njegovega starša. Polica se lahko pritrdi na levo stranico elementa TNE, pri čemer je prva možna točka oddaljena 45 mm po Y-osi, ostale pa gredo naprej s korakom 32 mm do višine elementa TNE. X-koordinata referenčnih točk police je enaka debelini leve stranice. Z-koordinata pa je enaka nič.



Slika 13: Sestav tipa TNE sestavljen iz leve, desne stranice,stropa, dna[7]

Ena izmed možnih rešitev za opis sestava TNE je lahko spodnja XML koda:

```
<Assembly >
        <Type>TNE</Type>
        <Name>NotranjiElement</Name>
        <Parameter xsi:type="ParameterFixed">
                <Key>W</Key>
                <Description>Element se prilagodi na razpoložljivo širino, ki jo izračuna in
njegov starš v parametru WA</Description>
                <Value>425</Value>
                <Type>System.Double</Type>
        </Parameter>
        <Parameter xsi:type="ParameterFixed">
                <Key>H</Key>
                <Description>Višina elementa je vedno konstantna</Description>
                <Value>400</Value>
                <Type>System.Double</Type>
        </Parameter>
        <Parameter xsi:type="ParameterFixed">
                <Key>D</Key>
                <Description>Globina elementa je vedno konstantna</Description>
                <Value>450</Value>
                <Type>System.Double</Type>
        </Parameter>
        <Parameter xsi:type="ParameterFixed">
                <Key>THICKNESS</Key>
```

[7] Z rdečo barvo sta prikazani polici, ki nista del osnovnega sestava TNE, ampak postaneta del sestava šele, ko jih tja postavi uporabnik.

```
            <Description>Debelina materiala za strop,dno, levo in desno
stranico</Description>
            <Value>18</Value>
            <Type>System.Double</Type>
        </Parameter>
        <Parameter xsi:type="ParameterFormula">
            <Key>W_TP#</Key>
            <Description>Širina horizontalne police, stropa, dna</Description>
            <Value>400</Value>
            <Type>System.Double</Type>
            <Formula>W - 2* THICKNESS</Formula>
        </Parameter>
        <SymbolData>./TNE.wmf</SymbolData>
        <RefPoints>
            <INDEX>0</INDEX>
            <Description>Polica</Description>
            <FormulaXCoord>THICKNESS</FormulaXCoord>
            <FormulaYCoord>Interval(45,32,H)</FormulaYCoord>
            <FormulaZCoord>0</FormulaZCoord>
        </RefPoints>
        <AcceptanceRules>
            <TypeOfAssembly>TP</TypeOfAssembly>
            <ParentChildLinks>
                <ParentRefPointIndex Index="0" SpecificSolutionIndex="0"/>
                <ChildRefPointIndex Index="0" SpecificSolutionIndex="0"/>
            </ParentChildLinks>
        </AcceptanceRules>
</Assembly>
```

Kot lahko vidimo, ima sestav TNE eno pravilo sprejemanja, ki pravi, da sprejme sestav tipa TP tako, da postavi referenčno točko sestava TP na referenčno točko sestava TNE. Poveže se torej levi spodnji zadnji kot police z ustrezno 3D točko na sestavu starša. Referenčna točka 0 uporablja že omenjeno statično funkcijo Interval in se razvije v 12 realnih 3D točk.

