LATVIJAS UNIVERSITĀTES
DATORIKAS FAKULTĀTE

LIETOTĀJA DEFINĒTU LAUKU MEHĀNISMS TDA RĪKU BŪVES PLATFORMĀ

BAKALAURA DARBS

Autors:
Jūlija Ovčiņņikova
Stud. apl.
jo08051

Vadītājs:
Profesors Dr.dat. Kārlis Čerāns
Latvijas Universitāte

RĪGA 2012

Anotācija

Darbā ietvaros ir izstrādāts un realizēts mehānisms lietotāju definēto lauku un skatījumu pievienošanai TDA rīku būves platformā veidotiem grafiskiem rīkiem.

Mehānisms sniedz iespēju papildināt rīku notācijas ar jauniem vizuāliem elementiem un nodrošināt to semantikas definēšanu, kā arī nodrošināt lietotājiem iespēju mainīt diagrammas elementu un atribūtu izskatu.

Izstrādātais mehānisms ir pārbaudīts OWLGrEd rīkā, definējot domēn-specifiskas OWL ontoloģiju vizualizācijas notācijas.

Atslēgvārdi: OWL, OWLGrEd, TDA, Protégé, Metamodelis, Lietotāja definētu lauku mehānisms, Semantika.

Abstract

User defined field mechanism in TDA tool building platform.

During the work time was developed and implemented a mechanism, that allows add user defined fields and views into graphical editors, that were created on TDA platform.

Mechanism provides an opportunity to add new notations and define its visual appearance and semantics, as well as providing users the opportunities to change the elements and attributes look.

Developed mechanism was tested inside OWLGrEd editors, by defining domen-specific OWL ontology visual notations.

Keywords: OWL, OWLGrEd, TDA, Protégé, Meta-model, User defined field mechanism, Semantics.

Saturs

Tabulu saraksts

Attēlu saraksts

Apzīmējumu saraksts

OWL – web ontology language. Ontoloģiju aprakstīšanas valoda, kas domāta semantiskajam tīmeklim.

RDF – metadatu modeļa specifikāciju kopa datu atspoguļošanai.

OWL ontoloģija – ontoloģijas valoda semantiskajam tīmeklim ar formāli definētu nozīmi. OWL ontoloģija ietver klases, īpašības un datu vērtības, kas tiek glabātas semantiskā tīmekļa dokumentu ietvaros.

Metamodelis – modelis, kas apraksta citu modeļu struktūru un darbības principus.

UML – valoda grafiskai objektorientētai modelēšanas aprakstīšanai.

TDA – Metamodeļiem un modeļu transformācijām balstīta modelēšanas rīku arhitektūra. Metamodeļi un modeļu transformācijas veido arhitektūras kodolu, un tiem tiek nodrošināta iespēja pieslēgt dažādus lietotāja saskarnes dziņus (piemēram, grafveida diagrammu dzini, lietotāja dialogu (formu) dzini un citus).

OWLGrEd redaktors – UML tipa grafiskais redaktors priekš OWL 2, kas atspoguļo ontoloģijas, izmantojot paplašinātu UML klašu diagrammas notāciju un piedāvā ontoloģiju rediģēšanas iespējas.

Ievads

OWL ontoloģiju[11,12] un semantisko tehnoloģiju pielietojumi kļūst aizvien populārāki līdz ar to nepieciešamas intuitīvas ontoloģiju attēlošanas un rediģēšanas iespējas. Parādās aizvien vairāk rīku, kas paredzēti darbam ar OWL/RDF ontoloģijām, piemēram Protégé [18]. Tajā skaitā ir daudzi rīki, kas ļauj attēlot ontoloģijas grafiski, pārsvarā visi no tiem izmanto UML klašu diagrammu notāciju (TopBraid Composer [21], Protégé [18]).

Daudzas pamata OWL konstrukcijas var tikt ērti attēlotas ar UML klašu notācijām. Bet OWL ontoloģijas sāk izmantot semantiskajās datu bāzēs un informāciju sistēmās, un rodas nepieciešamība pēc plašākas OWL notāciju definēšanas.

Līdz ar to OWL ontoloģiju plašu izmantošanu, tai skaita semantisko datu bažu specifikācijā paradās nepieciešamība pielāgot ontoloģiju redaktoru vizuālo izskatu un iespējas konkrētiem lietojumu apgabaliem. Tipiski lietotāju definētas notācijas var tikt balstītas uz OWL anotāciju konstrukcijas.

Latvijas Universitātes Matemātikas un Informātikas Institūts izveidoja TDA rīku būves platformu [4], kas ir balstīta uz metamodeļiem un modeļu transformācijām un ļauj salīdzinoši īsā laikā izveidot jaunus grafiskus modeļu izveides rīkus. OWLGrEd grafiskais redaktors[6,7] ir viens no šādiem rīkiem. OWLGrEd piedāvā paplašinātu UML notāciju OWL ontoloģiju attēlošanai un rediģēšanai, kas ļauj attēlot gandrīz visas OWL 2 konstrukcijas.

Līdz ar vienotu grafisko ontoloģiju attēlošanas notāciju, kas tiek piedāvāta OWLGrEd rīkā, atsevišķās situācijās ontoloģijas uztveramību var atvieglot to papildinoša specifiska, tieši konkrētai ontoloģijai vai kādam ontoloģiju kopumam piemērota notācija.

Viena no situācijām, kurā papildus notācija var būt noderīga, ir OWLGrEd izmantošana semantisko datubāzu shēmas uzdošanā.

Šī bakalaura darba ietvaros tiek piedāvāts lietotāja definētu lauku mehānisms, kas piedāvā iespēju kā ērti un vienkārši papildināt uz TDA platformas veidoto rīku, tajā skaitā arī OWLGrEd redaktora, notāciju klāstu un vizualizēt domēnu specifiskus anotāciju apgalvojumus.

Vairumu no lietotāja definētu lauku mehānisma iespējām varēja realizēt ar konfiguratora palīdzību, kas ir implementēts OWLGrEd redaktorā, bet ar jauniem OWL ontoloģiju lietojumiem ar konfiguratora iespējām sāka nepietikt.

Bakalaura darba mērķis bija izveidot mehānismu, kas ļautu ērti:

- Papildināt OWLGrEd redaktora notāciju ar jauniem laukiem un tiem atbilstošiem vizuālajiem efektiem;

- Nodrošināt jauno lauku semantikas atspoguļošanu importējot/eksportējot ontoloģijas uz jauno redaktoru;
- Pielietot lietotāju definētus stila uzstādījumus, lai specifiskā veidā attēlotu noteiktus laukus un elementus.

Lietotāja definētu lauku mehānisms ir realizēts kā spraudnis un var tikt instalēts OWLGrEd redaktorā.

Lietotājiem tiek piedāvāts ērts veids kā papildināt OWLGrEd redaktoru ar vajadzīgām lietām, neaiztiekot redaktora pamata struktūru, un noņemt pielikto funkcionalitāti, kad tā vairs nav vajadzīga, neradot nekādus zaudējumus.

Tiek atbalstīta arī jauno lauku pārnesamība no viena projekta citā, saglabājot jauno lauku informāciju teksta failā un ielādējot to jaunā projektā.

Ar lietotāja definēto lauku mehānisma ietvaros realizētu integrēto skatījumu mehānismu ir iespējams uzstādīt diagrammu izskatu, lai izceltu svarīgus elementus, paslēptu nevajadzīgās elementu daļas un padarītu diagrammas vieglāk caurskatāmas.

Iespējamie lietojumi lietotāja definētu lauku mehānismam ir:

- Anotāciju semantikas definēšana papildus UML konstrukcijām, tādām kā kompozīcija (composition) vai atvasināto apvienojumu īpašība (derived union);
- Datu bāzes savienojamības anotāciju definēšana klasēm, objektiem un datu īpašībām;
- Integritātes ierobežojumu notāciju definēšana.

Darbs sastāv no sešām nodaļām.

Pirmajā nodaļā tiek aprakstīta TDA rīku būves platforma ar tās svarīgākajiem metamodeļiem un dziņiem, kā arī tiek aprakstīts konfigurators, kas ir domāts jauno rīku definēšanai.

Otrajā nodaļā tiek aprakstīts OWLGrEd redaktors, kura ietvaros tika veidots lietotāju definēto lauku mehānisms un kas ir paredzēts OWL ontoloģiju grafiskajai vizualizācijai.

Trešajā nodaļā tiek aprakstīts Protégé ontoloģiju redaktors.

Ceturtajā nodaļā, kas ir darba pamata nodaļa, tiek aprakstīta lietotāju definēto lauku mehānisma realizācija un iespējamās lietotāju darbības.

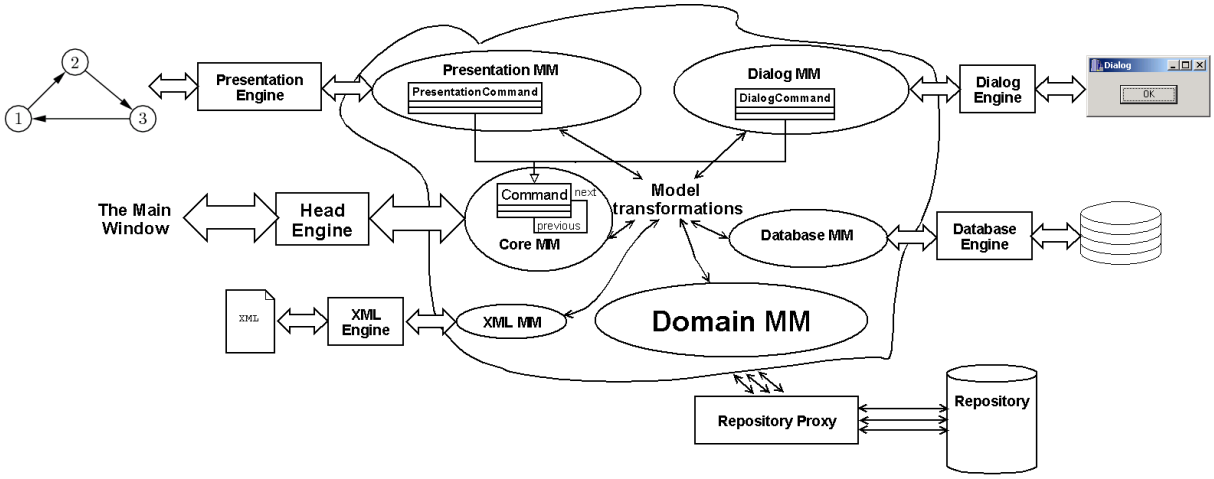
Piektajā nodaļā tiek aprakstīta lietotāja definētu lauku mehānisma izpildlaika daļa.

Sestajā nodaļā tiek aprakstīti mehānisma iespējamie lietojumi un piemēri.

1 TDA rīku būves platforma

TDA (*Transformation-Driven Architecture*) [4] ir uz metamodeļiem un modeļu transformācijām balstīta modelēšanas rīku arhitektūra, kas atbalsta vairākus prezentācijas metamodeļus. Transformācijas tiek izmantotas domēnu un prezentāciju sasaistē. Katrs prezentācijas metamodelis var tiks apstrādāts neatkarīgi ar tam atbilstošu dzini.

Transformācijas var komunicēt ar dziņiem, tas notiek ar notikumu (*Event*) un komandu (*Command*) palīdzību. Notikums ir transformācijas izsaukums no dziņa, un komanda ir dziņa izsaukšana no transformācijas. Katrs notikums atbilst kādai lietotāja darbībai.



Attēls 1.1 **TDA rīku būves platformas arhitektūra [4]**

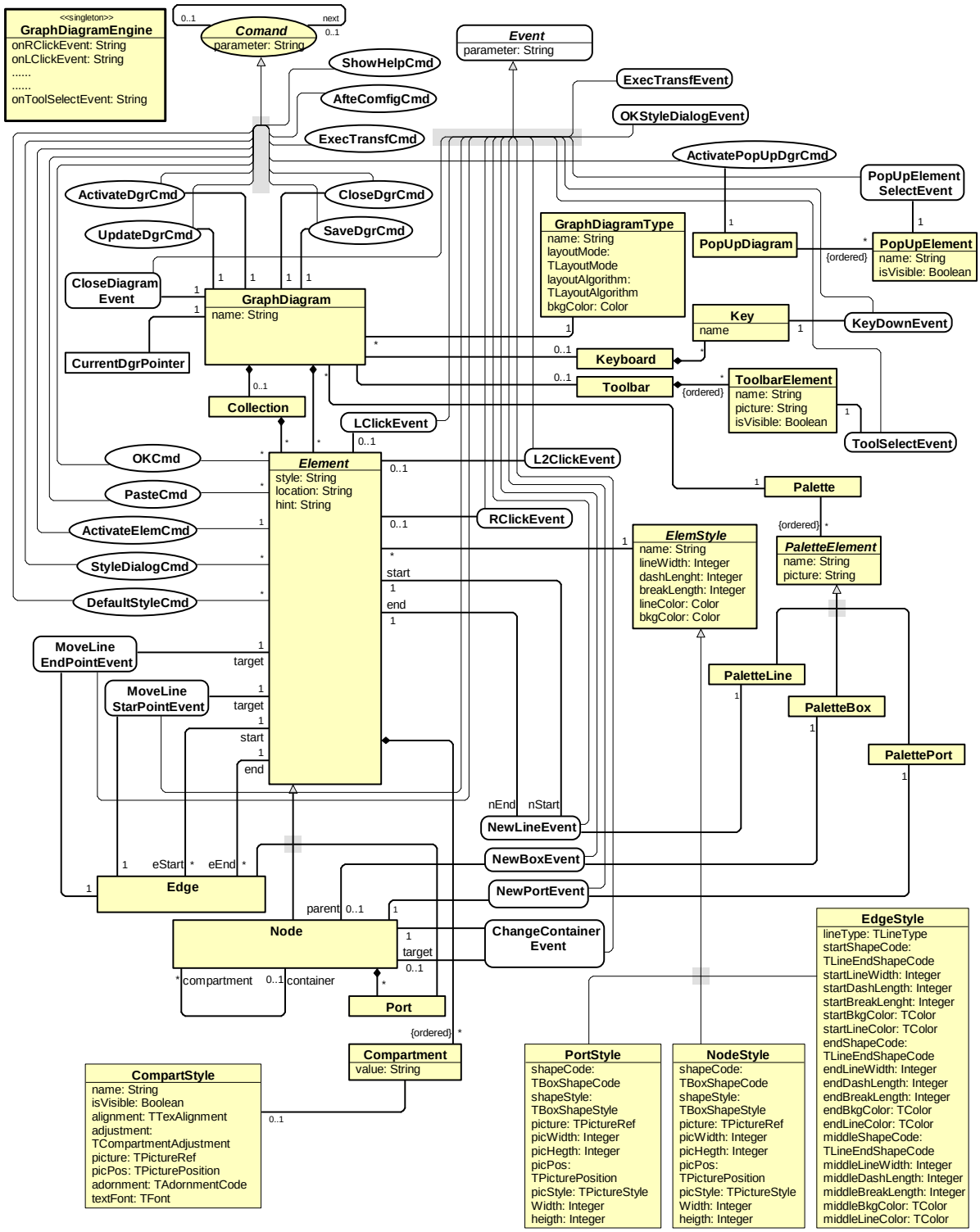
Neatņemama daļa no TDA rīku būves platformas ir tā saucamais kodola (*Core*) metamodelis ar tam atbilstošu vadības (*Head*) dzini. Visi prezentācijas metamodeļi tiek saistīti ar kodola metamodeli. Vadības dzinis nodrošina transformācijas un dziņus ar pakalpojumiem. Kodola metamodelis satur notikumu (*Event*) un komandu (*Command*) klases, vadības dzinis tiek izmantots kā notikumu un komandu menedžeris.

TDA rīku būves platformai ir pievienoti tādi prezentācijas metamodeļi kā grafveida diagrammu metamodelis, lietotāja dialogu (formu) metamodelis, kā arī var tikt pievienoti jauni prezentācijas metamodeļi.

1.1 Grafveida diagrammu metamodelis

Grafveida diagrammu metamodelis [5] ļauj atspoguļot domēna metamodeļa datus grafiskā veidā.

Grafveida diagrammu metamodelim atbilst Grafveida diagrammu dzinis, kura uzdevums ir vizualizēt Grafveida diagrammu metamodeļa instances, kā arī nodrošināt to diagrammu apstrādi redaktorā.



Attēls 1.2 **Grafveida diagrammu metamodelis [5]**

Elementi, kas var tikt atspoguļoti grafiski, pieder šādām klasēm: **GraphDiagram, Element, Compartment**. GraphDiagram klase atbilst diagrammām. Elements var būt mezgls

(*Node*), šķautne (*Edge*), ports (*Port*), brīva kaste (*FreeBox*), brīva līnija (*FreeLine*). Kompartments atbilst teksta laukam, kas var tikt pievienots mezglam, šķautnei vai līnijai.

Katram elementam un kompartmentam ir tieši viens stils, kas nosaka elementu atspoguļošanas veidu. Noklusētie elementu un kompartmentu stili tiek glabāti **ElemStyle** un **CompartStyle** klašu instancēs. Reālie elementu un kompartmentu stili tiek iekodēti Element un Compartment klašu instanču *style* atribūtos kā simbolu virknes, kas ļauj mainīt konkrētu elementu stilus izpildes laikā, nemainot noklusētos stilus. Stils sastāv no vairākiem stila atribūtiem, kas var atšķirties katram elementa tipam.

Katrai diagrammai ir noteikts tips, kas tiek glabāts **GraphDiagramType** klases instancē. Tajā pat instancē tiek glabāta arī diagrammu stila informācija.

Grafveida diagrammu metamodelim ir arī savi specifiski notikumi uz komandas, kas nodrošina darbību.

Grafveida diagrammai ir palete ar paletes elementiem, kas ir aizmetņi līnijām, kastām vai portiem.

Konkrēti šim metamodelim atbilstošu grafveida diagrammu piemēri būs redzami tālāk šajā darbā, piemēram, sadaļā par OWLGrEd redaktoru.

1.2 Dialogu loga metamodelis

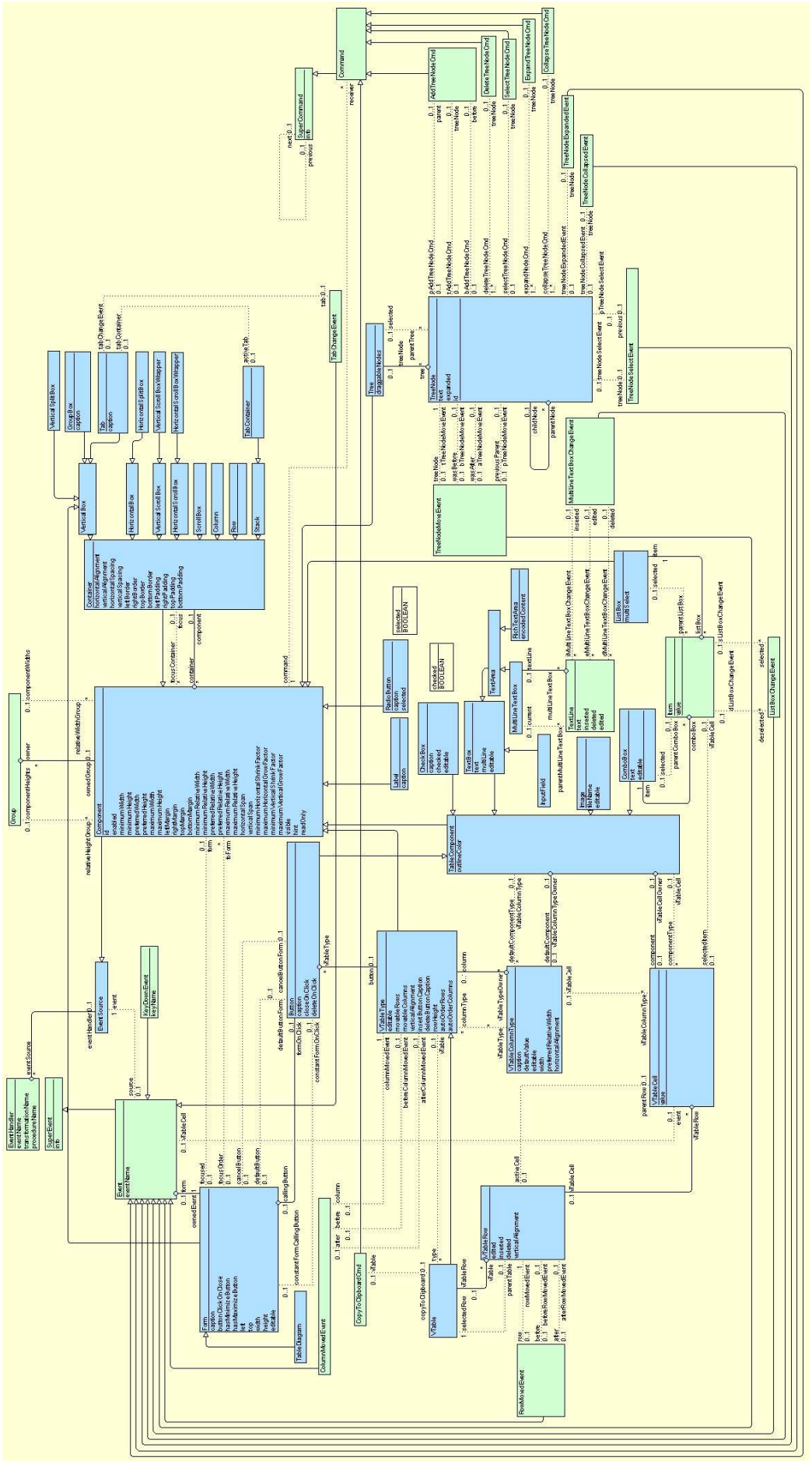
Dialogu loga metamodelis [9] un dzinis paredzēti, lai atspoguļotu lietotājiem dialoga logus. Metamodelis ir realizēts tādā veidā, ka, ja dzinim tiek padota kāda dialoga loga metamodeļa instance, tas spēj automātiski uzbūvēt dialoga logu. Tiek atbalstīta arī dialoga loga elementu izkārtojuma iespējas.

Svarīgi jēdzieni šajā metamodelī ir **komponente** (*compoment*) – grafisks elements, piemēram, virsraksts (*Label*), izvēles rūtiņa (*CheckBox*), ievadlauks (*InputField*). Komponentēm ir divi specifiski notikumi FocusGainedEvent un FocusLostEvent, kas iestājas, kad komponente iegūst vai zaudē fokusu. **Konteiners** (*container*) – specifiska komponente, kas var saturēt citas komponentes. Tipiskie konteinera piemēri ir forma (*Form*), grupu kaste (*GroupBox*), cilne (*Tab*).

Lai atspoguļotu dialoga logu, sākumā jāizveido forma, kas būs augstāka līmeņa konteiners. Konteineri var sadalīt vairākās daļās. Katra daļa var arī tikt rekursīvi sadalīta. Konteineri var saturēt gan redzamas komponentes, gan neredzamas, tādas kā rāmji (*frames*) un robežas (*borders*). Lai veidotu režģim līdzīgu izkārtojumu, bieži tiek izmantoti tādi elementi kā rindas (*Row*) un kolonnas (*Column*). Plaši tik lietoti arī tādi elementi kā tabulas (*Table*) un koki (*Tree*).

Svarīgākās dialogu loga dzinēja komandas ir **Show** – šī komanda lūdz dialogu dzini parādīt logu, kurš ir aprakstīts ar komponenšu koku. **Refresh** – izdzēš no formas visus dotās komponentes bērnus (iekļautās komponentes) un pielasa jaunus bērnus no repozitorija. **Close** – šī komanda lūdz dialogu dzini aizvērt doto formu. **Delete** – šī komanda lūdz dialogu dzini izdzēst doto formu (ar visām komponentēm) no repozitorija.

Svarīgākie dialogu loga dzinēja notikumi ir **Click** – iestājas, kad lietotājs uzklikšķina uz pogas. **Close** – iestājas, kad lietotājs uzklikšķina uz krustiņa loga augšā. **Change** – iestājas, kad lietotājs maina vērtību. Notikumam piekārtotā transformācija tiek izsaukta pēc tam, kad tikušas uzstādītas atbilstošo atribūtu (selected, checked, text, fileName) jaunās vērtības.



Attēls 1.3 **Dialogu loga metamodelis [9]**

1.3 Tipu daļas metamodelis

Tipu daļas metamodelis [10] paplašina grafveida diagrammu metamodeli un domāts, lai aprakstītu grafiskus elementus, to uzvedību un ierobežojumus.

Galvenās tipu daļas metamodeļa klases ir **GraphDiagramType, ElemType, NodeType, PortType, CompartType**. Šīs klases glabā meta informāciju par katru elementa tipu. Katram elementam ir saites uz tam atbilstošu tipu klasi, kas ļauj elementu darbības apskatīt instanču līmenī.

Katrai diagrammai ir noteikts tips, kas atrodams **GraphDiagramType** klases instancē. Diagrammas tipu savieno ar **Palette** klases instanci, lai norādītu, kādi paletes elementi būs šī tipa diagrammās. Katra elementa tips atrodams **ElemType** klases instancē. Elementa tipu savieno ar **PaletteElement** klases instanci. Katra kompartmenta tips atrodams **CompartType** klases instancē, kas tiek savienota ar noteiktu **ElemType** klases instanci.

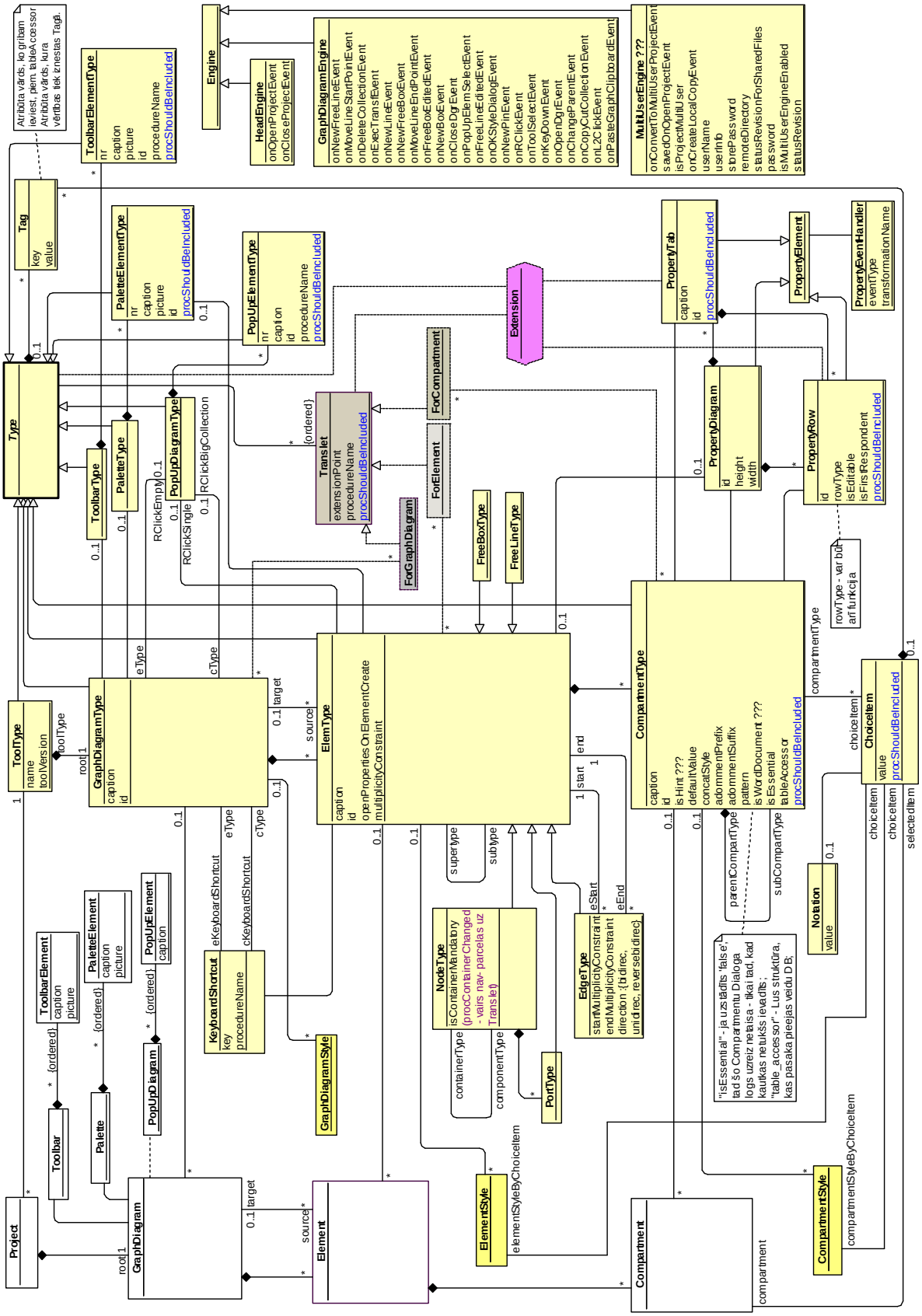
Lai nodefinētu konkrētu elementu uzvedību dažādās situācijas, tiek lietoti paplašinājuma punkti (*ExtensionPoint*).

Tiek atbalstīta tāda iespēja kā kompartmenta vērtības izveidošana no zemāka līmeņa kompartmentiem, bet lietotājiem tiek radītas tikai augšējā līmeņa kompartmentu vērtības.

Lai varētu automātiski ģenerēt dialoga logus un to struktūru, tiek ieviestas tādas klases kā **PropertyDiagram** – atbilst dialoga logam, **PropertyTab** – atbilst dialoga loga cilnei, **PropertyRow** – atbilst dialoga loga komponentei.

Gadījumā, ja ir jāatver vairāki dialoga logi dažādos līmeņos, ir jānodrošina loga atvēršana no cita loga. Tas tiek paveikts ar *calledPropertyRow-calledPropertyDiagram* asociācijām.

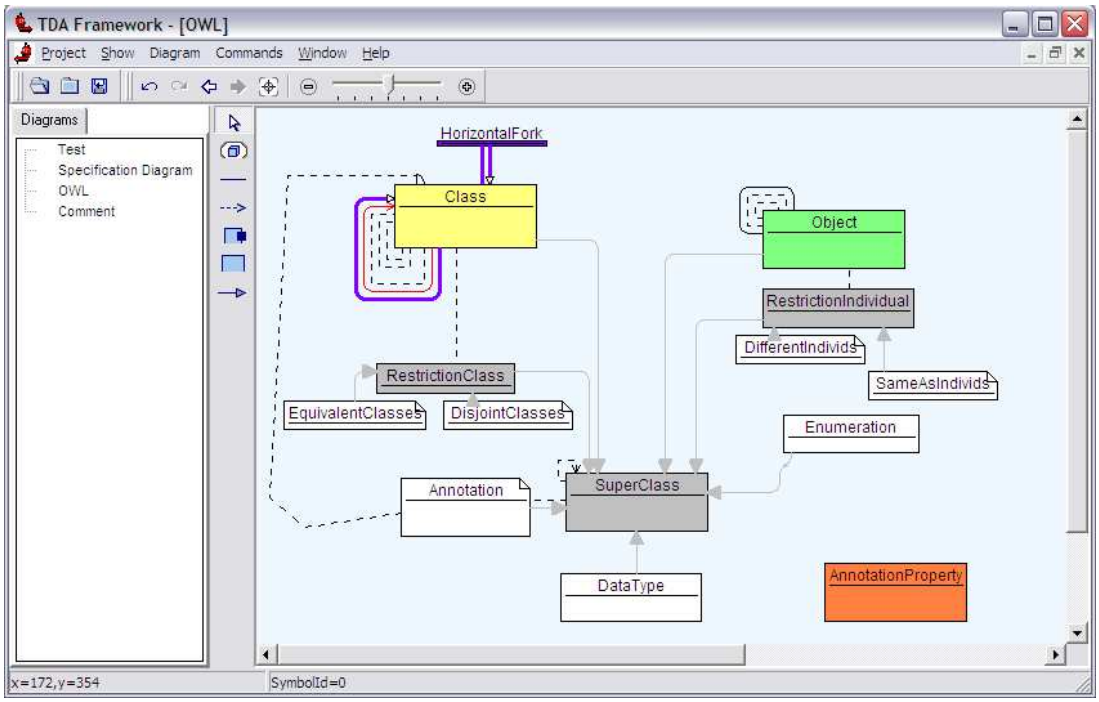
Pastāv iespēja dinamiski mainīt elementu un kompartmentu stilus. Visi iespējamie noteikta tipa stili glabājas vai nu elemStyle vai compartStyle klašu instancēs. Var uzstādīt nosacījumus, pie kādiem tipiem tiek uzstādīts noteikts stils. Tas ir paveicams ar elemStyleByChoiceItem un compartStyleByChoiceItem saitēm, kas sasaista stila instanci ar izvēles vienumu, kas ir pakārtots kādam laukam. Brīdī, kad šī lauka vērtība tiek uzstādīta par doto izvēles vienumu, tiek pārrēķināts elementa vai kompartmenta stils.



Attēls 1.4 **Tipu daļas metamodelis [10]**

1.4 Konfigurators

Konfigurators [8] ir paredzēts, lai ātri un vienkārši veidotu grafiskus tipus domēnu-specifiskām valodām. Lai to paveiktu, tiek izmantots tipu daļas metamodelis. Specifiski tipi tiek pielikti tipu daļas metamodelim kā instances. Ar konfiguratora palīdzību jaunie tipi tiek veidoti grafiski un to atribūti tiek pielikti ar dialogu logu palīdzību.



Attēls 1.5 **Jaunie tipi definēti ar konfiguratora palīdzību**

Ar konfiguratoru pieejami trīs elementu tipi: kastes, līnijas un īpašības. Kastes reprezentē mezglus (*Node*), līnijās – šķautnes (*Edge*), īpašības – kompartmentus.

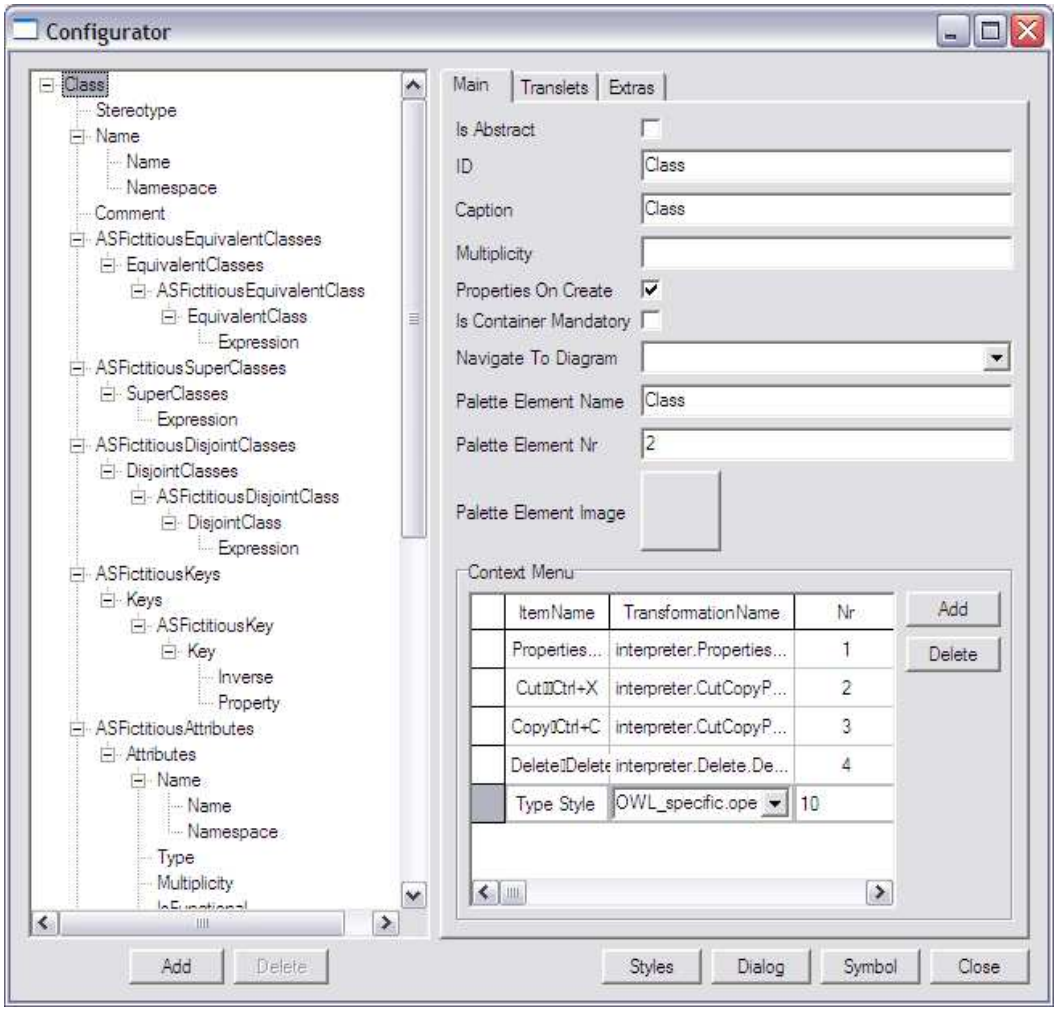
Konfigurators izmanto universālu interpretatoru, lai sadarbotos ar grafveida diagrammas dzini un dialogu loga dzini. Universāls interpretators ir universāla transformācija, kas interpretē tipu daļas metamodeli, lai nodrošinātu jaunu tipu darbību. Būtībā konfigurators ir tips, kas definē citus tipus.

Konfiguratora veidotas tipu daļas metamodeļa instances tiek interpretētas ar universālo interpretatoru. Konfiguratorā tiek izmantoti paplašinājuma punkti, kas ļauj lietotājam veidot savas transformācijas. Universālais interpretators noteiktos brīžos izsauc šīs transformācijas.

