



УНИВЕРЗИТЕТ У НОВОМ САДУ

ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА



УНИВЕРЗИТЕТ У НОВОМ САДУ

ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

НОВИ САД

Департман за рачунарство и аутоматику

Одсек за рачунарску технику и рачунарске комуникације

ЗАВРШНИ (BACHELOR) РАД

Кандидат: Милан Станкић

Број индекса: е12345

Тема рада: Једно решење графичког алата за генерисање конфигурационих датотека за повезивач са апсолутним пуњачем

Ментор рада: Проф. др Мирослав Поповић

Нови Сад, јун 2011. године



КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, РБР:			
Идентификациони број, ИБР:			
Тип документације, ТД:	Монографска документација		
Тип записа, ТЗ:	Текстуални штампани материјал		
Врста рада, ВР:	Завршни (Bachelor) рад		
Аутор, АУ:	Милан Станкић		
Ментор, МН:	Др Мирослав Поповић		
Наслов рада, НР:	Једно решење графичког алата за генерирање конфигурационих датотека за повезивач са апсолутним пуњачем		
Језик публикације, ЈП:	Српски / латиница		
Језик извода, ЈИ:	Српски		
Земља публиковања, ЗП:	Република Србија		
Уже географско подручје, УГП:	Војводина		
Година, ГО:	2011		
Издавач, ИЗ:	Ауторски репринт		
Место и адреса, МА:	Нови Сад; трг Доситеја Обрадовића 6		
Физички опис рада, ФО: (поглавља/страна/цитата/табела/ слика/графика/прилога)	8/33/0/0/20/0/0		
Научна област, НО:	Електротехника и рачунарство		
Научна дисциплина, НД:	Рачунарска техника		
Предметна одредница/Кључне речи, ПО:	Повезивач, апсолутни пуњач, меморијска мала, графички едитор		
УДК			
Чува се, ЧУ:	У библиотеци Факултета техничких наука, Нови Сад		
Важна напомена, ВН:			
Извод, ИЗ:	У овом раду је описано једно решење имплементације графичког едитора конфигурационих датотека за повезивач са апсолутним пуњачем. Графички едитор је имплементиран кориштење Eclipse пакета са окружењем за графичко моделовање, GMF (Graphical Modeling Framework). Кроз појединачно поглавље врши се упознавање са GMF окружењем и потребним алатима.		
Датум прихватања теме, ДП:			
Датум одбране, ДО:			
Чланови комисије, КО:	Председник:	др Михајло Катона	
	Члан:	др Јелена Ковачевић	Потпис ментора
	Члан, ментор:	др Мирослав Поповић	



KEY WORDS DOCUMENTATION

Accession number, ANO:		
Identification number, INO:		
Document type, DT:	Monographic publication	
Type of record, TR:	Textual printed material	
Contents code, CC:	Bachelor Thesis	
Author, AU:	Milan Stankić	
Mentor, MN:	Miroslav Popović, PhD	
Title, TI:	One solution of graphical tool for generating configuration files for the linker with the absolute loader	
Language of text, LT:	Serbian	
Language of abstract, LA:	Serbian	
Country of publication, CP:	Republic of Serbia	
Locality of publication, LP:	Vojvodina	
Publication year, PY:	2011	
Publisher, PB:	Author's reprint	
Publication place, PP:	Novi Sad, Dositeja Obradovica sq. 6	
Physical description, PD: (chapters/pages/ref./tables/pictures/graphs/appendices)	8/33/0/0/20/0/0	
Scientific field, SF:	Electrical Engineering	
Scientific discipline, SD:	Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, S/KW:	Linker, absolute loader, memory map, graphical editor	
UC		
Holding data, HD:	The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, N:		
Abstract, AB:	This paper describes one solution of graphical tool for generating configuration files for the linker with the absolute loader. Graphical editor is implemented using Eclipse package with a graphical environment for modeling, GMF (Graphical Modeling Framework). Through individual chapter shall be meeting with the GMF environment and necessary tools.	
Accepted by the Scientific Board on, ASB:		
Defended on, DE:		
Defended Board, DB:	President:	Mihajlo Katona, PhD
	Member:	Jelena Kovečević, PhD
	Member, Mentor:	Miroslav Popović, PhD
		Menthor's sign

SADRŽAJ

1.	Uvod.....	1
2.	Teorijske osnove	3
2.1	Povezivač sa absolutnim punjačem.....	3
2.2	Memorijska mapa nivoa	5
2.3	Format mem.xml datoteka.....	6
3.	Programsko okruženje	10
4.	Koncept rešenja.....	11
4.1	Metamodel i domen generator model.....	12
4.2	Model alata.....	13
4.3	Grafički model.....	14
4.4	Model povezivanja	15
4.5	Generator model.....	15
4.6	Integracija u CLIDE	16
5.	Implementacija.....	18
5.1	Memorymapeditor.....	19
5.2	Memorymapeditor.edit	21
5.3	Memorymapeditor.editor.....	21
5.4	Memorymapeditor.diagram.....	22
5.5	MemoryMapParser.....	23
5.6	MemoryMapWriter	24
5.7	Editor i integracija u CLIDE	24
6.	Verifikacija	26
7.	Zaključak	27
8.	Literatura.....	29

SPISAK SLIKA

Slika 1 - Struktura pokrivača.....	4
Slika 2 - Podela memorije	5
Slika 3 - Podela statičke memorije.....	6
Slika 4 - GMF razvojni tok	10
Slika 5 - GMF razvojna ploča	11
Slika 6 - Ecore dijagram.....	12
Slika 7 - Metamodel (ecore model).....	13
Slika 8 - Domen generator model	13
Slika 9 - Model alata	14
Slika 10 - Grafički model.....	14
Slika 11 - Model povezivanja	15
Slika 12 - Generator model	16
Slika 13 - Paketi modela	19
Slika 14 - UML sprega modela	19
Slika 15 - UML klasa modela	20
Slika 16 - UML klasa modela sa međusobnim vezama	20
Slika 17 - UML klasa rukovaoca	21
Slika 18 - Klase paketa memory	21
Slika 19 - Paketi dijagrama	22
Slika 20 - UML dijagram klasa pakete diagram.edit.part	23
Slika 21 - Grafički editor.....	24
Slika 22 - Sistem događaja, GEF i GMF	25

SKRAĆENICE

GMF	- <i>Graphical Modeling Framework</i>
EMF	- <i>Eclipse Modeling Framework</i>
GEF	- <i>Graphical Editing Framework</i>
MVC	- <i>Model-View-Control</i>

1. Uvod

Ovaj rad predlaže jedno rešenje za implementaciju grafičkog editora memorijskih mapa. Grafički editor će biti implementiran korištenjem Eclipse paketa za modelovanje grafičkog okruženja, GMF.

Povezivač koji proizvodi sliku programa za apsolutni punjač ima složen zadatak. Potrebno je omogućiti punjenje više različitih modula za obradu signala u isto vreme. Ovaj problem se rešava podelom memorije DSP-a na zone koje su organizovane hijerarhijski. Svaka zona može primiti jedan modul koji vrši tip obrade za koju je zona namenjena. Na osnovu datoteke koja opisuje zone povezivač proizvodi sliku modula koju je moguće puniti zajedno sa drugim modulima koji vrše drugi tip obrade, i upisuju se u drugu zonu.

Datoteke sa opisom zona postaju komplikovanije sa porastom broja modula, pa se javlja potreba za grafičkim alatom za proizvodnju i prilagođavanje ovih datoteka. Ovaj grafički alat bi trebao da da sliku podele memorije.

Razvoj programske podrške na osnovu modela (engl. *Model-Driven Software Development*) je sve popularniji poslednjih godina. Osnovni cilj industrije je da se stvore alati koji omogućavaju generisanje kompletног koda za rešavanje datog problema na osnovu odgovarajućeg modela. Ova automatizacija razvoja dovodi do povećanja kvaliteta koda, smanjenja programerskih grešaka pa samim tim i povećenjem produktivnosti.

Nekoliko Eclipse projekata omogućava generisanje koda na osnovu modela. Eclipse Modeling Framework (EMF) pruža mogućnosti modelovanja i generisanja koda za Eclipse aplikacije koje su zasnovane na strukturnim modelima podataka. Na modele iz EMF-a naslanjaju se GEF komponente koje omogućavaju realizaciju osnovnih i naprednih funkcionalnosti editora zasnovanih na MVC (Model-View-Control) arhitekturi.[1]

Graphical Modeling Framework (GMF) komponenta na osnovu modela vrši generisanje celokupnog koda za grafički editor. Unutar GMF se nalaze delovi u kojima je sadržan EMF i modeli struktura podataka koje će se koristiti, kao i delovi sa GEF komponentama vezanim za prikaz editora i kontrolu.