Sedaj si oglejmo še kako izgleda sestav TP:

```
<Assembly >
        <Type>TP</Type>
        <Name>Polica</Name>
        <Parameter xsi:type="ParameterExternal">
            <Key>THICKNESS#</Key>
            <Value>18</Value>
            <Type>System.Double</Type>
        </Parameter>
        <Parameter xsi:type="ParameterExternal">
            <Key>W_TP#</Key>
            <Value>400</Value>
            <Type>System.Double</Type>
        </Parameter>
        <Parameter xsi:type="ParameterExternal">
```

```
            <Key>D#</Key>
            <Value>450</Value>
            <Type>System.Double</Type>
        </Parameter>
        <Parameter xsi:type="ParameterFormula">
            <Key>W</Key>
            <Description>Širina</Description>
            <Value>400</Value>
            <Type>System.Double</Type>
            <Formula>W_TP#</Formula>
        </Parameter>
        <Parameter xsi:type="ParameterFormula">
            <Key>D</Key>
            <Description>Globina</Description>
            <Value>450</Value>
            <Type>System.Double</Type>
            <Formula>D#</Formula>
        </Parameter>
        <Parameter xsi:type="ParameterFormula">
            <Key>H</Key>
            <Description>Globina</Description>
            <Value>450</Value>
            <Type>System.Double</Type>
            <Formula>THICKNESS#</Formula>
        </Parameter>
        <SymbolData>128;0;0</SymbolData> <!—Možen simbol v eni barvi(temno rdeča)-->
        <RefPoints>
            <INDEX>0</INDEX>
            <Description>Levi spodnji kot</Description>
            <FormulaXCoord>0</FormulaXCoord>
            <FormulaYCoord>0</FormulaYCoord>
            <FormulaZCoord>0</FormulaZCoord>
        </RefPoints>
        <NamingIndex>1</NamingIndex>
</Assembly>
```

Polica je definirana z debelino materiala, širino in globino starša. Simbol za predstavitev police je kar RGB vrednost, kar pomeni, da program zagotavlja predstavitev police v ustrezni barvi. Sestav TP ne sprejema nobenega drugega sestava, potrebuje pa nujno vsaj eno referenčno točko, ki se uporablja, kadar uporabnik vstavlja polico v sestav tipa TNE.

2.5 Knjižnica

Organizacija knjižnice je z vidika aplikacije zelo pomembna. Vsi osnovni sestavi so shranjeni v glavnem imeniku knjižnice. Vsak tip sestava je predstavljen v svojem imeniku, ki mora biti enak imenu tipa. Znotraj imenika posameznega tipa se lahko nahaja več različnih sestavov, ki pa morajo biti vsi istega tipa. Če ima sestav v knjižnici komponente, se le-te sklicujejo na zunanji sestav preko relativne poti. Sestav A v imeniku A vsebuje komponento AC1, ki se

sklicuje na sestav B, ki se nahaja v imeniku B. Referenčna pot je podana kot *../B/B.xml*. Na takšen način lahko sestavimo optimalno knjižnico brez nepotrebnega podvajanja sestavov.

Pri tem naj omenim, da zaradi optimizacije pri sestavljanju vgradne omare knjižnico v celoti preberemo ob zagonu programa. Na ta način se pohitri iskanje in postavljanje sestavov v omari.

Knjižnico za primer iz prejšnjega poglavja lahko zapišemo kot:

Imenik TNE → TNE.xml, TNE.emf

Imenik TP → TP.xml

Torej bi imeli dva imenika TNE in TP, ki vsebujeta vsak svojo XML datoteko z istim imenom, kot je ime imenika in ostale resurse, na katere se lahko XML datoteke sklicujejo.

3 GastonVO vtičnik za *SolidWorks*

3.1 Kratka predstavitev *SolidWorks* okolja

3.1.1 O programu *SolidWorks*

Razvojno okolje *SolidWorks* je bilo prvič lansirano na tržišče leta 1995. V 15-letni rasti se je razvila tudi podpora za pisanje programov, ki avtomatizirajo *SolidWorks* funkcionalnost. Podobno kot pri Microsoft Office okolju je možno pisanje makrojev v jeziku Visual Basic 6. V zadnjem času pa je omogočena že tudi podpora za razvojno okolje .NET (Visual Studio predloga za C# vtičnik).

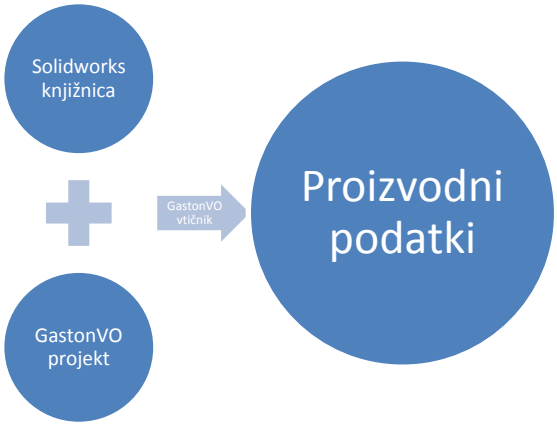
3.1.2 *SolidWorks* API arhitektura

Osnovni *SolidWorks* dokumenti so: sestav (*Assembly*), sestavni del (*Part*), delavniška risba (*Drawing*). Part je najmanjši del, ki vsebuje geometrijo (mrežo). Sestav vsebuje poljubno število sestavnih delov in jih umešča v prostor preko transformacijskih matrik. Delavniška risba pa prikazuje 2D poglede bodisi sestava bodisi sestavnega dela.