Tipiem tiek glabātas divu veidu instances. Pirmā veida instances definē tipus un ir statiskas. Otra veida instances, atbilst grafu diagrammas metamodelīm, tiek veidotas dinamiski un atbilst elementiem, ar kuriem darbojas gala lietotājs. Ideja, kas ir realizēta konfiguratorā, ir saistīt statiskās instances ar dinamiskām saišu palīdzību. Lai to varētu izdarīt, novilktas trīs *presentation-target_type* saites starp GrapgDiagram un GraphDiagramType, Element un ElemType, Compartment un CompartType klasēm.

Konfiguratora definētas valodas sastāv no diviem diagrammas tipiem. Viens diagrammas tips definē prototipus diagrammu aizmetņa (*seed*) elementiem un ilustrē atkarības starp tiem. Šīs diagrammas tips ir specifikācijas diagrammas. Specifikācijas diagrammās var tikt atrasti trīs elementu tipi – aizmetnis (*seeds*), līnijs (*lines*), specializācijas (*specializations*). Aizmetņi ir elementi, kas definē aizmetņa prototipus, līnijas – elementi, kas definē atkarības prototipus, specializācijas – līnijas, kas tiek izmantotas, lai nodefinētu elementu mantojamību no vecāka elementa. Otrs diagrammas tips definē elementu prototipus. Šajā diagrammā var tikt atrasti seši elementu tipi – kaste (*Box*), līnija (*Line*), brīva kaste (*FreeBox*), brīva līnija (*FreeLine*), ports (*Port*) un specializācija (*Specializations*). Līnijas un specializācijas var tikt saistītas ar pārējiem elementiem, atskaitot specializācijas. Visi iespējamie savienoti pāri ir jāatspoguļo metamodelī, lai noteiktu, kādi elementi var tikt savienoti savā starpā.

Ar konfiguratora palīdzību var veidot ne tikai jaunus tipus un to atribūtus, bet arī noteikt tipa uzvedību, dialoga loga parametrus, pielikt tipam un atribūtiem specifiskus paplašinājuma punktus, kā arī noteikt dotā tipa elementu un kompartmentu attēlošanas stilus.



Attēls 1.6 **Atribūti definēti ar konfiguratora palīdzību**

2 OWLGrEd redaktors

OWLGrEd [6,7] ir UML tipa grafiskais redaktors priekš OWL 2 (Web ontology language) [16], kas atspoguļo ontoloģijas, izmantojot paplašinātu UML klašu diagrammas [15] notāciju un piedāvā ontoloģiju rediģēšanas iespējas.

OWLGrEd redaktors ir uzbūvēts uz TDA platformas bāzes.

OWLGrEd redaktorā UML klašu diagrammas notācija tiek paplašināta ar OWL Mančesteras sintaksi [17], kas padara notāciju kompaktāku un saprotamāku. OWLGrEdā notācija pārsvarā tika pārņemta no UML klašu diagrammas notācijas. Tika ieviestas šādas jaunās notācijas [7]:

- Jaunie lauki klasei – ekvivalenta klase (*equivalent class*), virs klase (*super class*), nepārklājoša klase (*disjoint class*);
- Asociācijām un atribūtiem lauki ekvivalentam, virs un nepārklājošam īpašībām, kā arī lauki īpašību aprakstīšanai;
- Iespēja veidot anonīmas klases, kas nesatur klases nosaukumu, bet satur ekvivalento klašu izteiksmes;
- Savienotāji (connectors) aksiomu vizualizācijai, tādi kā *equivelent*, *disjoint*;
- Savienotāji, kas atspoguļo objektu īpašību ierobežojumus, tādus kā *some*, *only*, *exactly*.

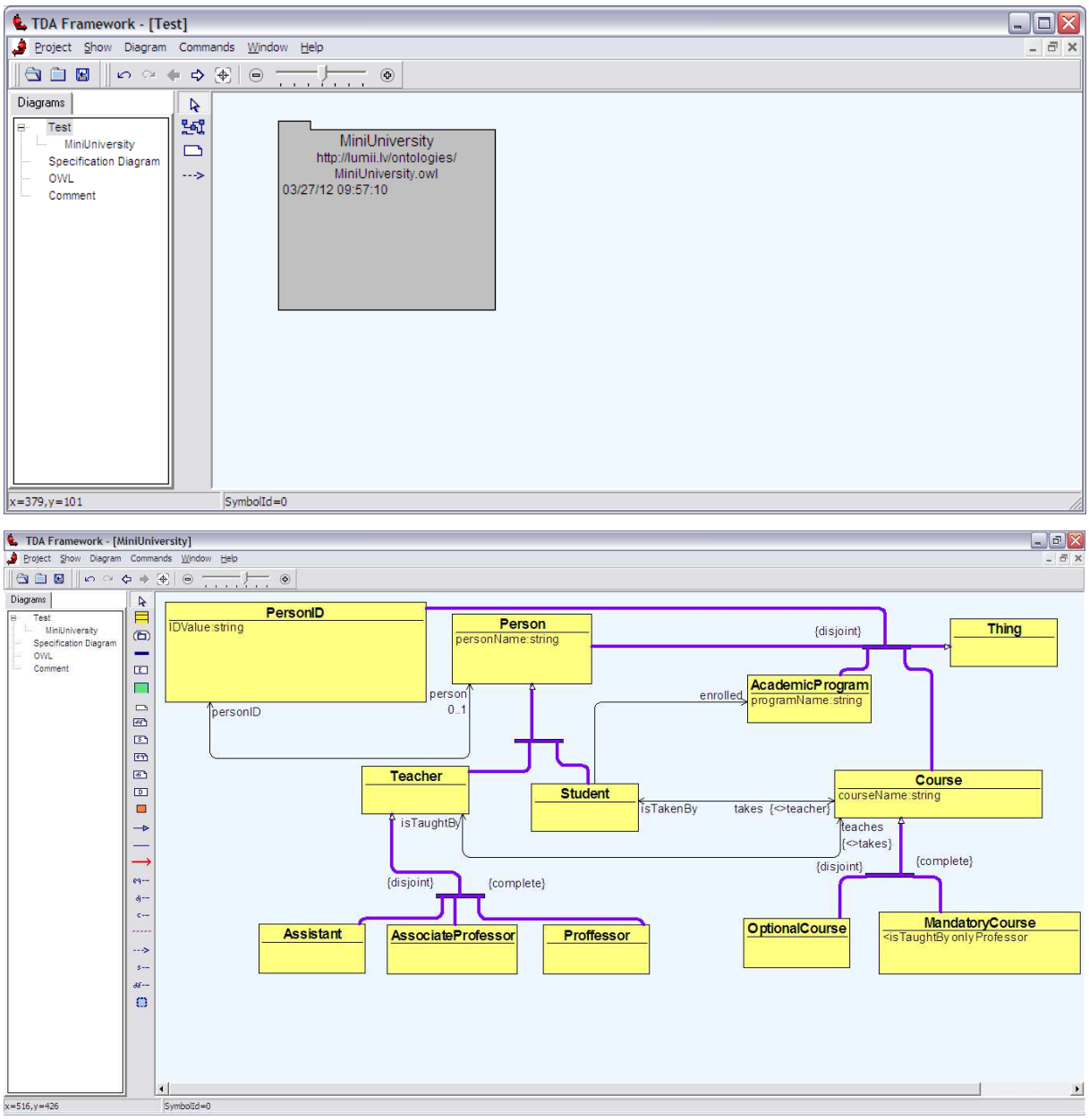
Šīs jaunās notācijas palīdz definēt klašu izteiksmes kompaktā tekstuālā veidā.

OWLGrEd redaktors piedāvā grafisko refaktoringu, kas ļauj rediģēt grafiskas notācijas, nemainot semantiku. Tipiski piemēri ir vispārinājums (*generalization*) un dakša (*fork*), kas var tikt izmantoti, lai apvienotu vairākas līnijas vienā. Ontoloģija var tikt sadalīta vairākās diagrammās, kur katra diagramma attēlo ontoloģiju no cita skatījuma vai attēlo ontoloģijas apakškopu.

Ir atbalstītas arī tādas funkcionalitātes kā vairāki diagrammas izkārtojuma veidi un meklēšanas mehānisms.

OWLGrEd redaktorā ir realizēts atbalsts ontoloģiju importam no Protégé, kā arī eksportam uz Protégé. Tiek izmantoti divi principi:

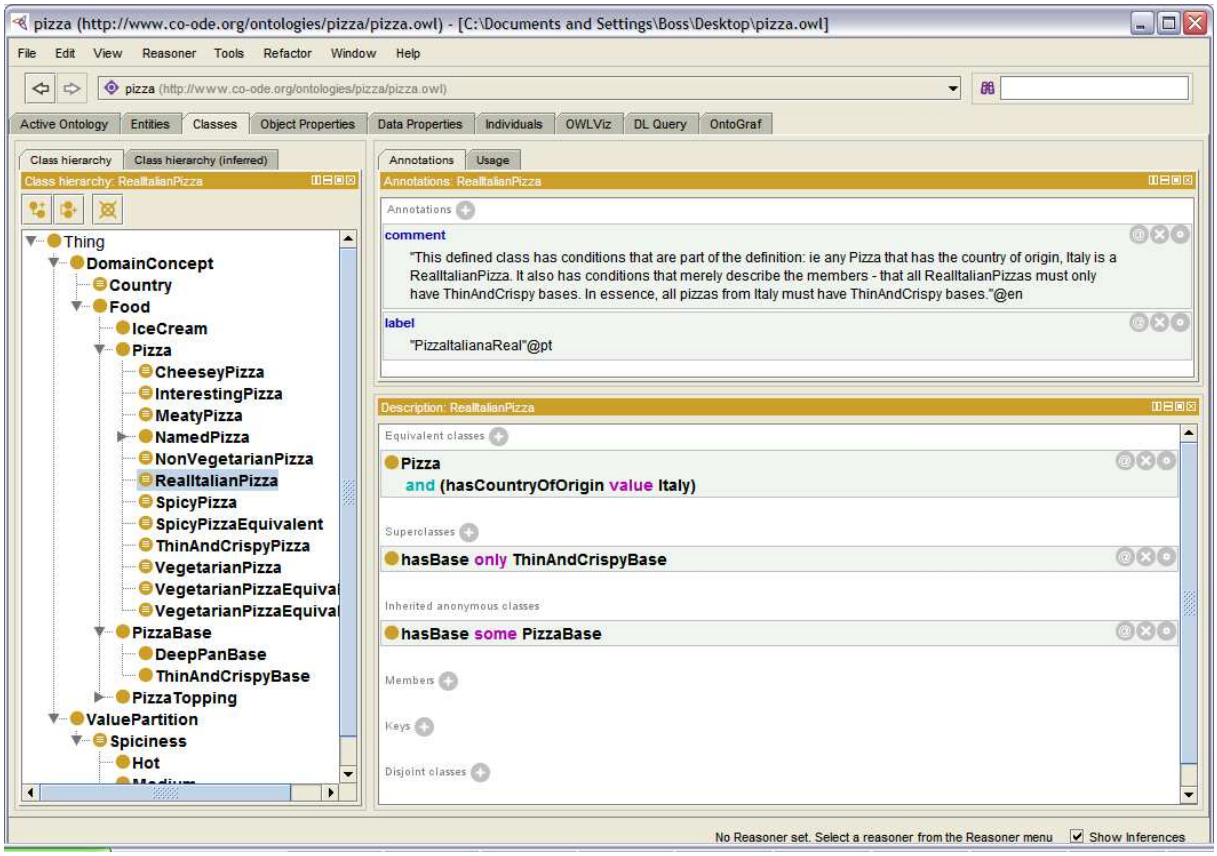
- Ontoloģijas autorēšana (ontoloģija tiek veidota un rediģēta OWLGrEd redaktorā, tad tā tiek eksportēta uz Protégé vai nu analīzei vai lai nodotu to citam ontoloģijas apstrādes rīkam);
- Ontoloģijas vizualizācija (ontoloģija, kas tikusi ielādēta no Protégé, tiek attēlota grafiski, lai iegūtu saprotamu vizuālo izskatu).



Attēls 2.1 **OWLGrEd redaktorā veidotas diagrammas piemērs**

3 Protégé

Protégé [18] ir atvērta pirmkoda platforma, kas piedāvā komplektu ar rīkiem, kas paredzēti domēnu modeļu un uz zināšanām balstītu lietojumu konstruēšanu ar ontoloģiju palīdzību. Savā kodolā Protégé īsteno lielu klāstu ar zināšanām modelētām struktūrām un rīcībām, kas atbalsta ontoloģiju veidošanu, vizualizāciju un navigāciju vairākos attēlošanas formātos. Protégé var tikt paplašināts ar iespējām piedāvāt domēna draudzīgo atbalstu zināšanu modeļu izveidei. Protégé paplašinājums tiek realizēts ar spraudņa veida arhitektūru un uz Javas balstītu API (Application Programming Interface), kas ir paredzēts uz zināšanām balstītu rīku un aplikāciju veidošanai.



Attēls 3.1 **Protégé redaktora klases cilne**

Protégé atbalsta divus veidus kā veidot ontoloģijas:

- Protégé-Frames [19] redaktors ļauj lietotājiem veidot ietvaru balstītas ontoloģijas. Šajā modelī ontoloģija sastāv no klašu kopuma, kas veidots kategorizācijas struktūrā, lai atspoguļotu domēnu nozīmīgus konceptus, slotu kopuma, piesaistītu pie klasēm, lai aprakstītu tās īpašības un attiecības, un klašu instanču kopuma.

- Protégé-OWL [20] redaktors ļauj lietotājiem veidot ontoloģijas Semantiskajam tīmeklim. OWL ontoloģija var saturēt klašu, īpašību un instanču aprakstus. OWL formāta semantika specificē, kā iegūt to loģiskās sekas, piemēram, faktus, kas būtiski netiek norādīti ontoloģijā, bet iegūti no semantikas. Protégé-OWL redaktors atbalsta šādas iespējas:
    - Ielādēt un saglabāt OWL un RDF ontoloģijas;
    - Rediģēt un vizualizēt klases, īpašības un likumus;
    - Definēt klases aprakstus kā OWL izteiksmes;
    - Rediģēt OWL indivīdus.

Protégé var tikt definēti klases, īpašības, indivīdi, datu tipi.

Protégé visām izteiksmēm izmanto „Mančesteras OWL sintaksi”.

Darbā izmantots Protégé 4.1, kas atbalsta pilnu OWL 2.

4 Lietotāja definētu lauku mehānisma definīcija

Lietotāja definētu lauku mehānisms ir spraudnis, kas var tikt pievienots uz TDA platformas veidotiem rīkiem. Mehānismu var uzstādīt gan atsevišķa projekta līmenī, gan visa rīka līmenī.

Ar lietotāja definētu lauku mehānisma palīdzību var izveidot tā saucamos profilus, kas satur lietotāja definētu informāciju par rīkam/projektam papildus pieliktajām īpašībām.

Profila ietvaros var tikt izveidoti jauni lauki, kuros tiks ieliktas papildus notācijas. Lauks ir grafisks vienums, kas ietver:

- Lauka tipu (piemēram, ievada lauks, izvēles rūtiņa);
- Lauka izskatu (piemēram, redzamību un burtu izmēru);
- Diagrammās simbolu un citu lauku vizuālus efektus (tādus kā simbola krāsa un forma);
- Lauka semantiku (kādām OWL aksiomām vai anotāciju aksiomām atbilst lauka vērtība).

Katrs lauks tiek saistīts ar konteksta tipu, kas apzīmē, kādā vietā redaktorā veidotajos simbolos jaunais lauks tiks izveidots. Pēc noklusējuma OWLGrEd redaktorā tiek piedāvāti šādi konteksta tipi:

- „Class” – lauks tiek veidots zem klases;
- „Attribute” – lauks tiek veidots zem klases atribūta;
- „Object” – lauks tiek veidots zem objekta jeb indivīda;
- „Association” – lauki tiek veidoti asociācijas abos galos.

Pastāv iespēja pielikt papildus konteksta tipus.

Katram redaktoram rīka definētais var izveidot savus noklusētos konteksta tipus, kurus atļaut lietotāja veidotus paplašinājumus.

Katram laukam var tikt izvēlēts veids, kā tās attēlosies īpašību dialogā. Lietotājam tiek piedāvāta iespēja izvēlēties no:

- InputField – vienkāršs ievadlauks;
- InputField+Button – detalizējams lauks;
- CheckBox – izvēles rūtiņa [14] ar „patiess” un „aplams” vērtībām;
- ComboBox – kombinētais lodziņš (apvieno ievadlauka un sarakstlodziņa iespējas);
- ListBox – sarakstlodziņš;
- TextArea – vairākrindu ievadlauks;

- TextArea+Button – detalizējams vairākrindu ievadlauks;
- Tukšais lauks.

Ja lauka veids ir *InputField+Button* vai *TextArea+Button*, tiek piedāvāta iespēja veidot apakšlaukus. Ja lauka veids ir *ChechBox, ComboBox* vai *ListBox*, tiek piedāvāta iespēja veidot izvēles vienumus, kurus varēs izvēlēties kā lauka vērtības no īpašību loga.

Laukiem var tikt uzstādīti lietotāja definēti stili – veids, kā lauka informācija tiek attēlota diagrammā. Stilus var uzstādīt gan lietotāja definētiem laukiem, gan jau eksistējošiem (stilu uzlikšana būs atkarīga no lietotāja definēto lauku izvēles vienumiem). Uzstādāmus stila elementus nosaka grafveida diagrammu metamodelis [5,13]. Konkrētie stila elementi atrodami ***Attēls 4.2*** Elementu un lauku stila vienumi [13] nodaļā 4.1.13.

Stili var tikt uzstādīti arī no skatījumiem, kas tiek pievienoti konkrētam profilam un var tikt pielietoti katrai diagrammai atsevišķi. Pastāv iespēja uzstādīt stilus gan elementiem, gan laukiem. Skatījumi var tikt veidoti kā noklusētie, šajā gadījumā skatījumu diagrammām nav jāpielieto, stili uzstādās automātiski.

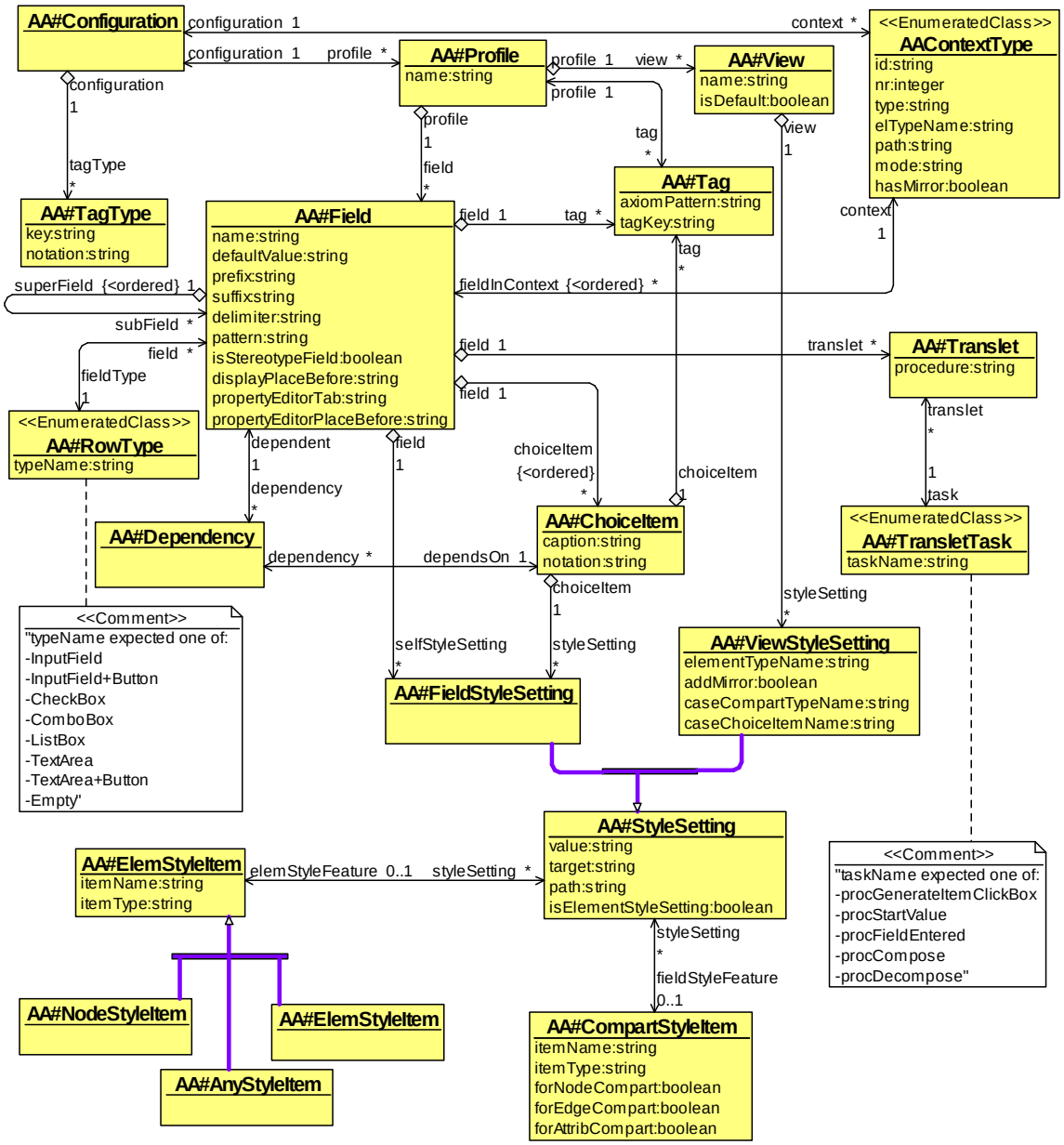
Jauno lauku semantika tiek glabāta tagos. Semantiku var nodefinēt profila, lauka vai izvēles vienumu līmenī.

Var nodefinēt sarakstu ar īpašībām, kas tiks pievienoti rīkam uzstādīšanas laikā, kā arī jebkurā laikā var pievienot jaunas īpašības vai dzēst nevajadzīgās. Līdzīgi var nodefinēt konfigurāciju, ar kuru strādās lietotāja definētu lauku mehānisms.

4.1 Metamodelis

Lai realizētu lietotāja definētu lauku mehānismu, bija jāievieš metamodeļa papildinājums ar jaunām klasēm. Jaunieviestās klases tiek apzīmētas ar „AA” prefiksu.

Galvenās klases ir **AA#Profile**, kas satur visus lietotāja definētos profilus; **AA#Field**, kas satur lietotāja definētus laukus; **AA#ChoiceItem**, kas satur lietotāja definētus lauku izvēles vienumus un **AA#StyleSetting**, kas satur dažāda veida lietotāju definētus stilus.



Attēls 4.1 **Lietotāja definētu lauku mehānisma metamodelis**

4.1.1 AA#Profile

AA#Profile klase satur visus lietotāja definētus profilus no konkrēta projekta.

Profils ir papildus parametru kopums, kas nosaka jauno notāciju, semantiku un stilu uzstādījumus.

Tabula 4.1 **AA#Profile klases atribūti**

Atribūts	Vērtības tips	Apraksts
name	String	Profila nosaukums.

4.1.2 AA#Field

AA#Field klase satur visus lietotāja definētus laukus, kas ir piesaistīti kādam profilam un veido jaunas notācijas.

Tabula 4.2 **AA#Field klases atribūti**

Atribūts	Vērtības tips	Apraksts
name	String	Lauka nosaukums.
defaultValue	String	Noklusēta vērtība, kas tiek rādīta īpašību dialogā un diagrammā.
prefix	String	Prefikss, kas tiek piekārtots visiem šāda tipa elementiem.
suffix	String	Sufikss, kas tiek piekārtots visiem šāda tipa elementiem.
delimiter	String	Atdalītais starp vairākiem dotā lauka elementiem.
pattern	String	Simbolu saraksts, kurus ir atļauts izmantot, veidojot lauka vērtības.
isStereotypeField	boolean	Pazīme, ka lauks ir stereotips un no tā vērtības var būt atkarīga cita lauka attēlošanas veids.
displayPlaceBefore	String	Lauka nosaukums, pirms kura liekams jaunais lauks diagrammā.
propertyEditorPlaceBefore	String	Lauka nosaukums, pirms kura liekams jaunais lauks īpašību dialoglogā.
propertyEditorTab	String	Cilnes nosaukums, kurā ir liekams jaunais lauks īpašību dialoglogā.

4.1.3 AA#ChoiceItem

AA#ChoiceItem klase satur visus izvēles vienumus. Izvēles vienumi ir piekārtoti laukiem un no tiem var būt atkarīgi semantikas un stila uzstādījumi.

Tabula 4.3 **AA#ChoiceItem klases atribūti**

Atribūts	Vērtības tips	Apraksts
caption	String	Izvēles vienuma vērtība.
notation	String	Notācijas vērtība.

4.1.4 AA#RowType

AA#RowType klase satur lietotāja definēto lauku atspoguļošanas veidus.

Tabula 4.4 **AA#RowType klases atribūti**

Atribūts	Vērtības tips	Apraksts
typeName	String	Lauka atspoguļošanas veids.

Lauka atspoguļošanas veids var pieņemt vērtības: *InputField*; *InputField+Button*; *CheckBox*; *ComboBox*; *ListBox*; *TextArea*; *TextArea+Button*; *Empty*.

4.1.5 AA#Tag

AA#Tag klase satur visus lietotāja veidoto lauku semantikas definējumus, kas pieder noteiktam semantikas veidam.

Tabula 4.5 **AA#Tag klases atribūti**

Atribūts	Vērtības tips	Apraksts
axiomPattern	String	Semantikas definīcija.
tagKey	String	Semantikas veids.

4.1.6 AA#TagType

AA#TagType klase satur visus semantikas veidus, kas tiek piedāvāti lietotājam lauku semantikas definēšanai.

Tabula 4.6 **AA#TagType klases atribūti**

Atribūts	Vērtības tips	Apraksts
key	String	Semantikas veids.
notation	String	Semantikas nosaukums, kas tiek redzams lietotājam.
rowType	String	Semantikas lauka tips, kas tiek atspoguļots lietotājam.

4.1.7 AA#Translet

AA#Translet klase satur visas transletu definīcijas, kas tiek piekārtotas lietotāja definētam laukam. Ar transleta palīdzību var kādā noteiktā brīdī izsaukt kādu procedūru.

Tabula 4.7 **AA#Translet klases atribūti**

Atribūts	Vērtības tips	Apraksts
procedure	String	Ceļš līdz procedūrai, kas ir jāizsauc ar transletu.

4.1.8 AA#TransletTask

AA#TransletTask klase satur visus transleta veidus.

Tabula 4.8 **AA#TransletTask klases atribūti**

Atribūts	Vērtības tips	Apraksts
taskName	String	Transleta veids.

Transleta veids var pieņemt vērtības:

- procGenerateItemsClickBox – transleta procedūra tiek izsaukta lauka izvēles vienumu veidošanas brīdī;
- procStartValue – transleta procedūra tiek izsaukta lauka noklusētās vērtības aprēķināšanas brīdī;
- procFieldEntered – transleta procedūra tiek izsaukta, kad laukā tiek ierakstīta vērtība;
- procCompose – transleta procedūra tiek izsaukta lauka vērtības veidošanas brīdī no apakš kompartmentiem;
- procDecompose – transleta procedūra tiek izsaukta lauka vērtības sadalīšanas brīdī apakš kompartmentos.

4.1.9 AA#ContextType

AA#ContextType klase satur vērtības, kas apzīmē, zem kā lietotāja definēts lauks tiks izveidots. Var būt gan no **ElemType**, gan **CompartType** klases.

Tabula 4.9 **AA#ContextType klases atribūti**

Atribūts	Vērtības tips	Apraksts
type	String	Vērtība, kas apzīmē, zem kā jaunais lauks tiks izveidots.
nr	Integer	Kārtas numurs, ar kādu konteksta tips tiek rādīts lietotāja definēto lauku mehānismā.
elTypeName	String	Elementa tipa identifikators, zem kura atrodas konteksta tips.
path	String	Ceļš no elementa tipa līdz konteksta tipam.
mode	String	Stilizēšanas režīms, kas piemīt konteksta tipam.

		Var pieņemt vērtības: Element; Group; Group Item; Text.
hasMirror	Boolean	Pazīme, ka konteksta tipam ir apgriezts tips un lauks ir jāveido abos konteksta tipos.

Atribūta *mode* „Element" vērtība nozīmē, ka konteksta tips pieder **ElemType** klasei un ir stilizējams. Group – konteksta tips ir grupas elements un ir stilizējams. Group Item – pats konteksta tips ir stilizējams, bet apakšlauki nav. Text – konteksta tips un to apakš lauki ir nestilizējami.

4.1.10 AA#View

AA#View klase satur visus skatījumus, kas definē elementu un kompartmentu atspoguļošanas veidus un ir piesaistīti pie noteikta profila.

Tabula 4.10 **AA#View klases atribūti**

Atribūts	Vērtības tips	Apraksts
name	String	Skatījuma nosaukums.
isDefault	Boolean	Pazīme, ka skatījums jāpielieto pēc noklusējuma.

4.1.11 AA#StyleSetting

AA#StyleSetting klase satur visus lietotāja uzstādītus stila vienumus.

Tabula 4.11 **AA#StyleSetting klases atribūti**

Atribūts	Vērtības tips	Apraksts
value	String	Stila vērtība.
target	String	Mērķa kompartmenta tips, kuram uzstāda stilu.
path	String	Ceļš līdz mērķa kompartmenta tipam.
isElementStyleSetting	String	Pazīme, ka stils ir priekš elementa tipa.

AA#FieldStyleSetting klase ir apakšklase no AA#StyleSetting klases, satur visus stila uzstādījumus, kas tiek uzlikti no profila. Ja instance ir piekārtota laukam, tad stils uzstādās lietotāja definētam laukam. Ja instance ir piekārtota izvēles vienumam, tad stils ir uzstādāms kādam citam tipam atkarībā no jauna lauka vērtības.

AA#ViewStyleSetting klase, apakšklase no AA#StyleSetting klases, satur visus stila uzstādījumus, kas tiek uzlikti no skatījuma.

Tabula 4.12 **AA#ViewStyleSetting klases atribūti**

Atribūts	Vērtības tips	Apraksts
elementTypeName	String	Elementa tips, zem kura atrodas kompartmenta tips, kuram tiek uzstādīts stils.
addMirror	Boolean	Pazīme, ka stils jāuzliek arī pretējā kompartmenta tipā.
caseCompartTypeName	String	Kompartmenta tipa identifikators, no kura ir atkarīgs skatījuma stils.
caseChoiceItemName	String	Izvēles vienuma nosaukums, no kura ir atkarīgs skatījuma stils.

4.1.12 AA#CompartStyleItem

AA#CompartStyleItem klase satur visus iespējamos stila nosaukumus, kas var tikt piekārtoti kompartmenta tipam (laukiem).

Tabula 4.13 **AA#CompartStyleItem klases atribūti**

Atribūts	Vērtības tips	Apraksts
itemName	String	Stila nosaukums.
itemType	String	Stila Vērtības tips.
forNodeCompart	Boolean	Pazīme, ka stila tips var tikt uzlikts mezgla tipa kompartmenta tipiem.
forEdgeCompart	Boolean	Pazīme, ka stila tips var tikt uzlikts šķautnes tipa kompartmenta tipiem.
forAttribCompart	Boolean	Pazīme, ka stila tips var tikt uzlikts nestilizējamiem kompartmenta tipiem.

4.1.13 AA#ElemStyleItem

AA#ElemStyleItem klase satur iespējamos stila nosaukumus, kas var tikt piekārtoti elementa tipiem.

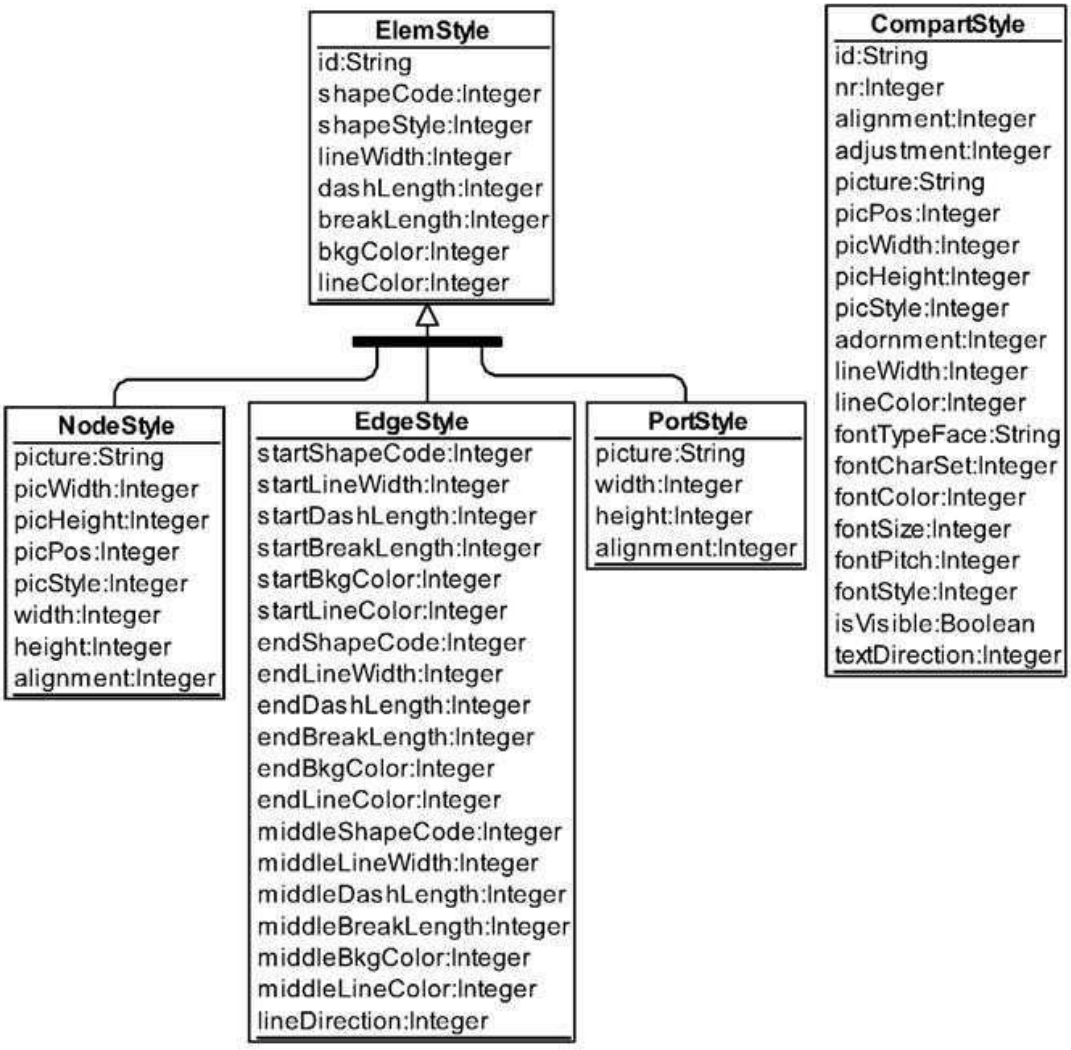
Tabula 4.14 **AA#ElemStyleItem klases atribūti**

Atribūts	Vērtības tips	Apraksts
itemName	String	Stila nosaukums.
itemType	String	Stila vērtības tips.

AA#NodeStyleItem klase ir apakšklase no AA#ElemStyleItem klases, kas satur iespējamos stila nosaukumus, kas var tikt piekārtoti mezgla tipa elementiem.

AA#EdgeStyleItem klase ir apakšklase no AA#ElemStyleItem klases, kas satur iespējamos stila nosaukumus, kas var tikt piekārtoti šķautņu tipa elementiem.

AA#AnyElemStyleItem klase, apakšklase no AA#ElemStyleItem klases, kas satur iespējamos stila nosaukumus, kas var tikt piekārtoti gan mezgla, gan šķautņu tipa elementiem.



Attēls 4.2 **Elementu un lauku stila vienumi [13]**

Elementu un kompartmentu iespējamie stila vienumi tiek parādīti attēlā ***Attēls 4.2*** Elementu un lauku stila vienumi [13].

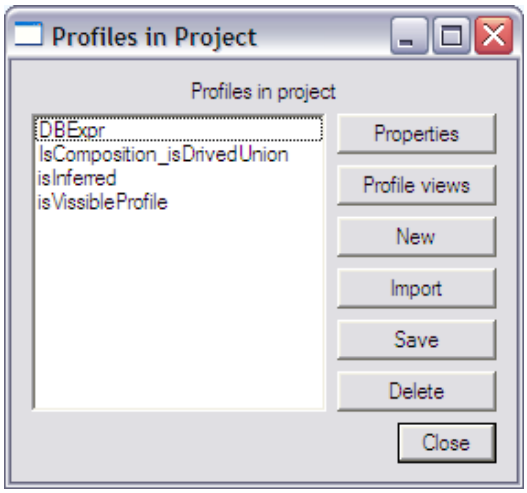
4.1.14 AA#Configuration

AA#Configuration klasei tiek piesaistīti visi konteksta tipi, profili, tagu tipi, kas tiek izmantoti dotā projekta ietvaros. Katram projektam ir tieši viena konfigurācija.

4.2 Profilu pārvaldības logs

Uzsākot darbu ar lietotāja definēto lauku mehānismu, tiek parādīts logs ar navigācijas iespējām. Tiek piedāvātas šādas iespējas:

- Izveidot jaunu profilu;
- Importēt profilu no teksta faila;
- Eksportēt profilu teksta failā;
- Dzēst profilu;
- Apskatīt un rediģēt profilu;
- Atvērt formu ar profilam piesaistītiem skatījumiem.