2. Teorijske osnove

2.1 Povezivač sa absolutnim punjačem

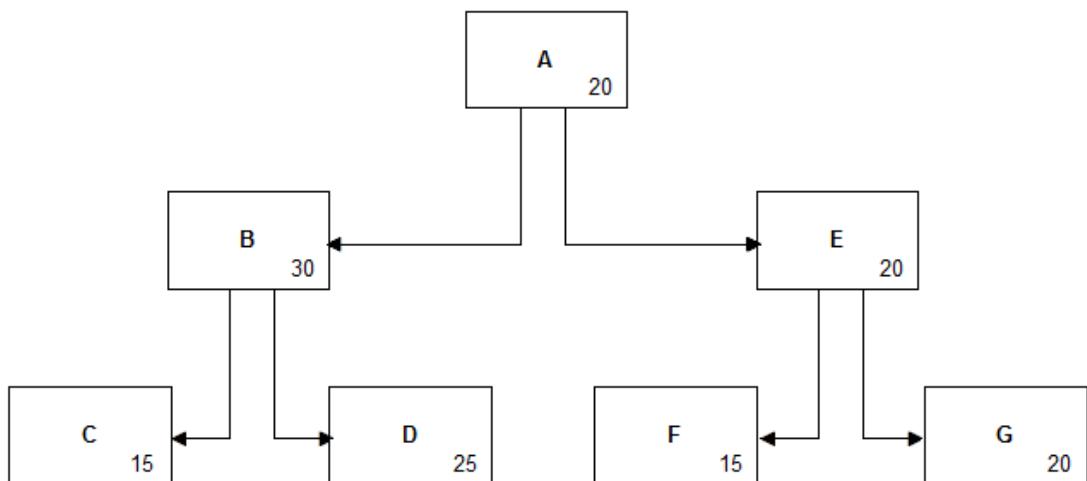
Programi koje generiše asemblerski ili neki drugi program višeg nivoa, pisani su tako da im je svaka instrukcija pisana relativno u odnosu na prvu instrukciju programa, čija je adresa nula. To znači da svi programi imaju početnu adresu nula, što je u multiprogramskim uslovima nemoguće realizovati. Da bi se ovi programi mogli izvršiti definiše se posebna komponenta programske podrške čiji je zadatak da puni programe na različite adrese u operativnoj memoriji, odakle ih procesor računarskog sistema izvršava.

Povezivač je program koji obavlja vezivanje svih programa i potprograma zajedno. Međutim, umesto smeštanja relokacionog i vezanog teksta direktno u memoriju, on smešta tu informaciju u obliku datoteke na spoljnu memoriju. Izlazna datoteka je smeštena na spoljnu memoriju i uobičajen naziv za nju je modul punjenja. Punjač modula fizički puni modul punjanja u memoriju. Povezivač obavlja funkciju dodele, relokacije i spajanja, a punjač modula samo funkciju punjenja.

Postoje dve klase povezivača. Najprostiji tip povezivača razvija modul punjenja sličan izvršivom kodu kod absolutnog punjača. To označava da se vrši dodela specifične memorijске zone programu u vreme kad su programi i potprogrami vezani zajedno. Pošto modul ovog tipa liči na sliku memorije, naziva se i modulom memorijске slike.[2]

U najvećem broju slučajeva svi potrebni programi i programi se pune u memoriju u isto vreme. U tom slučaju ukupna vrednost potrebne memorije za sve programe i potprograme može da pređe raspoloživu vrednost memorije, što je čest slučaj kod velikih programa, pa se mogu javiti poteškoće.

Ukoliko se eksplisitno zna koji program i potprogrami pozivaju druge, moguće je realizovati strukturu POKRIVAČA (overlay) kojima se identifikuju međusobno isključivi potprogrami. Ideja prekrivača je prikazana na Slika 1.



Slika 1 - Struktura pokrivača

Ukoliko bi svi programi bili u memoriji istovremeno, bila bi potrebna zona od 160 memorijskih jedinica. Na šemici se vidi da program A poziva potprograme B i E, program B poziva potprograme C i D, a E poziva F i G. Uočava se da B i E neće nikad biti zajedno potrebni u memoriji, kao ni C i D, odnosno F i G. Drugim rečima, u najgorem slučaju, istovremeno je potreban memorijski prostor za programe A, B i D, u iznosu od 75 memorijskih jedinica, što predstavlja znatnu uštedu memorijskog prostora.

Ideja sa pokrivačima našla je primenu kod pojedinih DSP procesora. Ovaj rad se odnosi konkretno na editor memorijskih mapa za familiju DSP procesora kompanije *Cirrus Logic* (CS47XXX, CS48LXX, CS485XX, CS497XX, CS498XX i CS4953X). Sistemska programska podrška za navedene familije procesora, u odnosu na vrstu aplikacije, posmatra se u pet namenskih nivoa. To su operativni sistem, dekoderski nivo, nivo među-obrade (Mid-Processor Module), nivo završne obrade (Post-Processor Module) i nivo procesa virtualizacije (Virtual Processing Module).

Svakom nivou je dodeljena jedinstvena zona (segment) radne memorije. Time je omogućeno nezavisno učitavanje svakog nivoa u programsku memoriju nezavisno od drugih modula, a time i dobijanje željene kombinacije različitih tipova obrade. Ovim se smanjuje veličina potrebne ROM memorije, jer se čuvaju samo pojedinačni moduli umesto svih predviđenih kombinacija modula. Tako na primer ako je potrebna promena dekodera usled

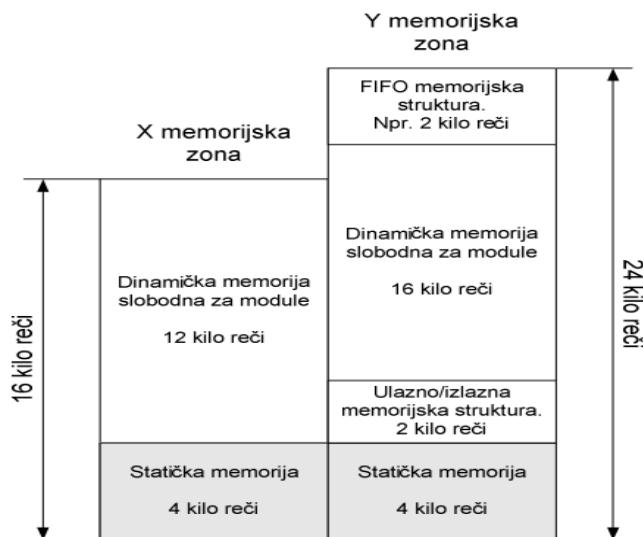
promene podataka na ulazu u sistem, nije neophodno ponovno učitavanje OS-a, ili bilo kojih drugih modula u nivoima među i završne obrade. Neka je na DSP-ju dekodovan mp3 ulazni vektor i neka su pored mp3 dekodera prisutni OS (os nivo) i BassManagement (ppm nivo) moduli. Neka se u jednom trenutku na ulasku u sistem nađe test vektor u aac formatu, tada zbog postojanja nivoa neće biti potrebno na DSP ponovo učitati OS i BassManagement zajedno sa aac modulom, već će se samo izvršiti promena modula u dekoderskom nivou.

2.2 Memorija nivoa

Svaka od navedenih familija procesora ima poboljšanu Harvard arhitekturu, tj. ima dve odvojene memorije za podatke, X i Y, i programsku memoriju. Njihova veličina varira u zavisnosti od pod verzije procesora. Programskim okruženjem (engl. framework) je dodatno uvedena logička podela memorije na dva segmenta:

1. memorija rezervisana za statičku alokaciju
2. dinamički alocirana memorija (heap) koja se dodeljuje pojedinim modulima na njihov zahtev (malloc).

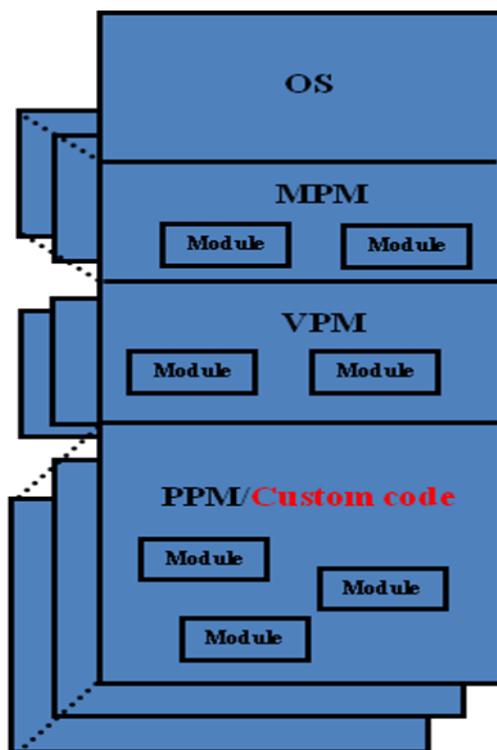
Slika 2 prikazuje primer podele memorije.