Za upravljanje z osnovnimi dokumenti skrbi vmesnik *IModelDoc*, ki obstaja v specializiranih različicah za vsak tip dokumenta (vmesniki *IDrawingDoc*, *IAssemblyDoc*, *IPartDoc*). Podrobnosti o API nivoju so na voljo v programu *SolidWorks* [9].

3.1.3 Osnovno delovanje vtičnika *GastonVO*

Namen vtičnika *GastonVO* za *SolidWorks* je pripraviti končne podatke za proizvodnjo: 3D CAD sestav, delavniška dokumentacija ter kosovnica. Program *GastonVO* temelji na enostavnem opisovanju parametrov vgradne omare, medtem ko proizvodni modul zapolni manjkajoče podatke, ki jih proizvodnja potrebuje, da lahko uspešno sestavi vgradno omaro. Osnovna ideja je prikazana na sliki spodaj. Vtičnik *GastonVO* za program *SolidWorks* generira vgradno omaro na podlagi zapisa v projektni datoteki programa *GastonVO* in SolidWorks knjižnice, ki jo pripravi proizvajalec. Podrobnosti implementacije bodo prikazane v poglavju 3.3.

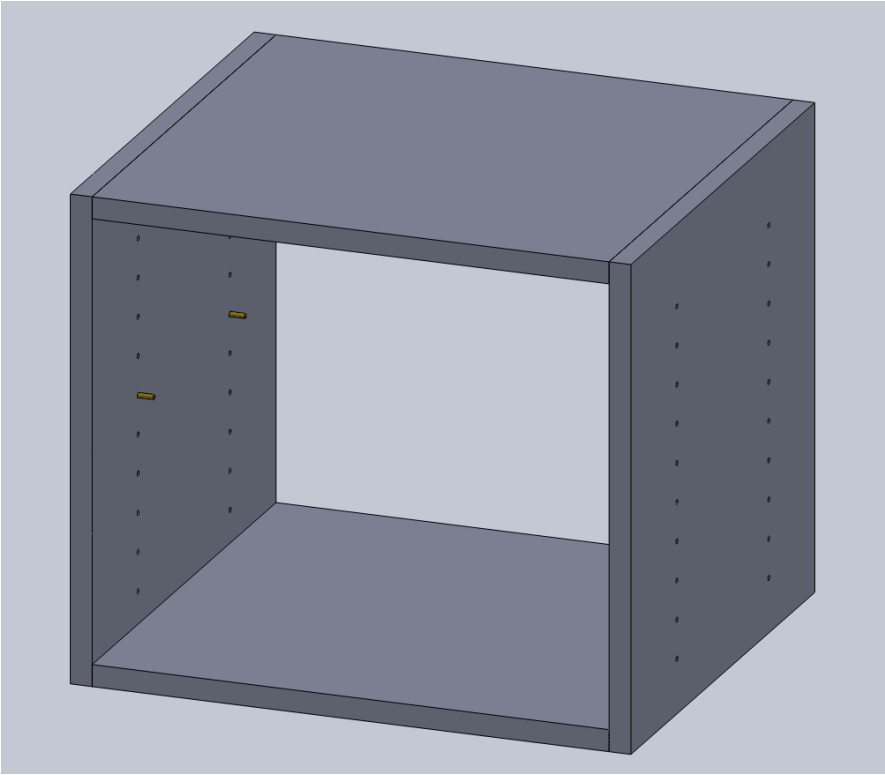


Slika 14: Proces pridobivanja proizvodnih podatkov

3.2 Knjižnica osnovnih sestavov

3.2.1 Analogija z knjižnico *GastonVO*

Vtičnik *GastonVO* za *SolidWorks* potrebuje analogno vnaprej pripravljeno knjižnico enako kot pri samostojnem programu *GastonVO*. Vsak definiran tip sestava potrebuje svojo ustrezno *SolidWorks* predstavitev. Strogo gledano bi dejansko potrebovali za vsak tip sestava znotraj knjižnice *GastonVO* sestav v *SolidWorks* knjižnici. Vzemimo sestav TNE iz poglavja 2.4. Sestav *TNE.xml* potrebuje ustrezno prestavitev kot *SolidWorks* sestav *TNE.sldasm*. Pri razvoju XML opisa sestava smo zagotovili minimalno informacijo, ki jo program potrebuje zato, da se lahko v sestav TNE vstavlja polico TN. Ničesar pa nismo povedali o dodatnih ostalih lastnostih sestava TNE, saj so z vidika programa *GastonVO* nepomembne za izračun cene. Sestav TNE ima dve vrsti lukenj na levi in desni stranici, kamor se pritrdi police TN. Za vsako polico potrebujemo še 4 dodatne sestavne dele za pritrditev na sestav TNE, prikazuje spodnja slika.



Slika 15: *SolidWorks* sestav TNE[8]

3.2.2 Priprava *SolidWorks* knjižnice osnovnih sestavov

Najprej moram omeniti, da se termin sestav z vidika programa *GastonVO* v *SolidWorks*-u razbije na dva dela, in sicer sestavni del in sestav. *GastonVO* takšne delitve ne pozna, saj je ne

[8] Sestav na sliki je izmišljen.