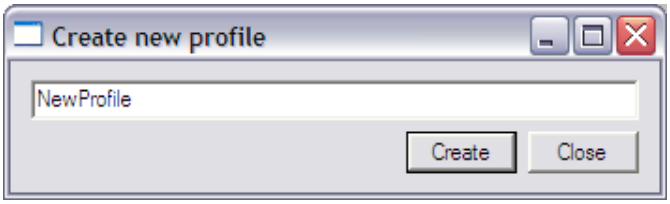
Attēls 4.3 **Profilu pārvaldības logs**

Piekļūt profilu pārvaldības logam var no projekta diagrammas rīku joslas, nospiežot Ⓟ pogu.

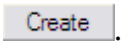
4.2.1 Jaunu profilu izveide

Lai izveidotu jaunu profilu no profilu pārvaldības loga, jānospiež 

poga. Tālāk tiek atvērts jauns logs, kur lietotājam tiek piedāvāts ievadīt jauna profila nosaukumu.



Attēls 4.4 **Jaunu profilu izveides logs**

Pēc jauna profila nosaukuma ievadīšanas jānospiež poga 

. Nospiežot pogu tiek izveidota **AA#Profile** instance, kas atbilst jaunajam profilam un tipu daļas **Extension**

instance, ar tipu *aa#Profile*. **AA#Profile** klases instance tiek piekārtota **AA#Configuration** klases instancei, kas atbilst dotā projekta konfigurācijai. Tiek atvērts lietotāja definētu lauku pārvaldības logs, kas tiek aprakstīts nodaļā 4.5.

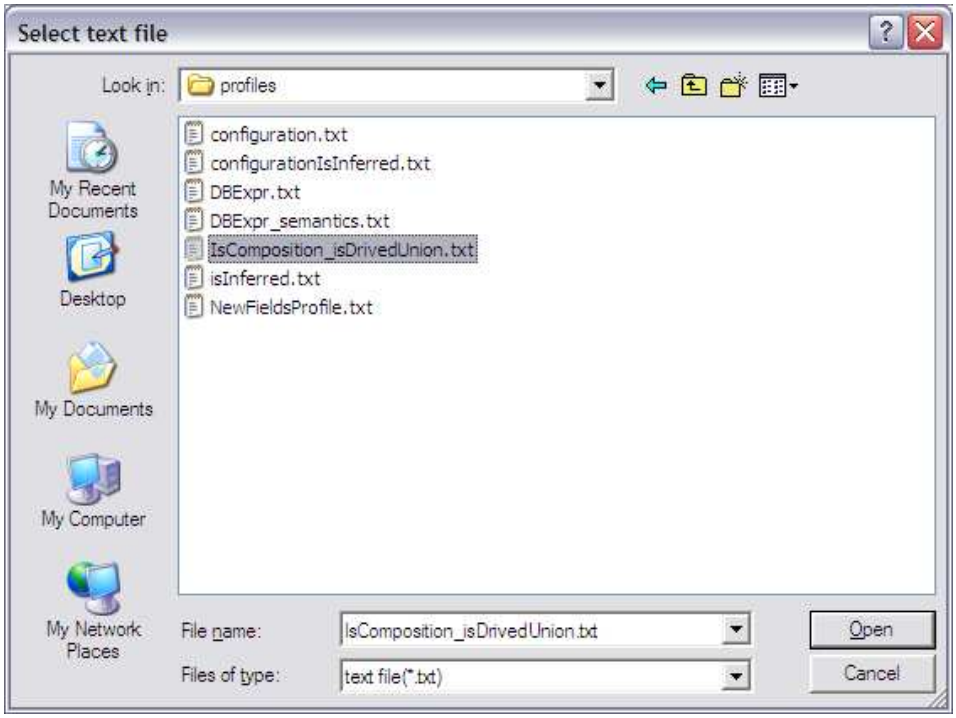
4.2.2 Profilu dzēšana

Lai izdzēstu profilu no profilu pārvaldības loga, jānospiež Delete poga. Nospiežot pogu tiek dzēstas visas instances, kas atrodas zem dzēšamas profila instances. Instances tiek dzēstas gan no lietotāja definēto lauku mehānisma metamodeļa (ar „AA" prefiksu), gan no tipu daļas metamodeļa.

Profila nosaukums tiek aizvākts no izveidoto profilu saraksta.

4.2.3 Profilu imports no teksta faila

Lai importētu profilu no teksta faila no profilu pārvaldības logā, jānospiež Import poga. Tālāk tiek atvērts failu meklēšanas logs un piedāvāts izvēlēties teksta failu, kas satur iekodēto informāciju par importēto profilu.



Attēls 4.5 **failu meklēšanas logs**

Nospiežot Open pogu, tiek pārbaudīts, vai profils ar tādu nosaukumu jau neeksistē. Ja tāds profils eksistē, tad no jauna tas netiek importēts un tiek izvadīts kļūdas ziņojums:

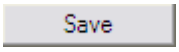
Attēls 4.6 **kļūdas ziņojums „Profils ar tādu nosaukumu jau eksistē”**

Tiek arī pārbaudīts, vai visi profilā izmantojamie konteksta tipi ir pieejami dotajā projektā. Ja kāds konteksta tips nav pieejams, tad profils netiek importēts un tiek izvadīts kļūdas ziņojums:

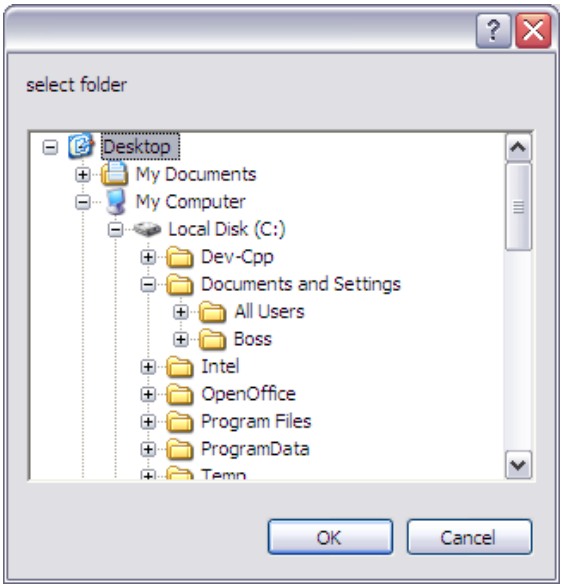
Attēls 4.7 **kļūdas ziņojums „Pietrūkst konteksta tipu”**

Ja kāda konteksta tipa pietrūkst, tad tas ir jāielādē projektā, sīkāk tas tiek aprakstīts 4.6 nodaļā. Ja tiek izietas visas pārbaudes, tiek importētas visas lietotāja definēto lauku mehānisma metamodeļa instances un tiek veikta to transformācija uz tipu daļu, kas tika aprakstīta 5.2 nodaļā.

4.2.4 Profilu eksports teksta failā

Lai eksportētu profilu teksta failā, profilu pārvaldības logā jāizvēlas vajadzīgais profils no saraksta un jānospiež 

poga. Tālāk tiek jautāts, vai profilu vajadzēs ielādēt automātiski. Ja vajadzēs, tad profils tiek saglabāts „AutoLoad” mapē, un spraudņa instalācijas brīdī tas tiks ielādēts automātiski. Ja nav vajadzības pēc profila automātiskas ielādes, tad tiek atvērts logs ar iespēju izvēlēties profila eksporta teksta faila izveidošanas vietu.



Attēls 4.8 **Logs profila eksporta teksta faila izveidošanas vietas izvēlei**

Pēc noklusējuma tiek piedāvāts saglabāt profilu tajā pašā mapē, kur ir visi lietotāja definēto lauku mehānisma izejas koda faili.

Profila eksporta faila nosaukums tiek veidots šādi: [**ProfileName]_[month]_[date]_[year]_[hour]_[minute]_[second]**, lai izslēgtu iespēju izveidot vairākus teksta failus ar vienādu nosaukumu.

Profila eksporta teksta faila saturs ir šāds:

```
return {
  [2000821135] = {
    ["class"] = "AA#ChoiceItem",
    ["properties"] = {
      ["caption"] = "false"
    },
    ["links"] = {

    }
  },
  [2000820605] = {
    ["class"] = "AA#Profile",
    ["properties"] = {
      ["name"] = "NewProfile"
    },
    ["links"] = {
      ["field"] = {
        [1] = 2000820652
      }
    }
  },
  [2000819844] = "AA#RowType[typeName=CheckBox]",
  [2000820652] = {
    ["class"] = "AA#Field",
    ["properties"] = {
      ["pattern"] = "a-zA-Z0-9-_",
      ["displayPlaceBefore"] = "Name",
      ["propertyEditorTab"] = "Main",
      ["propertyEditorPlaceBefore"] = "Name",
      ["name"] = "NewUsedDefinedField"
    },
    ["links"] = {
      ["choiceItem"] = {
        [1] = 2000821134,
        [2] = 2000821135
      },
      ["context"] = {
        [1] = 2000819833
      },
      ["fieldType"] = {
        [1] = 2000819844
      }
    }
  },
  [2000821134] = {
    ["class"] = "AA#ChoiceItem",
    ["properties"] = {
      ["caption"] = "true"
    },
    ["links"] = {

    }
  },
```

```
      [2000819833] = "AA#ContextType[id=Class]"
    }, 2000820605
```

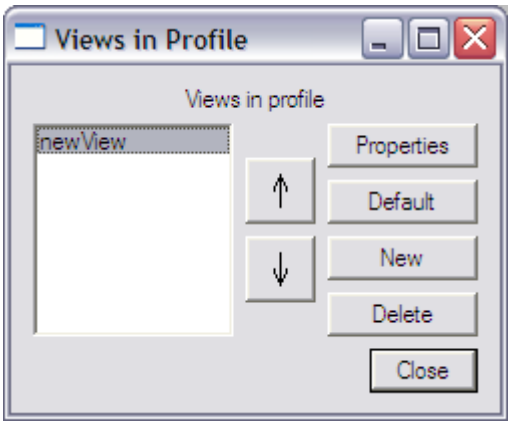
Profila eksporta teksta faila struktūra ir šāda:

- [2000821135] ir instances identifikators;
- Zem ["class"] glabājas klases nosaukums, kurai pieder instance;
- Zem ["properties"] glabājas saraksts ar instances atribūtiem un to vērtībām;
- Zem ["links"] glabājas saraksts ar instances saitēm.

4.3 Skatījumu pārvaldības logs

Ja no profila pārvaldības loga tiek izvēlēts apskatīt formu ar profilam piesaistītiem skatījumiem, nospiežot 

pogu tiek atvērts logs ar profilam piesaistītiem skatījumiem un to navigācijas iespējām.

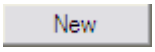


Attēls 4.9 **Skatījumu pārvaldības logs**

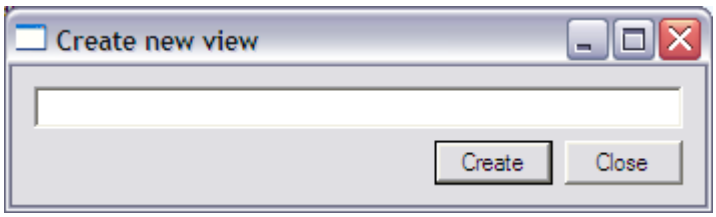
Skatījumu konfigurācijas logā tiek piedāvātas šādas navigācijas iespējas:

- Izveidot jaunu skatījumu;
- Dzēst skatījumu;
- Apskatīt un rediģēt skatījumu.

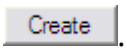
4.3.1 Jaunu skatījumu izveide

Lai izveidotu jaunu skatījumu, skatījumu pārvaldības logā jānospiež 

poga. Tālāk tiek atvērts jauns logs, kur lietotājam tiek piedāvāts ievadīt jauna skatījuma nosaukumu.

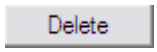


Attēls 4.10 **Jaunu skatījumu izveides logs**

Pēc jauna skatījuma nosaukuma ievadīšanas jānospiež poga 

. Nospiežot pogu tiek izveidota **AA#View** klases instance, kas atbilst jaunajam skatījumam un tipu daļas **Extension** klases instancei, ar tipu *aa#View*. **AA#View** klases instance tiek piekārtota **AA#Profile** klases instancei, kas atbilst profilam, zem kura tiek veidots jauns skatījums. Tiek atvērts skatījuma rediģēšanas dialogu logs, kas tiek aprakstīts nodaļā 4.3.3.

4.3.2 Skatījumu dzēšana

Lai izdzēstu skatījumu no skatījumu pārvaldības logā, jānospiež 

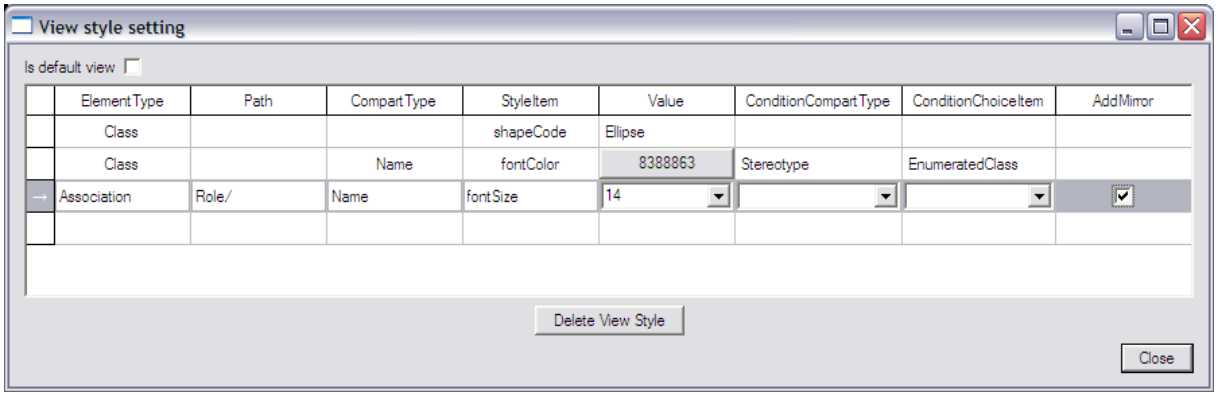
poga. Nospiežot pogu tiek dzēstas visas instances, kas atrodas zem dzēšamās skatījuma instances. Instances tiek dzēstas gan no lietotāja definēto lauku mehānisma metamodeļa, gan no tipu daļas metamodeļa.

Skatījuma nosaukums tiek aizvākts no izveidoto skatījumu saraksta.

4.3.3 Skatījuma rediģēšanas logs

Lai apskatītu vai rediģētu skatījumu, no skatījumu pārvaldības loga, jānospiež 

poga. Nospiežot pogu, tiek atvērta tabulas veida forma ar stilu sarakstu, kas tiek uzstādīti no dotā skatījuma.



Attēls 4.11 **Skatījuma rediģēšanas logs**

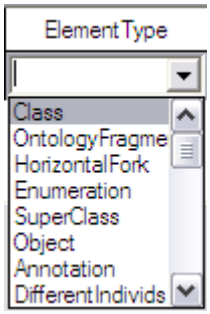
Šajā formā ir iespējams uzstādīt stilus visiem kādas diagrammas elementiem, visiem noteikta veida diagrammas kompartmentiem, noteikta veida diagrammas kompartmentiem, kuriem izpildās konkrētie nosacījumi.

Formā tiek piedāvāts padarīt skatījumu par noklusēto (iezīmējot *Is default view* izvēles rūtiņu). Tas nozīmē, ka skatījuma stili tiks piekārtoti visiem elementiem un kompartmentiem visās diagrammās.

Tālāk lietotājam tabulas veidā tiek atspoguļoti visi stila uzstādījumi no dotā skatījuma. Tabula satur šādas kolonnas:

- **ElementType** – elementa tips, kuram tiek piekārtots stils vai zem kura atrodas kompartmenta tips, kuram tiek piekārtots stils;
- **Path** – ceļš no elementa tipa līdz kompartmenta tipam, kuram tiek piekārtots stils;
- **CompartType** – kompartmenta tips, kuram tiek piekārtots stils;
- **StyleItem** – piekārtota stila nosaukums;
- **Value** – piekārtota stila vērtība;
- **ConditionCompartType** – kompartmenta tips, no kura ir atkarīga stila uzstādīšana;
- **ConditionChoiceItem** – izvēles vienums, no kura ir atkarīga stila uzstādīšana;
- **AddMirror** – pazīme, ka dotajam kompartmenta tipam ir apgriezts kompartmenta tips, kuram arī ir jāpieliek stils.

Lai izveidotu jaunu elementa stila instanci, tukšajā tabulas rindā kolonnā „ElementType” no piedāvātā saraksta ir jāizvēlas elementa tips, kuram tiks uzstādīts stils.



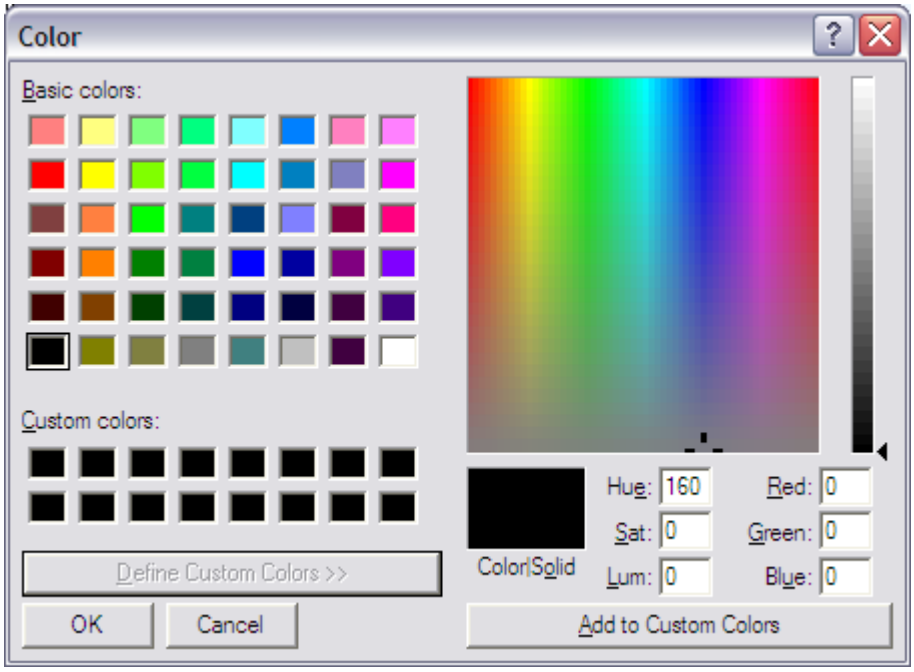
Attēls 4.12 **Elementa tipa izvēle no saraksta**

Izvēloties elementa tipu, tiek veidota **AA#ViewStyleSetting** klases instance ar šādiem parametriem: *elementTypeName* vienāds ar izvēlētā elementa tipa nosaukumu, *isElementStyleSetting* vienāds ar „true” (pazīme, ka stils ir domāts elementiem). Izveidotā instance tiek piesaistīta skatījuma instancei no **AA#View** klases.

Pēc elementa tipa izvēles kolonnā „Style Item” tiek piedāvāti stila vienumi, kas var tikt uzstādīti konkrētajam elementu tipam.

Kad tiek izvēlēts kāds vienums, **AA#ViewStyleSetting** klases instance tiek savienota ar **AA#ElemStyleItem** klases instanci, kurai *itemName* atribūta vērtība ir vienāda ar izvēlēto vērtību.

Pēc stila vienuma izvēles kolonnā „Value" tiek piedāvātas šī stila iespējamās vērtības, ja tādas eksistē. Ja tiek izvēlēts kāds stila vienums, kas uzstāda krāsas, kolonnas šūnā tiek attēlota poga, kuru nospiežot tiek atvērts krāsu dialoga logs.



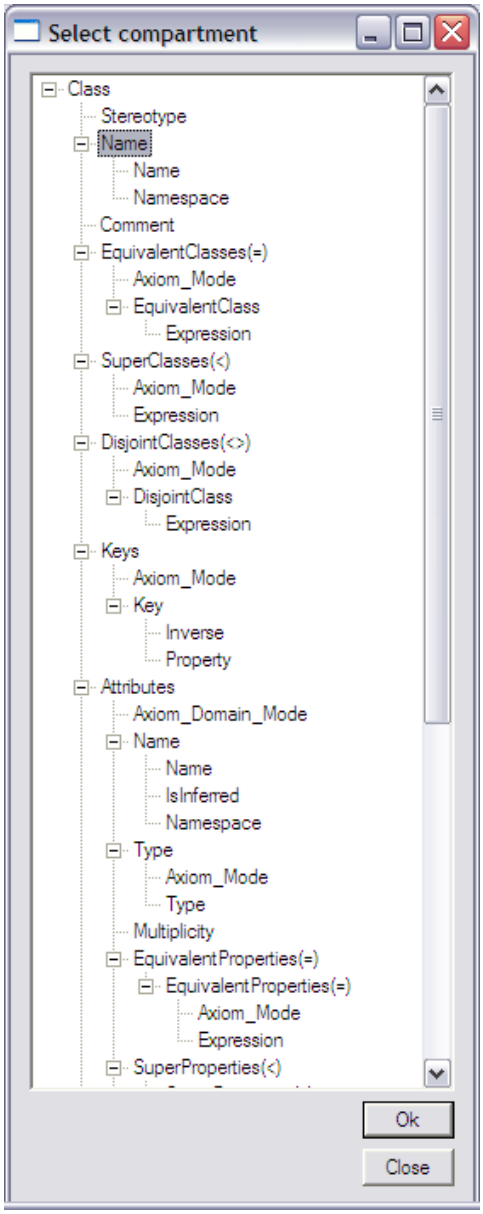
Attēls 4.13 **krāsu dialoga logs**

Kad tiek izvēlēta vai ierakstīta kāda stila vērtība, tā tiek ierakstīta **AA#ViewStyleSetting** klases instances atribūtā *value*.

Lai izveidotu jaunu kompartmenta stila instanci, tukšajā tabulas rindā kolonnā „ElementType" no piedāvātā saraksta ir jāizvēlas elementa tips, zem kura atrodas vēlamais kompartmenta tips.

Izvēloties elementa tipu, tiek veidota **AA#ViewStyleSetting** klases instance ar šādiem parametriem: *elementTypeName* vienāds ar izvēlētā elementa tipa nosaukumu, *isElementStyleSetting* vienāds ar „true" (pazīme, ka stils pagaidām ir domāts elementiem). Izveidotā instance tiek piesaistīta skatījuma instancei no **AA#View** klases.

Pēc elementa tipa izvēles kolonnā „CompartType" tiek izveidota poga „set compartType", kuru nospiežot tiek atvērts koks ar visu kompartmenta tipu struktūru, kas atrodas zem dotā elementa tipa.



Attēls 4.14 **Koks ar kompartmenta tipu struktūru**

Uz kompartmenta tipa izvēles **AA#ViewStyleSetting** klases instances atribūtā *target* tiek ierakstīts izvēlētā kompartmenta tipa nosaukums. Atribūta *isElementStyleSetting* vērtība tiek nomainīta uz „false” (pazīme, ka stils ir domāts kompartmentiem). Atribūtā *path* tiek ierakstīts ceļš no elementa tipa līdz izvēlētajam kompartmenta tipam.

Pēc kompartmenta tipa izvēles kolonnā „Style Item” tiek piedāvātas stila komponentes, kas var tikt uzstādītas konkrētajam kompartmenta tipam.

Kad tiek izvēlēta kāda komponente, **AA#ViewStyleSetting** klases instance tiek savienota ar **AA#CompartmentStyleItem** klases instanci, kurai *itemName* atribūta vērtība ir vienāda ar izvēlēto vērtību.

Pēc stila komponentes izvēles kolonnā „Value” tiek piedāvātas šī stila iespējamās vērtības, ja tādas eksistē. Ja tiek izvēlēta kāda stila komponente, kas uzstāda krāsas, kolonnas šūnā tiek attēlota poga, kuru nospiežot tiek atvērts krāsu dialoga logs.

Kad tiek izvēlēta vai ierakstīta kāda stila vērtība, tā tiek ierakstīta **AA#ViewStyleSetting** klases instances atribūtā *value*.

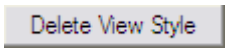
Ja kompartmenta tipam ir apgriezts kompartmenta tips, ir iespējams norādīt, ka tam arī ir jāpiekārto stils, iezīmējot kolonnā „AddMirror" izvēles rūtiņu. Šādā situācijā **AA#ViewStyleSetting** klases instances atribūta *addMirror* vērtība tiek uzstādīta uz „true". Ja izvēles rūtiņa netiek iezīmēta, tad apgrieztam kompartmenta tipam stils netiek piekārtots.

Pastāv iespēja norādīt, ka stils tiek uzstādīts tikai tiem kompartmentiem, kuriem viena elementa ietvaros kāds cits kompartments pieņem noteiktu vērtību. Tas ir paveicams, izvēloties no „ConditionCompartType" kolonnā piedāvātajiem variantiem kompartmenta tipu, no kura būs atkarīga stila uzlikšana. Tiek piedāvāti tikai tie kompartmenta tipi, kas ir vienā līmenī ar doto kompartmenta tipu un kuriem ir definēti izvēles vienumi (*choiceItem*). Tālāk no „ConditionChoiceItem" kolonnas ir jāizvēlas noteikts vienums. Pēc šo darbību paveikšanas **AA#ViewStyleSetting** klases instances atribūtā *caseCompartTypeName* tiek ierakstīts kompartmenta tipa nosaukums no „ConditionCompartType" kolonnas un atribūtā *caseChoiceItemName* tiek ierakstīts izvēlētais vienums.

Ar skatījuma rediģēšanas formu var uzstādīt ne tikai stilus, bet arī papildus prefiksus un sufiksus konkrētiem kompartmentiem. Tas ir izdarāms, izvēloties no „Style Item" kolonnā piedāvātajiem stiliem vienu no:

- prefix-inside (pieliek prefiksu pēc jau pastāvošiem prefiksiem),
- prefix-ouside (pieliek prefiksu pirms jau pastāvošiem prefiksiem),
- prefix-inPlace (pieliek prefiksu jau pastāvošo prefiksu vietā), vai
- suffix-inside (pieliek sufiksu pirms jau pastāvošiem sufiksiem),
- suffix-ouside (pieliek sufiksu pēc jau pastāvošiem sufiksiem),
- suffix-inPlace (pieliek sufiksu jau pastāvošo sufiksu vietā).

Kolonnā „Value" ir jāieraksta prefiksa/sufiksa vērtība.

Skatījumu rediģēšanas logā pastāv iespēja dzēst stilu. Tas ir paveicams, novietojot kursoru vajadzīgajā tabulas rindā un nospiežot 

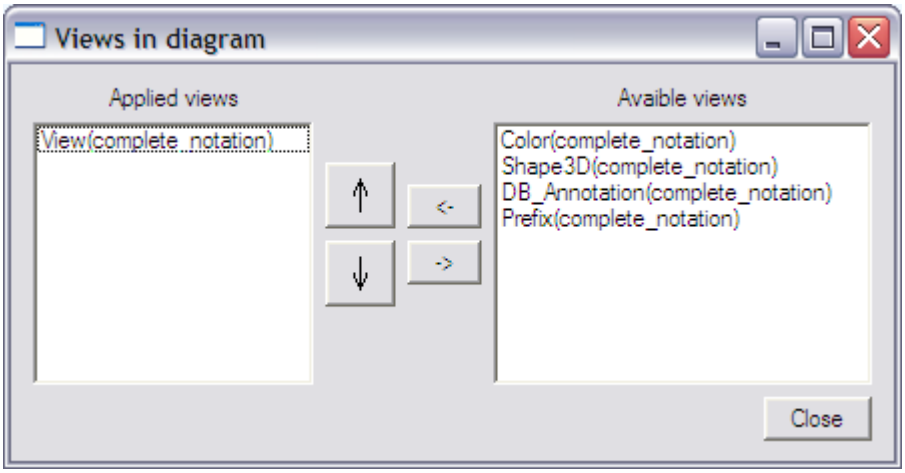
pogu. Nospiežot pogu, **AA#ViewStyleSetting** klases instance, kas atbilst dotajam stilam, tiek dzēsta.

Aizverot skatījuma rediģēšanas logu, visas izdarītās izmaiņas tiek transformētas uz tipu daļas metamodeli pēc nodaļā 5.2 aprakstītā veida.

4.4 Skatījumu pielietošana diagrammai

Skatījumus no visiem profiliem projektā ir iespējams pielietot diagrammām.

Katras diagrammas rīkjoslā atrodas  ikona, kuru nospiežot atvērts logs ar visiem pielietotajiem un pieejamajiem dotās diagrammas skatījumiem.



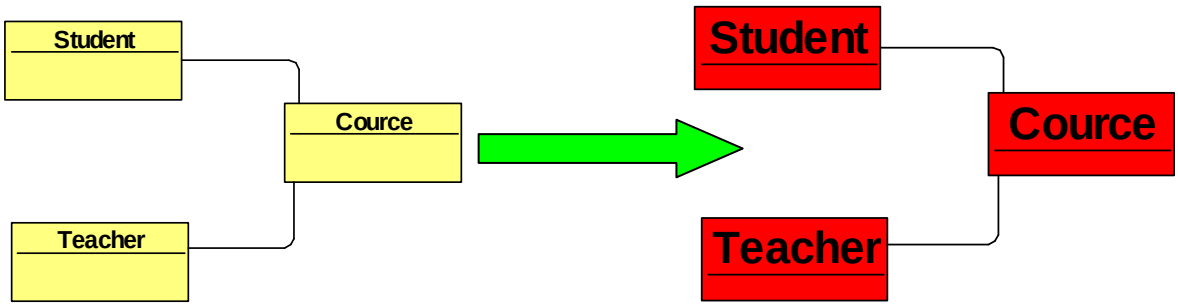
Attēls 4.15 **Skatījumu pielietošanas logs**

Šajā formā ir iespējams veikt tādas darbības kā pielietot dotajai diagrammai kādu skatījumu, noņemt skatījumu no diagrammas, pārvietot skatījumu augstāk vai zemāk (uzstādīt augstāku/zemāku prioritāti).

Par skatījumu tiek attēlota tāda informācija kā skatījuma nosaukums un profila nosaukums, kuram pieder skatījums. Sintakse ir šāda: **[skatījums_nosaukums]([Profila_nosaukums])**.

Pielietot dotajai diagrammai kādu skatījumu, var izvēlēties to no pieejamo skatījumu saraksta un nospiežot pogu  . Nospiežot pogu, skatījums tiek pārvietots no pieejamo skatījumu saraksta uz pielietoto skatījumu sarakstu. Starp **Extension** klases instanci, kas atbilst dotajam skatījumam un **GraphDiagram** klases instanci, kas atbilst dotajai diagrammai, tiek novilkta saite *aa#activeExtension* pie Extension un *graphDiagram* pie GraphDiagram. Tālāk tiek veikta stilu pārrēķināšana, kas ir aprakstīta nodaļā 5.4.

Piemēram, ja mums ir skatījums, kas uzstāda klašu kastes krāsu par sarkanu un klašu nosaukumiem burtu izmēru uz 20 punktiem, pēc skatījuma pielietošanas diagrammā elementi izskatīsies šādi:



Attēls 4.16 **Skatījuma pielietošanas piemērs**

Noņemt no dotās diagrammas kādu skatījumu var izvēloties to no pielietoto skatījumu saraksta un nospiežot pogu  . Nospiežot pogu tiek noņemta saite starp **Extension** klases instanci, kas atbilsts dotajam skatījumam, un **GraphDiagram** klases instanci, kas atbilst dotajai diagrammai. Tālāk tiek veikta stilu pārrēķināšana, kas ir aprakstīta nodaļā 5.4.

Stili no skatījumiem tiek uzstādīti atbilstoši skatījuma prioritātei. Prioritāti nosaka tas, kādā secībā skatījumi tiek pievienoti diagrammai. Šo secību nosaka *aa#activeExtension* saites secības uzstādīšana. Prioritātes uzstādīšana izpaužas šādi: ja diagrammai ir pievienoti divi skatījumi, kas ietekmē vienu un to pašu stila elementu, tiks uzstādīts tā skatījuma stils, kuram saite uz diagrammu ir pielikta augstāk.

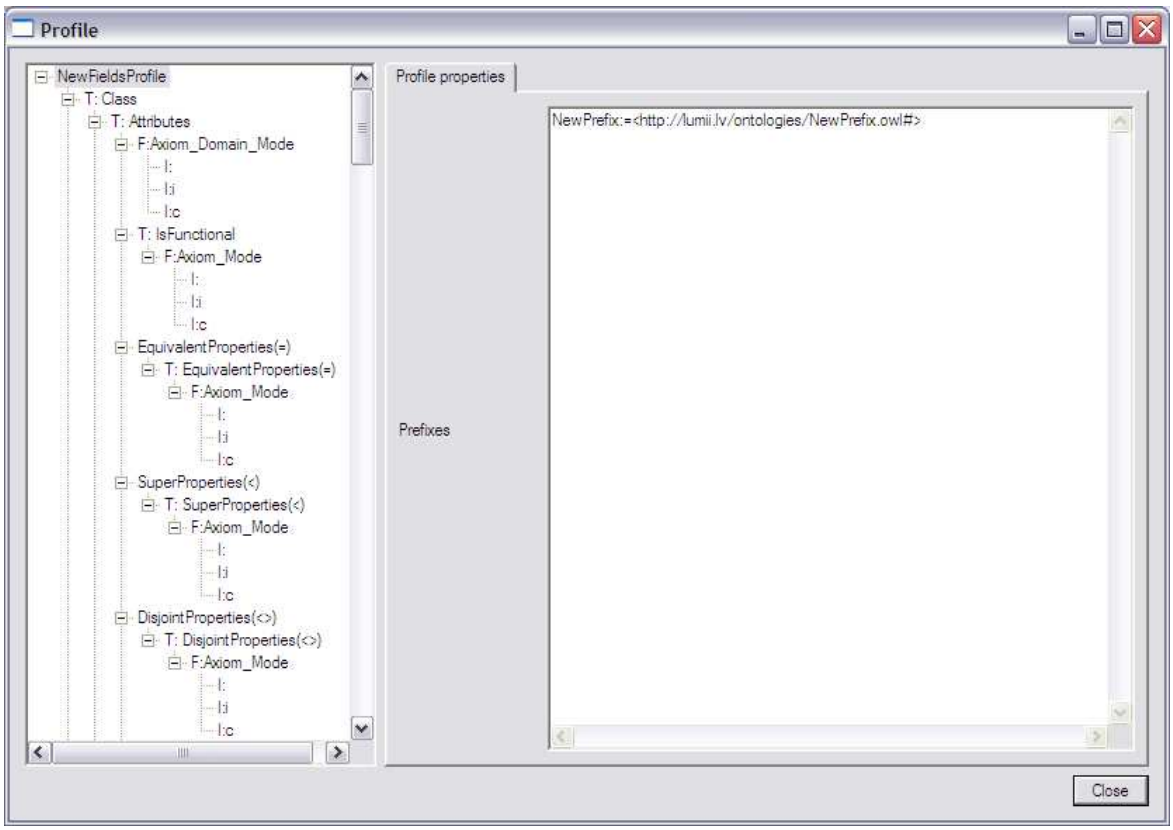
Pastāv iespēja vienu skatījumu pacelt augstāk par citu. Tas ir izdarāms, izvēloties skatījumu no pielietoto skatījumu saraksta un nospiežot pogu  . Nospiežot pogu, skatījums pielietotu skatījumu sarakstā tiek pārvietots par vienu pozīciju augstāk. Tālāk tiek veikta stilu pārrēķināšana, kas ir aprakstīta nodaļā 5.4.

Līdzīgi notiek arī gadījumos, ja skatījums tiek pārvietots zemāk. Tas ir izdarāms, izvēloties skatījumu no pielietoto skatījumu saraksta un nospiežot pogu  .

4.5 Lietotāja definētu lauku pārvaldības logs

Lai atvērtu lietotāja definētu lauku pārvaldības logu, profilu pārvaldības logā jāizvēlas vajadzīgais profils no saraksta un jānospiež 

poga. Tālāk tiek atvērts logs ar iespēju apskatīt un rediģēt lietotāja definētus laukus un to īpašības.



Attēls 4.17 **Lietotāja definētu lauku pārvaldības logs**

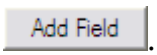
Labajā formas pusē tiek attēlots koks ar pieejamajiem kompartmenta tipiem, izveidotiem laukiem un izvēles vienumiem. Kokā atrodas virsotne, kas atbilst profilam. Visi kompartmenta tipi, zem kuriem var tikt veidoti jaunie lauki, tiek apzīmēti ar prefiksu „**T:** ”. Visi lietotāja izveidotie lauki tiek apzīmēti ar prefiksu „**F:** ”. Visi lietotāja izveidotie izvēles vienumi tiek apzīmēti ar prefiksu „**I:** ”. Ja kokā atrodas kāda virsotne bez prefiksiem, tad tā ir papildus virsotne, kas tiek izmantota, lai attēlotu koka struktūru.

Atverot lietotāja definētu lauku pārvaldības logu, tiek atvērts profila īpašību logs, kur lietotājs var nodefinēt ontoloģijas prefiksus, kas tiks ņemti vērā ontoloģijas importa laikā no Protégé redaktora. Prefiksu definēšana tiek pierakstīta šādā formā: **[PrefixName]:=<URI>**. Piemērām *NewPrefix:=<http://lumii.lv/ontologies/NewPrefix.owl#>*.

Katra prefiksa definīcija tiek ierakstīta savā **AA#Tag** klases instancē, kur *tagKey= "owl_Import_Prefixes"* un *axiomPattern* atribūts vienāds ar prefiksa definīciju (*axiomPattern='NewPrefix:=<http://lumii.lv/ontologies/NewPrefix.owl#>').* Izveidotā instance tiek piesaistīta pie **AA#Profile** klases instances, kas atbilst atvērtajam profilam. Sinhronizācijas laikā **AA#Tag** klases instanču, kas ir piesaistītās projekta profiliem, *axiomPattern* atribūta saturs tiek notranslēts uz vienu **Tag** klases instanci ar *key= owl_Import_Prefixes*, kur *value* atribūts satur visu prefiksu deklarācijas. Dotā instance tiek piesaistīta pie **ToolType** klases instances. Instances *value* atribūts tiek veidots šādi:

Prefix([AA#Tag_instance_axiomPattern_value])\nPrefix([AA#Tag_instance_axiomPattern_value]). Piemēram: *Prefix(owlFields:=<http://lumii.lv/ontologies/owlFields.owl#>)\nPrefix(NewPrefix:=<http://lumii.lv/ontologies/NewPrefix.owl#>).*

4.5.1 Lietotāja definēto lauku izveide

Lai izveidotu lietotāja definētu lauku, no koka jāizvēlas konteksta tips (virsotne ar „**T:** ” prefiksu), zem kura jāveido lauks un jānospiež poga 

.