Slika 2 - Podela memorije

Statička memorija je podeljena ne segmente koji su dodeljeni odgovarajućim nivoima programske podrške. Veličina statičke memorije koja je dodeljena segmentu zavisi od verzije programskog okruženja, tipa ogruženja (single/dual decode - sd/dd) i DSP jezgra na koje se odnose (A/B), ... U zavisnosti od odabrane verzije programskog okruženja postoje definisane

*.mem.xml datoteke koje opisuje podelu statičke memorije. Statička podela memorije je data na Slika 3.



Slika 3 - Podela statičke memorije

2.3 Format mem.xml datoteka

Opis memorijskih mapa je dat u XML formatu. Ova datoteka definiše memorijski prostor koji je dodeljen modulima. Celokupan memorijski prostor je izdeljen na opsege koji su definisani klasama. Klasa opisuje jedan memorijski opseg. Klasa sadrži:

1. početnu adresu u memorijskom prostoru
2. krajnju adresu u memorijskom prostoru ili veličinu memorije
3. listu aliasa
4. listu podklasa
5. vezu ka srodnim zonama

Klase su hijerarhijski struktuirane. Svaka klasa može imati podklase i nadklasu. Klasa takođe može biti i član pokrivača (overlay). Svaka mem.xml datoteka kao koren mora imati tag <memory_map>. Ovaj tag može sadržati jedan ili više <class> tagova u okviru kojih su opisane klase. Ove klase najčešće opisuju fizički odvojene memorije.

Za *Cirrus Logic* DSP familiju procesora definisane su četiri memorijske zone; X, Y, L i P. Ove memorijske zone su definisane kao top klase. X i Y memorije su 32-bitne memorijske zone koje su predviđene za podatke. P je memorijska zona koja je predviđena za programsку

memoriju. L memorijska zona je 64-bitna memorija i obuhvata X i Y memorijske blokove sa istim adresama. Veza između L i X-Y memorije može biti definisana na nekoliko načina.

Jednostavna mem.xml datoteka je data u nastavku.

```
<memory_map>
    <class name="CODE" image_section="P" output_type="RAM">
        <start>0x0</start>
        <size>0xf000</size>
    </class>
    <class name="X" image_section="X" output_type="RAM">
        <start>0x0</start>
        <size>0xF000</size>
    <class name="Y" image_section="Y" output_type="RAM">
        <start>0x0</start>
        <size>0xF000</size>
    <class name="L" image_section="L" output_type="RAM">
        <xzone>X</xzone>
        <yzone>Y</yzone>
    </class>
</memory_map>
```

Kao što se može videti iz primera klase su jednostavne za definisanje i čitanje. U navedenom primeru ne postoje podklase, već samo glavne nezavisne zone. L memorijska zona je definisana preko tagova `<xzone>` i `<yzone>` i ona je definisana preko cele X odnosno Y zone.

```
<class name="CODE" image_section="P" output_type="RAM">
    <start>0x0</start>
    <size>0x2000</size>
    <subclass>
        <class name="CODE_OS" >
            <start>CODE.s</start>
            <size>0x200</size>
        </class>
        <class name="CODE_DEC" >
            <start>$</start>
            <size>3072</size>
        </class>
        <class name="CODE_MPM" >
            <start>$</start>
            <end>0x1cfe</end>
        </class>
        <class name="CODE_PPM" >
            <start>$</start>
            <size>1</size>
        </class>
    </subclass>
</class>
```

U prethodno navedenom primeru vidi se definisanje podklasa. Memorijski prostor koji zauzima osnovna klasa CODE izdeljen je na klase CODE_OS, CODE_DEC, CODE_MPM i CODE_PPM. U podklasi CODE_OS za postavljanje početka memorijskog segmenta korišten je simbol CODE.s. `<ClassName>.s` i `<ClassName>.e` su simboli koji predstavljaju adrese početka, odnosno kraja memorijskom segmentu dodeljenog klasi `ClassName`. Početna adresa podklase

CODE_MPM je definisana simbolom \$. Simbol \$ predstavlja adresu kraja poslednjeg definisanog memorijskog segmenta uvećenu za 1. Iz gornjeg primera se primećuje da je definisanje vrednosti adresa moguće korišćenjem dekadnog, odnosno heksadecimalnog brojnog sistema.

Naredni primer pokazuje upotrebu aliasa.

```
<class name="CODE_DEC_JTABLE" >
  <start>0x1F70</start>
  <end>0x1FFD</end>
  <aliases>
    <alias name="CODE_AAC_JTABLE" />
    <alias name="CODE_AC3_JTABLE" />
    <alias name="CODE_DTS_JTABLE" />
    <alias name="CODE_SGEN_JTABLE" />
    <alias name="CODE_HDCD_JTABLE" />
  </aliases>
</class>
```

Dodatno alias omogućava smeštanje prevedenog koda u odgovarajući segment statičke memorije bez izmena na izvornom kodu. Jednostavnim definisanjem promenljive overlay_type prilikom poziva povezivača vrši se izbor segmenta u koji će se smestiti prevedeni izvorni kod. Broj aliasa za jednu klasu nije ograničen.

Projekti generisane C prevodiocem u svojim segmentima emituju posebne aliase CODE_OVLY, XDATA_OVLY, YDATA_OVLY i LDATA_OVLY. Ovi aliasi su definisane u specijalnom <supported_overlays> tagu.

```
<supported_overlays>
  <overlay name="dec">
    <aliases>
      <alias name="CODE_OVLY" class="CODE_DEC"/>
      <alias name="LDATA_OVLY" class="L_DEC"/>
      <alias name="XDATA_OVLY" class="X_DEC"/>
      <alias name="YDATA_OVLY" class="Y_DEC"/>
    </aliases>
  </overlay>
  <overlay name="mpm">
    <aliases>
      <alias name="CODE_OVLY" class="CODE_MPM"/>
      <alias name="LDATA_OVLY" class="L_MPM"/>
      <alias name="XDATA_OVLY" class="X_MPM"/>
      <alias name="YDATA_OVLY" class="Y_MPM"/>
    </aliases>
  </overlay>
</supported_overlays>
```

```
</aliases>
</overlay>
</supported_overlays>
```

U okviru definisanja klase koje opisuju memoriske segmnte moguće je koristiti matematičke izraze.

- Simboli u izrazima:

Početnoj i krajnjoj adresi se može pristupiti preko imena klase kome sledi .s za početak ili .e za kraj. Znak \$ predstavlja trenutnu memorisku adresu uvećanu za jedan. Definiše se kao <poslednja definisana klasa>.e + 1.

- Aritmetičke operacije u izrazima:

Sabiranje i oduzimanje mogu koristiti brojeve ili simbole.

- Funkcije u izrazima:

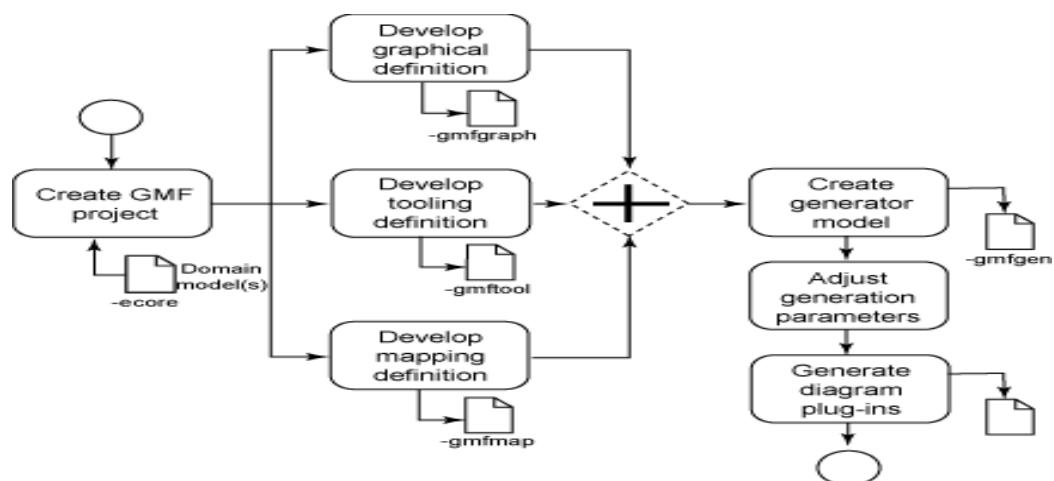
min(<izraz>,<izraz>) - vraća minimalnu vrednost

max(<izraz>,<izraz>) - vraća maksimalnu vrednost

3. Programsко окруžење

Prilikom izrade grafičkog editora korišten je Eclipse razvojno okruženje i programski paket GMF. Graphical Modeling Framework (GMF) je počeo kao Eclipse tehnološki projekat koji ima za cilj da obezbedi generičke komponente i infrastrukturu za razvoj grafičkog editora. Pre GMF napor koji je bio potreban da se izradi prilagođeni, specifičan editor najčešće je bio neekonomičan za većinu vizuelnih notacija. GMF je nastao kao rezultat potrebe da se na lakši način razvije grafički editor koristeći GEF i osnovni EMF model.