potrebuje. Sestavni del je v XML datoteki predstavljen kot sestav, ki nima nobene komponente ampak samo geometrijo.

Pri pripravi osnovnih sestavov lahko ločimo 2 primera:

- fiksni sestavi (stopnička kot dodatek omari),
- parametrični sestavi (vsi sestavi, ki so variabilni glede na vhodne parametre).

Za fiksne sestave je priprava *SolidWorks* sestavnega dela/sestava trivialna. Potrebno je samo umestiti sestavni del/sestav v koordinatni sistem, kot je prikazano na sliki na strani 15.

Za parametrične sestave pa je potrebno zagotoviti, da jih v fazi generiranja ustrezno preoblikujemo glede na vrednosti parametrov, ki jih dobimo iz projektne datoteke programa *GastonVO*. Način vnosa teh parametrov na nivoju *SolidWorks* sestava/sestavnega dela je dokaj zahteven. CAD modeliranje je dokaj kompleksno in vključuje ogromno število sistemov enačb in neenačb ter omejitev. Z nekaj naprednejšega programerskega znanja je mogoče nastavljati poljubne dimenzije na različnih nivojih sestava ali sestavnega dela. Če vemo, katera dimenzija predstavlja širino sestava, jo lahko znotraj *SolidWorks* okolja poiščemo in nastavimo. Takšna rešitev bi lahko funkcionirala, toda pojavljale bi se nenehne težave, če bi se struktura posameznih sestavov spreminjala, kar bi posledično pomenilo neprestano popravljanje programa. To pa je nesprejemljivo.

Z uporabo uporabniško definiranih parametrov (*Custom Properties*) v *SolidWorks*-u lahko izpostavimo tiste značilke (*features*) posameznega sestava/sestavnega dela, ki predstavljajo vhodne parametre. Lastnosti za polico TP, ki jo lahko vstavimo v TNE in se lahko predeluje po vseh treh dimenzijah (širina, višina, globina), bi lahko definirali tako, kot je prikazano na spodnji sliki.

	Property Name	Type	Value / Text Expression	Evaluated Value
1	W	Text	"D1@Sketch4@dno.SLDPRT"	425
2	H	Text	"D2@Sketch4@dno.SLDPRT"	18
3	D	Text	"D1@Boss-Extrude1@dno.SLDPRT"	400
4				

Slika 16: Poljubne lastnosti v *SolidWorks*

V programu *GastonVO* je uporabnik definiral širino, globino in višino notranjega korpusa, kar pomeni, da je potrebno te vrednosti nastaviti tudi v *SolidWorks* sestavu/sestavnem delu. Programer pričakuje izpostavljene ustrezne lastnosti, če manjkajo, program javi napako. Risar v *SolidWorks*-u pa mora zagotoviti, da izpostavi vse parametre, ki jih določenemu sestavu lahko spreminjamo.