Nospiežot pogu, tiek izveidota **AA#Field** klases instance, kas atbildīs jaunam laukam. Šī instance tiek piesaistīta **AA#Profile** klases instancei, kas atbilst atvērtajam profilam, ja lauks ir pirmā līmeņa (ja laukam nav lietotāja definētu apakšlauku) vai arī lietotāja definētam virs laukam. Tiek atvērta forma ar lauku rediģēšanas iespējām.

Atvērtajā logā tiek piedāvātas turpmāk nosauktās iespējas:

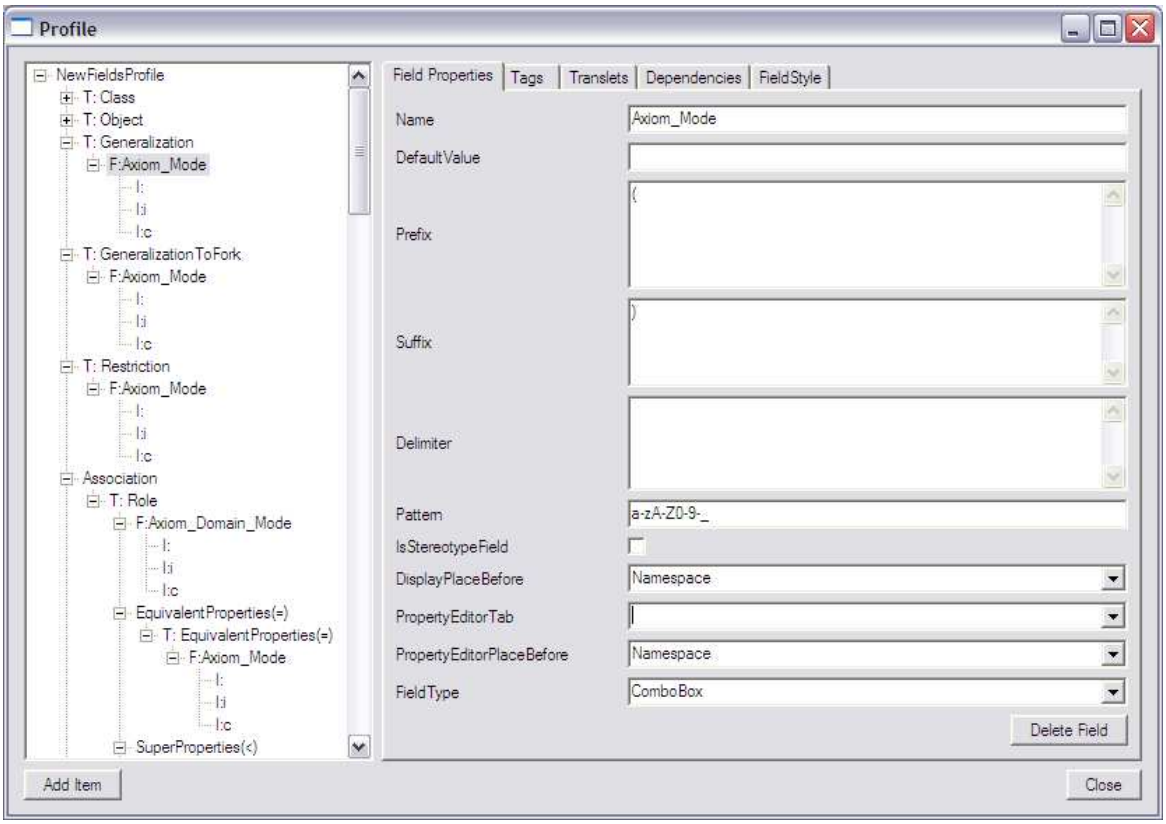
- Rediģēt lauka īpašības;
- Uzstādīt lauka tagus;
- Uzstādīt lauka transletus;
- Uzstādīt lauka atkarības;
- Uzstādīt lauka stilus.

4.5.1.1 Lauka īpašību rediģēšana

Tiek piedāvāta iespēja rediģēt šādas īpašības:

- Lauka nosaukumu (Name);
- Noklusēto vērtību (Default value);
- Lauka prefiksu (Prefix);
- Lauka sufiksu (Suffix);
- Atdalītāju (Delimiter);
- Simbolu šablonu – simbolu sarakstu, kurus ir atļauts izmantot, veidojot lauka vērtības (Pattern);
- Pazīmi, ka lauks ir stereotips – ja no tā ir atkarīgs cita lauka attēlošanas veids (IsStereotypeField);
- Vietu diagrammā, kur attēlosies jauns lauks – tiek piedāvāts izvēlēties no tā paša līmeņa kompartmentu tipiem (DisplayPlaceBefore) kā veidotajam laukam;

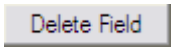
- Cilni, kurš tiks attēlots lauks īpašību dialogā (PropertyEditorTab);
- Vietu īpašību dialogā, kur attēlosies lauks (PropertyEditorPlaceBefore);
- Lauka attēlošanas veidu īpašību dialogā (FieldType);



Attēls 4.18 **Lauka īpašību rediģēšanas logs**

Ja lauka attēlošanas veids ir uzlikts uz „CheckBox", „ComboBox" vai „ListBox", laukam var tikt veidoti izvēles vienumi (ChoiceItem). Pie „CheckBox" lauka tipa automātiski tiek izveidoti divi izvēles vienumi ar *true* un *false* vērtībām.

Ja lauka attēlošanas veids ir uzlikts uz „InputField+Button" vai „TextArea+Button", laukam var tikt veidoti apakšlauki.

Lietotāja veidotus laukus var arī izdzēst. Lai to izdarītu, ir jānospiež 

poga. Nospiežot pogu tiek izvadīts logs ar visiem apakšlaukiem un izvēles vienumiem, kas ir piesaistīti dzēšamam laukam un lūgumu apstiprināt lauka dzēšanu.

Attēls 4.19 **lauka dzēšanas apstiprināšanas logs**

Ja tiek nospiesta 

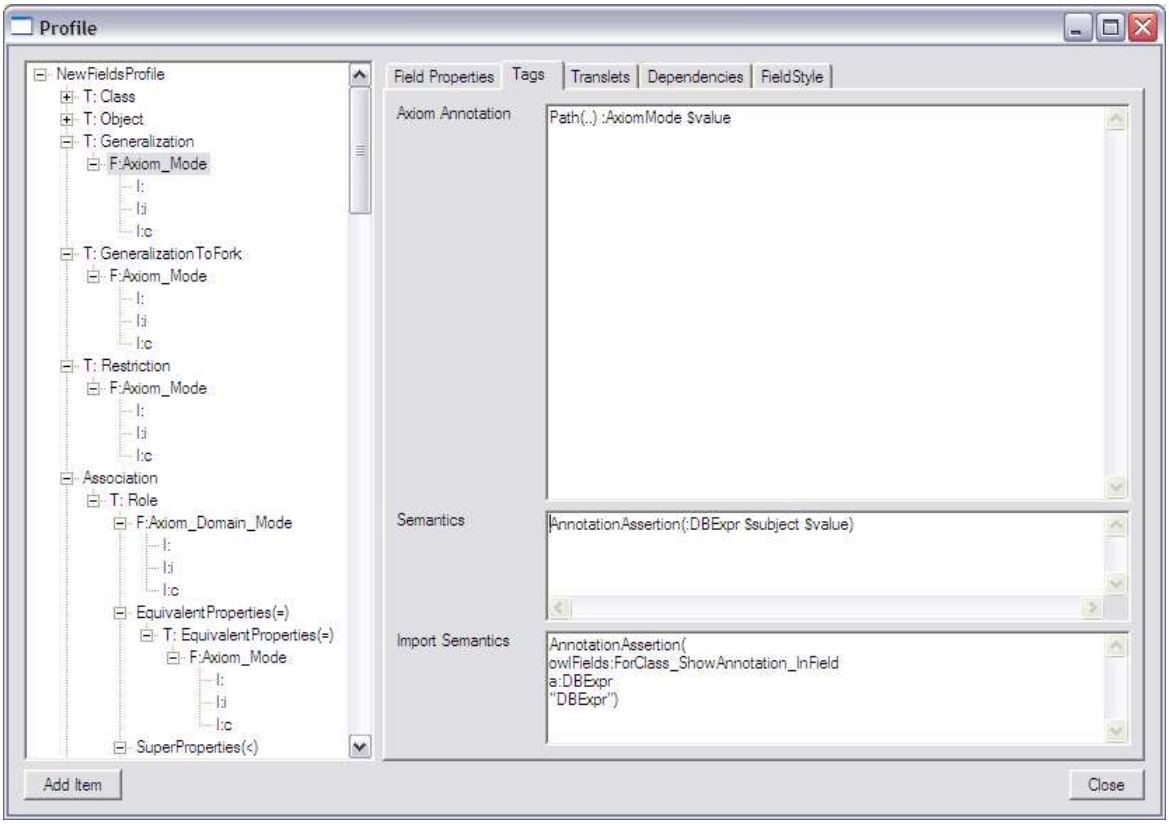
poga, tad lauks netiek dzēsts. Ja tiek nospiesta 

poga, tad lauks ar izvēles vienumiem un apakšlaukiem tiek dzēsts no lietotāja definēto lauku koka. Tiek dzēsta **AA#Field** klases instance, kas atbilst dzēšamajam laukam, kā arī tiek dzēstas visas pārējās instances, kas ir saistītas ar doto instanci. Tiek dzēstas šādu klašu instances: AA#Field, AA#ChoiceItem, AA#Tag, AA#Translet, AA#FieldStyleSetting, AA#Dependency, ja tie ir saistīti ar lauka instanci.

4.5.1.2 Lauka tagu uzstādīšana

Tagi ir paredzēti, lai definētu semantiku priekš eksporta un importa no Protégé.

Lai uzstādītu lauka tagus, no koka ir jāizvēlas vajadzīgais lauks un jāatver „Tags" cilne.



Attēls 4.20 **Lauka tagu uzstādīšanas logs**

Šajā cilnē tiek piedāvāts nodefinēt to veida semantiku, kas ir norādīta **AA#TagType** klases instancēs. Semantikas nosaukums tiek ņemts no **AA#TagType** klases instances *notation* atribūta.

Pēc noklusējuma tiek piedāvāti trīs semantikas veidi:

- **Axiom Annotation** – anotāciju ierakstīšana konteksta tipa kompartmentā eksportā uz Protégé redaktoru;

- **Semantics** – papildus anotāciju definīcijas, kas tiek pieliktas eksportam uz Protégé redaktoru;
- **ImportSemantics** – semantikas deklarācija, kas ļauj lietotāja definēto lauku atpazīšanu importā no Protégé;

Ja semantikas attēlošanas veids ir vienkāršs teksta lauks (*textArea*), tad katram laukam var būt tikai viena šāda veida semantikas definīcija. Ja semantikas attēlošanas veids ir daudzrindu teksta lauks (*textArea+Button*), tad katrai rindai tiek veidota sava instance.

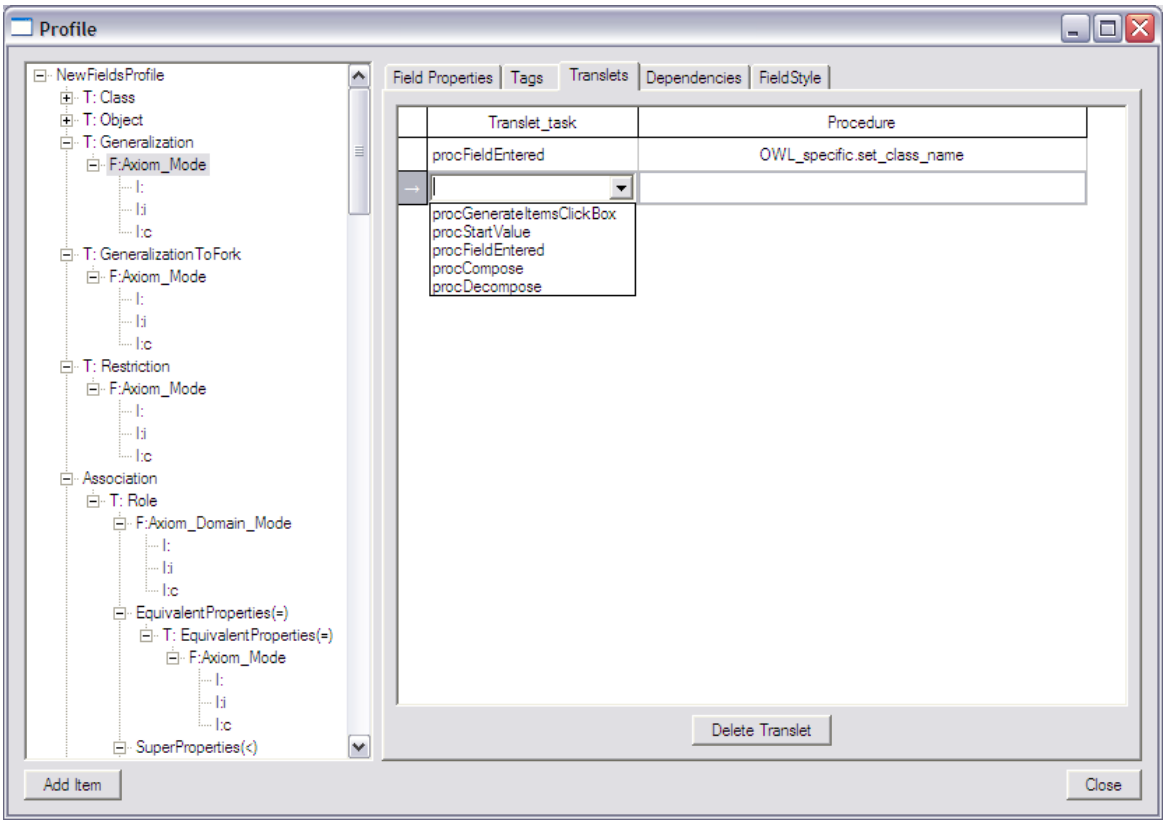
Semantiku definīcijām tiek veidotas **AA#Tag** klases instances, kur *tagKey* atribūts tiek ņemts no **AA#TagType** klases instances, kas atbilst dotajam semantikas veidam un *axiomPattern* ir lietotāju semantikas definīcija. **AA#Tag** klases instances tiek piesaistītas dotajam laukam.

Semantiku definīcijas var veidot, rediģēt un dzēst.

4.5.1.3 Lauka transletu uzstādīšana

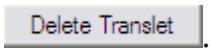
Transleti ir domāti, lai kādā noteiktā brīdī izsauktu specifisku procedūru.

Lai uzstādītu lauka transletus, no koka ir jāizvēlas vajadzīgais lauks un jāatver „Translets” cilne.



Attēls 4.21 **Lauka transletu uzstādīšanas logs**

Lai izveidotu jaunu transletu, jāizvēlas transleta tips no „Translet_task” kolonnas un jāuzraksta izsaucamās procedūras ceļš kolonnā „Procedure”. Izvēloties transleta tipu, tiek veidota jauna **AA#Translet** klases instance. Tā tiek pievienota **AA#Field** klases instancei, kas atbilst izvēlētajam laukam, un **AA#TransletTask** klases instancei, kas atbilst „Translet_task” kolonnas izvēlētajai vērtībai. Ierakstot procedūru, tā tiek ierakstīta **AA#Translet** klases instances *procedure* atribūtā.

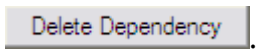
Lai izdzēstu transletu, tabulā ir jāiezīmē rinda, kas satur vajadzīgo transletu un jānospiež poga 

. Nospiežot pogu tiek dzēsta **AA#Translet** klases instance, kas atbilst dzēšamajam transletam.

4.5.1.4 Lauka atkarību uzstādīšana

Atkarības ir domātas, lai norādītu, kas dotā lauka attēlošanā ir atkarīgs no kāda cita lietotāja definēta lauka, kas ir atzīmēts kā stereotips. Par stereotipu drīkst būt tādi lietotāja definēti lauki, kas satur vismaz vienu izvēles vienumu.

Izvēloties stereotipa lauku un izvēles vienumu, no kā būs atkarīgs tekošā lauka atspoguļošanas veids, tiek veidota **AA#Dependency** klases instance, kas tiek sasaistīta ar doto lauku un izvēlēto vienumu.

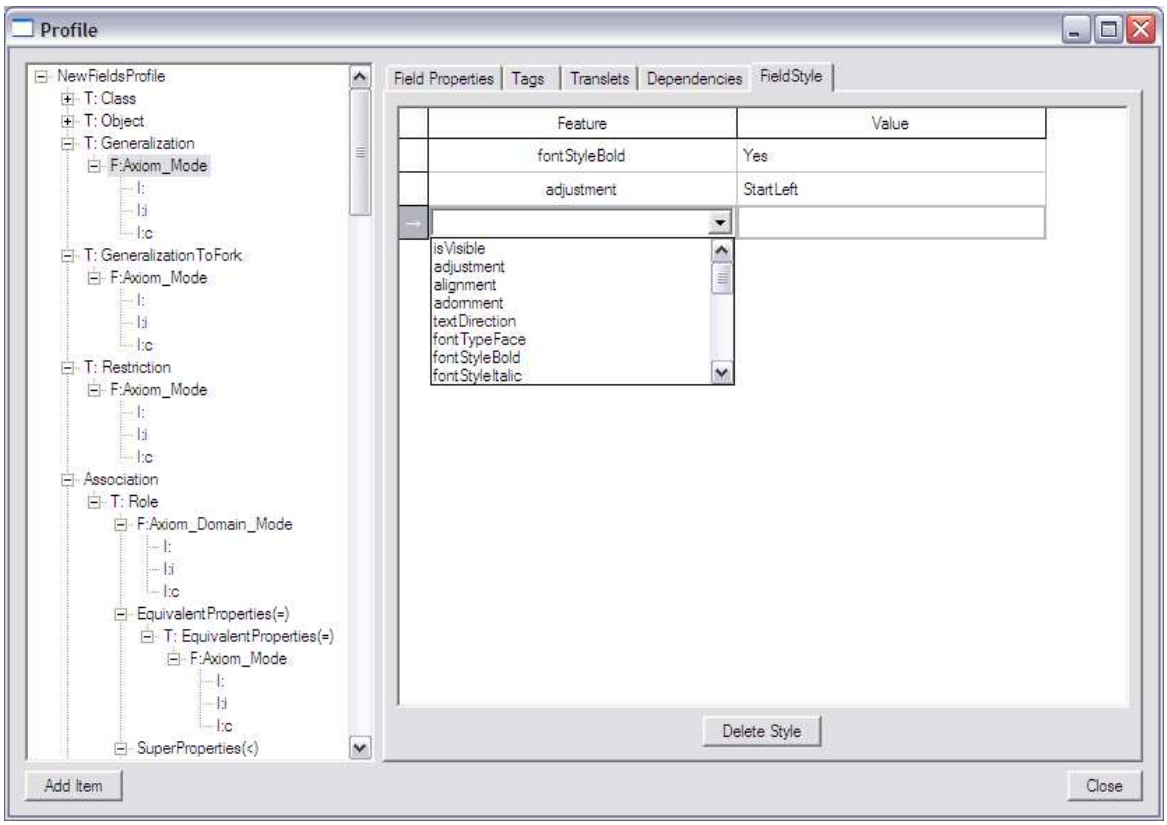
Atkarības var tikt dzēstas. Tas ir paveicams, izvēloties vajadzīgo atkarību un nospiežot pogu 

. Nospiežot pogu tiek dzēsta **AA#Dependency** klases instance, kas atbilst dotajai atkarībai.

4.5.1.5 Lauka stilu uzstādīšana

Stili ir domāti, lai grafiski izdalītu laukus.

Lai uzstādītu stilus, no koka ir jāizvēlas vajadzīgais lauks un jāatver „FieldStyle” cilne.



Attēls 4.22 **Lauka stilu uzstādīšanas logs**

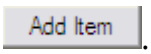
Lai izveidotu jaunu stilu, jāizvēlas stila komponente no „Feature" kolonnas. Izvēloties komponenti tiek izveidota **AA#FieldStyleSetting** klases instance, kas tiek piesaistīta **AA#Field** klases instancei, kas atbilst dotajam laukam, un **AA#CompartStyleItem** klases instancei, kurai *itemName* atribūta vērtība sakrīt ar izvēlēto komponenti no „Feature" kolonnas. Tālāk jāizvēlas no saraksta vai jāieraksta stila komponentes vērtība kolonnā „Value". Izvēloties vērtību, tā tiek ierakstīta **AA#FieldStyleSetting** klases instances *value* atribūtā. Pēc vērtības ierakstīšanas pastāv iespēja vērtību nomainīt uz citu.

Lai izdzēstu stilu, tabulā ir jāiezīmē rinda, kas satur vajadzīgo stilu, un jānospiež poga Delete Style. Nospiežot pogu tiek dzēsta **AA#FieldStyleSetting** klases instance, kas atbilst dzēšamajam stilam.

4.5.2 Izvēles vienumu izveide

Ja lietotāja veidotajam laukam attēlošanas veids ir uzlikts uz „CheckBox", „ComboBox" vai „ListBox", laukam var tikt veidoti izvēles vienumi (*ChoiceItem*).

Izvēles vienumi ir paredzēti, lai definētu lauka iepriekš definētas vērtības.

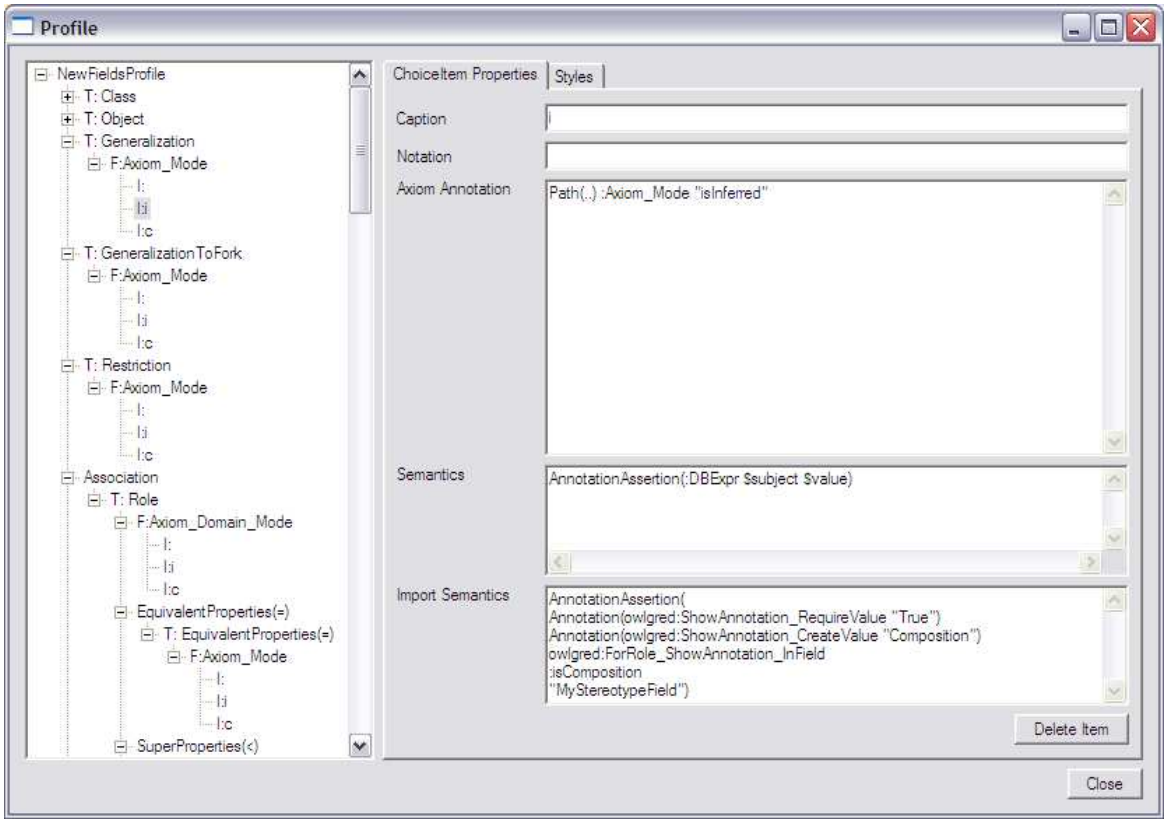
Lai izveidotu lauka izvēles vienumu, no koka ir jāizvēlas vajadzīgais lauks un jānospiež poga 

. Nospiežot pogu tiek izveidota **AA#ChoiceItem** klases instance, kas atbildīs jaunajam izvēles vienumam.

Tiek atvērta forma ar izvēles vienuma iespējamajiem uzstādījumiem. Tiek piedāvāts uzstādīt izvēles vienuma īpašības un no izvēles vienuma atkarīgus stilus.

4.5.2.1 Izvēles vienuma īpašību rediģēšana

Lai rediģētu izvēles vienuma īpašības, no koka ir jāizvēlas vajadzīgais izvēles vienums un jāatver „ChoiceItem Properties" cilne.



Attēls 4.23 **Izvēles vienuma īpašību rediģēšanas logs**

Šeit tiek piedāvāts ierakstīt izvēles vienuma nosaukumu (*Caption*), izvēles vienuma notāciju – vērtību, kas tiks atspoguļota grafiskajā diagrammā (*Notation*). Ja notācija netiek ierakstīta, tad atspoguļojas izvēles vienuma nosaukums.

Tiek piedāvāts arī nodefinēt tā veida semantiku, kas ir norādīta **AA#TagType** klases instancēs. Semantikas nosaukums tiek ņemts no **AA#TagType** klases instances *notation* atribūta.

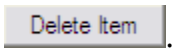
Pēc noklusējuma tiek piedāvāti trīs semantikas veidi:

- **Axiom Annotation** – anotāciju ierakstīšana konteksta tipa kompartmenta eksportā uz Protégé redaktoru;
- **Semantics** – papildus anotāciju definīcijas, kas tiek pieliktas eksportam uz Protégé redaktoru;
- **ImportSemantics** – semantikas deklarācija, kas ļauj lietotāja definēto lauku atpazīšanu importā no Protégé redaktora.

Ja semantikas attēlošanas veids ir vienkāršs teksta lauks (*textArea*), tad katram laukam var būt tikai viena šāda veida semantikas definīcija. Ja semantikas attēlošanas veids ir daudzrindu teksta lauks (*textArea+Button*), tad katrai rindai tiek veidota sava instance.

Semantiku definīcijām tiek veidotas **AA#Tag** klases instances, kur *tagKey* atribūts tiek ņemts no **AA#TagType** klases instances, kas atbilst dotajam semantikas veidam un *axiomPattern* atribūts ir lietotāju semantikas definīcija. **AA#Tag** klases instances tiek piesaistītas dotajām izvēles vienumam.

Semantiku definīcijas var veidot, rediģēt un dzēst.

Lai izdzēstu izvēles vienumu, jānospiež poga 

. Nospiežot pogu tiek izvadīts logs ar visiem atkarīgajiem uzstādījumiem, kas ir piesaistīti dzēšamam izvēles vienumam un lūgumu apstiprināt vienuma dzēšanu.

Ja tiek nospiesta 

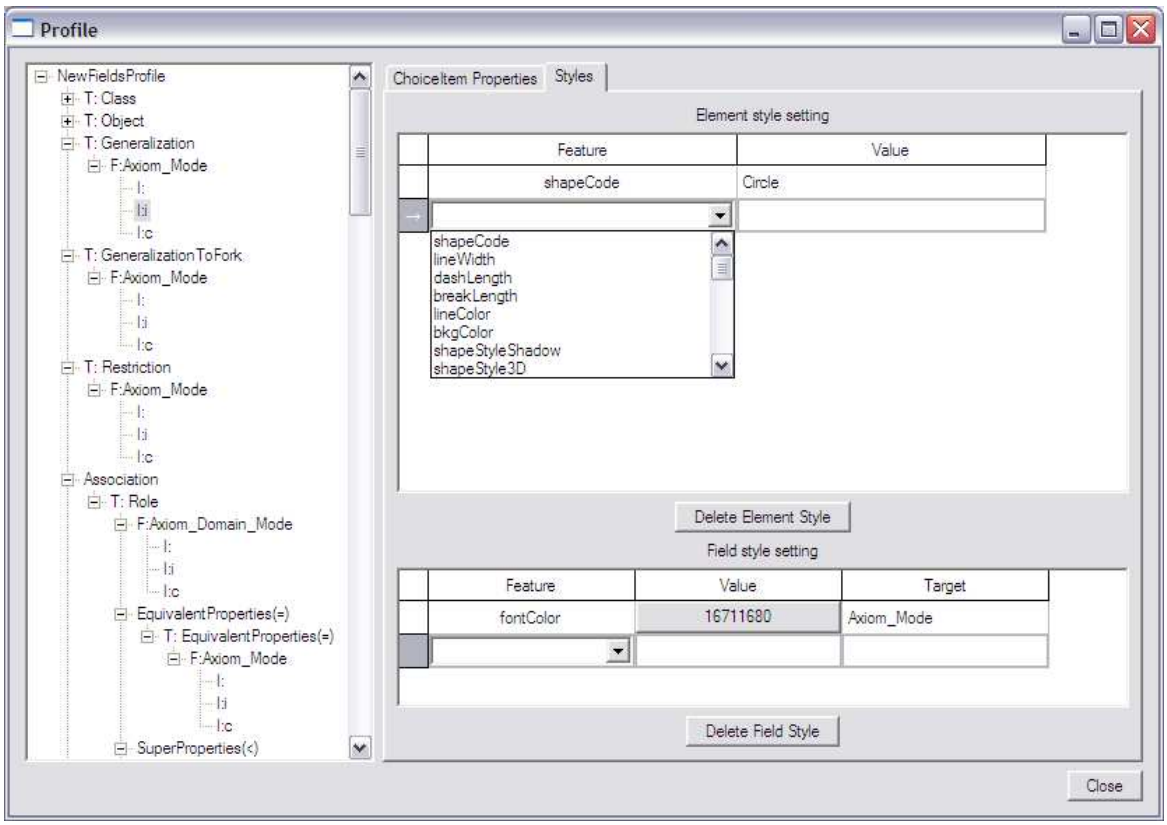
poga, tad izvēles vienums netiek dzēsts. Ja tiek nospiesta 

poga, tad izvēles vienums ar visiem tam piesaistītajiem parametriem tiek dzēsts no lietotāja definēto lauku koka. Tiek dzēsta **AA#ChoiceItem** klases instance, kas atbilst dzēšamajam izvēles vienumam, kā arī tiek dzēstas visas pārējās instances, kas ir saistītas ar doto instanci. Tiek dzēstas šādas klašu instances: AA#Tag, AA#FieldStyleSetting, AA#Dependency, ja tie ir saistīti ar izvēles vienuma instanci.

4.5.2.2 No izvēles vienuma atkarīgu stilu uzstādīšana

Lai uzstādītu no izvēles vienuma atkarīgus stilus, no koka ir jāizvēlas vajadzīgais izvēles vienums un jāatver „Styles” cilne.

Tiek piedāvāts uzstādīt atkarīgus kompartmenta vai elementa tipu stilus. Stils tiek uzstādīts tam elementa tipam, zem kura ir izvēlētais lietotāja definētais lauks vai arī tam kompartmenta tipam, kas atrodas tajā pašā līmenī kā izvēlētais lietotāja definētais lauks.



Attēls 4.24 **No izvēles vienuma atkarīgu stilu uzstādīšanas logs**

Lai izveidotu stilu elementa tipam „Element style setting" tabulā, kolonnā „Feature" jāizvēlas stila komponente. Tiek piedāvātas tās stila komponentes, kas var tikt uzstādītas konkrētajam elementa tipam. Izvēloties stila komponenti tiek veidota **AA#FieldStyleSetting** klases instance ar *isElementStyleSetting* atribūta vērtību vienādu ar „true" (pazīme, ka stils ir pieliekams elementa tipam). Instance tiek piesaistīta **AA#ElemStyleItem** klases instancei, kurai *itemName* atribūta vērtība sakrīt ar „Feature" kolonnā izvēlēto stila komponentes nosaukumu un **AA#ChoiceItem** klases instances, kas atbilst dotajam izvēles vienumam. Tālāk kolonnā „Value" tiek piedāvāts izvēlēties no saraksta vai ierakstīt stila komponentes vērtību. Vērtība tiek ierakstīta **AA#FieldStyleSetting** klases instances *value* atribūtā. Pēc vērtības ierakstīšanas pastāv iespēja vērtību nomainīt uz citu.

Lai izdzēstu stilu, tabulā ir jāiezīmē rinda, kas satur vajadzīgo stilu un jānospiež poga Delete Element Style. Nospiežot pogu, tiek dzēsta **AA#FieldStyleSetting** klases instance, kas atbilst dzēšamajam stilam.

Lai izveidotu stilu kompartmenta tipam „Field style setting" tabulā, kolonnā „Feature" jāizvēlas stila komponente. Tiek piedāvātas tās stila komponentes, kas var tikt uzstādītas konkrētajam kompartmenta tipam. Izvēloties stila komponenti, tiek veidota **AA#FieldStyleSetting** klases instance ar *isElementStyleSetting* atribūta vērtību vienādu ar

„false” (pazīme, ka stils ir pieliekams kompartmenta tipam). Instance tiek piesaistīta **AA#CompartStyleItem** klases instancei, kurai *itemName* atribūta vērtība sakrīt ar „Feature” kolonnā izvēlēto stila komponentes nosaukumu un **AA#ChoiceItem** klases instances, kas atbilst dotajam izvēles vienumam. Tālāk kolonnā „Value” tiek piedāvāts izvēlēties no saraksta vai ierakstīt stila komponentes vērtību. Vērtība tiek ierakstīta **AA#FieldStyleSetting** klases instances *value* atribūtā. Pēc vērtības ierakstīšanas pastāv iespēja vērtību nomainīt uz citu. Tālāk kolonnā „Target” tiek piedāvāts izvēlēties mērķa kompartmenta tipu, kuram tiks uzstādīts stils. Mērķa kompartmenta tipi ir tie kompartmenti, kas atrodas ar izvēlēto lietotāja definētu lauku vienā līmenī (ir kopīgs vecāks kompartmenta tips vai elementa tips). Izvēloties mērķa kompartmenta tipu, tā nosaukums tiek ierakstīts **AA#FieldStyleSetting** klases instances *target* atribūtā.

Lai izdzēstu stilu, tabulā ir jāiezīmē rinda, kas satur vajadzīgo stilu un jānospiež poga 

. Nospiežot pogu, tiek dzēsta **AA#FieldStyleSetting** klases instance, kas atbilst dzēšamajam stilam.

Izveidotie stili tiek pielietoti brīdī, kad lietotāja definētājā laukā tiek uzstādīts izvēles vienums, no kura ir atkarīgs stils. Mainot izvēles vienumu, stili tiek pārrēķināti attiecīgi jaunajam izvēles vienumam.

4.5.3 Apakšlauku izveide

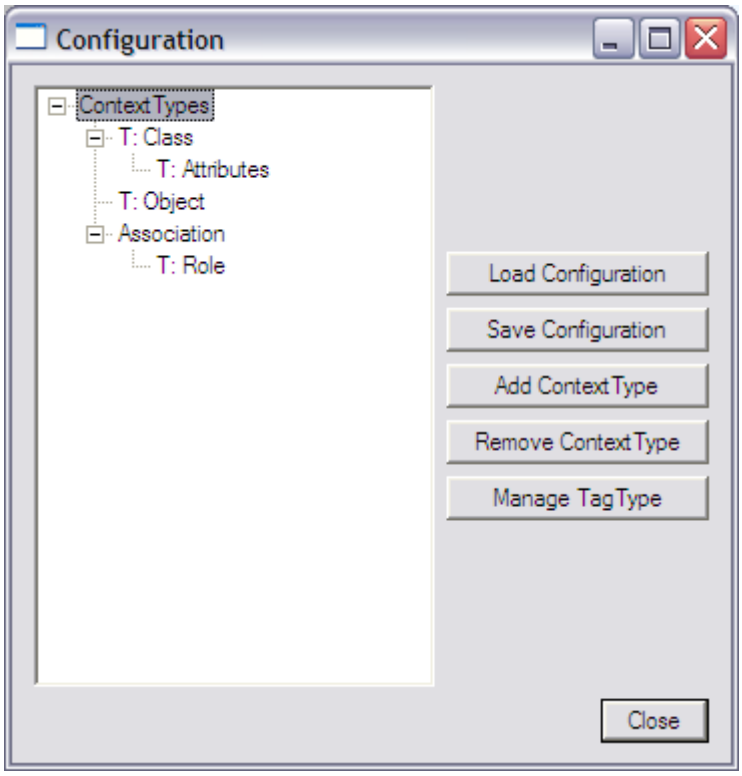
Ja lietotāja veidotajam laukam attēlošanas veids ir uzlikts uz „InputField+Button” vai „TextArea+Button”, laukam var tikt veidoti apakšlauki.

Apakšlaukiem var tikt veiktas visas tās darbības, kas pirmā līmeņa laukiem, izņemot stila uzstādījumus, kas apakšlaukiem netiek paredzēti.

4.6 Konfigurācijas pārvaldības logs

Lietotāju definētu lauku mehānismā lietotājiem tiek piedāvātas iespējas norādīt, kādu konfigurāciju ir jālieto konkrētajā projektā.