Danas GMF se sastoji od razvojnog okruženja za alate (engl. tooling framework) i podrške za izvršenje (engl. runtime). Podrška za izvršenje rukuje zadacima vezanim za premošćavanje EMF i GEF. On takođe pruža veliki broj usluga i sprega za programirnaje aplikacija (API) kako bi se omogućilo razvijanje bogatih grafičkih editora. Alati pružaju razvoj softvera na osnuvnu modela kroz definisanje grafičkih elemenata, alata u okviru editora, i mapiranje između modela i prikaza.[3] GMF razvojni tok je prikazan na Slika 4.



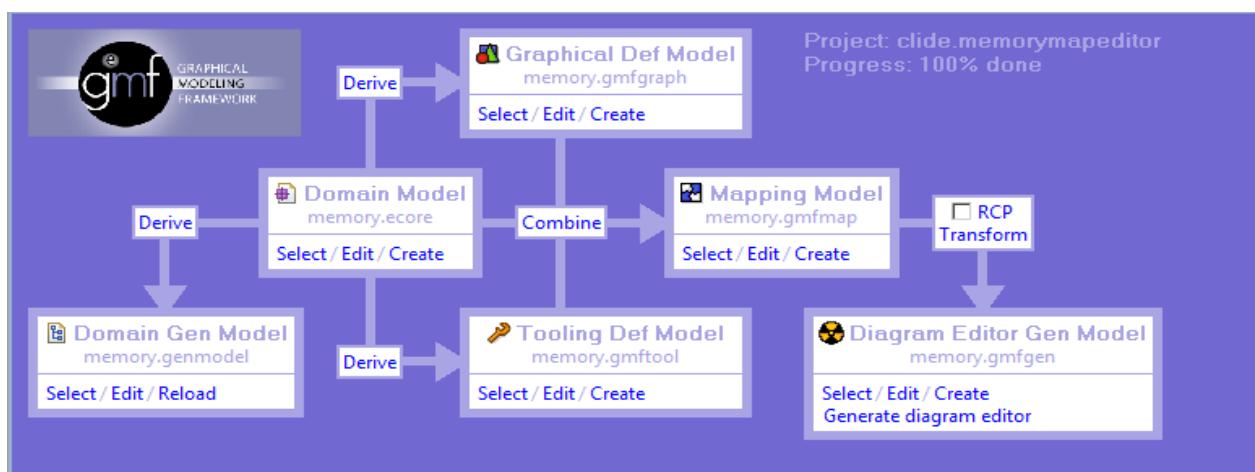
Slika 4 - GMF razvojni tok

4. Koncept rešenja

U ovom poglavlju će biti opisani koraci u razvoju grafičkog editora koji zasnivaju se na razvojnom toku koji je prikazan na Slika 4. Pored objašnjenja implementacije model biće dodatno objašnjeni alati koji se koriste u toku razvoja GMF editora.

Izrada GMF editora počinje od metamodela (.ecore datoteka) i domen generatoriog modela (.genmodel datoteka). Ova dva modela potiču iz EMF. Formiranje generatoriog modela se vrši na osnovu metamodela. Da bi generisali kompletan grafički editor neophodno je definisati nekoliko modela:

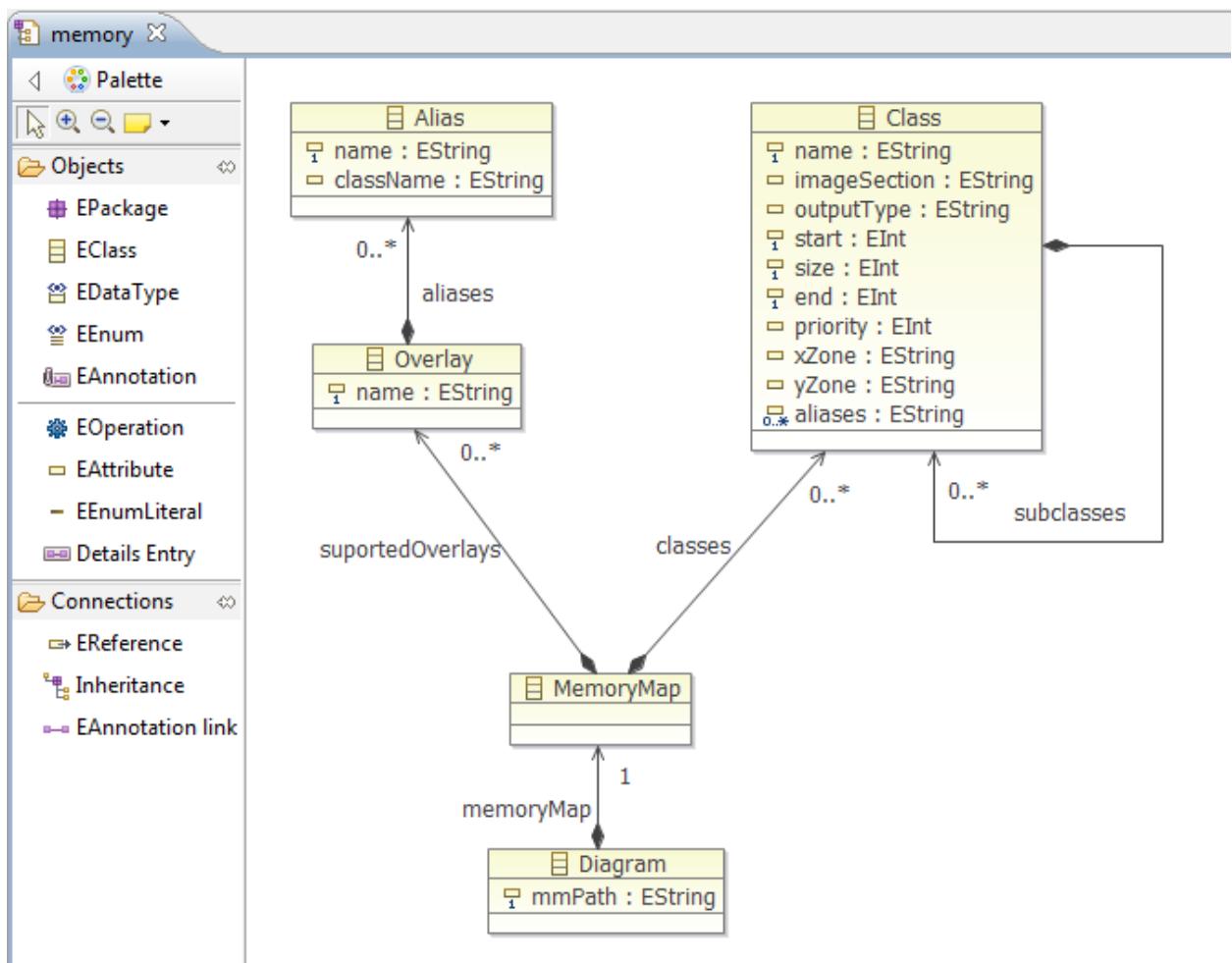
- .gmfgraph - ovaj model definiše grafičke prikaze, uključujući oblike (shapes) i grafičke čvorove i njihove međusobne veze
- .gmftool - ovaj model definiše paletu alata i ostale alate
- .gmfmap - ovaj model definiše povezivanje prethodna dva modela sa metamodelom



Slika 5 - GMF razvojna ploča

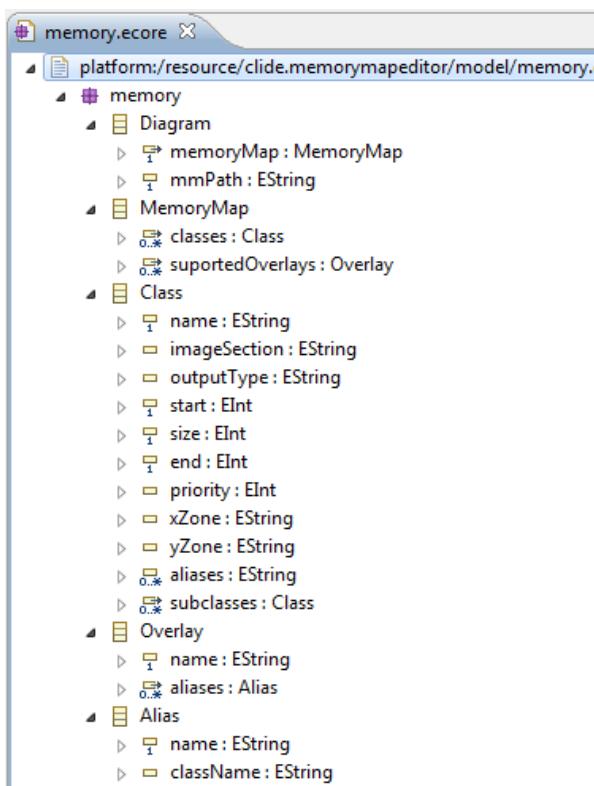
4.1 Metamodel i domen generator model

Prvi korak u formiranju modela je razvijanje metamodela. Metamodel je EMF model koji sadrži sve klase i sprege koje su neophodne za čuvanje informacija. Metamodel predstavlja model podataka nad kojima se vrši neka obrada. Ecore ili metamodel se u GMF razvijaju crtanjem dijagrama klasa i njihovih međusobnih veza. U okviru GMF je uvedeno kreiranje ecore dijagrama koji na vrlo jednostavan način omogućavaju formiranje modela. Prikaz ecore dijagrama za konkretan problem je dat na Slika 6.