Ob takšni zasnovi *SolidWorks* sestavov/sestavnih delov se pojavi še problem skladnosti imen parametrov. V XML datoteki so imena parametrov shranjena v polju ključ. Spodnji dve rešitvi nam zagotavljata, da se imena parametrov skladajo oziroma, da se ve, kateri parameter je potrebno uporabiti za nastavljanje parametrov v *SolidWorks*-u.

- Definiranje imen parametrov pred izdelavo knjižnice. To pomeni, da ima vsak vhodni parameter sestava v knjižnici programa *GastonVO* natančno enako lastnost v *SolidWorks* sestavu/sestavnem delu. Ta rešitev pomeni veliko discipline in nujno usklajevanje pri izdelavi obeh knjižnic.
- Preslikovalna datoteka, ki pove, kako se slikajo imena parametrov posameznih *GastonVO* sestavov v lastnosti *SolidWorks* sestavov/sestavnih delov.

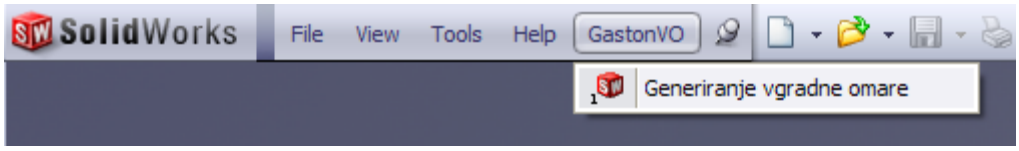
Druga rešitev je na videz bolj naravna, saj dopušča, da se knjižnici sestavita ločeno. Dejansko pa gre za vprašanje, ali v primeru napak popravljamo bodisi imena parametrov v eni ali drugi knjižnici bodisi popravljamo preslikovalno datoteko. Če *SolidWorks* knjižnico uporablja še kakšna druga aplikacija, je nujna druga rešitev.

Organizacija knjižnice je popolnoma analogna kot pri programu *GastonVO*. Vsak tip sestava ima svoj imenik, ki vsebuje vse potrebne *SolidWorks* datoteke, ki jih določen tip potrebuje (3D model, tehnična dokumentacija itn). Podobno kot pri problemu skladnosti imen parametrov je potrebno zagotoviti preslikavo *GastonVO* tipa v ustrezen *SolidWorks* sestav.

3.3 *GastonVO* vtičnik

V prejšnjem poglavju sem analiziral organizacijo *SolidWorks* knjižnice in ustrezno povezovanje sestavov programa *GastonVO* z *SolidWorks* sestavi/sestavnimi deli. Sedaj pa prehajam k opisu delovanja vtičnika.

Znotraj *SolidWorks* okolja se pojavi nova rubrika *GastonVO*, kjer poženemo ukaz za generiranje vgradne omare.



Slika 17: Integriran meni kot vtičnik *GastonVO* v okolju *SolidWorks*

Po izbiri ustreznega *GastonVO* sestava se prične postopek generiranje 3D *SolidWorks* CAD sestava. Algoritem za generiranje sestava v *SolidWorks*-u je dokaj enostaven, saj mora v bistvu narediti samo en obhod čez drevesno strukturo sestava in zgraditi analogno drevesno strukturo → sestav v *SolidWorks*-u. Najprej se celoten *GastonVO* sestav prebere v razred sestav, ki ga nato pošlje funkciji *FillComponents_rec*, ki nato rekurzivno zgradi vgradno omaro. Preden pokličemo funkcijo za generiranje moramo še ustvariti korenski *SolidWorks* sestav, ki ga bo napolnila rekurzivna funkcija. Spodnja C# koda zajema:

```
public void Generate(string gastonAssemblyPath, string outputDir)
{
    //najprej preberemo GastonVO podatke v razred Assembly
    Assembly assembly = Assembly.Open(gastonAssemblyPath);

    string rootFileName = outputDir + assembly.Name + ".SLDASM";
```

```
    //ustvarimo nov document za delo s sestavi
    ModelDoc2 rootAssemblyModelDoc =_swAddin.SwApp.NewDocument();

    //save new empty assembly
    rootAssemblyModelDoc.Extension.SaveAs(rootFileName)

    //rekurzivno napolni korenski SolidWorks sestav
    FillComponents_rec(assembly, rootAssemblyModelDoc);

}
```