Pastāv iespējas nodefinēt konteksta tipus (vietas, zem kurām var tikt izveidoti jauni lauki) un tagu tipus, kas tiks lietoti projektā.



Attēls 4.25 **Konfigurācijas pārvaldības logs**

Piekļūt konfigurāciju pārvaldības logam var no projekta diagrammas rīku joslas, nospiežot © pogu.

Pēc pogas nospiešanas tiek atvērts logs, kur labajā pusē ir redzams koks ar visiem konteksta tipiem, kuriem šobrīd ir atļauts veidot laukus. Pēc noklusējuma var veidot jaunus laukus zem klases (class), klases atribūta (attribute), objekta (object) un asociācijas abos galos (role).

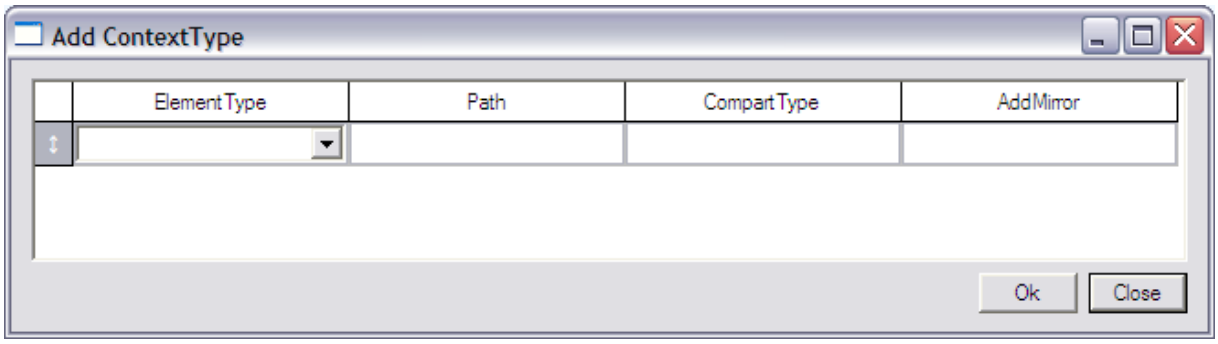
Kokā visi elementi, zem kuriem var tikt veidoti jauni lauki, grafiski tiek atzīmēti ar „**T:** ” prefiksu. Koka lapas, kurām nav „**T:** ” prefiksa, ir palīgelementi, zem kuriem laukus veidot nevar un kas atrodas kokā tikai tāpēc, lai atspoguļotu ceļu no elementa tipa līdz konteksta tipam.

Šajā logā tiek piedāvātas tādas darbības kā:

- Ielādēt konfigurāciju no teksta faila;
- Saglabāt konfigurāciju teksta failā;
- Pievienot konteksta tipu;
- Noņemt konteksta tipu;
- Pārvaldīt tagu tipus.

4.6.1 Konteksta tipu pievienošana

Lai pievienotu konteksta tipu no konfigurāciju pārvaldības logā, jānospiež Add ContextType poga. Nospiežot pogu, tiek atvērta tabulas veida forma, kur ir iespējams izvēlēties vajadzīgo konteksta tipu.

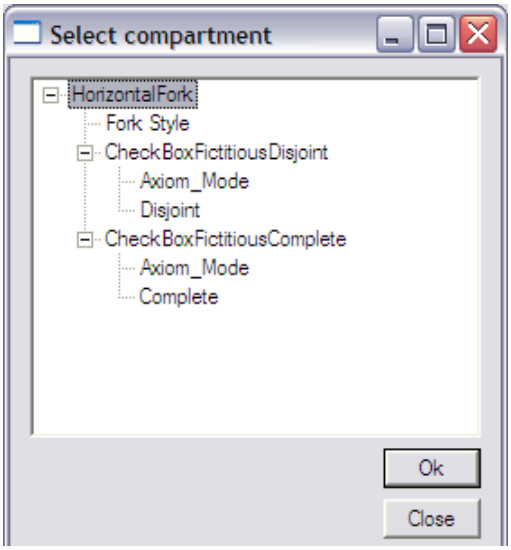


Attēls 4.26 **Konteksta tipu pievienošanas logs**

Šeit kā konteksta tipi var tikt izvēlēti gan elementa tipi, gan kompartmenta tipi.

Ja ir nepieciešams pievienot konteksta tipiem elementa tipu, tad no tabulas kolonnas „ElementType" ir jāizvēlas vajadzīgais elementa tips. Izvēloties elementa tipu, tiek veidota **AA#ContextType** klases instance ar šādām atribūtu vērtībām: *type* vienāds ar izvēlēto elementa tipa nosaukumu, *mode* vienāds ar „Element", kas nozīmē, ka konteksta tips ir no ElemType klases.

Lai pievienotu konteksta tipu, kas ir no **CompartType** klases, vispirms ir jāizvēlas elementa tips, zem kura ir vajadzīgais kompartmenta tips un tad „CompartType" kolonnā tiek parādīta poga set CompartType. Nospiežot pogu, tiek atvērts koks ar visu kompartmentu tipu struktūru, kas ir zem dotā elementa tipa.



Attēls 4.27 **Koks ar kompartmentu tipu struktūru**

Pēc kompartmenta tipa izvēles, **AA#ContextType** klases instances *elTypeName* atribūtā tiek ierakstīts elementa tipa nosaukums, *type* – izvēlētā kompartmenta tipa nosaukums, *path* – ceļš no elementa tipa līdz kompartmenta tipam, *mode* – viena no turpmāk nosauktajām vērtībām: „Group”, ja izvēlētais kompartmenta tips ir atzīmēts kā grupa; „Group Item” – ja pats kompartmenta tips ir stilizējams, bet apakš kompartmenti nav; „Text” – ja kompartmenta tips nav stilizējams.

Tabulas veida formā pastāv iespēja norādīt, ka kompartmenta tipam ir apgriezts kompartmenta tips, kurš arī ir jāiekļauj konteksta tipā. To var izdarīt, iezīmējot izvēles rūtiņu kolonnā „AddMirror”.

4.6.2 Konteksta tipu noņemšana

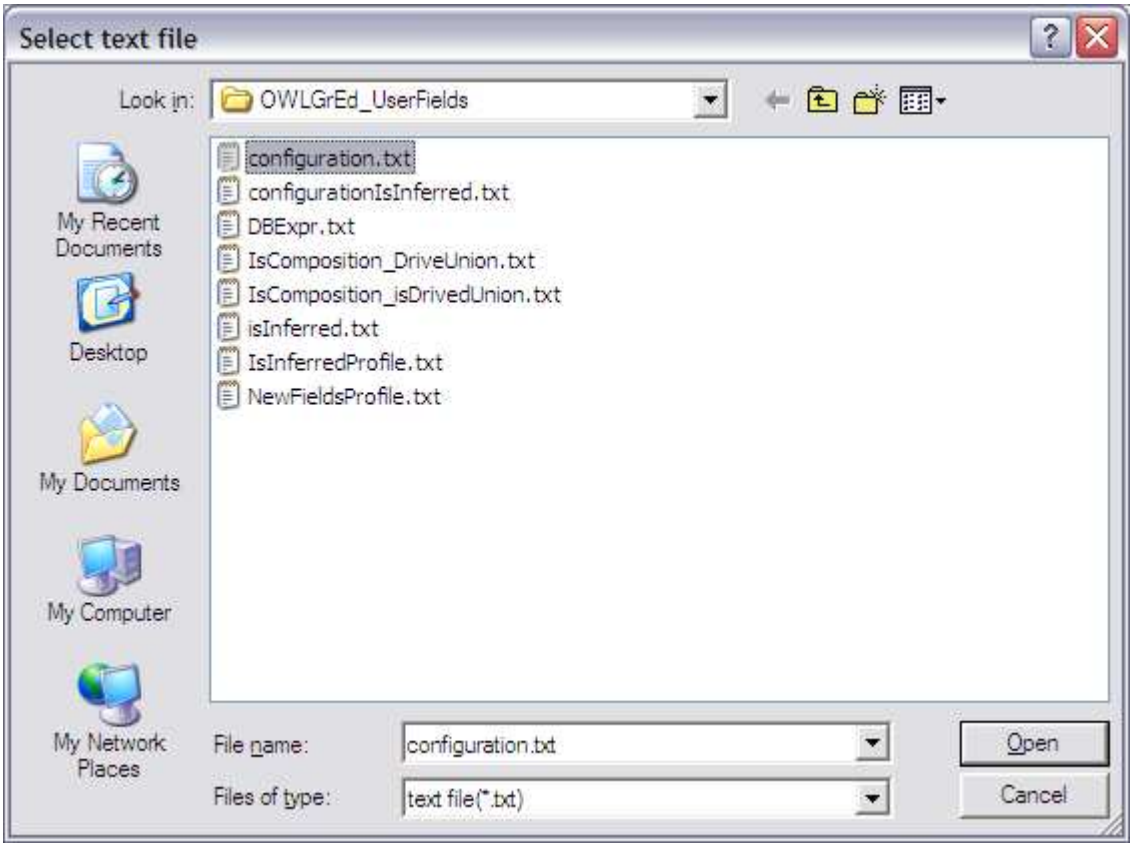
Lai noņemtu konteksta tipu no projekta, jāizvēlas vajadzīgais konteksta tips no konfigurācijas pārvaldības koka un jānospiež poga Remove ContextType. Nospiežot pogu tiek pārbaudīts, vai kādā profilā no projekta zem izvēlētā konteksta tipa nav izveidots kāds lietotāja definēts lauks. Ja tāds lauks ir izveidots, tad šo konteksta tipu neatļauj dzēst. Tiek rādīts paziņojums:

Attēls 4.28 **Ziņojums, ka konteksta tipu nedrīkst dzēst**

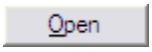
Ja zem konteksta tipa nav izveidots lietotāja definēts lauks, tad tas tiek dzēsts no kontekstu tipu koka un tiek dzēsta **AA#ContextType** klases instance, kas atbilst dotajam konteksta tipam.

4.6.3 Konfigurācijas ielādēšana no teksta faila

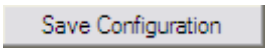
Lai ielādētu konfigurāciju no teksta faila, konfigurācijas pārvaldības logā jānospiež Load Configuration poga. Tālāk tiek atvērts failu meklēšanas logs un piedāvāts izvēlēties teksta failu, kas satur iekodēto informāciju par ielādēto konfigurāciju.



Attēls 4.29 **Konfigurācijas failu meklēšanas logs**

Pēc  pogas nospiešanas tiek veikta kontekstu tipu un tagu tipu ielādēšana. Tiek veiktas pārbaudes, vai kāds konteksta vai tagu tips nav ielādēts divas reizes. Ja ir, tad dublikāti tiek aizvākti.

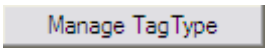
4.6.4 Konfigurāciju saglabāšana teksta failā

Lai saglabātu konfigurāciju teksta failā, konfigurāciju pārvaldības logā jānospiež  poga. Tālāk tiek atvērts logs ar iespēju izvēlēties konfigurācijas eksporta teksta faila izveidošanas vietu.

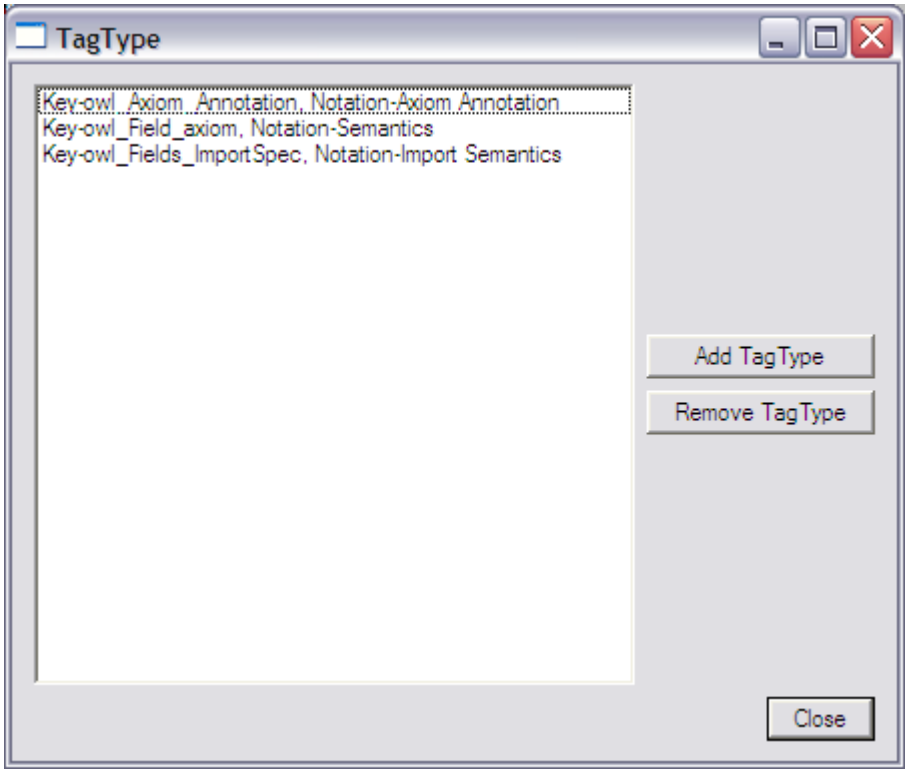
Pēc noklusējuma tiek piedāvāts saglabāt konfigurāciju tajā pašā mapē, kur ir visi lietotāja definēto lauku mehānisma izejas koda faili.

Konfigurācijas eksporta faila nosaukums tiek veidots šādi: **configuration_[month]_[date]_[year]_[hour]_[minute]_[second]**, lai izslēgtu iespēju izveidot vairākus teksta failus ar vienādiem nosaukumiem.

4.6.5 Tagu tipu pārvaldīšana

Lai piekļūtu tagu tipu pārvaldības logiem, konfigurāciju pārvaldības logā jānospiež 

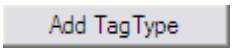
poga. Nospiežot pogu, tiek atvērts logs ar tagu tipu pārvaldības iespējām.



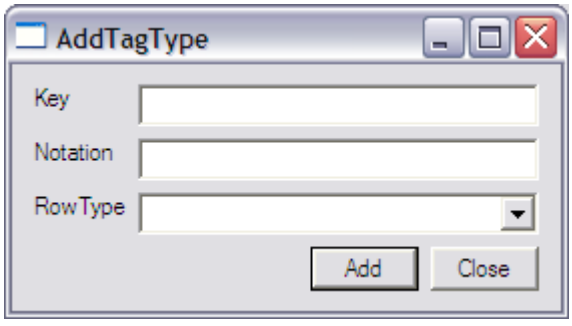
Attēls 4.30 **Tagu tipu pārvaldīšanas logs**

Šajā logā tiek attēloti visi tagu tipi, kas pašlaik ir piesaistīti pie projekta konfigurācijas. No tagu tipu pārvaldības loga tiek piedāvātas šādas iespējas:

- Pievienot tagu tipu;
- Noņemt tagu tipu.

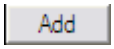
Lai pievienotu jaunu tagu tipu projekta konfigurācijai, no tagu tipu pārvaldības loga ir jānospiež 

poga. Nospiežot pogu, tiek atvērta jauna forma tagu tipa informācijas ievadīšanai.

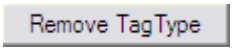


Attēls 4.31 **Jaunu tagu tipu pievienošanas logs**

Tiek piedāvāts ievadīt sekojošas vērtības: *Key* – Semantikas veids; *Notation* – semantikas nosaukums, kas redzams lietotājam profila logos; *RowType* – semantikas lauka tips, kas tiek atspoguļots lietotājām.

Uz 

pogas nospiešanu, tiek izveidota jauna **AA#TagType** klases instance, kas tiek piesaistīta projekta konfigurācijai.

Lai noņemtu tagu tipu no projekta konfigurācijas, tagu tipu pārvaldības logā no saraksta ir jāizvēlas vajadzīgais tagu tips un jānospiež 

poga. Uz pogas nospiešanu tiek izdzēsta **AA#TagType** klases instance, kas atbilda dzēšamam tagu tipam.

4.7 Lietotāja definētu lauku mehānisma uzstādīšana un noņemšana

Lietotāja definētu lauku mehānisma uzstādīšana ir ļoti vienkārša.

Lai uzstādītu mehānismu vienā projektā, ir jāiekopē „OWLGrEd_UserFields” mape, kas satur visu, projekta mapē „Plugins”, piemēram, OWLGrEd\Projects\NewProject\Plugins.

Lai uzstādītu mehānismu visā OWLGrEd rīkā, zem OWLGrEd\Tools\OWLGrEd ir jāizveido mape „Plugins” un jāievieto tajā „OWLGrEd_UserFields” mape.

Lietotāja definētu lauku mehānisma uzstādīšanas laikā projektā tiek izveidota **Extension** klases instance ar tipu „Plugin” (*type=Plugin*), kas atbilst visam mehānismam un zem kuras tiek piekārtoti visi projektā atrodamie profili.

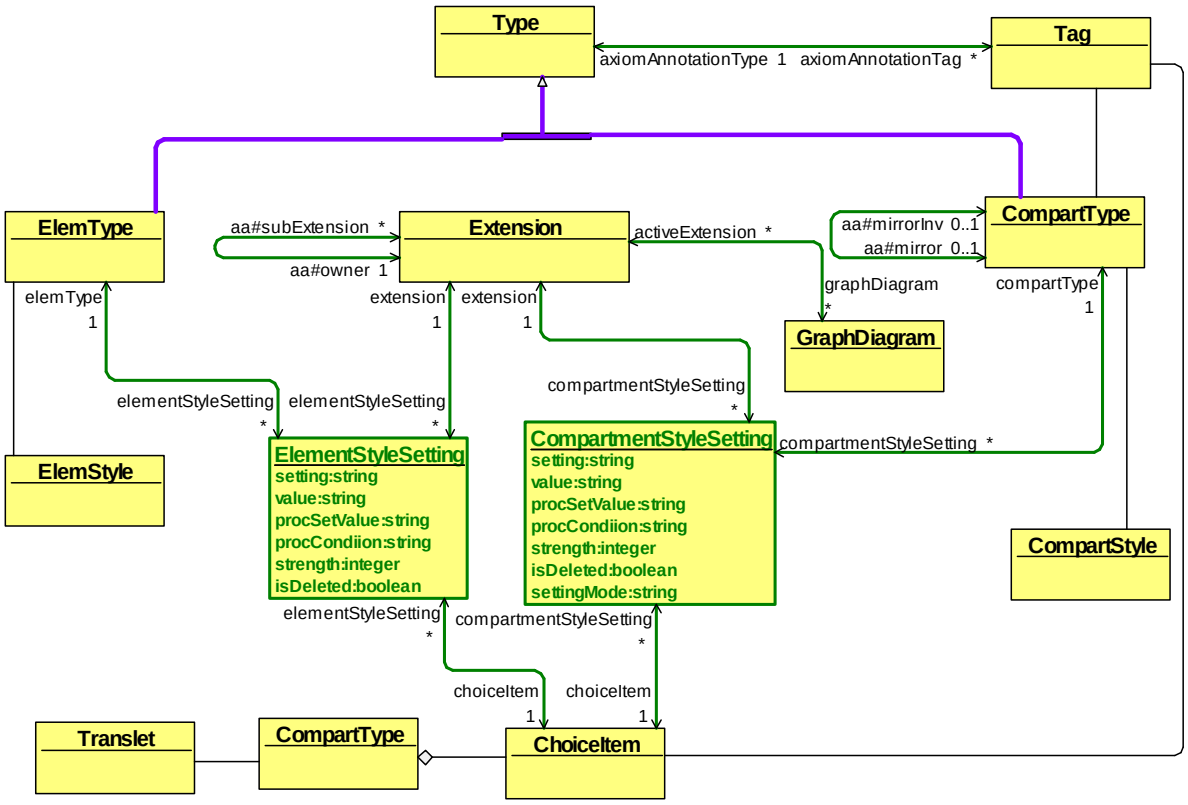
Lietotāja definētu lauku mehānisma noņemšana ir tikpat vienkārša kā tā uzstādīšana. Viss, kas ir jāizdara, jāizdzēš mape „OWLGrEd_UserFields” un visas papildus īpašības, kas tika pievienotas ar mehānisma palīdzību, tiks dzēstas.

5 Lietotāja definētu lauku mehānisma izpildlaika daļas apraksts

5.1 Tipu daļas metamodeļa papildinājumi

Lai realizētu lietotāja definētu lauku mehānismu, bija nepieciešams papildināt tipu daļas metamodeli ([10], skatīt nodaļu 1.3) ar jauniem elementiem.

Papildinājumi bija nepieciešami arī, lai nodrošināt stilu uzstādīšanu atsevišķu stila komponenšu līmenī, dažādu stila uzstādīšanas mehānismu (choice item, view) sadarbību, atļautu paslēpt apakš kompartmentus tekstuālos kompartmentos un atļautu pielikt kompartmentu prefiksus un sufiksus.



Attēls 5.1 **Tipu daļas metamodeļa papildinājumi**

Lai varētu saglabāt tipu stilu un prefiksu/sufiksu uzstādījumus atkarībā no lietotāja definēta lauka vērtības vai arī skatījuma pielietojumā diagrammai, tika izveidotas divas klases **ElementStyleSetting** un **CompartmentStyleSetting**.

ElementStyleSetting klase satur visus nosacījuma stilus, kas tiek piekārtoti elementiem.

Tabula 5.1 **ElementStyleSetting klases atribūti**

Atribūts	Vērtības tips	Apraksts
setting	String	Stila nosaukums.
value	String	Stila vērtība.
procSetValue	String	Procedūras nosaukums, kas izrēķina stila vērtību.
procCondition	String	Procedūras nosaukums, kas noskaidro, vai ir realizēti visi nosacījumi, lai stils tiktu uzlikts.
strength	Integer	Stila uzstādīšanas stiprums. Skatījumu stiliem tiek piekārtots stiprums 3, izvēles vienuma stiliem – 5, skatījuma un izvēles vienuma stiliem – 10.
isDeleted	Boolean	Pazīme, ka stils ir nodzēsts, bet vēl nav noņemts

		no tipiem. Tiek lietots, ja profila vai skatījuma pārvaldības logā tika dzēsts kāds stila vienums.

No ElementStyleSetting klases tiek novilktas saites uz **ElemType**, **Extension**, **ChoiceItem** klasēm. Ja no ElementStyleSetting klases instances ir saite uz ChoiceItem klases instanci, tad stils ir atkarīgs no kāda izvēles vienuma. Ja no ElementStyleSetting klases instances ir saite uz Extension klases instanci, tad stils ir atkarīgs no skatījuma pielietošanas diagrammai, kurā tas atrodas. Ja no ElementStyleSetting ir saites uz abām šo klašu instancēm, tad stils ir atkarīgs gan no izvēles vienuma, gan no skatījuma pielietošanas diagrammai.

CompartmentStyleSetting klase satur visus nosacījuma stilus un prefiksus/sufiksus, kas tiek piekārtoti kompartmentiem.

Tabula 5.2 **CompartmentStyleSetting klases atribūti**

Atribūts	Vērtības tips	Apraksts
setting	String	Stila nosaukums.
value	String	Stila vērtība.
procSetValue	String	Procedūras nosaukums, kas izrēķina stila vērtību.
procCondition	String	Procedūras nosaukums, kas noskaidro, vai ir realizēti visi nosacījumi, lai stils tiktu uzlikts.
strength	Integer	Stila uzstādīšanas stiprums. Skatījumu stiliem tiek piekārtots stiprums 3, izvēles vienuma stiliem – 5, skatījuma un izvēles vienuma stiliem – 10.
isDeleted	Boolean	Pazīme, ka stils ir nodzēsts, bet vēl nav noņemts no tipiem. Tiek lietots, ja profila vai skatījuma pārvaldības logā tika dzēsts kāds stila vienums.
settingMode	String	Tiek lietots, ja tiek uzstādīts prefikss vai sufikss. Apzīmē vietu, kur prefikss/sufikss tiek likts. Var pieņemt vērtības: inside; outside; inPlace.

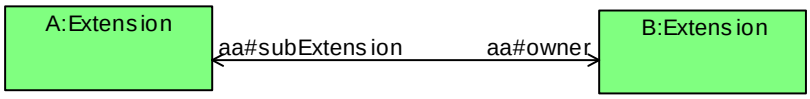
No **CompartmentStyleSetting** klases tiek novilktas saites uz **CompartType**, **Extension**, **ChoiceItem** klasēm. Ja no **CompartmentStyleSetting** klases instances ir saite uz **ChoiceItem** klases instanci, tad stils ir atkarīgs no kāda izvēles vienuma. Ja no

CompartmentStyleSetting klases instances ir saite uz **Extension** klases instanci, tad stils ir atkarīgs no skatījuma pielietošanas diagrammai, kurā tas atrodas. Ja no **CompartmentStyleSetting** ir saites uz abām šo klašu instancēm, tad stils ir atkarīgs gan no izvēles vienuma, gan no skatījuma pielietošanas diagrammai.

Stila vienumi tiek piekārtoti katram elementam vai kompartmentam atsevišķi. Lai atceltu kādu stila vienuma uzstādīšanu, vienkārši jāizdzēš ElementStyleSetting vai CompartmentStyleSetting klases instance, kas atbilst stila vienumam.

Lai nodrošinātu apgriezto kompartmenta tipa (piemēram, asociāciju gadījumā pretēja gala *InvRole*) atrašanu, tika izveidota saite no **CompartType** klases pašu uz sevi ar *aa#mirror* lomas vārdu vienā galā un *aa#mirrorInv* otrā.

Lai nodrošinātu profilu un skatījumu hierarhiju, tika izveidota saite no **Extension** klases pašu uz sevi ar *aa#owner* lomas vārdu vienā galā un aa#*subExtension* otrā. Ja no vienas Extension klases instances „A" ir saite uz citu Extension klases instanci „B", tas nozīmē, ka „A" instancei ir pakārtota „B" instance.



Attēls 5.2 **Extension klases attiecība**

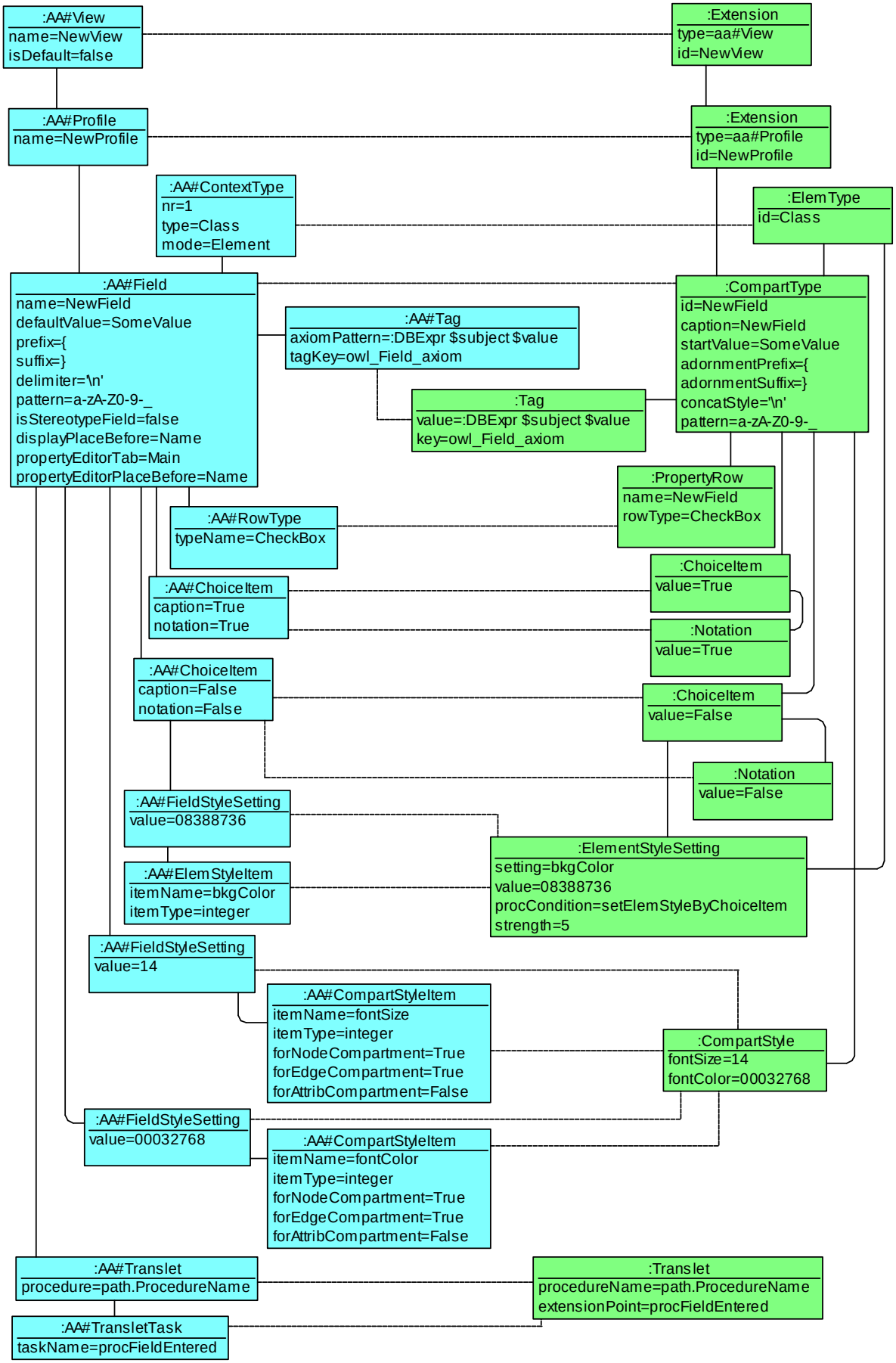
Lai nodrošinātu skatījumu pielietošanu diagrammai, tika izveidota saite no **Extension** klases uz **GraphDiagram** klasi, ar *aa#activeExtension* lomas vārdu pie Extension un *graphGiagram* pie GraphDiagram. Viens skatījums var būt pielietots vairākām diagrammām un vienai diagrammai var būt pielietoti pēc patikas daudz skatījumi.

Lai varētu saglabāt lietotāja definēto lauku semantiku, tika izveidota saite no **Type** klases uz **Tag** klasi, ar *axiomAnnotationType* lomas vārdu pie Type un *axiomAnnotatiomTag* pie Tag, kur Type instance ir tā, kurā līdz šim glabājās konteksta tipa semantika, Tag instance ir jauna lauka semantikas definīcija.

5.2 Transformācija no lietotāja definēto lauku mehānisma metamodeļa uz tipu daļu

Šajā nodaļā tiek aprakstīts, par kādām instancēm no tipu daļas metamodeļa tiek pārvērstas lietotāja definēto lauku mehānisma metamodeļa daļas instances.

Tiek būtiski izmantota Extension klase, kura formāli netiek izmantota universālajā transformācijā, un kurā tiek likti elementi gan pats lietotāju definēto lauku mehānisms, gan profili, gan profila skatijūmi.



Attēls 5.3 **Transformācija no lietotāja definēto lauku mehānisma metamodeļa uz tipu daļu**

AA#Profile klases instances tiek pārvērstas par **Extension** klases instancēm ar tipu „aa#Profile” (*type=aa#Profile*), kur AA#Profile klases instances nosaukums (*name=NewProfile*) pārvēršas par Extension klases instances identifikatoru (*id=NewProfile*).

AA#Field klases instances tiek pārvērstas par **CompartType** klases instancēm, kur:

Lauka nosaukums (*name=NewField*)	pārvēršas par	Identifikatoru un virsrakstu (*id=NewField, caption=NewField*)
Noklusētā vērtība (*defaultValue=SomeValue*)		Sākuma vērtību (*startValue=SomeValue*)
Prefikss (*prefix={*)		Prefiksu (*adornmentPrefix={*)
Sufikss (*suffix=}*)		Sufiksu (*adornmentSuffix=}*)
Atdalītais (*delimiter='\n'*)		Salikšanas stilu (*concatStyle='\n'*)
Šablons (*pattern=a-zA-Z0-9-_*)		Šablonu (*pattern=a-zA-Z0-9-_*)

AA#ChoiceItem klases instances tiek pārvērstas par **ChoiceItem** un **Notation** klašu instancēm, kur AA#ChoiceItem klases instances virsraksts (*caption=True*) pārvēršas par ChoiceItem klases instanci ar attiecīgu vērtību (*value=True*). AA#ChoiceItem klases instances notācija (*notation=True*) pārvēršas par Notation klases instanci ar attiecīgu vērtību (*value=True*).

AA#RowType klases instances tiek pārvērstas par **PropertyRow** klases instancēm, kur AA#RowType klases instances tipa nosaukums (*typeName=CheckBox*) pārvēršas par PropertyRow klases instances rindas tipu (*rowType=CheckBox*). PropertyRow klases instances nosaukums (*name=NewField*) ir vienāds ar AA#Field klases instances, kurai ir piekārtots rindas tips, nosaukumu.

AA#Tag klases instances tiek pārvērstas par **Tag** klases instancēm, kur:

Aksiomas šablons (*axiomPattern= DBExpr $subject $value*)	pārvēršas par	Vērtību (*value=:DBExpr $subject $value*)
Taga atslēga (*tagKey=owl_Field_axiom*)		Atslēgu (*key=owl_Field_axiom*)

AA#Translet un **AA#TransletTask** klašu instances tiek pārvērstas par **Translet** klases instanci, kur AA#Translet klases instances procedūra (*procedure=path.ProcedureName*) pārvēršas par Translet klases instances procedūras nosaukumu

(*procedureName=path.ProcedureName*). AA#TransletTask klases instances, kurai ir piesaistīta dota AA#Translet instance, uzdevuma nosaukums (*taskName=procFieldEntered*) tiek pārvērsts par Translet klases instances pagarinājuma punktu (*extensionPoint=procFieldEntered*).

AA#ContextType klases instances tiek pārvērstas par saiti starp **CompartType** vai **ElemType** klases instanci, kas atbilst AA#ContextType klases instances tipam (*type=Class*) un CompartType instanci, kas atbilst AA#Field klases instancei.

AA#View klases instances tiek pārvērstas par **Extension** klases instancēm ar tipu „aa#View" (type=aa#View), kur AA#View klases instances nosaukums (name=NewView) pārvēršas par Extension klases instances identifikatoru (id=NewView).

Stilu transformācija atšķiras gadījumos, kad stila instance atbilst lietotāju definēta lauka stilam, vai kad stila instance atbilst CompartType vai ElemType klases instancēm un uzstādās, ja tiek izpildīti noteiktie nosacījumi.

Ja tiek transformētas lietotāja definēta lauka stila instances, tad visas **AA#StyleSetting** un **AA#CompartStyleItem** klašu instances tiek pārvērstas par **CompartStyle** klases instances atribūtu vērtībām. AA#CompartStyleItem klases instances vienības nosaukums (*itemName=fontColor*) un AA#StyleSetting klases instances vērtība (*value = 00032768*) pārvēršas par CompartStyle klases instances atribūtu *fontColor* ar vērtību *00032768* (*fontColor=00032768*)

Ja tiek transformētas stilu instances, kas atbilst CompartType vai ElemType klases instancēm un uzstādās, ja tiek izpildīti noteiktie nosacījumi, tad visas **AA#StyleSetting** un **AA#CompartStyleItem** klašu instances tiek pārvērstas par **CompartmentStyleSetting** klases instancēm, visas **AA#StyleSetting** un **AA#ElemStyleItem** klašu instances tiek pārvērstas par **ElementStyleSetting** klases instancēm.

Vienības nosaukums (*itemName=bkgColor*)	pārvēršas par	Uzstādījumu (*setting= bkgColor*)
Vērtība (*value=08388736*)		Vērtību (*value=08388736*)

5.3 Jaunu notāciju semantikas eksports un imports no Protégé

Lai nodrošinātu jaunu notāciju savietojamību ar Protégé, bija jāievieš specifiski semantikas veidi.

Pašreiz tiek piedāvāti trīs semantikas veidi:

- **Axiom Annotation** – semantika jaunu notāciju pielikšanai aksiomu definīcijai eksporta laikā uz Protégé;
- **Semantics** – semantika papildus anotāciju definīcijai eksporta laikā uz Protégé;
- **ImportSemantics** – semantikas deklarācija, kas atļauj jaunu notāciju atpazīšanu importā no Protégé;

5.3.1 Axiom Annotation

Anotācijas aksiomas tiek izmantotas, lai nodefinētu integritātes ierobežojumus, kas sīkāk tiek aprakstīti nodaļā 6.1.4.

Semantika tiek pierakstīta šādi:

Path(Name/Role) (*ObjectSomeValuesFrom*)<- : isInferred $value, vai

Path(..) : isInferred „true”,

kur *Path(..)* ir ceļš no lauka vai elementa, zem kura tiek pielikts lietotāja definētais lauks, līdz tam laukam, zem kura atrodas translets semantikas savākšanai priekš eksporta uz Protégé. **(*ObjectSomeValuesFrom*)<-** ir neobligāta vērtība, kas var tikt izmantota, lai noteiktu kādam anotācijas tipam tiks pielikts ierobežojums, ja ir iespējami vairāki anotāciju tipi. **Axiom_Mode** ir anotācijas īpašības nosaukums, **$value** ir pazīme, ka īpašības vērtība ir jāņem no lauka, kas definē jaunu notāciju. **$value** var tikt aizvietota ar jebkuru lietotāja definētu simbolu virkni.