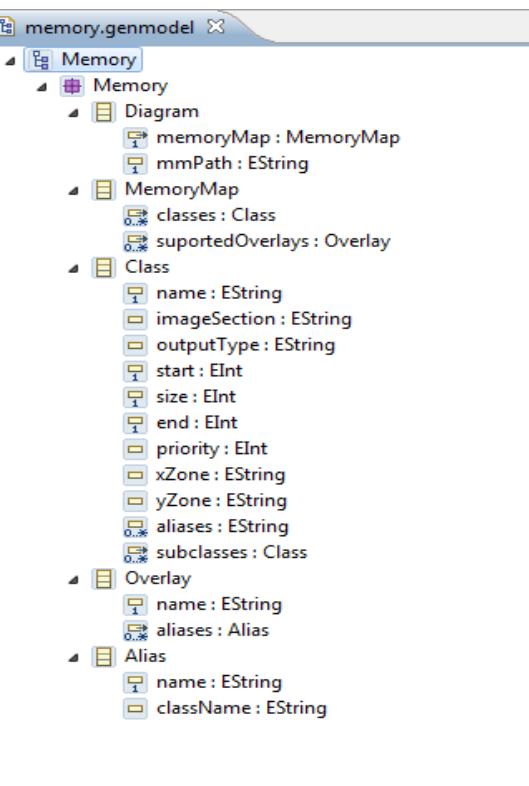


Slika 6 - Ecore dijagram

Na Slika 6 se vidi struktura klase koje će biti potrebne za reševanje konkretnog problema. Prikazane klase omogućavaju rukovanje podacima iz mem.xml datotekama i one su osnova celokupnog modela podataka. Prilikom crtanja dijagrama istovremeno se formira i .ecore datoteka sa modelom. Izgled .ecore datoteke je dat na Slika 7. Da bi se moglo izvršiti generisanje koda klase i sprega koje opisuju model podataka neophodno je stvoriti domen generator model. Ovaj model sa dobija na osnovu metamodela (pogledati GMF razvojnu ploču, Slika 5). Izgled .genmodel datoteke je dat na Slika 8.



Slika 7 - Metamodel (ecore model)



Slika 8 - Domen generator model

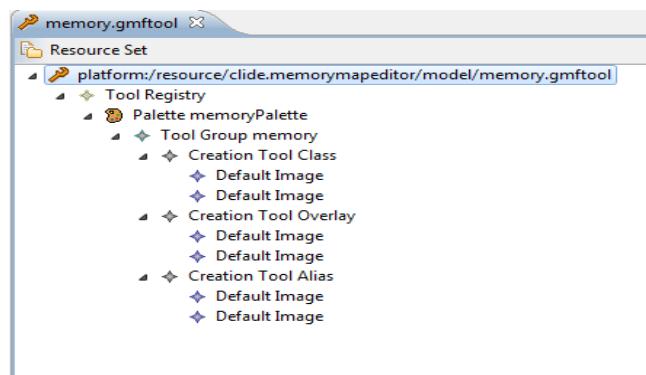
Generisani kod modela sadrži sledeće:

- Model - nudi Java interfejse i implementacije klasa za sve klase u modelu, kao i fabrike (engl. factory)
- Adapteri - generiše implementacije klasa koje vrše prilagođenje klasa modela za prikaz ili izmenu (ItemProviders klase)

Domen generatori model ima mogućnost i generisanje osnovnih klasa za testiranje generisanih klasa modela.

4.2 Model alata

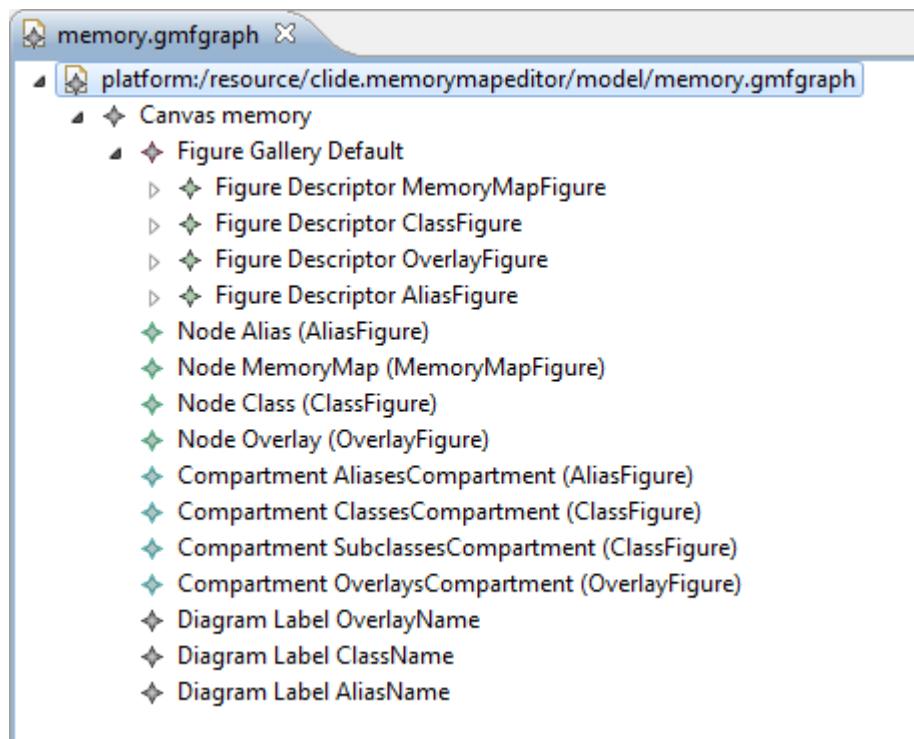
Dijagrami obično uključuju palete i druge alate za kreiranje i rad sa sadržajem dijagrama. Paleta je skup tastera koji omogućava dodavanje elemenata na dijagram a samim tim i dodavanje instanci objekata u model podataka. Svrha modela alata je da definiše sve elemente koji će se nalaziti u paleti alata. Model trenutno uključuje elemente za palete, trake sa alatima (toolbar), kao i različite menije za definisanje dijagrama. U sadašnjoj verziji GMF, koristi se samo paleta alata. Prikaz .gmftool datoteke dat je na Slika 9.