Funkcija *Generate* je dejansko predpriprava za rekurzivno funkcijo *FillComponents_rec*, ki ustvarja in dodaja *SolidWorks* komponente na podlagi podatkov iz projektne datoteke *GastonVO*. Spodnja koda predvideva, da vtičnik ob zagonu prebere preslikovalno datoteko in napolni razpršilni tabeli oziroma C# slovarja *type_mapping* in *parameter_mapping*. V prvi tabeli naredimo poizvedbo o preslikavi tipa *GastonVO* sestava, ki nam pove, v katerem imeniku v *SolidWorks* knjižnici se nahaja sestav, ki ustreza trenutnemu *GastonVO* tipu. Druga tabela pa hrani informacijo o slikanju tipov in uporablja 2 ključa. Prvi je enak kot pri preslikavi tipov (tip *GastonVO* sestav), drugi pa predstavlja ime parametra znotraj sestava *GastonVO*. Na podlagi obeh podatkov dobimo ustrezen znakovni niz, ki predstavlja ime uporabniško definiranega parametra, ki ga program ustrezno nastavi. Jedro rekurzivne funkcije je prikazano na spodnjem segmentu kode. Funkcija *InsertFromLibrary* prekopira *Solidworks* sestav/sestavni del v delovni imenik, odpre sestav ali sestavni del[9] in ga doda staršu, nastavi transformacijsko matriko, glede na vrednosti iz *GastonVO* ustrezno nastavi uporabniško definirane parametri in pokliče zopet rekurzivno funkcijo *FillComponents_rec*, tokrat z novim *GastonVO* sestavom in novim *SolidWorks* sestavom. Funkcija *InsertNewAssembly* kreira prazen sestav, ga doda staršu in mu nastavi transformacijsko matriko.

```
Dictionary<string, string> type_mapping = new Dictionary<string,string>();

Dictionary<string, Dictionary<string, string>> parameter_mapping
      = new Dictionary<string,Dictionary<string,string>>();

private void FillComponents_rec(
      Assembly assembly,
      ModelDoc2 assemblyModelDoc
      )
{
    foreach (ComponentType component in assembly.Components)
    {
        if( type_mapping.ContainsKey(assembly.Type.ToString()))
            InsertFromLibrary(
                    component,
                    assemblyModelDoc,
                    type_mapping[assembly.Type.ToString()].First,
                    type_mapping[assembly.Type.ToString()].Second);
        else
```

[9] Osnovni sestav v GastonVO je v splošnem sestav v SolidWorks-u. Sestavni del je samo v primeru, ko pokliče funkcijo list v drevesni strukturi.

```
                InsertNewAssembly(component, assemblyModelDoc);
        }
}
```

Opisani algoritem je enostavna rešitev za generiranje poljubnih sestavov znotraj okolja *SolidWorks*. Predpogoj za delovanje vtičnika je pravilno sestavljena delujoča knjižnica s preslikovalno datoteko in ustrezen vhod. Skrajna preprostost programa zagotavlja stabilno delovanje, minimizira napako in nepredvidljive situacije ter omogoča enostaven prehod med različnimi verzijami *SolidWorks* okolja.

4 Sklepne ugotovitve

V prvem delu diplomske naloge sem s problematiko opisa vgradnih omar predstavil neposredno in posredno delovanje programa *GastonVO*. Ta omogoča hitro sestavo in prodajo vgradne omare v pohištvenih salonih ter posredovanje podatkov o prodani vgradni omari proizvajalcu v digitalni obliki. *GastonVO* je bil predstavljen z nivoja abstrakcije in povezovanja podatkov za opisovanje različnih elementov, ki so parametrično definirani, kar pomeni, da vgradne omare še zdaleč niso edina problemska domena, ki jo program pokriva. XML shemo lahko uporabimo kot shrambo za opis poljubnega področja, kjer je ena izmed potrebnih zahtev sestavljanje na podlagi osnovnih sestavov, ki so preddefinirani v knjižnici.