Ja *Restriction* elementam tiek pielikts jauns lauks *Axiom_Mode* ar **i** un **c** izvēles vienumiem un zem **i** vienuma tiks nodefinēta šāda semantika **Path(Name/Role) :isInferred „true”**, tad eksporta laikā, ja *Axiom_Mode* lauka vērtība būs vienāda ar **i**, *Restriction* elementam atbilstošas aksiomas izskatīsies šādi:

SubClassOf(Annotation(:isInferred "true") :ClassA ObjectSomeValuesFrom(:role :ClassB))

Declaration(Annotation(:isInferred "true") ObjectProperty(:role))

Aksiomām tika pielikta papildus anotācija *Annotation(:isInferred "true").*

Ja semantikas definīcijā tiktu norādīts, ka ierobežojumu anotācijas ir liekamas tikai pie *ObjectSomeValuesFrom* aksiomām (**Path(Name/Role) (*ObjectSomeValuesFrom*)<- :isInferred „true”**), tad *ObjectProperty* aksioma nesaturētu ierobežojumu anotācijas:

SubClassOf(Annotation(:isInferred "true") :ClassA ObjectSomeValuesFrom(:role :ClassB))
Declaration(ObjectProperty(:role))

5.3.2 Semantics

Šī veida semantika tiek izmantota, lai papildus notācijas atspoguļotu kā anotācijas definīcijas. Šis semantikas veids ir pieliekams klasēm, atribūtiem, objektiem un asociāciju galiem.

Semantika tiek pierakstīta šādi:

AnnotationAssertion(:DBExpr $subject $value)

kur **AnnotationAssertion** ir pieliekamas aksiomas veids, **DBExpr** ir pieliekamas anotācijas tips, **$subject** ir lauka vai elementa vērtība, **$value** ir lauka vērtība, kas definē jaunu notāciju. **$value** var tikt aizvietota ar jebkuru lietotāja definētu simbolu virkni.

Lai, piemēram, nodefinētu **isComposition UML** notācijas semantiku, kas tiek sīkāk aprakstīta nodaļā 6.1.1, ir jāuzraksta šāda semantikas definīcija: **AnnotationAssertion(:isComposition $subject "true")**, kas eksporta laikā pārvēršas par *AnnotationAssertion(:isComposition :roleForClass "true").*

Datu bāzes savienojamības semantikas no 6.1.3 nodaļas izskatīsies šādi: **AnnotationAssertion(:DBExpr $subject $value)**, kas eksporta laikā pārvēršas par *AnnotationAssertion(:DBExpr :AcademicProgram "XProgram").*

5.3.3 ImportSemantics

Dotā semantika ir domāta, lai importa laikā no Protégé varētu atpazīt to notāciju aksiomas, kas tika pievienotas ar *Semantics* semantikas veida palīdzību. Šis semantikas veids ir pieliekams klasēm, atribūtiem, objektiem un asociāciju galiem.

Semantika tiek pierakstīta šādi:

- Tipiski pie laukiem pierakstāma:

  AnnotationAssertion(
  owlFields:ForClass_ShowAnnotation_InField
  namespace:DBExpr
  "DBExpr")

kur **namespace** un **owlFields** ir iepriekš definētās vārdu telpas, **owlFields:ForClass_ShowAnnotation_InField** ir anotāciju īpašība, kas norāda, zem kā atrodas lauks, kurā jāieraksta semantikas vērtība. **ForClass_ShowAnnotation_InField** –

lauks ir zem klases, **ForRole_ShowAnnotation_InField** – lauks ir zem asociācijas gala, **ForAttr_ShowAnnotation_InField** – lauks ir zem atribūta, **ForIndiv_ShowAnnotation_InField** – lauks ir zem objekta. **DBExpr** ir anotācijas veids. **"DBExpr"** ir lauka nosaukums, kurā ir jāieraksta semantika.

- Tipiski pie izvēles vienuma pierakstāma:

  AnnotationAssertion(
  Annotation(owlFields:ShowAnnotation_RequireValue "True")
  Annotation(owlFields:ShowAnnotation_CreateValue "Composition")
  owlFields:ForRole_ShowAnnotation_InField
  namespace:isComposition
  "MyStereotypeField")

kur **owlFields:ShowAnnotation_RequireValue** norāda, ka anotācija attēlojama konkrētajā laukā tikai ja tajā ir konkrēta vērtība, **owlFields:ShowAnnotation_CreateValue**, norāda, kāda vērtība ir jāuzliek mērķa vērtības laukā.

5.4 Stilu uzstādīšana

Kad tiek iedarbināts stilu uzstādīšanas mehānisms, tiek savākti visi elementi un kompartmenti, kam ir jāuzstāda stils. Stils tiek rēķināts katram elementam un kompartmentam atsevišķi.

Ja stils tiek uzstādīts elementam, tad tiek savāktas visas **ElementStyleSetting** klases instances, kas ir piesaistītas pie dotā elementa tipa instances. Katram stila vienumam tiek noskaidrots, vai dotajam elementam izpildās visi nosacījumi, lai uzstādītu stilu. Nosacījumi var būs šādi:

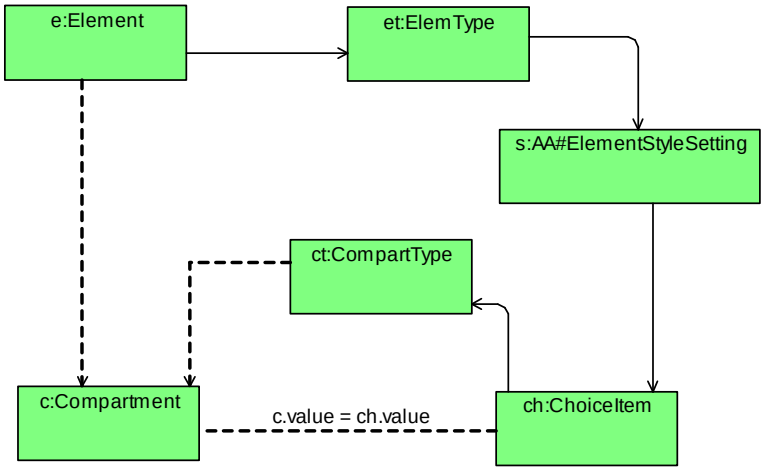
1. Attiecīgajā lietotāju definētā laukā tiek uzstādīta vērtība, kas sakrīt ar **ChoiceItem** klases instances vērtību, kam ir piesaistīta stila instance;
2. Diagrammai, kurā atrodas elements, tiek pielikts skatījums, kas atbilst **Extension** klases instancei, kas ir piesaistīta stila instancei, vai arī **Extension** klases instancei, kas atbilst noklusētajam stilam;
3. Ja ir jāizpildās (1) un (2) nosacījumiem.

Lai pārbaudītu (1) nosacījuma izpildi elementam, ir jāpalaiž transformācija **setElemStyleBySetting**, kurai tiek padots noteikts elements un konkrēta stila vienuma instance. Transformācija no stila vienuma atrod:

- izvēles vienumu (ChoiceItem);
- kompartmenta tipu, kuram ir pakārtots atrastais izvēles vienums;

- kompartmentu, kas ir ar atrasto tipu un atrodas zem dotā elementa.

Lai nosacījums izpildītos, atrastā kompartmenta vērtībai ir jāsakrīt ar izvēles vienuma vērtību.

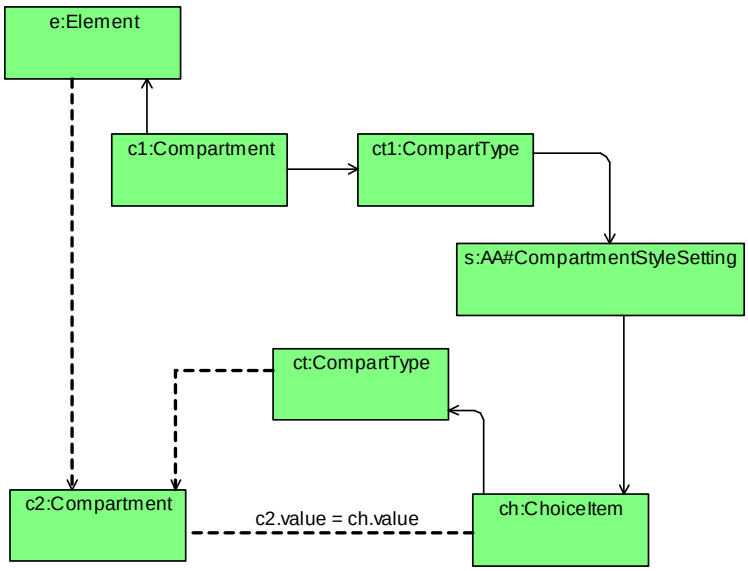


Attēls 5.4 **setElemStyleBySetting transformācijas instanču diagramma**

Kompartmentu stila uzstādīšanas nosacījumu pārbaude ir līdzīga. Transformācijai **setCompartStyleByChoiceItem** tiek padots kompartments un stila instance. Transformācija no stila vienuma atrod:

- izvēles vienumu (ChoiceItem);
- kompartmenta tipu, kuram ir pakārtots atrastais izvēles vienums;
- kompartmentu, kas ir ar atrasto tipu un atrodas zem tā paša elementa kā padotais kompartments.

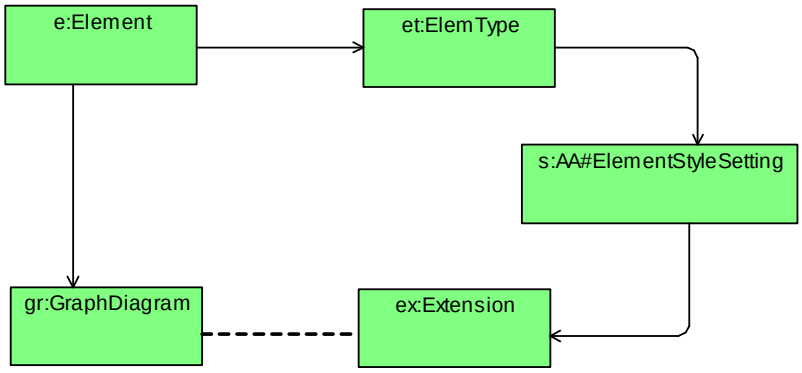
Lai nosacījums izpildītos, atrastā kompartmenta vērtībai ir jāsakrīt ar izvēles vienuma vērtību.



Attēls 5.5 **setCompartStyleBySetting transformācijas instanču diagramma**

Lai pārbaudītu (2) nosacījuma izpildi (diagrammai, kurā atrodas elements, tiek pielikts skatījums), elementam ir jāpalaiž transformācija **setElemStyleByExtension**, kurai tiek padots noteikts elements un konkrēta stila vienuma instance. Transformācija no stila vienuma atrod skatījumu, kuram ir piekārtots stila vienums, un no elementa atrod diagrammu, kurā atrodas elements.

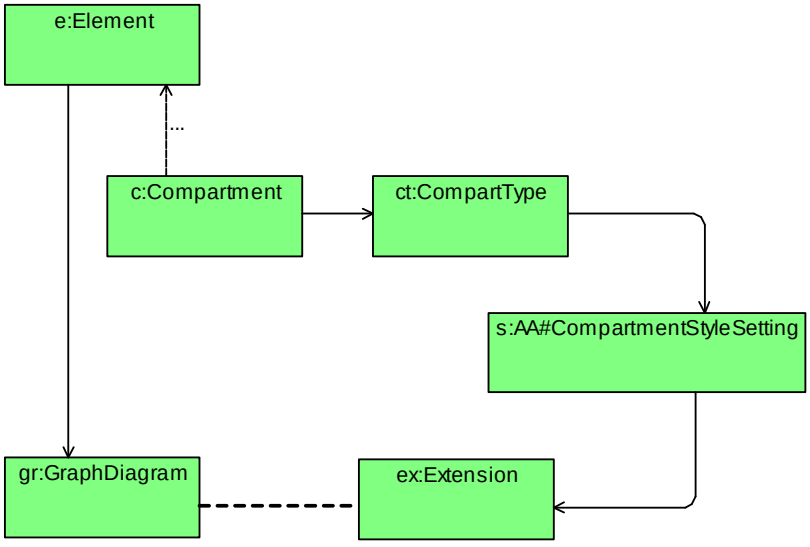
Lai nosacījums izpildītos, ir jābūt novilktai saitei starp atrasto diagrammu un skatījumu.



Attēls 5.6 **setElemStyleByExtension transformācijas instanču diagramma**

Kompartmentu stila uzstādīšanas nosacījumu pārbaude ir līdzīga. Transformācijai **setCompartStyleByExtension** tiek padots kompartments un stila instance. Transformācija no stila vienuma atrod skatījumu, kuram ir piekārtots stila vienums, no kompartmenta elementu, zem kura ir kompartments. Un no elementa atrod diagrammu, kurā atrodas elements.

Lai nosacījums izpildītos, ir jābūt novilktai saitei starp atrasto diagrammu un skatījumu.



Attēls 5.7 **setCompartStyleByExtension transformācijas instanču diagramma**

Visi stila vienumi, kuriem izpildās nepieciešamie nosacījumi, tiek sakārtoti pēc to prioritātēm.

Pirms jauna stila uzstādīšanas ir svarīgi saglabāt tos stila uzstādījumus, kas dotajam elementam jau tika piekārtoti. Lai to izdarītu, no elementa *style* atribūta tiek nolasīts un atkodēts pašreizējais elementa stils. Visas atkodētās stila vērtības tiek ierakstītas jaunā **ElemStyle** klases instancē, un tālāk nepieciešamie stili tiek aizstāti ar jaunām vērtībām. Beigās stils tiek piekārtots elementam.

Kompartmentiem stila uzstādīšana notiek līdzīgi, tikai tiek ņemtas instances no **CompartmentStyleSetting** klases.

Citādi tiek apstrādātas tās **CompartmentStyleSetting** klases instances, kas satur prefiksu vai sufiksu uzstādījumus. Šeit tieši tāpat tiek noskaidrots, vai izpildās visi nosacījumi prefiksa/sufiksa pielikšanai. Prefikss/sufikss tiek pielikts kompartmenta vērtībai. Pastāv dažādas prefiksa/sufiksa pielikšanas vietas:

- prefix-inside (pieliek prefiksu pēc jau pastāvošajiem prefiksiem);
- prefix-ouside (pieliek prefiksu pirms jau pastāvošajiem prefiksiem);
- prefix-inPlace (pieliek prefiksu jau pastāvošo prefiksu vietā);
- suffix-inside (pieliek sufiksu pirms jau pastāvošajiem sufiksiem);
- suffix-ouside (pieliek sufiksu pēc jau pastāvošajiem sufiksiem);
- suffix-inPlace (pieliek sufiksu jau pastāvošo sufiksu vietā).

Pēc prefiksu/sufiksu pielikšanas tiek pārrēķināta kompartmenta vērtība, kā arī to kompartmentu vērtības, zem kuriem atrodas dotais kompartments.

6 Lietojumi un piemēri

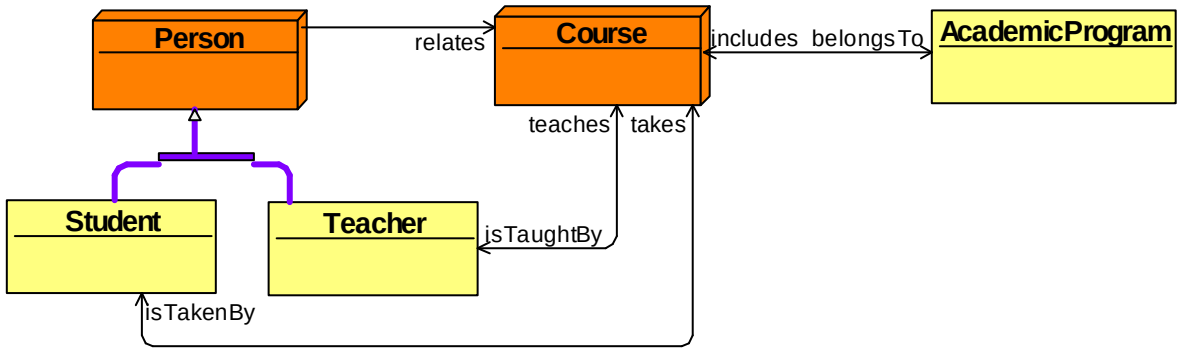
6.1 Iespējamie lietojumi lietotāja definētu lauku mehānismam

Iespējamie lietojumi lietotāja definētu lauku mehānismam ir:

- Anotāciju semantikas definēšana papildus UML konstrukcijām, tādām kā kompozīcija (*composition*) vai atvasināto apvienojumu īpašība (*derived union*);
- Datu bāzes savienojamības anotāciju definēšana klasēm, objektiem un datu īpašībām;
- Integritātes ierobežojumu notāciju definēšana.

6.1.1 Svarīgo klāšu izcelšana

Var gadīties situācijas, kad no ontoloģijas ir jāizceļ kādas svarīgas klases.



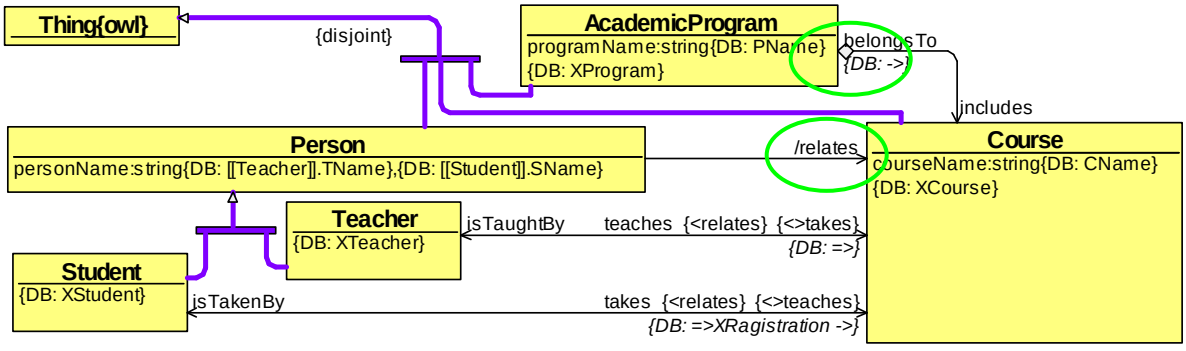
Attēls 6.1 **Svarīgo klašu izcelšanas piemērs**

Lai to izdarītu ar lietotāja definēto lauku mehānisma palīdzību, klases tipam ir jāizveido jauna izvēlas rūtiņa „isImportant”, kur zem „true” izvēles vienuma var piemēram nodefinēt klašu kastīšu krāsu padarīt par oranžu un formu par 3D.

6.1.2 Anotāciju semantikas definēšana papildus UML konstrukcijām

Kompozīcijas (*composition*) nozīmē to, ka objekts tiek veidots no citiem objektiem, tas ir, objekti tiek iekļauti citā objektā – konteinerā, un eksistē tikai kamēr eksistē konteiners. Kompozīcija parasti tiek attēlota ar rombu (dimantu) līnijas galā pie konteinera klases.

Atvasināta apvienojumu īpašība (*derived union*) ir abstrakta īpašība un tiek lietota, lai apvienotu vienu vai vairākas kolekcijas. Atvasināta apvienojumu īpašība parasti tiek attēlota, pieliekot „/” simbolu pirms kolekcijas nosaukuma.



Attēls 6.2 **Papildus UML konstrukciju attēlošanas piemērs**

Kompozīcijas vai atvasināto apvienojumu īpašības notācijas nav paredzētas OWL sintaksē.

Lai pieliktu kompozīcijas vai atvasināto apvienojumu īpašības notācijas ar lietotāja definētu lauku mehānismu, zem asociāciju galiem jāizveido divi jauni lauki: *isComposition* un

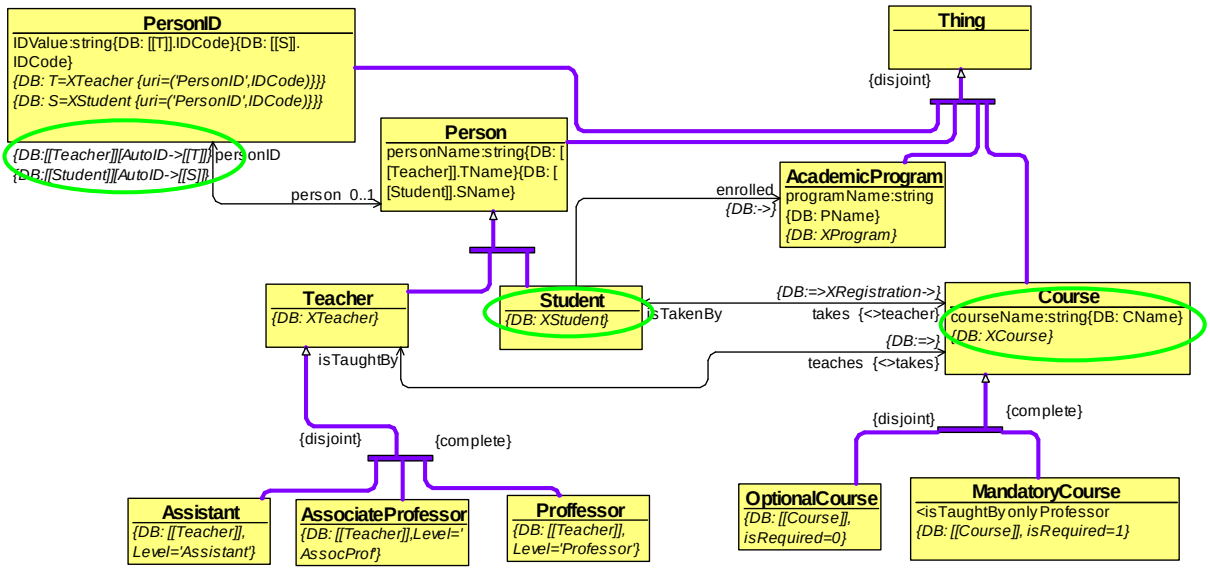
isDerivedUnion, ar *true* un *false* vērtībām. Iestājoties *true* vērtībai, *isComposition* laukā asociācijas pretējā galā jāpieliek rombs (dimants), un uz *isDerivedUnion* lauka *true* vērtības jāpieliek „/” prefikss asociācijas gala nosaukumam.

Ir jādefinē arī semantika notāciju eksportam un Protégé. Semantikas definēšana notiek pēc nodaļās 5.3.2 un 5.3.3 aprakstītajiem principiem. Dotajam piemēram eksporta laikā tiek pieliktas divas papildus anotācijas:

AnnotationAssertion(:isComposition:belongsTo "true")
AnnotationAssertion(:isDerivedUnion :relates "true")

6.1.3 Datu bāzes savienojamības anotāciju definēšana

Ja OWL ontoloģijas veidā tiek attēlota relāciju datu bāzē ietvertās informācijas konceptuālā struktūra, tad dabiski ir OWL ontoloģiju papildināt ar anotācijām, kas ontoloģijas entītijām norada atbilstošos datu bāzes objektus (tabulas, laukus, saites). Viens no šādiem anotācijas valodām ir RBD2OWL [22], šeit ilustrējam specifisko notāciju RBD2OWL anotāciju attēlošanai.



Attēls 6.3 **Datu bāzes savienojamības anotāciju attēlošanas piemērs**

Ir jānodefinē jauni **DBExpr** daudzrindu lauki ar prefiksu „{DB: ” un sufiksu „}” zem Klases, Atribūta un Asociāciju galiem. Lauku saturs zem klasēm un asociāciju galiem tiek grafiski izdalīti ar slīpraksta palīdzību.

Ir jādefinē arī semantika notāciju eksportam uz Protégé. Semantikas definēšana notiek pēc nodaļās 5.3.2 un 5.3.3 aprakstītajiem principiem. Dotajam piemēram eksporta laikā tiek pieliktas šādas papildus anotācijas:

AnnotationAssertion(owlFields:DBExpr MiniUniversity:AcademicProgram "XProgram")
AnnotationAssertion(owlFields:DBExpr MiniUniversity:Assistant "[[Teacher]],Level='Assistant'")

AnnotationAssertion(owlFields:DBExpr MiniUniversity:AssociateProfessor "[[Teacher]],Level='AssocProf'")
AnnotationAssertion(owlFields:DBExpr MiniUniversity:MandatoryCourse "[[Course]], isRequired=1")
AnnotationAssertion(owlFields:DBExpr MiniUniversity:OptionalCourse "[[Course]],isRequired=0")
AnnotationAssertion(owlFields:DBExpr MiniUniversity:PersonID "S=XStudent {uri=('PersonID',IDCode)}}")
AnnotationAssertion(owlFields:DBExpr MiniUniversity:PersonID "T=XTeacher {uri=('PersonID',IDCode)}}")
AnnotationAssertion(owlFields:DBExpr MiniUniversity:Proffessor "[[Teacher]],Level='Professor'")
AnnotationAssertion(owlFields:DBExpr MiniUniversity:Student "XStudent")
AnnotationAssertion(owlFields:DBExpr MiniUniversity:Teacher "XTeacher")
AnnotationAssertion(owlFields:DBExpr MiniUniversity:enrolled "->")
AnnotationAssertion(owlFields:DBExpr MiniUniversity:personID "[[Student]][AutoID->[[S]]")
AnnotationAssertion(owlFields:DBExpr MiniUniversity:personID "[[Teacher]][AutoID->[[T]]")
AnnotationAssertion(owlFields:DBExpr MiniUniversity:takes "=>XRegistration->")
AnnotationAssertion(owlFields:DBExpr MiniUniversity:teaches "=>")
AnnotationAssertion(owlFields:DBExpr MiniUniversity:IDValue "[[T]].IDCode")
AnnotationAssertion(owlFields:DBExpr MiniUniversity:IDValue "[[S]].IDCode")
AnnotationAssertion(owlFields:DBExpr MiniUniversity:courseName "CName")
AnnotationAssertion(owlFields:DBExpr MiniUniversity:personName "[[Teacher]].TName")
AnnotationAssertion(owlFields:DBExpr MiniUniversity:personName "[[Student]].SName")
AnnotationAssertion(owlFields:DBExpr MiniUniversity:programName "PName")

6.1.4 Integritātes ierobežojumu notāciju definēšana

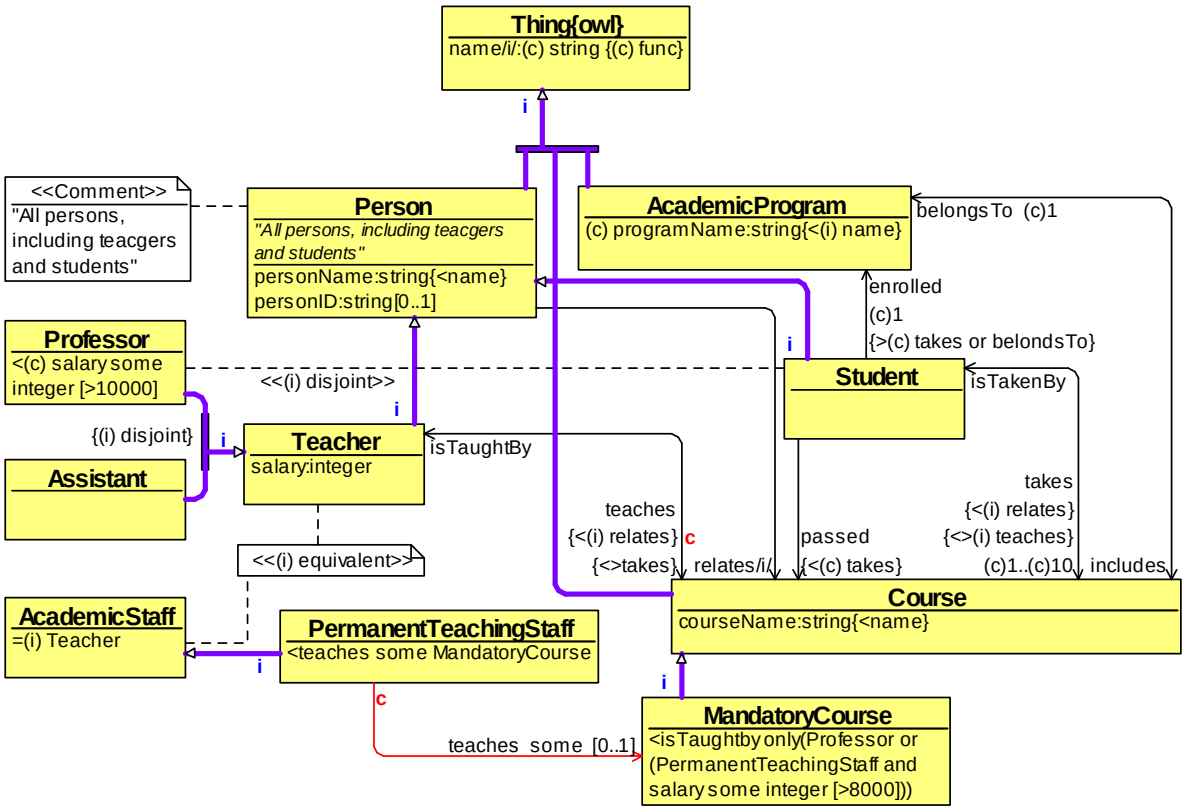
StarDog OWL/RDF DBMS [23] piedāvā jaunu OWL lietojumu – kā shēmas valodu RDF datu bāzēm. Tas var tikt izdarīts, sadalot OWL aksiomas atvērtas un aizvērtas pasaules pieņēmumos.

Atvērtas pasaules pieņēmumi nozīmē, ka, ja kādas izteiksmes vērtība nav norādīta, tad mēs pieņēmam, ka izteiksmes vērtība eksistē, bet mums nav zināma. Ar aizvērtas pasaules pieņēmumiem, ja kādas izteiksmes vērtība nav norādīta, tad šī izteiksme ir aplama.

Tradicionāli OWL lieto atvērtas pasaules semantiku, bet pēdējos gados arvien biežāk rodas situācijas, kad atsevišķās situācijās ir jānorāda integritātes ierobežojumi. Piemēram, informāciju sistēmas specifikācijā, ja personai netiek norādīts tālruņa numurs, situācija, kad tiek pieņemts, ka numurs eksistē, bet nav zināms un patiesībā tas nav norādīts, var izraisīt kļūdas.

Ar lietotāja definētu lauku mehānismu var tikt izveidotas papildus notācijas, kas atbalstīs gan atvērtas, gan aizvērtas pasaules pieņēmumu implementāciju.

Lai pieliktu integritātes ierobežojumus, visiem laukiem, kam var tikt norādīti ierobežojumi, ir jāizveido apakšlauks „Axiom-Mode” ar **i** un **c** izvēles vienumiem, kur **i** (saīsinājums no *inferred*) atbilst atvērtas pasaules pieņēmumiem un **c** (saīsinājums no *constreint*) – aizvērtas pasaules pieņēmumiem.



Attēls 6.4 **Integritātes ierobežojumu notāciju attēlošanas piemērs**

Ir jādefinē arī semantika notāciju eksportam uz Protégé. Semantikas definēšana notiek pēc nodaļā 5.3.1 aprakstīta principa. Dotajam piemēram eksporta laikā tiek veidotas šādas papildus anotācijas, kas tiek ieliktas aksiomās:

EquivalentClasses(Annotation(ic:isInferred "true") ic:AcademicStaff ic:Teacher)
SubClassOf(Annotation(ic:isInferred "true") ic:Assistant ic:Teacher)
DisjointClasses(Annotation(ic:isInferred "true") ic:Assistant ic:Professor)
SubClassOf(Annotation(ic:isInferred "true") ic:MandatoryCourse ic:Course)
SubClassOf(Annotation(ic:isInferred "true") ic:PermanentTeachingStaff ic:AcademicStaff)
SubClassOf(Annotation(ic:isConstreint "true") ic:PermanentTeachingStaff ObjectSomeValuesFrom(ic:teaches ic:MandatoryCourse))
SubClassOf(Annotation(ic:isInferred "true") ic:Professor ic:Teacher)
DisjointClasses(Annotation(ic:isInferred "true") ic:Professor ic:Assistant)
DisjointClasses(Annotation(ic:isInferred "true") ic:Professor ic:Student)
SubClassOf(Annotation(ic:isInferred "true") ic:Student ic:Person)
DisjointClasses(Annotation(ic:isInferred "true") ic:Student ic:Professor)
EquivalentClasses(Annotation(ic:isInferred "true") ic:Teacher ic:AcademicStaff)
SubClassOf(Annotation(ic:isInferred "true") ic:Teacher ic:Person)
Declaration(Annotation(ic:isInferred "true") ObjectProperty(ic:isTaughtBy))
SubObjectPropertyOf(Annotation(ic:isConstreint "true") ic:passed ic:takes)
SubObjectPropertyOf(Annotation(ic:isInferred "true") ic:takes ic:relates)
DisjointObjectProperties(Annotation(ic:isInferred "true") ic:takes ic:teaches)
Declaration(Annotation(ic:isConstreint "true") ObjectProperty(ic:teaches))
SubObjectPropertyOf(Annotation(ic:isInferred "true") ic:teaches ic:relates)
DisjointObjectProperties(Annotation(ic:isInferred "true") ic:teaches ic:takes)
Declaration(Annotation(ic:isConstreint "true") DataProperty(ic:programName))
SubDataPropertyOf(Annotation(ic:isInferred "true") ic:programName ic:name)
FunctionalDataProperty(Annotation(ic:isConstreint "true") owl:name)
DataPropertyRange(Annotation(ic:isConstreint "true") owl:name xsd:string).

6.1.5 Citi lietojumi

Kā jau tika minēts iepriekš, uz doto brīdi konfigurators spēj nodrošināt daļu no lietotāja definētu lauku mehānisma iespējām, bet konfigurators ir paredzēts, lai veidotu jaunus tipus un to atribūtus un gala lietotājiem nav paredzēts to piegādāt. Lietotāja definētu lauku mehānisms, tieši pretēji, ir paredzēts gala lietotājiem. Pirms mehānismu nodod gala lietotājam, ar spraudņa konfigurācijas mehānismu var nodefinēt, ar kādiem tipiem un atribūtiem tam ir atļauts darboties.

Kaut gan ar konfiguratora palīdzību var veidot jaunus atribūtus, tas neatbalsta jauno lauku semantikas eksportu uz Protégé. Lietotāja definētu lauku mehānismā šī problēma ir atrisināta. Lietotājs pats var noteikt, kādās situācijas kādu semantiku ir nepieciešams nosūtīt. Piemēram, sūtīt lauka semantiku, ja tam ir uzstādīta kāda noteikta vērtība.

Lietotāja definētu lauku mehānisms piedāvā smalkākās stilu uzstādīšanas iespējas. Lietotājam nav jāuzstāda visi stila vienumi, var uzstādīt tikai tos vienumus, kas ir nepieciešami, pārējie vienumi tiks nolasīti no tekošiem stila uzstādījumiem. Var uzstādīt stilus konkrētiem elementiem un atribūtiem, turklāt nav nepieciešams vērt stilu uzstādīšanas logu. Var piesaistīt stila uzstādīšanu lietotāja definēta lauka konkrētai vērtībai. Tai pašai vērtībai var piesaistīt arī citus uzstādījumus, tādus kā semantikas ģenerēšanu priekš eksporta uz Protégé. Šo iespēju var izmantot, ja, piemēram, kāda klase ir grafiski jāizdala no citām. Tad viss, kas atliek, ir piesaistīt vajadzīgos stila vienumus lauka vērtībai un vajadzīgajam elementam ierakstīt laukā šo vērtību.

Ir pieejamas arī tādas iespējas kā konkrētu atribūtu papildus prefiksu un sufiksu uzstādīšana. Prefiksi un sufiksi var tikt lietoti, lai uzlabotu diagrammās caurskatāmību, izceltu kādus konkrētus atribūtus vai apzīmētu kādu jaunu notāciju, piemēram, pie UML atvasinātas apvienojumu īpašības (*derived union*), kas tika apzīmēta ar „/” zīmi pirms nosaukuma.

Lietotāja definētu lauku mehānismā ir pieejama iespēja paslēpt atsevišķus atribūtus. Konfiguratorā ir piedāvāta tikai augšējā līmeņa atribūtu paslēpšana. Ar Lietotāja definētu lauku mehānismu var paslēpt jebkurus atribūtus neatkarīgi no tā, cik dziļi tie atrodas.

Būtiska lieta ir lietotāja definētu lauku mehānisma integrētais skatu mehānisms, ar kura palīdzību var tikt definēti elementu un atribūtu attēlošanas veidi. Turklāt katrai diagrammai var nodefinēt savu izskatu, pielietojot vienu vai vairākus lietotāju definētus skatījumus, kas satur noteiktus stila uzstādījumus. Var tikt veidoti arī noklusētie skatījumi, kas ļauj pielietot stilus jauniem un no Protégé importētiem elementiem. Piemēram, ja ontoloģijas klases satur daudz informācijas, pirms tās importa var izveidot skatījumu, kas palielina klašu kastes izmērus, lai ontoloģija kļūtu lasāmāka. Ja kādai diagrammai skatījums nav vajadzīgs, to var

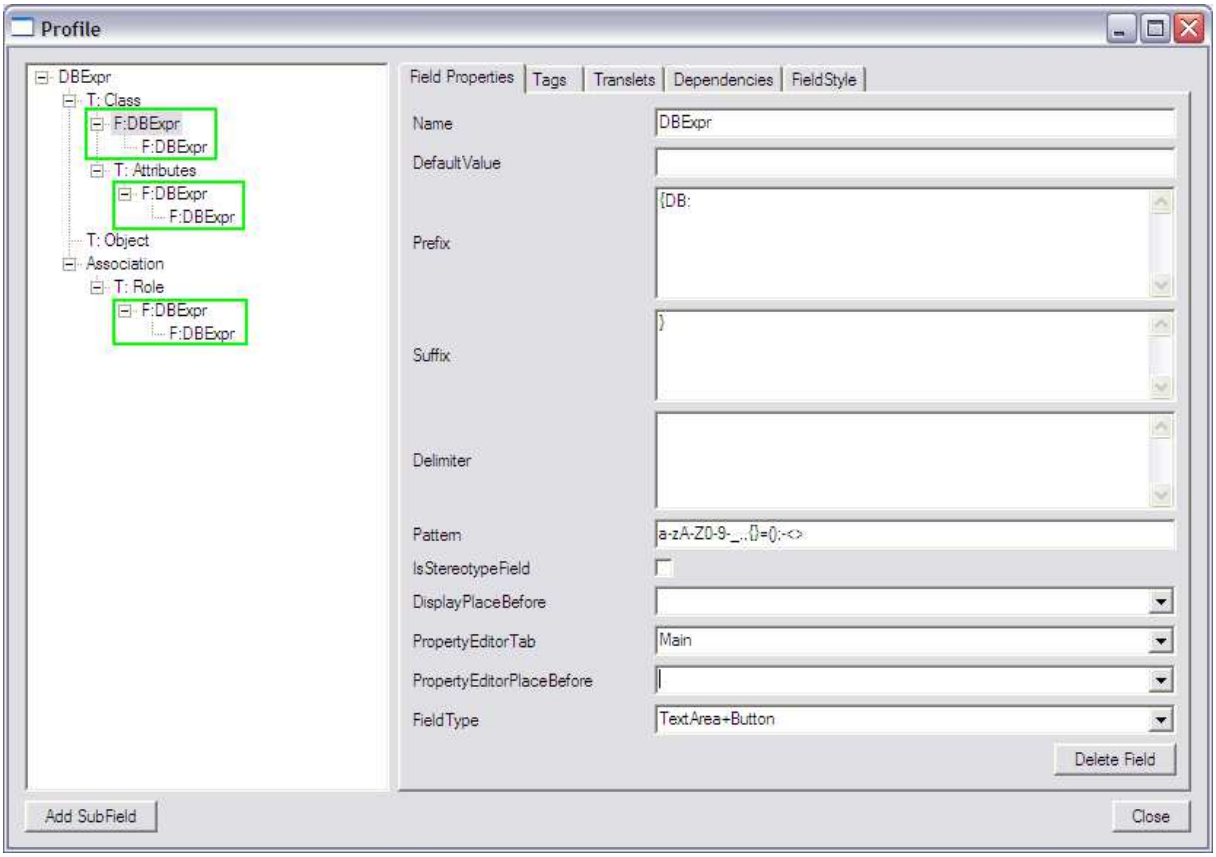
noņemt, un tiks noņemti visi stili, kurus saturēja skatījums. Konfigurators neatbalsta šādas iespējas.