Slika 9 - Model alata

4.3 Grafički model

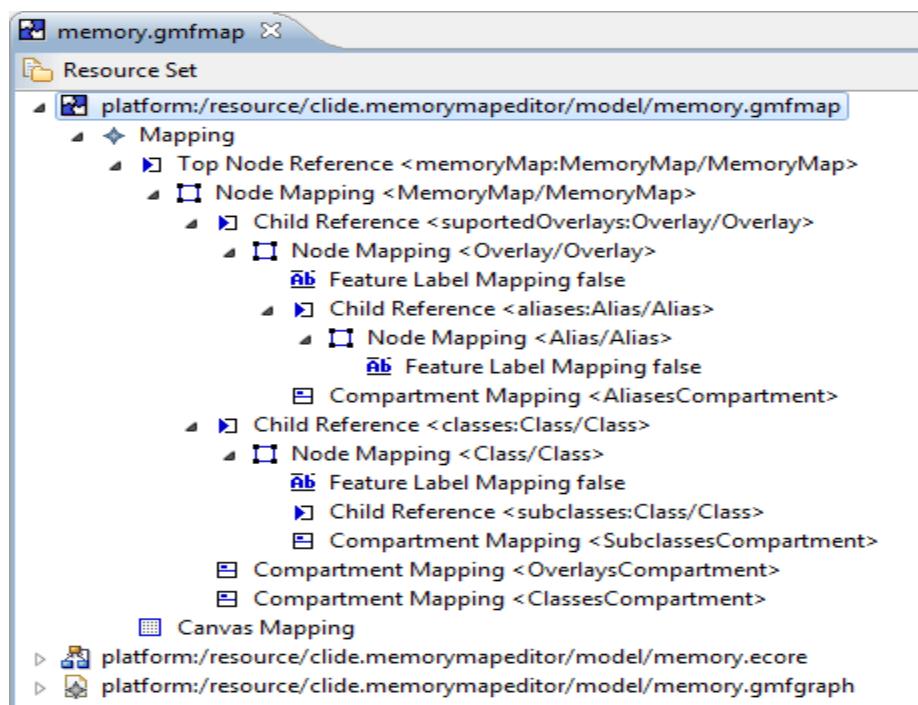
Grafički model ili model grafičke definicije se sastoji iz dva dela i definiše grafičke elemente koji se nalaze na dijagramu. Prvi deo je galerija figura koja definiše figure koje se pojavljaju na dijagramu (oblici, labele, linije, itd). Drugi deo se odnosi na definisanje čvorova, veza, pregrada i dijagram labela. Bitna stavka je da definisane figure mogu biti naknadno korištene u razvoju nekog drugog editora. Mnogi dijagrami zahtevaju sličan izgled elemenata kao što su zaobljeni pravougaonici sa centriranom labelom ili veze sa punom linijom i strelicom na kraju. Razvijene grafičke komponente je moguće sačuvati i iskoristiti ih u realizaciji novog editora. Grafički model editora dat je na Slika 10.



Slika 10 - Grafički model

4.4 Model povezivanja

Verovatno navažniji i najkompleksniji od svih modela u GMF je model povezivanja (mapping model). Elementi iz definicije dijagrama (alati i grafički elementi dijagrama) su mapirani na model domena. Povezivanje se vrši tako što se za svaki čvor na dijagramu vežu odgovarajući grafički objekti, akcije koje se vezuju za njega a potiču iz modela alata, kao i strukture podataka koje se vezuju za grafički objekat. U ovom modelu se međusobno povezivanje grafičkih objekata definisanih u grafičkom modelu. Ukoliko postoji neka ograničenja koja se tiču međusobnih veza ona se definišu u modelu povezivanja. Mapiranje modela predstavlja stvarni dijagram definicija i koristi se za generisanje modela generatora koda editora (.gmfgen model). Model povezivanja za implementirani editor dat je na Slika 11.

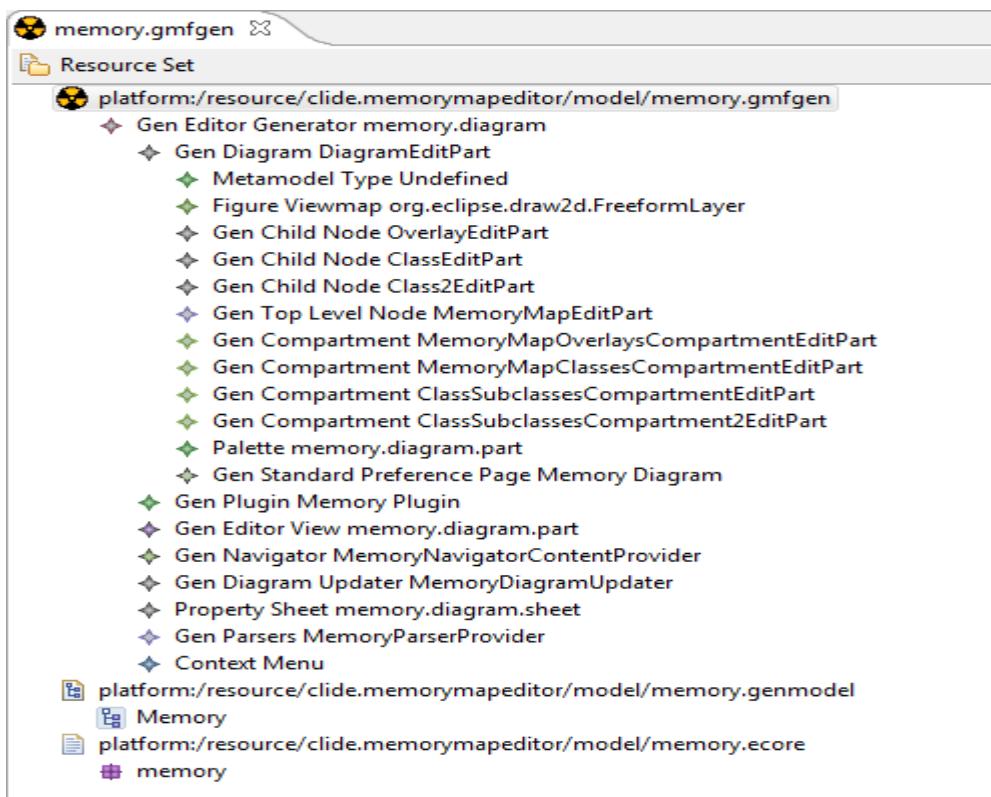


Slika 11 - Model povezivanja

4.5 Generator model

Generator model na osnovu informacija iz modela povezivanja vrši generisanje koda editora. Ovaj model radi na sličan način kao domen generator model iz EMF. Oba ova modela se mogu reproducovati i osvežavati iz svojih izvornih modela, .ecore i .gmfmap modeli. U okviru ovog modela moguće je vršiti dodatna podešavanja vezana za editor kao što su podešavanje ekstenzija datoteka u kojima se čuvaju dijagrami ili da li će elementi na dijagramu biti slagani

proizvoljno po površini dijagrama ili u po nekom unapred definisanom rasporedu. Izgled datoteke sa generator modelom je prikazan na Slika 12.



Slika 12 - Generator model

4.6 Integracija u CLIDE

CLIDE (Cirrus Logic Integrated Development Environment) je integrисано razvojno okruženje za razvoj DSP aplikacija bazirano na Eclipse platformi. Pored editovanja programskog koda u C-u i asembleru CLIDE omogуćava prikaz registara, prikaz memorije, prikaz adresa koda, evaluaciju izraza i kontrolisano izvršenje programa na nivou C jezika i asemblerskog jezika. Pored navedenog CLIDE podržava i simulaciju i kontrolu razvojne ploče.

GMF kao i ostali Eclipse paketi koristi plug-in koncept. Prilikom generisanja editora formira se plug-in koji omogуćava formiranje dijagrama. Ideja je da se novo nastali dijagram "veže" za neki postojeći projekat. Taj koncept nije bio pogodan za tekuću verziju CLIDE pa su se morale vršiti odgovarajuće izmene i prilagođenja.

Integracija u CLIDE se sastoji od nekoliko koraka

- kreiranje novog projekta (MemoryMapProject)
- kreiranje odgovarajućih čarobnjaka (engl. wizard) za stvaranje novog projekta
- pisanje parsera koji bi preuzimao sadržaj iz postojećih mem.xml datoteka i punio domen model (klasa *MemoryMapParser*)

- pisanje emitera koja bi omogućio formiranje mem.xml datoteke na osnovu domen modela (klasa *MemoryMapWriter*)
- izmena klasa koje je generisao GMF u cilju raznih prilagođenja

5. Implementacija

U ovom poglavlju će ukratko biti opisane klase koje su implementirane u okviru grafičkog editora. Veliki broj klasa je generisan uz pomoć GMF modela. Da bi se izvršila bilo kakva promena, odnosno bilo kakvo prilagođenje neophodno je upoznati se sa strukturama generisanih klasa. Jednostavnim redefinisanjem generisanih funkcija moguće je u značajnoj meri izmeniti funkcionalnost. Ukoliko izvršimo redefinisanje neke metode neophodno je to naglasiti GMF da prilikom ponovnog generisanja koda, npr. usled promene metamodela, ne vrati staro telo redefinisane funkcije. Izgled redefinisane funkcije:

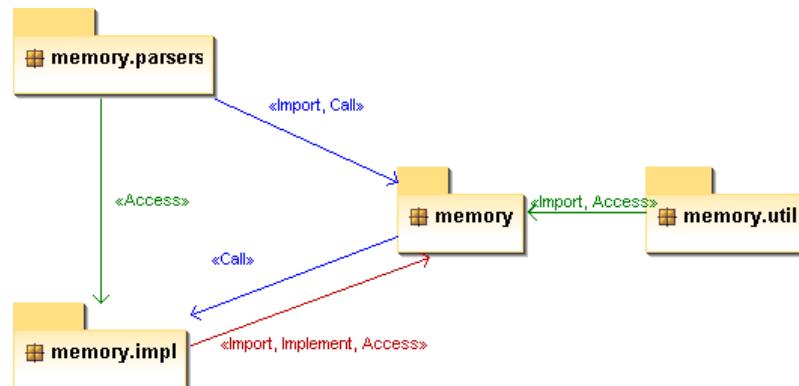
```
/**
 * Create a new instance of domain element associated with canvas.
 * <!-- begin-user-doc -->
 * <!-- end-user-doc -->
 * @generated NOT
 */
private static Diagram createInitialModel() {
    Diagram diagram = MemoryFactory.eINSTANCE.createDiagram();
    MemoryMap memoryMap = MemoryFactory.eINSTANCE.createMemoryMap();
    diagram.setMemoryMap(memoryMap);
    return diagram;
}
```

Inicijalno generisane funkcije sadrže anotaciju *generated* iznad tela funkcija. Brisanjem blokovskog komentara ili dodavanje reči NOT, kao u navedenom primeru, vršimo signaliziranje GMF da ne menja telo funkcije prilikom generisanja koda.