Glavna pomanjkljivost trenutnega programa *GastonVO* je, da še ne omogoča 3D prikaza sestava (vgradne omare). Stranko bi veliko bolj prepričal 3D pogled vgradne omare, vendar trenutno zasnova XML sheme še ne vsebuje dovolj podatkov za generiranje geometrije parametričnih elementov. Fiksne elemente, ki jih ni možno spreminjati, lahko brez težav preddefiniramo, medtem ko parametrični končni elementi z geometrijo prinesejo s seboj veliko dodatnih težav.

XML podatke pri parametričnih elementih bi bilo potrebno razširiti z dodatno informacijo za izgradnjo 3D geometrije(CAD operacije) in z uporabo ustrezne knjižnice, ki takšne operacije podpira zgraditi 3D geometrijo. Takšna razširitev ustrezno poveča kompleksnost podatkov in tudi samega programa, povečajo se zahteve glede strojne opreme. Generiranje 3D podatkov je izjemno zahtevna operacija in časovno veliko bolj potratna od samega prikaza podatkov. Če je parametričnih končnih elementov veliko, lahko postane proces prikaza predogleda preveč požrešen in na koncu neuporaben z vidika programa *GastonVO*, ki temelji na hitrem posredovanju podatkov.

V drugem delu naloge je predstavljena osnovna funkcionalnost delovanja *GastonVO* vtičnika za *SolidWorks*, ki na podlagi preddefinirane *SolidWorks* knjižnice in *GastonVO* projekta generira ustrezno 3D CAD predstavitev v okolju *SolidWorks*. Osnovna ideja programa *GastonVO* je bila, da se zagotovi minimalna količina informacije, ki še omogoča 3D vizualizacijo, kar pomeni, da je možno na podlagi preddefiniranih elementov izdelati 3D geometrijo sestava (vgradne omare).

Dosedanji razvoj vtičnika omogoča samo generiranje 3D geometrije, manjka pa še nastavljanje ustreznih materialov (npr. material leve, desne stranice, hrbtišča vgradne omare,…). Tako lahko dobimo tudi foto-realistični prikaz sestava z realnimi materiali.

Možna nadgradnja v *SolidWorks*-u je tudi preddefinirana tehnična dokumentacija za vsak osnovni element, ki jo pripravi tehnolog, risar v podjetju in deluje kot predloga, ki jo izpolni vtičnik z vpisom ustreznih parametrov. Tako dobi proizvajalec podatke, ki jih lahko posreduje direktno delavcu za strojem.

Za podjetje predstavlja takšna rešitev veliko podatkovno obremenitev, zato potrebuje tudi ustrezen sistem za hranjenje in dokumentiranje projektov znotraj podjetja. Takšno rešitev omogoča program *PDMWorks Enterprise* [10]. Po končanem generiranju se sestav doda v PDM sistem, katerega funkcionalnost omogoča enostaven pregled vseh projektov glede na različne kriterije, ki so definirani znotraj konkretnega PDM-sistema za konkretno podjetje.

Literatura

[1] Microsoft .NET. *Microsoft.* http://www.microsoft.com/NET/.

[2] Fast Lightweight Expression Evaluator. *CodePlex.* http://flee.codeplex.com/.

[3] DotNetZip. *CodePlex.* http://dotnetzip.codeplex.com/.

[4] Xsd2Code .net class generator from XSD schema. *CodePlex.* http://xsd2code.codeplex.com/.

[5] Crystal Reports. SAP. http://www.crystalreports.com/.

[6] Introduction to XML Schema. *W3Schools.com.* http://www.w3schools.com/schema/schema_intro.asp.

[7] Introducing XML Serialization. *MSDN Visuall Studio Developer Center.* Microsoft. http://msdn.microsoft.com/en-us/library/182eeyhh.aspx.

[8] Generalized Type Conversion. *MSDN Visual Studio Developer Center.* Microsoft. http://msdn.microsoft.com/en-us/library/ayybcxe5.aspx.

[9] **SolidWorks Corp.** *SolidWorks 2010 API Help.*

[10] SolidWorks Enterprise PDM. *SolidWorks.com.* http://www.solidworks.com/sw/products/data-management-software-pdm.htm.

[11] SolidWorks 3D CAD Design Software. *SolidWorks.* Dassault Systèmes SolidWorks Corp. http://www.solidworks.com.

[12] XML Tutorial. *W3Schools.com.* http://www.w3schools.com/xml/default.asp.