Lietotāju ērtībai, ja atribūtam ir apgriezts atribūts, jaunie lauki ir jāveido tikai vienam atribūtam, apgrieztam tie tiks pievienoti automātiski.

6.2 Lietotāja definētu lauku mehānisma darbības piemērs

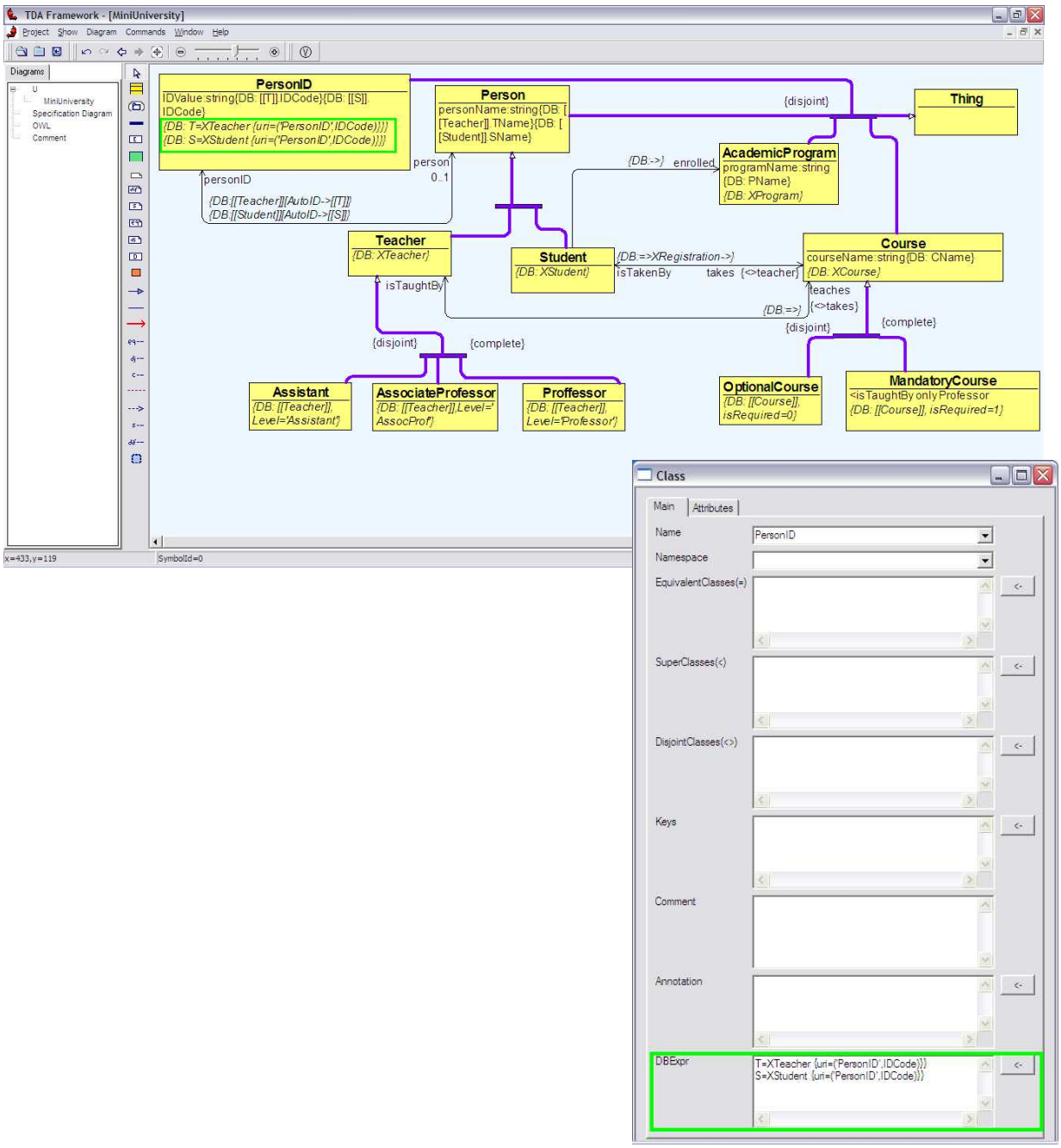
Aplūkosim dažus piemērus, kā darbojas lietotāja definētu lauku mehānisms.

Piemēram, lai nodefinētu datu bāzes savienojamības anotācijas, bija jāizveido **DBExpr** daudzrindu teksta ievades lauki. Jaunie lauki tika veidoti zem klases, atribūta un asociācijas galos. Tad lietotāja definētu lauku pārvaldības logs izskatīsies šādi:



Attēls 6.5 **Lietotāja definētu lauku pārvaldības logs datu bāzes savienojamības anotācijām**

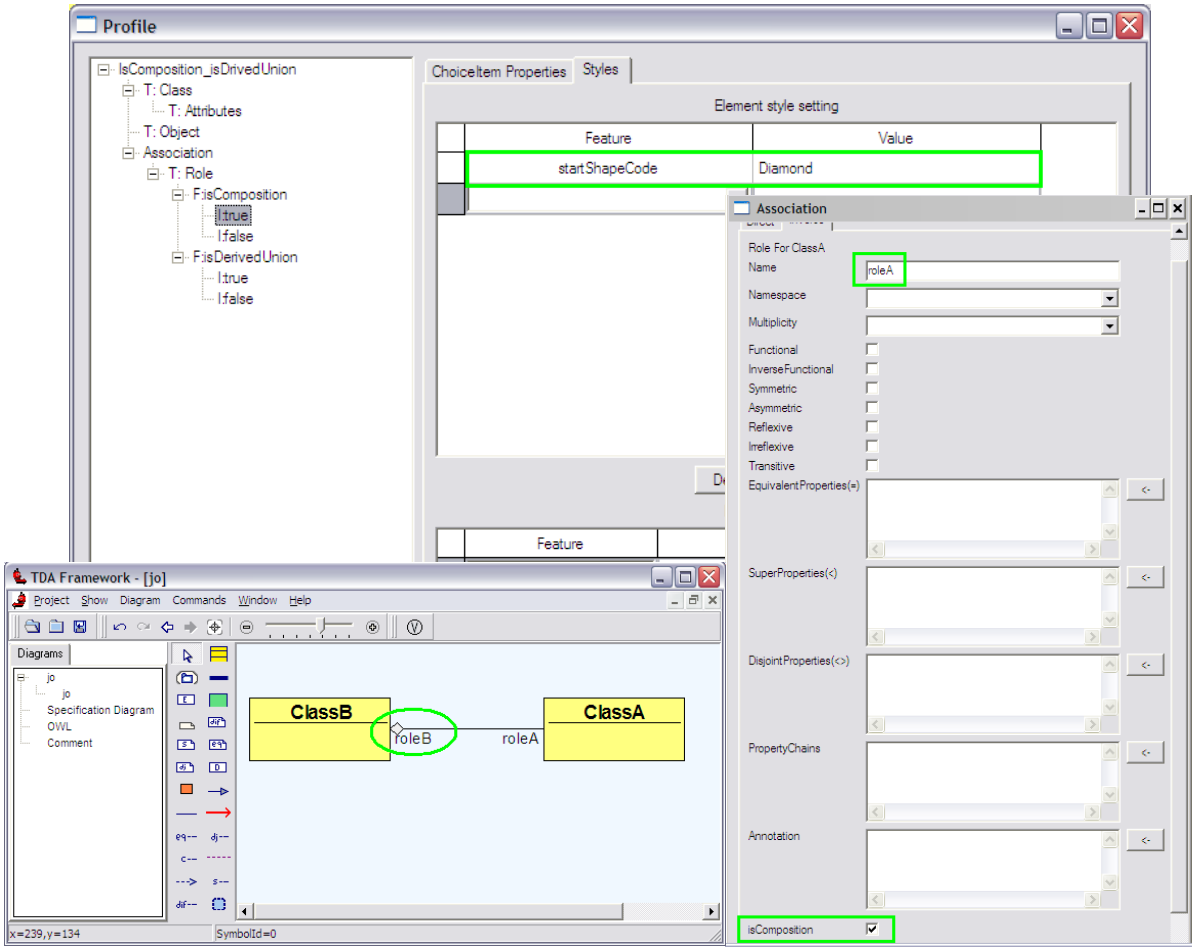
Un klases, atribūta uz asociāciju īpašību dialoga logos parādīsies jauns lauks **DBExpr**. Klases un asociāciju galu lauki tika izcelti ar slīprakstu, lai atšķirtu tos no cita veida anotācijām. Klases īpašību logs izskatīsies šādi:



Attēls 6.6 **Datu bāzes savienojamības anotāciju notāciju definēšanas piemērs**

Un attiecīgi diagrammā datu bāzes savienojamības anotācijas rādīsies slīprakstā.

Lai izveidotu UML kompozīcijas notāciju zem asociāciju galiem, bija jāizveido izvēles rūtiņa (*CheckBox*) un zem *true* izvēles vienumā jānodefinē asociācijas stils:



Attēls 6.7 **UML kompozīcijas notāciju definēšanas piemērs**

Tika izvelēts *StartShapeCode* stila vienums, tās nozīmē, ka stils tiek pielietots pretējam galam. Tas ir, ja *roleA* tika uzstādīts *isComposition* vienāds ar *true*, tad grafiski stils atspoguļosies *roleB* līniju galā.

Nobeigums

Darba izpildes gaitā visi izvirzītie mērķi tika sasniegti.

Darba gaitā tika izpētītas OWL ontoloģijas, to pielietošanas iespējas un nākotnes vīzijas. Tika izpētītas OWL ontoloģiju nākotnē iespējamās izmantošanas iespējas un papildinājumi, kas ir nepieciešami, lai šos lietojumus realizētu.

Tika detalizēti izpētīta TDA rīku būves platforma un tās galvenās sastāvdaļas, izpētīts OWLGrEd grafiskais redaktors un tās OWL ontoloģiju vizualizācijas iespējas.

Darba gaitā tika izveidots lietotāju definēto lauku mehānisms, kas nodrošina iespēju papildināt OWLGrEd redaktoru ar lietotāju definētām notācijām. Tika realizēts jauno notāciju semantikas definēšanas mehānisms, kas ļauj lietotājiem pašiem noteikt kādu semantiku un kādos gadījumos ir jāģenerē.

Lietotāju definēto lauku mehānismā tika realizēts profilu mehānisms, kas ļauj definēt tā saucamos profilus, kas satur jauno īpašību definējumus. Papildus profilu mehānismam tika izveidots konfigurācijas mehānisms, kas ļauj definēt, zem kuriem atribūtiem un elementiem ir atļauts veidot jaunus laukus, un kāda tipa semantika ir pieejama definēšanai.

Šī darba ietvaros pirmo reizi TDA platformā ir piedāvāts universāls stilu uzstādīšanas mehānisms, kas darbojas atsevišķu stila komponenšu līmenī.

Ar izveidotu lietotāju definēto lauku mehānismu kļuva iespējams vizualizēt OWL ontoloģiju jaunos lietojumus. Par to liecina jauno notāciju veiksmīgs vizualizācijas mēģinājums. Tika izveidoti profili, kas definē anotāciju semantiku papildus UML konstrukcijām, tādām kā kompozīcija un atvasināta apvienojumu īpašība, datu bāzes savienojamības anotācijas, integritātes ierobežojumu notācijas.

Lietotāju definēto lauku mehānisma uzstādīšana ir vienkārša un intuitīva.

Nākotnē tiek plānots lietotāju definēto lauku mehānismu pilnveidot, lai to varētu izmantot jebkurā uz TDA platformas veidotajā rīkā, uzlabot semantikas definēšanas iespējas, lai lietotājiem nav jādefinē gan eksporta, gan importa semantikas, realizēt atkarības, kas šobrīd netiek atbalstītas, pilnveidot atkarīgo stilu uzstādīšanas mehānismu, lai atļautu iespēju uzstādīt atkarīgus stilus dažādos līmeņos.

Literatūra

1. **K. Cerans, R. Liepins, A. Sprogis, J. Ovcinnikova, G. Barzdins**. Domain-Specific OWL Ontology Visualization with OWLGrEd **Proceedings of** *9th Extended Semantic Web Conference 2012*, May 27 – 31, Heraklion, Crete, GR.
2. **K. Cerans, G. Barzdins, R. Liepins, J. Ovcinnikova, S. Rikacovs, A. Sprogis**. Graphical Schema Editing for StarDog OWL/RDF Databases using OWLGrEd/S **Proceedings of** *OWLED Workshop of 9th Extended Semantic Web Conference 2012*, May 27 – 28, Heraklion, Crete, GR.
3. **K. ČERĀNS, J. OVČIŅŅIKOVA, R. LIEPIŅŠ, A. SPROĢIS**. Advanced OWL 2.0 Ontology Visualization in OWLGrEd **Iesniegts publicēšanai** *Tenth International Baltic Conference on Databases and Information Systems 2012*, July 8 – 11, Vilnius, Lithuania.
4. **J. Barzdins, S. Kozlovics, E. Rencis**. The Transformation-Driven Architecture. **Proceedings of** *DSM'08 Workshop of OOPSLA 2008*. Nashville, USA, 2008, pp. 60–63.
5. **J. Barzdins, K. Cerans, S. Kozlovics, E. Rencis, A. Zarins**. A Graph Diagram Engine for the Transformation- Driven Architecture. **Proceedings of** *MDDAUI'09 Workshop of International Conference on Intelligent User Interfaces 2009*, Sanibel Island, Florida, USA, 2009, pp. 29–32.
6. **J. Bārzdiņš, G. Bārzdiņš, K. Čerāns, R. Liepiņš, A. Sproģis**. OWLGrEd: a UML Style Graphical Editor for OWL, **Proceedings of** *the 1st Workshop on Ontology Repositories and Editors for the Semantic Web*, Hersonissos, Crete, Greece, May 31st, 2010
7. **J. Bārzdiņš, G. Bārzdiņš, K. Čerāns, R. Liepiņš, A. Sproģis**. OWLGrEd: a UML Style Graphical Notation and Editor for OWL 2, **Proceedings of** *OWLED 2010, OWL: Experiences and Directions, 7th International Workshop*, San Francisco, California, USA, 21-22 June 2010
8. **A. Sproģis**. The Configurator in DSL Tool Building, *Scientific Papers, University of Latvia, Vol. 756. Computer Science and Information Technologies*. Riga, Latvia, 2010, pp. 173–192.
9. **Sergejs Kozlovics.** A Dialog Engine Metamodel for the Transformation-Driven Architecture, *Scientific Papers, University of Latvia, Vol. 756. Computer Science and Information Technologies*. Riga, Latvia, 2010, pp. 151–170.
10. **J. Barzdins, K. Cerans, S. Kozlovics, L. Lace, R. Liepins, E. Rencis, A. Sprogis, A. Zarins.** An MDE-Based Graphical Tool Building Framework, *Scientific Papers, University of Latvia, Vol. 756. Computer Science and Information Technologies*. Riga, Latvia, 2010, pp. 121–138.

11. **Smith, M. K.; Welty, C.; and McGuiness, D.** OWL Web Ontology Language Guide, 2004.
12. **Motik, B; Patel-Schneider P.F; Parsia B.** OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax, 2009
13. **J. Barzdins, K. Cerans, S. Kozlovics, E. Rencis, A. Zarins.** Graph Diagram Engine for the Transformation-Driven Architecture, *Scientific Papers, University of Latvia, Vol. 756. Computer Science and Information Technologies*. Riga, Latvia, 2010, pp. 139–149.
14. *Lielā terminu vārdnīca.* Pieejams internetā: http://www.termini.lv
15. *Vikipēdijas raksts par Klašu diagrammām.* Pieejams internetā: http://en.wikipedia.org/wiki/Class_diagram
16. *Vikipēdijas raksts par Web Ontology Language.* Pieejams internetā: http://en.wikipedia.org/wiki/Web_Ontology_Language
17. *OWL 2 Web Ontology Language Manchester Syntax.* W3C Working Group Note, 27 October 2009. Pieejams internetā: http://www.w3.org/TR/owl2-manchester-syntax/
18. *Protégé.* Pieejams internetā: http://protege.stanford.edu/
19. *Protégé-frames*. Pieejams internetā: http://protege.stanford.edu/overview/protege-frames.html
20. *Protégé-owl.* Pieejams internetā: http://protege.stanford.edu/overview/protege-owl.html
21. *TopBraid Composer*. Pieejams internetā: http://www.topquadrant.com/products/TB_Composer.html
22. **K.Čerāns, G.Būmans**, RDB2OWL: a RDB-to-RDF/OWL Mapping Specification Language, *Databases and Information Systems VI*, IOS Press 2011, p.139-152. **SCOPUS**
23. *Stardog*. Pieejams internetā: http://stardog.com/

Pielikumi

1. pielikums

CD disks ar lietotāja definētu lauku mehānismu

Darbam tiek pievienots CD disks ar lietotāja definētu lauku mehānisma izejas kodiem un OWLGrEd redaktoru ar iebūvētu strādājošu lietotāja definētu lauku mehānismu.

Lietotāja definētu lauku mehānismu izejas kodi var tikt atrasti mapē „OWLGrEd_UserFields” un paredzēti koda apskatīšanas nolūkiem. OWLGrEd redaktors atrodas mapē „OWLGrEd” un tajā jau ir iebūvēts lietotāja definēto lauku mehānisms, kas atrodas mapē „OWLGrEd\Tools\OWLGrEd\Plugins”.

Lai palaistu OWLGrEd redaktoru, no „OWLGrEd/Bin” ar dubultklikšķi jāpalaiž **tda.exe**. Lai izveidotu jaunu projektu, jāizvēlas „Project->New project”. Lai atvērtu eksistējošo projektu, jāizvēlas „Project->Open project”. OWLGrEd redaktorā tika pielikts testa projekts „Example_UserFields” ar dažiem mehānisma darbības piemēriem.

Lai uzsāktu darbu ar lietotāja definētu lauku mehānismu, no rīkjoslas ir jānospiež Ⓟ poga. Lai izveidotu jauno profilu atvērtajā profilu pārvaldības logā, jānospiež **New** poga. Lai ielādētu profilu no teksta faila, jānospiež **Import** poga un jāizvēlas vēlamais fails.

Lietotāja definētu lauku mehānismam tika pievienoti trīs profilu faili: **DBExpr** – datu bāzes savienojamības anotāciju definēšanai, **IsComposition_isDerivedUnion** – anotāciju semantikas definēšanai papildus UML konstrukcijām un **IC** – integritātes ierobežojumu notāciju definēšana. Lai varētu ielādēt **IC** profilu, ir jāielādē papildus konteksta tipi. Tas ir paveicams no rīkjoslas, nospiežot © pogu un ielādējot **configuration.txt** failu.

2. pielikums

Lietotāja definētu lauku mehānisma koda piemēri

Metamodeļa papilinājuma funkcija

```
function completeMetamodel()
print("complete STARTED")
  --klases
  lQuery.model.add_class("AA#Profile")
  lQuery.model.add_class("AA#View")
  lQuery.model.add_class("AA#Field")
  lQuery.model.add_class("AA#RowType")
  lQuery.model.add_class("AA#Dependency")
  lQuery.model.add_class("AA#ChoiceItem")
  lQuery.model.add_class("AA#TransletTask")
  lQuery.model.add_class("AA#ContextType")
  lQuery.model.add_class("AA#Translet")
  lQuery.model.add_class("AA#Tag")
  lQuery.model.add_class("AA#StyleSetting")
  lQuery.model.add_class("AA#FieldStyleSetting")
```

```
lQuery.model.add_class("AA#ViewStyleSetting")
lQuery.model.add_class("AA#CompartStyleItem")
lQuery.model.add_class("AA#ElemStyleItem")
lQuery.model.add_class("AA#NodeStyleItem")
lQuery.model.add_class("AA#EdgeStyleItem")
lQuery.model.add_class("AA#AnyElemStyleItem")
lQuery.model.add_class("AA#Configuration")
lQuery.model.add_class("AA#TagType")
--atributi
lQuery.model.add_property("AA#Profile", "name")
lQuery.model.add_property("AA#Profile", "isUniversal")
lQuery.model.add_property("AA#Profile", "tagTabName")

lQuery.model.add_property("AA#Field", "name")
lQuery.model.add_property("AA#Field", "defaultValue")
lQuery.model.add_property("AA#Field", "prefix")
lQuery.model.add_property("AA#Field", "suffix")
lQuery.model.add_property("AA#Field", "delimiter")
lQuery.model.add_property("AA#Field", "pattern")
lQuery.model.add_property("AA#Field", "isStereotypeField")
lQuery.model.add_property("AA#Field", "displayPlaceBefore")
lQuery.model.add_property("AA#Field", "propertyEditorTab")
lQuery.model.add_property("AA#Field", "propertyEditorPlaceBefore")

lQuery.model.add_property("AA#RowType", "typeName")

lQuery.model.add_property("AA#ChoiceItem", "caption")
lQuery.model.add_property("AA#ChoiceItem", "notation")

lQuery.model.add_property("AA#TransletTask", "taskName")

lQuery.model.add_property("AA#ContextType", "nr")
lQuery.model.add_property("AA#ContextType", "type")
lQuery.model.add_property("AA#ContextType", "elTypeName")
lQuery.model.add_property("AA#ContextType", "path")
lQuery.model.add_property("AA#ContextType", "mode")
lQuery.model.add_property("AA#ContextType", "hasMirror")
lQuery.model.add_property("AA#ContextType", "id")

lQuery.model.add_property("AA#Translet", "procedure")

lQuery.model.add_property("AA#TagType", "key")
lQuery.model.add_property("AA#TagType", "notation")
lQuery.model.add_property("AA#TagType", "rowType")

lQuery.model.add_property("AA#Tag", "axiomPattern")
lQuery.model.add_property("AA#Tag", "tagKey")

lQuery.model.add_property("AA#StyleSetting", "value")
lQuery.model.add_property("AA#StyleSetting", "target")
lQuery.model.add_property("AA#StyleSetting", "isElementStyleSetting")
lQuery.model.add_property("AA#StyleSetting", "path")

lQuery.model.add_property("AA#ViewStyleSetting", "elementTypeName")
lQuery.model.add_property("AA#ViewStyleSetting", "conditionCompartType")
lQuery.model.add_property("AA#ViewStyleSetting", "conditionChoiceItem")
lQuery.model.add_property("AA#ViewStyleSetting", "addMirror")

lQuery.model.add_property("AA#View", "name")
lQuery.model.add_property("AA#View", "isDefault")

lQuery.model.add_property("AA#CompartStyleItem", "itemName")
lQuery.model.add_property("AA#CompartStyleItem", "itemType")
lQuery.model.add_property("AA#CompartStyleItem", "forNodeCompart")
lQuery.model.add_property("AA#CompartStyleItem", "forEdgeCompart")
lQuery.model.add_property("AA#CompartStyleItem", "forAttribCompart")
lQuery.model.add_property("AA#CompartStyleItem", "isStyleItem")

lQuery.model.add_property("AA#ElemStyleItem", "itemName")
lQuery.model.add_property("AA#ElemStyleItem", "itemType")
--linki
```

```
  lQuery.model.add_link("AA#Field", "fieldInContext", "context", "AA#ContextType")
  lQuery.model.add_link("AA#Field", "field", "fieldType", "AA#RowType")
  lQuery.model.add_link("AA#Field", "dependent", "dependency", "AA#Dependency")

  lQuery.model.add_link("AA#ChoiceItem", "dependsOn", "dependency",
"AA#Dependency")

  lQuery.model.add_link("AA#Translet", "translet", "task", "AA#TransletTask")

  lQuery.model.add_link("AA#Field", "subField", "superField", "AA#Field")

  lQuery.model.add_link("AA#ElemStyleItem", "elemStyleFeature", "styleSetting",
"AA#StyleSetting")
  lQuery.model.add_link("AA#CompartStyleItem", "fieldStyleFeature", "styleSetting",
"AA#StyleSetting")

  lQuery.model.add_link("AA#ContextType", "context", "profile", "AA#Profile")
  lQuery.model.add_link("AA#TagType", "tagType", "profile", "AA#Profile")
  lQuery.model.add_link("AA#Configuration", "configuration", "profile",
"AA#Profile")

  --kompozicijas
  lQuery.model.add_composition("AA#Field", "field", "profile", "AA#Profile")
  lQuery.model.add_composition("AA#View", "view", "profile", "AA#Profile")
  lQuery.model.add_composition("AA#ViewStyleSetting", "styleSetting", "view",
"AA#View")
  lQuery.model.add_composition("AA#ChoiceItem", "choiceItem", "field", "AA#Field")
  lQuery.model.add_composition("AA#Field", "subField", "superField", "AA#Field")
  lQuery.model.add_composition("AA#Translet", "translet", "field", "AA#Field")
  lQuery.model.add_composition("AA#Tag", "tag", "field", "AA#Field")
  lQuery.model.add_composition("AA#Tag", "tag", "choiceItem", "AA#ChoiceItem")
  lQuery.model.add_composition("AA#Tag", "tag", "profile", "AA#Profile")
  lQuery.model.add_composition("AA#FieldStyleSetting", "selfStyleSetting", "field",
"AA#Field")
  lQuery.model.add_composition("AA#FieldStyleSetting", "styleSetting",
"choiceItem", "AA#ChoiceItem")
  lQuery.model.add_composition("AA#ContextType", "context", "configuration",
"AA#Configuration")
  lQuery.model.add_composition("AA#TagType", "tagType", "configuration",
"AA#Configuration")

  --virsklases
  lQuery.model.set_super_class("AA#FieldStyleSetting", "AA#StyleSetting")
  lQuery.model.set_super_class("AA#ViewStyleSetting", "AA#StyleSetting")
  lQuery.model.set_super_class("AA#NodeStyleItem", "AA#ElemStyleItem")
  lQuery.model.set_super_class("AA#EdgeStyleItem", "AA#ElemStyleItem")
  lQuery.model.set_super_class("AA#AnyElemStyleItem", "AA#ElemStyleItem")

  lQuery.create("AA#Configuration")


  lQuery.create("AA#ContextType", {
              nr = 01,type = "Class",mode = "Element",id = "Class"
              }):link("configuration", lQuery("AA#Configuration"))
  lQuery.create("AA#ContextType", {
              nr = 02,type = "Role",elTypeName = "Association",mode = "Group"
              ,hasMirror = 1,id = "Association/Role"
              }):link("configuration", lQuery("AA#Configuration"))
  lQuery.create("AA#ContextType", {
              nr = 03,type = "Attributes",elTypeName = "Class",mode = "Group Item"
              ,id = "Class/Attributes"
              }):link("configuration", lQuery("AA#Configuration"))
  lQuery.create("AA#ContextType", {
              nr = 04,type = "Object"mode = "Element",id = "Object"
              }):link("configuration", lQuery("AA#Configuration"))

  lQuery.create("AA#TransletTask", {
              taskName = "procGenerateItemsClickBox"})
  lQuery.create("AA#TransletTask", {
              taskName = "procStartValue"})
  lQuery.create("AA#TransletTask", {
```

```
            taskName = "procFieldEntered"})
  lQuery.create("AA#TransletTask", {
            taskName = "procCompose"})
  lQuery.create("AA#TransletTask", {
            taskName = "procDecompose"})


  lQuery.create("AA#RowType", {
            typeName = "InputField"})
  lQuery.create("AA#RowType", {
            typeName = "InputField+Button"})
  lQuery.create("AA#RowType", {
            typeName = "CheckBox"})
  lQuery.create("AA#RowType", {
            typeName = "ComboBox"})
  lQuery.create("AA#RowType", {
            typeName = "ListBox"})
  lQuery.create("AA#RowType", {
            typeName = "TextArea"})
  lQuery.create("AA#RowType", {
            typeName = "TextArea+Button"})
  lQuery.create("AA#RowType", {
            typeName = ""})


  lQuery.create("AA#CompartStyleItem", {itemName = 'isVisible', itemType =
'integer', forNodeCompart = '1', forEdgeCompart = '1', forAttribCompart = '1'})
  lQuery.create("AA#CompartStyleItem", {itemName = 'adjustment', itemType =
'integer', forNodeCompart = '1', forEdgeCompart = '1', forAttribCompart = '0'})
  lQuery.create("AA#CompartStyleItem", {itemName = 'alignment', itemType =
'integer', forNodeCompart = '1', forEdgeCompart = '1', forAttribCompart = '0'})
  lQuery.create("AA#CompartStyleItem", {itemName = 'adornment', itemType =
'integer', forNodeCompart = '1', forEdgeCompart = '1', forAttribCompart = '0'})
  lQuery.create("AA#CompartStyleItem", {itemName = 'textDirection', itemType =
'integer', forNodeCompart = '1', forEdgeCompart = '1', forAttribCompart = '0'})
  lQuery.create("AA#CompartStyleItem", {itemName = 'fontTypeFace', itemType =
'string', forNodeCompart = '1', forEdgeCompart = '1', forAttribCompart = '0'})
  lQuery.create("AA#CompartStyleItem", {itemName = 'fontStyleBold', itemType =
'boolean', forNodeCompart = '1', forEdgeCompart = '1', forAttribCompart = '0'})
  lQuery.create("AA#CompartStyleItem", {itemName = 'fontStyleItalic', itemType =
'boolean', forNodeCompart = '1', forEdgeCompart = '1', forAttribCompart = '0'})
  lQuery.create("AA#CompartStyleItem", {itemName = 'fontStyleStrikeout', itemType =
'boolean', forNodeCompart = '1', forEdgeCompart = '1', forAttribCompart = '0'})
  lQuery.create("AA#CompartStyleItem", {itemName = 'fontStyleUnderline', itemType =
'boolean', forNodeCompart = '1', forEdgeCompart = '1', forAttribCompart = '0'})
  lQuery.create("AA#CompartStyleItem", {itemName = 'fontSize', itemType =
'integer', forNodeCompart = '1', forEdgeCompart = '1', forAttribCompart = '0'})
  lQuery.create("AA#CompartStyleItem", {itemName = 'fontColor', itemType =
'integer', forNodeCompart = '1', forEdgeCompart = '1', forAttribCompart = '0'})
  lQuery.create("AA#CompartStyleItem", {itemName = 'fontPitch', itemType =
'integer', forNodeCompart = '0', forEdgeCompart = '0', forAttribCompart = '0'})
  lQuery.create("AA#CompartStyleItem", {itemName = 'fontCharSet', itemType =
'integer', forNodeCompart = '1', forEdgeCompart = '1', forAttribCompart = '0'})
  lQuery.create("AA#CompartStyleItem", {itemName = 'lineWidth', itemType =
'integer', forNodeCompart = '0', forEdgeCompart = '0', forAttribCompart = '0'})
  lQuery.create("AA#CompartStyleItem", {itemName = 'lineColor', itemType =
'integer', forNodeCompart = '0', forEdgeCompart = '0', forAttribCompart = '0'})
  lQuery.create("AA#CompartStyleItem", {itemName = 'picture', itemType = 'string',
forNodeCompart = '1', forEdgeCompart = '1', forAttribCompart = '0'})
  lQuery.create("AA#CompartStyleItem", {itemName = 'picStyle', itemType =
'integer', forNodeCompart = '1', forEdgeCompart = '1', forAttribCompart = '0'})
  lQuery.create("AA#CompartStyleItem", {itemName = 'picWidth', itemType =
'integer', forNodeCompart = '1', forEdgeCompart = '1', forAttribCompart = '0'})
  lQuery.create("AA#CompartStyleItem", {itemName = 'picHeight', itemType =
'integer', forNodeCompart = '1', forEdgeCompart = '1', forAttribCompart = '0'})
  lQuery.create("AA#CompartStyleItem", {itemName = 'picPos', itemType = 'integer',
forNodeCompart = '1', forEdgeCompart = '1', forAttribCompart = '0'})
  lQuery.create("AA#CompartStyleItem", {itemName = 'prefix-inside', itemType =
'boolean', forNodeCompart = '1', forEdgeCompart = '1', forAttribCompart = '1'})
  lQuery.create("AA#CompartStyleItem", {itemName = 'prefix-outside', itemType =
'boolean', forNodeCompart = '1', forEdgeCompart = '1', forAttribCompart = '1'})
```

```
  lQuery.create("AA#CompartStyleItem", {itemName = 'prefix-inPlace', itemType =
'boolean', forNodeCompart = '1', forEdgeCompart = '1', forAttribCompart = '1'})
  lQuery.create("AA#CompartStyleItem", {itemName = 'suffix-inside', itemType =
'boolean', forNodeCompart = '1', forEdgeCompart = '1', forAttribCompart = '1'})
  lQuery.create("AA#CompartStyleItem", {itemName = 'suffix-outside', itemType =
'boolean', forNodeCompart = '1', forEdgeCompart = '1', forAttribCompart = '1'})
  lQuery.create("AA#CompartStyleItem", {itemName = 'suffix-inPlace', itemType =
'boolean', forNodeCompart = '1', forEdgeCompart = '1', forAttribCompart = '1'})

  lQuery.create("AA#AnyElemStyleItem", {itemName = 'shapeCode', itemType =
'integer'})
  lQuery.create("AA#AnyElemStyleItem", {itemName = 'lineWidth', itemType =
'integer'})
  lQuery.create("AA#AnyElemStyleItem", {itemName = 'dashLength', itemType =
'integer'})
  lQuery.create("AA#AnyElemStyleItem", {itemName = 'breakLength', itemType =
'integer'})
  lQuery.create("AA#AnyElemStyleItem", {itemName = 'lineColor', itemType =
'integer'})
  lQuery.create("AA#AnyElemStyleItem", {itemName = 'bkgColor', itemType =
'integer'})
  lQuery.create("AA#AnyElemStyleItem", {itemName = 'shapeStyleShadow', itemType =
'boolean'})
  lQuery.create("AA#AnyElemStyleItem", {itemName = 'shapeStyle3D', itemType =
'boolean'})
  lQuery.create("AA#AnyElemStyleItem", {itemName = 'shapeStyleMultiple', itemType =
'boolean'})
  lQuery.create("AA#AnyElemStyleItem", {itemName = 'shapeStyleNoBorder', itemType =
'boolean'})
  lQuery.create("AA#AnyElemStyleItem", {itemName = 'shapeStyleNoBackground',
itemType = 'boolean'})
  lQuery.create("AA#AnyElemStyleItem", {itemName = 'shapeStyleNotLinePen', itemType
= 'boolean'})

  lQuery.create("AA#NodeStyleItem", {itemName = 'alignment', itemType = 'integer'})
  lQuery.create("AA#NodeStyleItem", {itemName = 'picture', itemType = 'string'})
  lQuery.create("AA#NodeStyleItem", {itemName = 'picStyle', itemType = 'integer'})
  lQuery.create("AA#NodeStyleItem", {itemName = 'picPos', itemType = 'integer'})
  lQuery.create("AA#NodeStyleItem", {itemName = 'picWidth', itemType = 'integer'})
  lQuery.create("AA#NodeStyleItem", {itemName = 'picHeight', itemType = 'integer'})
  lQuery.create("AA#NodeStyleItem", {itemName = 'width', itemType = 'integer'})
  lQuery.create("AA#NodeStyleItem", {itemName = 'height', itemType = 'integer'})
  lQuery.create("AA#NodeStyleItem", {itemName = 'widthProc', itemType = 'boolean'})

  lQuery.create("AA#EdgeStyleItem", {itemName = 'lineType', itemType = 'string'})
  lQuery.create("AA#EdgeStyleItem", {itemName = 'lineDirection', itemType =
'integer'})
  lQuery.create("AA#EdgeStyleItem", {itemName = 'startShapeCode', itemType =
'integer'})
  lQuery.create("AA#EdgeStyleItem", {itemName = 'startLineWidth', itemType =
'integer'})
  lQuery.create("AA#EdgeStyleItem", {itemName = 'startDashLenght', itemType =
'integer'})
  lQuery.create("AA#EdgeStyleItem", {itemName = 'startBreakLenght', itemType =
'integer'})
  lQuery.create("AA#EdgeStyleItem", {itemName = 'startLineColor', itemType =
'integer'})
  lQuery.create("AA#EdgeStyleItem", {itemName = 'startBkgColor', itemType =
'integer'})
  lQuery.create("AA#EdgeStyleItem", {itemName = 'endShapeCode', itemType =
'integer'})
  lQuery.create("AA#EdgeStyleItem", {itemName = 'endLineWidth', itemType =
'integer'})
  lQuery.create("AA#EdgeStyleItem", {itemName = 'endDashLength', itemType =
'integer'})
  lQuery.create("AA#EdgeStyleItem", {itemName = 'endBreakLenght', itemType =
'integer'})
  lQuery.create("AA#EdgeStyleItem", {itemName = 'endLineColor', itemType =
'integer'})
  lQuery.create("AA#EdgeStyleItem", {itemName = 'endBkgColor', itemType =
'integer'})
```

```
  lQuery.create("AA#EdgeStyleItem", {itemName = 'middleShapeCode', itemType =
'integer'})
  lQuery.create("AA#EdgeStyleItem", {itemName = 'middleLineWidth', itemType =
'integer'})
  lQuery.create("AA#EdgeStyleItem", {itemName = 'middleDashLength', itemType =
'integer'})
  lQuery.create("AA#EdgeStyleItem", {itemName = 'middleBreakLength', itemType =
'integer'})
  lQuery.create("AA#EdgeStyleItem", {itemName = 'middleLineColor', itemType =
'integer'})
  lQuery.create("AA#EdgeStyleItem", {itemName = 'middleBkgColor', itemType =
'integer'})

  lQuery.model.add_class("AA#ElementStyleSetting")
  lQuery.model.add_class("AA#CompartmentStyleSetting")

  lQuery.model.add_property("AA#ElementStyleSetting", "setting")
  lQuery.model.add_property("AA#ElementStyleSetting", "value")
  lQuery.model.add_property("AA#ElementStyleSetting", "procSetValue")
  lQuery.model.add_property("AA#ElementStyleSetting", "procCondition")
  lQuery.model.add_property("AA#ElementStyleSetting", "strength")
  lQuery.model.add_property("AA#ElementStyleSetting", "isDeleted")

  lQuery.model.add_property("AA#CompartmentStyleSetting", "setting")
  lQuery.model.add_property("AA#CompartmentStyleSetting", "value")
  lQuery.model.add_property("AA#CompartmentStyleSetting", "procSetValue")
  lQuery.model.add_property("AA#CompartmentStyleSetting", "procCondition")
  lQuery.model.add_property("AA#CompartmentStyleSetting", "strength")
  lQuery.model.add_property("AA#CompartmentStyleSetting", "isDeleted")
  lQuery.model.add_property("AA#CompartmentStyleSetting", "settingMode")

  lQuery.model.add_link("AA#ElementStyleSetting", "aaElementStyleSetting",
"elemType", "ElemType")
  lQuery.model.add_link("AA#ElementStyleSetting", "aaElementStyleSetting",
"choiceItem", "ChoiceItem")
  lQuery.model.add_link("AA#ElementStyleSetting", "aaElementStyleSetting",
"extension", "Extension")

  lQuery.model.add_link("AA#CompartmentStyleSetting", "aaCompartmentStyleSetting",
"compartType", "CompartType")
  lQuery.model.add_link("AA#CompartmentStyleSetting", "aaCompartmentStyleSetting",
"choiceItem", "ChoiceItem")
  lQuery.model.add_link("AA#CompartmentStyleSetting", "aaCompartmentStyleSetting",
"extension", "Extension")

  lQuery.create("AA#TagType", {key = "owl_Field_axiom", notation = "Semantics",
rowType = "TextArea+Button"}):link("configuration", lQuery("AA#Configuration"))
  lQuery.create("AA#TagType", {key = "owl_Fields_ImportSpec", notation = "Import
Semantics", rowType = "TextArea"}):link("configuration",
lQuery("AA#Configuration"))
  lQuery.create("AA#TagType", {key = "owl_Axiom_Annotation", notation = "Axiom
Annotation", rowType = "TextArea"}):link("configuration",
lQuery("AA#Configuration"))

  --paslepjam IsComposition
  lQuery("CompartType[caption='IsComposition']"):attr("shouldBeIncluded",
"OWLGrEd_UserFields.owl_fields_specific.hide_for_OWL_Fields")

lQuery("CompartType[caption='IsComposition']/propertyRow"):attr("shouldBeIncluded",
"OWLGrEd_UserFields.owl_fields_specific.hide_for_OWL_Fields")
  lQuery("Compartment:has(/compartType[caption='IsComposition'])"):delete()
  lQuery("GraphDiagram:has(/graphDiagramType[id='OWL'])"):each(function(diagram)
    local cmd = lQuery.create("OkCmd")
    cmd:link("graphDiagram", diagram)
    utilities.execute_cmd_obj(cmd)
  end)


  lQuery.model.add_link("Extension", "aa#owner", "aa#subExtension", "Extension")
  lQuery.model.add_link("Extension", "aa#activeExtension", "graphDiagram",
"GraphDiagram")
```

```
  lQuery.model.add_link("Extension", "aa#notDefault", "aa#graphDiagram",
"GraphDiagram")

  lQuery.model.add_link("Type", "aa#type", "aa#tag", "Tag")

  lQuery.create("Extension", {id="OWL_Fields", type="Plugin"})

  local compart = lQuery("CompartType"):filter(function(obj)
    return lQuery(obj):find("/choiceItem/elemStyleByChoiceItem"):size() > 0
  end)
  compart:each(function(obj)
    lQuery.create("Translet", {extensionPoint = 'procFieldEntered', procedureName =
'OWLGrEd_UserFields.owl_fields_specific.setStyleSetting'}):link("type", obj)
  end)
  compart = lQuery("CompartType"):filter(function(obj)
    return lQuery(obj):find("/choiceItem/compartStyleByChoiceItem"):size() > 0
  end)
  compart:each(function(obj)
    lQuery.create("Translet", {extensionPoint = 'procFieldEntered', procedureName =
'OWLGrEd_UserFields.owl_fields_specific.setStyleSetting'}):link("type", obj)
  end)

  --pievienojam tagus pie toolType prieks importa/eksporta
  lQuery.create("Tag", {value = "OWLGrEd_UserFields.axiom.axiom", key =
"owlgred_export"}):link("type", lQuery("ToolType"))
  lQuery.create("Tag", {value =
"owlFields:=<http://owlgred.lumii.lv/__plugins/fields/2011/1.0/owlgred#>", key =
"owl_Fields_NamespaceDef"}):link("type", lQuery("ToolType"))
  lQuery.create("Tag", {key = "owl_Annotation_Import"}):link("type",
lQuery("ToolType"))
  lQuery.create("Tag", {key = "owl_Import_Prefixes"}):link("type",
lQuery("ToolType"))
  lQuery.create("Translet", {extensionPoint='RecalculateStylesInImport',
procedureName='OWLGrEd_UserFields.owl_fields_specific.setImportStyles'}):link("type
", lQuery("ToolType"))

  lQuery.model.add_link("CompartType", "aa#mirror", "aa#mirrorInv", "CompartType")

  --savienojam tiesos un inversos compartType
  local compartTypeRole =
lQuery("ElemType[id='Association']/compartType[id='Role']/subCompartType"):each(fun
ction(ct)
    local compatTypeInv =
lQuery("ElemType[id='Association']/compartType[id='InvRole']/subCompartType[id = '"
.. ct:attr("id") .. "']"):link("aa#mirrorInv", ct)
  end)

  local compartTypeRole =
lQuery("ElemType[id='Association']/compartType[id='Role']/subCompartType/subCompart
Type/subCompartType"):each(function(ct)
    local compatTypeInv =
lQuery("ElemType[id='Association']/compartType[id='InvRole']/subCompartType/subComp
artType/subCompartType[id = '" .. ct:attr("id") .. "']"):link("aa#mirrorInv", ct)
  end)

  local compartTypeLink =
lQuery("ElemType[id='Link']/compartType[id='Direct']/subCompartType"):each(function
(ct)
    local compatTypeInv =
lQuery("ElemType[id='Link']/compartType[id='Inverse']/subCompartType[caption = '"
.. ct:attr("caption") .. "']"):link("aa#mirrorInv", ct)
  end)

  lQuery("ElemType[id='Association']/compartType[id='Role']"):link("aa#mirror",
lQuery("ElemType[id='Association']/compartType[id='InvRole']"))
  lQuery("ElemType[id='Link']/compartType[id='Direct']"):link("aa#mirror",
lQuery("ElemType[id='Link']/compartType[id='Inverse']"))

  local dependentStylesTable = styleMechanism.dependentStylesTable()
  for i,v in pairs(dependentStylesTable) do
    local pathTable = styleMechanism.split(v[1], "/")
```

```
    local elem--lauks kuram ir translets
    for j,b in pairs(pathTable) do
      if j == 1 then elem = lQuery("ElemType[id='" .. b  .. "']")
      elseif j == 2 then elem = elem:find("/compartType[id='" .. b .. "']")
      else
        elem = elem:find("/subCompartType[id='" .. b .. "']")
      end
    end
    elem:find("/translet[extensionPoint='procFieldEntered']"):attr("procedureName",
"OWLGrEd_UserFields.owl_fields_specific.setDependentStyleSetting")
  end

  local pathContextType = tda.GetProjectPath() ..
"\\Plugins\\OWLGrEd_UserFields\\AutoLoad"
  local fileTable = syncProfile.attrdir(pathContextType)
  for i,v in pairs(fileTable) do
    serialize.import_from_file(v)
    --jaatrod profila vards
    --jaatrod tas profils, kam nav saderibas
    local profileName
    lQuery("AA#Profile"):each(function(obj)
      if lQuery("Extension[id='" .. obj:attr("name") .. "']"):is_empty() then
        profileName = obj:attr("name")
      end
    end)
    local ext = lQuery.create("Extension", {id = profileName, type =
"aa#Profile"})--:link("aa#owner", lQuery("Extension[id = 'OWL_Fields']"))
    lQuery("Extension[id = 'OWL_Fields']"):link("aa#subExtension", ext)
    syncProfile.syncProfile(profileName)
    styleMechanism.syncExtensionViews()
  end
  --importejam AA#ContextType
  local pathConfiguration = tda.GetProjectPath() ..
"\\Plugins\\OWLGrEd_UserFields\\AutoLoadConfiguration"
  local fileTableConfiguration = syncProfile.attrdir(pathConfiguration)
  for i,v in pairs(fileTableConfiguration) do
    serialize.import_from_file(v)
    local configuration = lQuery("AA#Configuration"):first()
    lQuery("AA#ContextType"):each(function(obj)
      obj:remove_link("configuration", obj:find("/configuration"))
    end)
    lQuery("AA#ContextType"):link("configuration", configuration)

    lQuery("AA#TagType"):each(function(obj)
      obj:remove_link("configuration", obj:find("/configuration"))
    end)
    lQuery("AA#TagType"):link("configuration", configuration)

    lQuery("AA#Configuration"):filter(function(obj)
      return obj:id() ~= configuration:id()
    end):delete()

    -- izmest dublikatus
     -- atrast visus AA#ContextType
     -- iziet cauri visiem. Ja ir vairki ar viendu id, tad izdzest to, kam nav
AA#Field
    lQuery("AA#ContextType"):each(function(obj)
      local id = obj:attr("id")
      local eq = lQuery("AA#ContextType[id = '" .. id .."']")
      if eq:size()>1 then
        if obj:find("/fieldInContext"):is_empty() then
          obj:delete()
        else
          eq:filter(function(ct)
            return ct:attr("id") ~= obj:attr("id")
          end):delete()
        end
      end
    end)

    lQuery("AA#TagType"):each(function(obj)
```

```
      local tt = lQuery("AA#TagType[key='" .. obj:attr("key") .. "'][notation='" ..
obj:attr("notation") .. "'][rowType='" .. obj:attr("rowType") .. "']")
      if tt:size()>1 then obj:delete() end
    end)
  end
  print("compele ENDED")
end
```