Grafički editor je realizovan po ugledu na GMF projekte u okviru 4 projekta: memorymapeditor, memorymapeditor.diagram, memorymapeditor.edit i memorymap.editor.

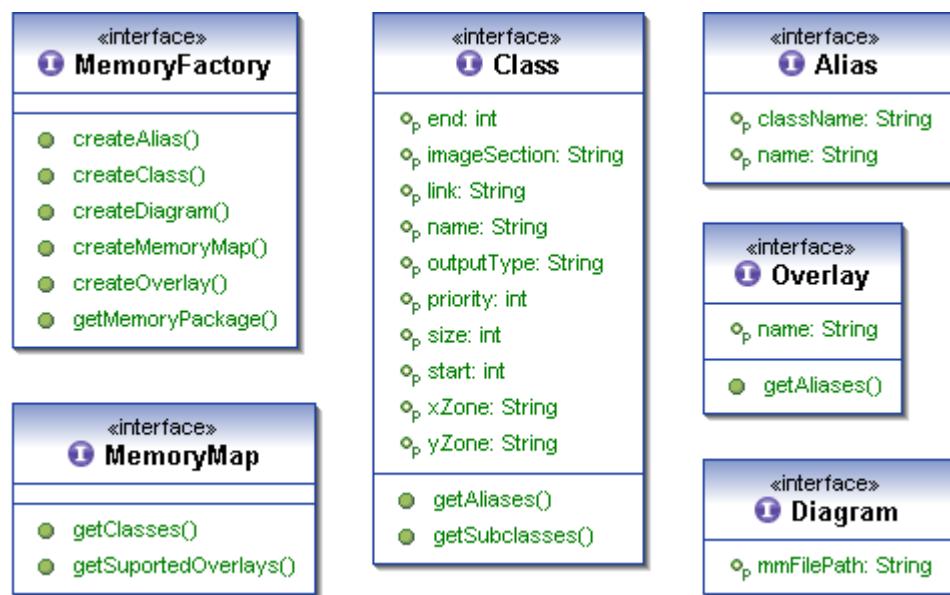
5.1 Memoriymapeditor

Projekat sa nazivom memoriymapeditor je polazni projekat za celokupan editor. On se formira kao GMF projekat i u okviru njega se definišu i formiraju modeli opisani u četvrtom poglavlju. Prilikom generisanje koda modela, na osnovu domen generatornog modela (.genmodel), u okviru ovog projekta se formiraju paketi koji sadrže klase opisane metamodelom (.ecore model). Veze između paketa su prikazane na Slika 13.



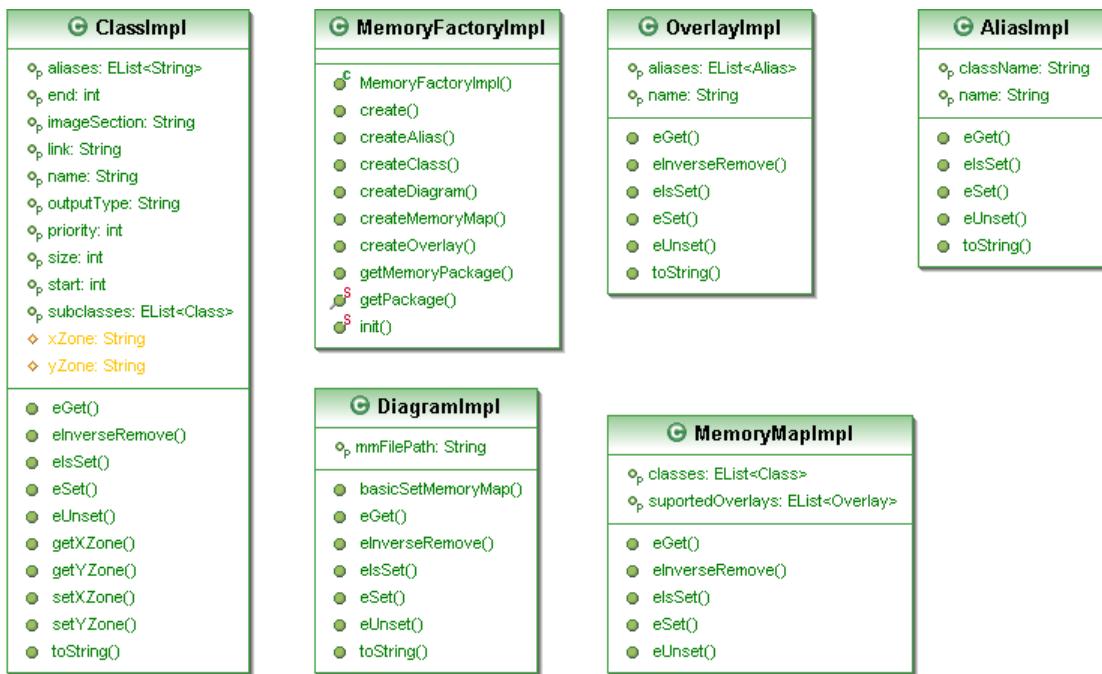
Slika 13 - Paketi modela

Paket memory sadrži sprege koje su generisani na osnovu metamodela a koji su formirani crtanjem Ecore dijagrama. Strukture sprega su prikazane na Slika 14.

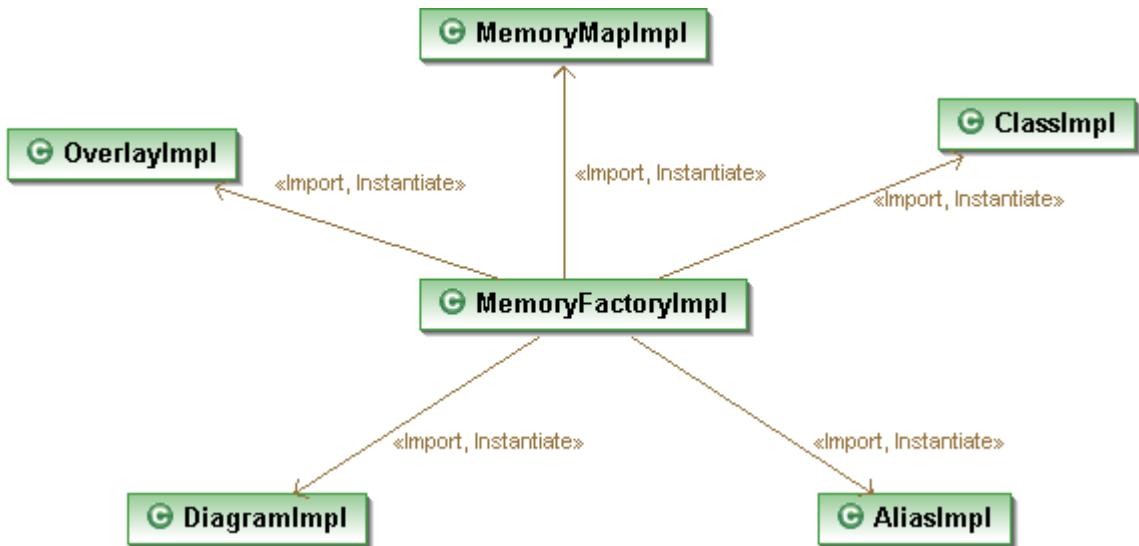


Slika 14 - UML sprega modela

Paket memory.impl sadrži klase u kojima su realizovane sprege iz paketa memory. Klase iz modela kao i njihove veze su prikazane na Slika 15 i Slika 16.