Profila pārvaldības loga ģenerējoša funkcija

```
--atver profila parvaldibas logu (profileName-profila nosaukums)
function Profile(profileName)
  local close_button = lQuery.create("D#Button", {
    caption = "Close"
  ,eventHandler = utilities.d_handler("Click", "lua_engine",
"lua.OWLGrEd_UserFields.Profile.close()")
  })

  local form = lQuery.create("D#Form", {
  id = "Profile"
  ,caption = "Profile"
  ,buttonClickOnClose = false
  ,cancelButton = close_button
  ,defaultButton = close_button
  ,eventHandler = utilities.d_handler("Close", "lua_engine",
"lua.OWLGrEd_UserFields.Profile.close()")
  ,component = {
    lQuery.create("D#HorizontalBox", {
      id = "HorForm"
      ,minimumWidth = 800
      ,component = {
        lQuery.create("D#VerticalBox", {
          id = "VerticalBoxWithTree"
          ,component = {
            lQuery.create("D#Tree", {
              id = "treeProfile",maximumWidth = 250,minimumWidth =
250,maximumHeight = 500,minimumHeight = 500
              ,treeNode = lQuery.create("D#TreeNode", {
                text = lQuery(prof):attr("name"),id = profId .. "
AA#Profile",childNode = create_items(profId),expanded = true
              })
              ,eventHandler = {utilities.d_handler("TreeNodeSelect", "lua_engine",
"lua.OWLGrEd_UserFields.Profile.treeEvent()")}
            })}
          })
        ,lQuery.create("D#VerticalBox", {id = "property"
          ,component = {
            lQuery.create("D#TabContainer", {
              minimumWidth = 250
              ,component = {
                lQuery.create("D#Tab", {
                  caption = "Profile properties"
                  ,component = {
                    lQuery.create("D#HorizontalBox", {
                      horizontalAlignment = 1
                      ,component = {
                        lQuery.create("D#Label", {caption = "Prefixes",
minimumWidth = 100})
                        ,lQuery.create("D#MultiLineTextBox", {
                          id = "ProfilePrefix"
                          ,textLine = {colectProfilePrefix(prof)}
                          ,eventHandler =
{utilities.d_handler("MultiLineTextBoxChange", "lua_engine",
"lua.OWLGrEd_UserFields.Profile.profilePrefix()")}
                        })}
                    })
                  }
                })
              }
            })
```

```
          }
        })
      }
    })
      ,lQuery.create("D#HorizontalBox", {
        horizontalAlignment = 1
    ,id = "closeForm"
        ,component = {
      lQuery.create("D#VerticalBox", {id = "buttons"})
      ,lQuery.create("D#VerticalBox", {
      id = "closeButton"
      ,horizontalAlignment = 1
      ,component = {close_button}})
      }
      })
    }
  }):link("command", utilities.enqued_cmd("D#Command", {info="ShowModal"}))
    :link("command", utilities.enqued_cmd("D#Command", {info="Delete"}))

  --pievieno ne pirma limena koka konteksta tipus
  createTreeChild(profId)
end
```

Elementa stila uzstādīšanas funkcija

```
-- element - elements, kam jauzstada stils
-- sourceType - stila uzstadisanas avots(Change-stilu definicijas maina profila vai
skatijuma, ViewRemove-tika nomemts skatijums
--      ViewApply - tika pielietots skatijums, ChoiceItem-stilu maina no
choiceItema)
-- sourceInformation - avots no kura ir atkariga stila uzstadisana(var but
choiceItem veca vertiba vai View)
-- external - pazime, ka ir areja stilu uzstadisana
-- parameterTable - pazime, ka stils tiek uzstadits importa laika
function ElemStyleBySettings(element, sourceType, sourceInformation, external,
parameterTable)

  graphDiagramEngine = require("lua_graphDiagram")

  owl_fields_specific2 = "OWLGrEd_UserFields.owl_fields_specific"

  --visi stili kuri ir japiekarto
  local elemStyles =
lQuery(element):find("/elemType/aaElementStyleSetting[isDeleted !=
1]"):map(function(objS)
    --izsaucam proceduru, kas noskaidro vai izpildas visi nosacijumi, lai
pielietotu stilu
    local result = assert(loadstring('return ' .. owl_fields_specific2..
'.'..lQuery(objS):attr("procCondition")..'(...)'))(element, objS, parameterTable)
    if result == true then
      return(objS)
    end
  end)

    --skatijumi, kas ir piekartoti diagramai kuraa ir padotais elements
    local viewW =
lQuery(element):find("/graphDiagram/aa#activeExtension"):map(function(obj)
      return obj
    end)
    --noklusetie skatijumi
    local defaultViews = lQuery("Extension[type='aa#View']"):map(function(obj)
      local l = 0
      obj:find("/aa#graphDiagram"):each(function(gd)
        if gd:id() == element:find("/graphDiagram"):id() then l = 1 end
      end)
      if lQuery("AA#View[name='" .. obj:attr("id") .. "']"):attr("isDefault") ==
"true" and l==0 then
        return obj
      end
    end)
```

```
    local view = lQuery.merge(viewW, defaultViews)

    --sakartojam stilus pec stipruma
    table.sort(elemStyles, function(x,y) return x:attr("strength") <
y:attr("strength") end)

    --sakartojam stilus pec view priaritatem
    local elemStyles2 = {}
    for i = #view,1,-1 do
      for j,b in pairs(elemStyles) do
        if tonumber(b:attr("strength")) == 3 and b:find("/extension"):id() ==
view[i]:id() then table.insert(elemStyles2, b) end
        if tonumber(b:attr("strength")) > 3 then break end
      end
    end
    for j,b in pairs(elemStyles) do
      if tonumber(b:attr("strength")) > 3 then table.insert(elemStyles2, b) end
    end

    --tekosa stila nolasisana

    --atrodam nokluseto stilu, kas ir pirmais pie elementa tipa
    local defStyle = lQuery(element):find("/elemType/elemStyle"):first()

    --noskaidrot stila tipu (node, edge)
    local styleType
    lQuery("NodeStyle"):each(function(obj)
      if lQuery(obj):id() == lQuery(defStyle):id() then styleType = "NodeStyle" end
    end)
    lQuery("EdgeStyle"):each(function(obj)
      if lQuery(obj):id() == lQuery(defStyle):id() then styleType = "EdgeStyle" end
    end)
    --izveidijam junu stilu
    local newStyle = lQuery.create(styleType, {})

    if styleType=="NodeStyle" then
      --ja elementam stils nav saglabats atributa style, tad jaunaja stila
parrakstam nokluseto
      if element:attr("style") == "#" or element:attr("style") == "" then
        local getAttributes = lQuery.model.property_list(styleType)
        for i,v in pairs(getAttributes) do
          lQuery(newStyle):attr(v, lQuery(defStyle):attr(v))
        end
      --ja elementam ir saglabats stils atributa style
      else
        --sadalam style atributa vertibas un ierakstam tas tabulaa
        local t = parseNodeStyle(element:attr("style"))
        --stilu vienumu nosaukumi, kas ir iekodeti style atributa
        local tt = nodeStyleAttribute()
        --stilu vienumu nosaukumi, kas ir ierakstami nodeStyle instance
        local ni = nodeStyleInstance()
        for i,v in pairs(tt) do
          --ierakstam stila vienumu instance
          if ni[v] ~= "" then newStyle:attr(v, t[i]) end
        end
        --atrodam kostes augstumu un platumu no elementa location atributa
        local location = parseLocation(element:attr("location"))
        newStyle:attr("width", location[1])
        newStyle:attr("height", location[2])
      end
    elseif styleType=="EdgeStyle" then
      --ja elementam stils nav saglabats atributa style, tad jaunaja stila
parrakstam nokluseto
      if element:attr("style") == "#" or element:attr("style") == "" then
        local getAttributes = lQuery.model.property_list(styleType)
        for i,v in pairs(getAttributes) do
          lQuery(newStyle):attr(v, lQuery(defStyle):attr(v))
        end
      --ja elementam ir saglabats stils atributa style
      else
        --sadalam style atributa vertibas un ierakstam tas tabulaa
```

```
        local t = parseEdgeStyle(element:attr("style"))
        --stilu vienumu nosaukumi, kas ir iekodeti style atributa
        local tt = edgeStyleAttribute()
        --stilu vienumu nosaukumi, kas ir ierakstami edgeStyle instance
        local ei = edgeStyleInstance()
        for i,v in pairs(tt) do
          --ierakstam stila vienumu instance
          if ei[v] ~= nil then  newStyle:attr(v, t[i]) end
        end
      end
    end

    -- sakam stila klasifikaciju

    --ja ir bijusi areja stilu uzstadisana, vai profila, skatijuma stila maina
    if external == 1 or sourceType == "Change" then
      --atrodam visus dzestos stila uzstadijumus, tiem jaunajaa stila usliekam
noklusetos
      local deletedElemStyles =
lQuery(element):find("/elemType/aaElementStyleSetting[isDeleted =
1]"):map(function(obj)
        return(obj)
      end)
      --ja bija kads lietotaja uzlikts stils, tad tas diemzel pazudis
      for i,v in pairs(deletedElemStyles) do
        if v:attr("setting") == "shapeStyleNoBorder" or v:attr("setting") ==
"shapeStyleShadow" or v:attr("setting") == "shapeStyle3D"
        or v:attr("setting") == "shapeStyleMultiple" or v:attr("setting") ==
"shapeStyleNoBackground" or v:attr("setting") == "shapeStyleNotLinePen" then
          lQuery(newStyle):attr("shapeStyle", defStyle:attr("shapeStyle"))
        else
          lQuery(newStyle):attr(v:attr("setting"),
defStyle:attr(v:attr("setting")))
        end
      end
      --uzliekam jaunus stilus
      setNewElementStyleSetting(elemStyles2, newStyle)

    -- ja tika pielietots view, vai view Tika samainiti vietam, vienkarsi
pasreizejam stilam uzlikt junus stilus
    elseif sourceType == "ViewApply" then
      setNewElementStyleSetting(elemStyles2, newStyle)

    --ja view tika noments, atrast tos stila uzstadijumus, kurus ietekmeja view un
tiem uzlits noklusetas vertibas
    -- uzlikt jaunus stilus
    elseif sourceType == "ViewRemove" then

      local elemTypeId = element:find("/elemType"):id()
      --atradisim visus stila uzstadijumus, kurus ietekmeja nonemtais view
      --sourceInformation ir nonetais skatijums
      local removedViewElemStyles =
sourceInformation:find("/aaElementStyleSetting"):filter(function(obj)
        return obj:find("/elemType"):id() == elemTypeId
      end)

      --uzliekam noklusetas vertibas
      removedViewElemStyles:each(function(obj)
        if obj:attr("setting") == "shapeStyleNoBorder" or obj:attr("setting") ==
"shapeStyleShadow" or obj:attr("setting") == "shapeStyle3D"
        or obj:attr("setting") == "shapeStyleMultiple" or obj:attr("setting") ==
"shapeStyleNoBackground" or obj:attr("setting") == "shapeStyleNotLinePen" then
          newStyle:attr("shapeStyle", defStyle:attr("shapeStyle"))
        else
          newStyle:attr(obj:attr("setting"), defStyle:attr(obj:attr("setting")))
        end
      end)

      --uzliekam jaunus stilus
      setNewElementStyleSetting(elemStyles2, newStyle)
```

```
    -- ja tika mainits choiceItems, visiem stiliem, ko ietekmeja veca vertiba
uzlikt noklusetas vertibas, tiem ko ietekme jauna vertiba-jaunas
    elseif sourceType == "ChoiceItem" then
      local elemTypeId = element:find("/elemType"):id()
      --sourceInformation - vecais choiceItem
      --atrodam visas stila instances, kas ir piesaistitas vecajam choiceItem
      choiceItemElemStyle =
sourceInformation:find("/aaElementStyleSetting"):filter(function(obj)
        return obj:find("/elemType"):id() == elemTypeId
      end)
      --uzliekam noklusetas vertibas
      choiceItemElemStyle:each(function(obj)
        if obj:attr("setting") == "shapeStyleNoBorder" or obj:attr("setting") ==
"shapeStyleShadow" or obj:attr("setting") == "shapeStyle3D"
        or obj:attr("setting") == "shapeStyleMultiple" or obj:attr("setting") ==
"shapeStyleNoBackground" or obj:attr("setting") == "shapeStyleNotLinePen" then
          newStyle:attr("shapeStyle", defStyle:attr("shapeStyle"))
        else
          newStyle:attr(obj:attr("setting"), defStyle:attr(obj:attr("setting")))
        end
      end)

      --uzliekam jaunus stilus
      setNewElementStyleSetting(elemStyles2, newStyle)
    end

    --pielietojam stilu
    local diagram = lQuery(element):find("/graphDiagram")
    --ja stils tiek piekartots atvertai diagrammai, vai ari stili uzstadas no
imports, tad ir jalieto stila parlinkisana
    if graphDiagramEngine.IsOpenDiagram( diagram:id() ) == "open" or
(parameterTable~=nil and parameterTable['import']=="true") then
      if element:find("/elemStyle/elemType"):id() ~= element:find("/elemType"):id()
then lQuery(element):find("/elemStyle"):delete()
      elseif lQuery(element):find("/elemStyle"):id() ~= defStyle:id() then
        lQuery(element):find("/elemStyle"):delete()
      end
      element:remove_link("elemStyle")
      element:link("elemStyle",newStyle)
      element:attr("style","#")
    --citos gadijumos var lietot UpdateStyleCmd komandu
    else
      local cmd = lQuery.create("UpdateStyleCmd")
      cmd:link("element", element)
        :link("elemStyle", newStyle)
        :link("graphDiagram", diagram)
        utilities.execute_cmd_obj(cmd)
      utilities.execute_cmd("SaveDgrCmd", {graphDiagram = diagram})
      newStyle:delete()
    end
end
```

Stilu uzstādīšanas nosacījumu pārbadies funkcijas

```
--noskaidro vai elementam ir japiekarto dotais stils no choiceItem (element-
elements, elemStyleSetting-stila uzstadijums)
function setElemStyleByChoiceItem(element, elemStyleSetting, parameterTable)
  local elemType = element:find("/elemType"):attr("id")
  local result = false
  local compartment =
lQuery(elemStyleSetting):find("/choiceItem/compartType/compartment"):filter(
    function(obj)
      if lQuery(obj):find("/element"):is_empty() then
        return lQuery(obj):find("/parentCompartment/element"):id() ==
lQuery(element):id()
      else
        return lQuery(obj):find("/element"):id() == lQuery(element):id()
      end
    end)
  lQuery(compartment):each(function(obj)
```

```
    local compartValue = lQuery(obj):attr("value")
    lQuery(elemStyleSetting):find("/choiceItem"):each(function(objI)
      if compartValue == lQuery(objI):attr("value") then result = true end
    end)
  end)
  return result
end

--noskaidro vai kompartmentam ir japiekarto dotais tils no choiceItem (compartment-
kompartments, compartStyleSetting-stila uzstadijums)
function setCompartStyleByChoiceItem(compartment, compartStyleSetting)
  local result = false
  local elemTypeAssociation =
lQuery(compartment):find("/parentCompartment/element/elemType"):attr("id")
  local elemTypeClassObject =
lQuery(compartment):find("/element/elemType"):attr("id")

  if lQuery(compartment):find("/parentCompartment"):is_not_empty() and
lQuery(compartment):find("/parentCompartment"):attr("isGroup")=="true" then
    local element = lQuery(compartment):find("/parentCompartment")
    local subCompartment =
lQuery(compartStyleSetting):find("/choiceItem/compartType/compartment"):filter(
      function(obj)
        return lQuery(obj):find("/parentCompartment"):id() == lQuery(element):id()
      end)
    lQuery(subCompartment):each(function(obj)
      local compartValue = lQuery(obj):attr("value")
      lQuery(compartStyleSetting):find("/choiceItem"):each(function(objI)
        if compartValue == lQuery(objI):attr("value") then result = true end
      end)
    end)
  elseif lQuery(compartment):find("/element"):is_not_empty() then
    local element = lQuery(compartment):find("/element")
    local subCompartment =
lQuery(compartStyleSetting):find("/choiceItem/compartType/compartment"):filter(
      function(obj)
        return lQuery(obj):find("/element"):id() == lQuery(element):id()
      end)
    lQuery(subCompartment):each(function(obj)
      local compartValue = lQuery(obj):attr("value")
      lQuery(compartStyleSetting):find("/choiceItem"):each(function(objI)
        if compartValue == lQuery(objI):attr("value") then result = true end
      end)
    end)
  end
  return result
end

--noskaidro vai elementam ir japiekarto dotais stils no skatijuma (element-
elements, elemStyleSetting-stila uzstadijums, parameterTable-tabula ar parametriem)
function setElemStyleByExtension(element, elemStyleSetting, parameterTable)
  local result = false
  local graphDiagram = element:find("/graphDiagram")
  local extension = elemStyleSetting:find("/extension")
  if extension:attr("id") == "Default" then result = true end
  graphDiagram:find("/aa#activeExtension"):each(function(ext)
    if extension:id() == ext:id() and element:find("/elemType"):id() ==
elemStyleSetting:find("/elemType"):id() then result = true end
  end)
  local l = 0
  extension:find("/aa#graphDiagram"):each(function(gd)
    if gd:id() == element:find("/graphDiagram"):id() then l = 1 end
  end)
  if lQuery("AA#View[name='" .. extension:attr("id") .. "']"):attr("isDefault") ==
"true" and l then result = true end
  if parameterTable~=nil and parameterTable["import"] == "true" and
elemStyleSetting:attr("setting")=="widthProc" then result = true end
  if (parameterTable==nil or parameterTable["import"] ~= "true") and
elemStyleSetting:attr("setting")=="widthProc" then result = false end
  return result
end
```

Aksiomu ģenerēšanas funkcija

```
--savac visas aksiomas prieks eksporta (diagram-eksportejama diagrama) (Class,
Attribute, Association, Object)
function axiom(diagram)
  String = ""
  local tag = lQuery("Tag[key = 'owl_Field_axiom']"):each(function(obj)
    local compartment = lQuery(obj):find("/type/compartment")
    compartment:each(function(objCom)
      local replacementValue = objCom:attr("value")
      local compartType = objCom
      local l = 0
      while l==0 do
        if lQuery(compartType):find("/compartType/elemType"):is_not_empty() then --
ja ir elements
          l=1
          if lQuery(compartType):find("/element/elemType"):attr("caption")==
"Class" and lQuery(compartType):find("/element/graphDiagram"):id() == diagram:id()
then
            local replacementSubject =
lQuery(compartType):find("/element/compartment/subCompartment:has(/compartType[id='
Name'])"):attr("value")
            local namespace
            if
lQuery(compartType):find("/element/compartment/subCompartment:has(/compartType[id='
Namespace'])"):is_not_empty() then
              namespace =
lQuery(compartType):find("/element/compartment/subCompartment:has(/compartType[id='
Namespace'])"):attr("value")
              if namespace~="" then replacementSubject = replacementSubject .. "{"
.. namespace .. "}" end
            end
            if replacementSubject~="" and replacementValue~="" then
              String = String .. "\n" .. gramar(obj:attr("value"),
replacementSubject, replacementValue)
            end
          elseif lQuery(compartType):find("/element/graphDiagram"):id() ==
diagram:id() then

lQuery(compartType):find("/element/compartment:has(/compartType/subCompartType[id='
Name'])"):each(function(objC)
              replacementSubject = lQuery(objC):attr("value")
              if replacementSubject~=""  and replacementValue~="" then
                String = String .. "\n" .. gramar(obj:attr("value"),
replacementSubject, replacementValue)
              end
            end)
          end
        elseif
lQuery(compartType):find("/parentCompartment/compartType"):attr("id") == "InvRole"
or lQuery(compartType):find("/parentCompartment/compartType"):attr("id") == "Role"
then
          l=1
          if
lQuery(compartType):find("/parentCompartment/element/graphDiagram"):id() ==
diagram:id() then
            replacementSubject =
lQuery(compartType):find("/parentCompartment/subCompartment/subCompartment:has(/com
partType[id='Name'])"):attr("input") ..
lQuery(compartType):find("/parentCompartment/subCompartment/subCompartment:has(/com
partType[id='Namespace'])"):attr("input")
            if replacementSubject~="" and replacementValue~="" then
              String = String .. "\n" .. gramar(obj:attr("value"),
replacementSubject, replacementValue)
            end
          end
        elseif
lQuery(compartType):find("/parentCompartment/compartType"):attr("id") ==
"Attributes" then
```

```
          l=1
          if
lQuery(compartType):find("/parentCompartment/parentCompartment/element/graphDiagram
"):id() == diagram:id() then
            replacementSubject =
lQuery(compartType):find("/parentCompartment/subCompartment/subCompartment:has(/com
partType[id='Name'])"):attr("value")
            local namespace
            if
lQuery(compartType):find("/parentCompartment/subCompartment/subCompartment:has(/com
partType[id='Namespace'])"):is_not_empty() then
              namespace =
lQuery(compartType):find("/parentCompartment/subCompartment/subCompartment:has(/com
partType[id='Namespace'])"):attr("value")
              if namespace~="" then replacementSubject = replacementSubject .. "{"
.. namespace .. "}" end
            end
            if replacementSubject~=""  and replacementValue~="" then
              String = String .. "\n" .. gramar(obj:attr("value"),
replacementSubject, replacementValue)
            end
          end
        else compartType = lQuery(compartType):find("/parentCompartment")  end
      end
    end)
    if lQuery(obj):find("/choiceItem"):is_not_empty() then
      local ciValue =  lQuery(obj):find("/choiceItem"):attr("value")
      local ciCompartment =
lQuery(obj):find("/choiceItem/compartType/compartment[value='" .. ciValue .. "']")
      ciCompartment:each(function(objCom)
        local compartType = objCom
        local replacementValue = objCom:attr("value")
        local l = 0
        while l==0 do
          if lQuery(compartType):find("/compartType/elemType"):is_not_empty() then
            l=1
            if lQuery(compartType):find("/element/elemType"):attr("caption")==
"Class" and lQuery(compartType):find("/element/graphDiagram"):id() == diagram:id()
then
              local replacementSubject =
lQuery(compartType):find("/element/compartment/subCompartment:has(/compartType[id='
Name'])"):attr("value")
              local namespace
              if
lQuery(compartType):find("/element/compartment/subCompartment:has(/compartType[id='
Namespace'])"):is_not_empty() then
                namespace =
lQuery(compartType):find("/element/compartment/subCompartment:has(/compartType[id='
Namespace'])"):attr("value")
                if namespace~="" then replacementSubject = replacementSubject ..
"{" .. namespace .. "}" end
              end
              if replacementSubject~="" and replacementValue~="" then
                String = String .. "\n" .. gramar(obj:attr("value"),
replacementSubject, replacementValue)
              end
            elseif lQuery(compartType):find("/element/graphDiagram"):id() ==
diagram:id() then

lQuery(compartType):find("/element/compartment:has(/compartType/subCompartType[id='
Name'])"):each(function(objC)
                replacementSubject = lQuery(objC):attr("value")
                if replacementSubject~="" and replacementValue~="" then
                  String = String .. "\n" .. gramar(obj:attr("value"),
replacementSubject, replacementValue)
                end
              end)
            end
          elseif
lQuery(compartType):find("/parentCompartment/compartType"):attr("id") == "InvRole"
```

```
or lQuery(compartType):find("/parentCompartment/compartType"):attr("id") == "Role"
then
            l=1
            if
lQuery(compartType):find("/parentCompartment/element/graphDiagram"):id() ==
diagram:id() then
              replacementSubject =
lQuery(compartType):find("/parentCompartment/subCompartment/subCompartment:has(/com
partType[id='Name'])"):attr("value")
              local namespace
              if
lQuery(compartType):find("/parentCompartment/subCompartment/subCompartment:has(/com
partType[id='Namespace'])"):is_not_empty() then
                namespace =
lQuery(compartType):find("/parentCompartment/subCompartment/subCompartment:has(/com
partType[id='Namespace'])"):attr("value")
                if namespace~="" then replacementSubject = replacementSubject ..
"{" .. namespace .. "}" end
              end
              if replacementSubject~=""  and replacementValue~="" then
                String = String .. "\n" .. gramar(obj:attr("value"),
replacementSubject, replacementValue)
              end
            end
          elseif
lQuery(compartType):find("/parentCompartment/compartType"):attr("id") ==
"Attributes" then
            l=1
            if
lQuery(compartType):find("/parentCompartment/parentCompartment/element/graphDiagram
"):id() == diagram:id() then
              replacementSubject =
lQuery(compartType):find("/parentCompartment/subCompartment/subCompartment:has(/com
partType[id='Name'])"):attr("value")
              local namespace
              if
lQuery(compartType):find("/parentCompartment/subCompartment/subCompartment:has(/com
partType[id='Namespace'])"):is_not_empty() then
                namespace =
lQuery(compartType):find("/parentCompartment/subCompartment/subCompartment:has(/com
partType[id='Namespace'])"):attr("value")

                if namespace~="" then  replacementSubject = replacementSubject ..
"{" .. namespace .. "}" end
              end
              if replacementSubject~="" and replacementValue~="" then
                String = String .. "\n" .. gramar(obj:attr("value"),
replacementSubject, replacementValue)
              end
            end
          else compartType = lQuery(compartType):find("/parentCompartment") end
        end
      end)
    end
  end)
  local ontologyFragment =
lQuery(diagram):find("/element/target"):each(function(obj)
    String = String ..  axiom2(String, obj)
  end)
  return String
end
```

Dokumentārā lapa

Bakalaura darbs ***„Lietotāja definētu lauku mehānisms TDA rīku būves platformā***” izstrādāts Latvijas Universitātes Datorikas fakultātē.

Ar savu parakstu apliecinu, ka iesniegto bakalaura darbu es esmu veikusi patstāvīgi un esmu izmantojusi tikai tajā norādītos palīglīdzekļus.

Autors: ***Jūlija Ovčiņņikova*** _____ __**.05.2012**.

Rekomendēju darbu aizstāvēšanai
Darba vadītājs: ***Dr.Dat. Kārlis Čerāns*** _____ __**.05.2012.**

Recenzents: ***Dr.Dat. Jānis Bičevskis***

Darbs iesniegts Datorikas fakultātē __.05.2012.

Metodiķe: Ārija Sproģe

Darbs aizstāvēts bakalaura gala pārbaudījuma komisijas sēdē

Komisijas sekretārs(-e):