Slika 15 - UML klasa modela

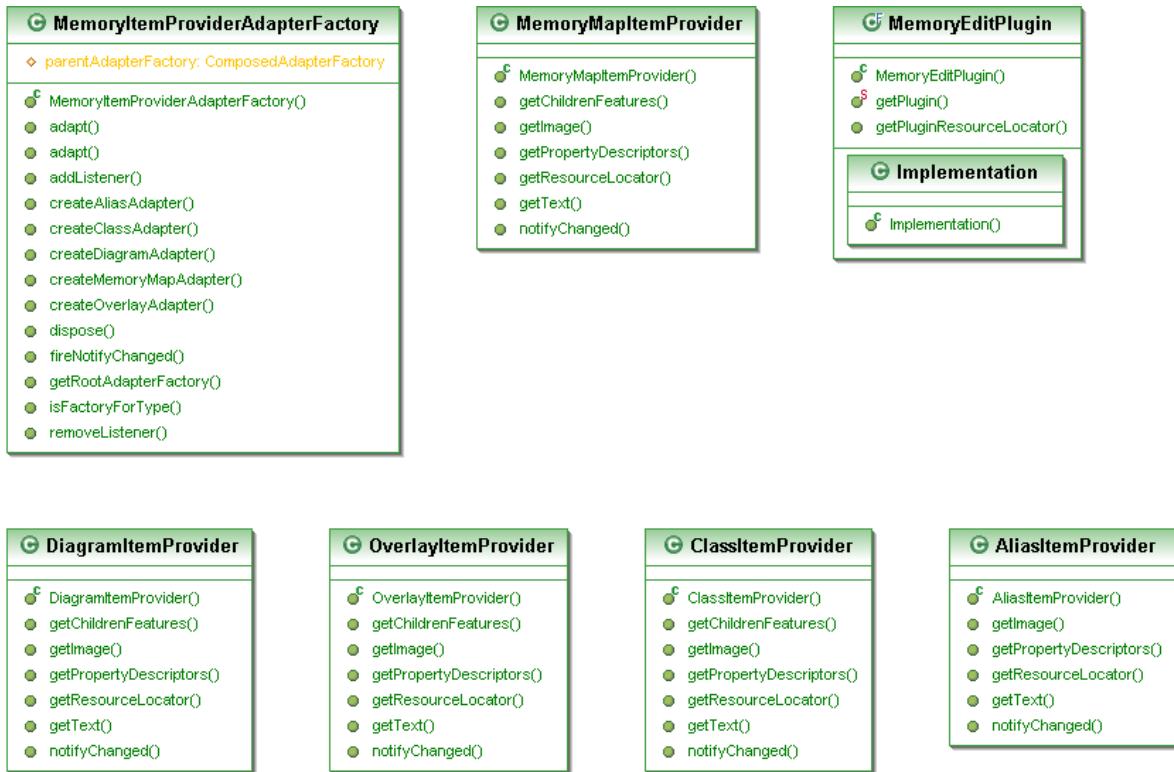


Slika 16 - UML klasa modela sa međusobnim vezama

Paket memory.parsers sadrži klase MemoryMapParser i MemoryMapWriter, opisane u poglavljima MemoryMapParseri i MemoryMapWriter.

5.2 Memorymapeditor.edit

Projekat koji nastaje generisanjem koda iz domen generatoriognog modela. U ovom projektu se nalazi paket memory.providers u kome se nalaze rukovaoci za grafičke objekte na dijagramu. Struktura klasa je prikazana na Slika 17.



Slika 17 - UML klasa rukovaoca

5.3 Memorymapeditor.editor

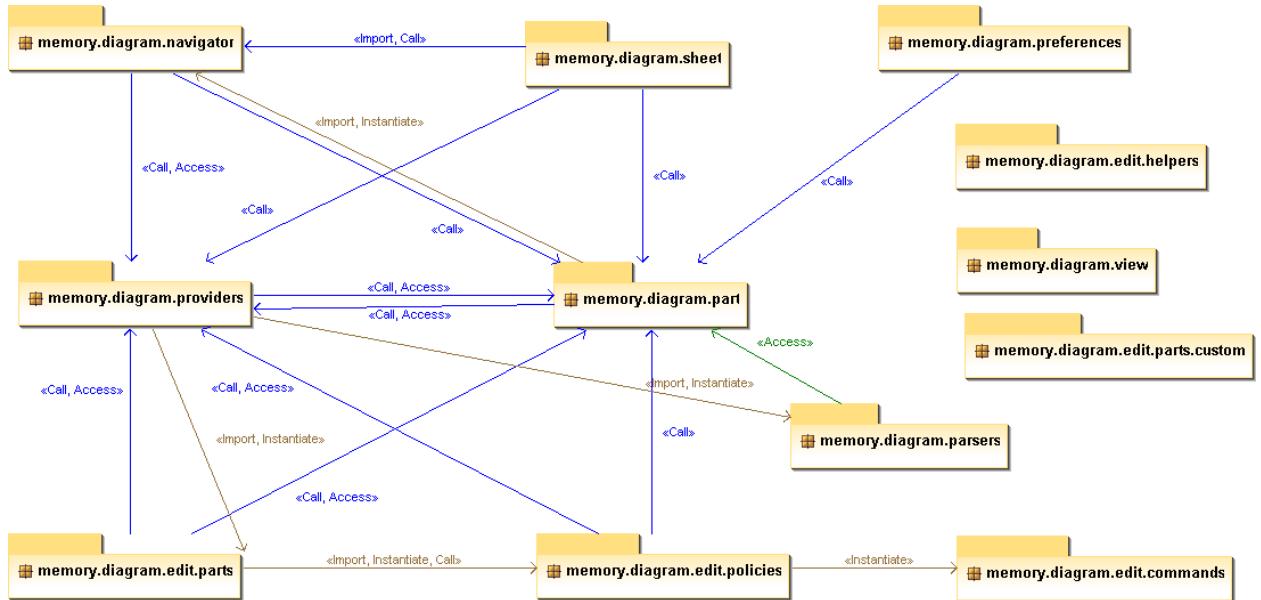
Projekat koji kao i prethodni nastaje generisanjem koda na osnovu domen generatoriognog modela.



Slika 18 - Klase paketa memory

5.4 Memoriymapeditor.diagram

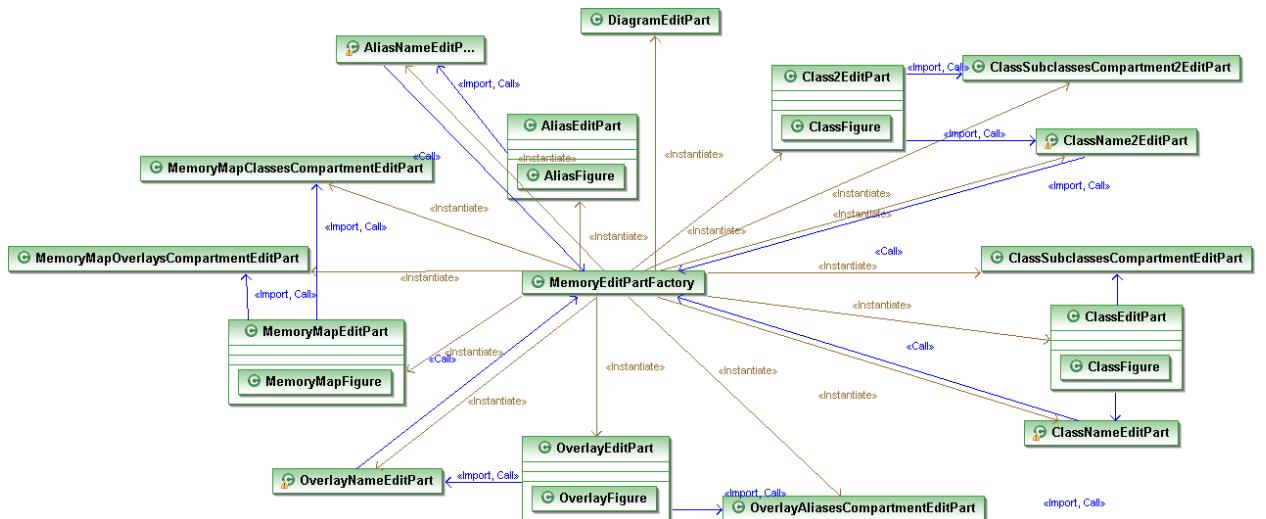
Memoriymapeditor.diagram je projekt koji nastaje po definisanju svih GMF modela. Ovaj projekat generiše generator model. Generisane klase omogućavaju rukovanje sa datotekama dijagrama kao i prikaz dijagrama. Struktura generisanih paketa data je na Slika 19.



Slika 19 - Paketi dijagrama

Paket memory.diagram.part sadrži klase koje se bave rukovanjem sa dijagramom. MemoryDiagramEditor, MemoryDocumentProvider i LoadResourceAction su samo neke klase iz ovog paketa.

Memory.diagram.edit.part je paket u kome su sadržane definicije grafičkih objekata. Definisanje grafičkih objekata u grafičkom modelu direktno se preslikava na klase iz ovog paketa. Ukoliko bi želeli da izvršimo neke izmene vezane za grafički prikaz bilo bi potrebno izmeniti grafički model ili redefinisati metode odgovarajućih klasa iz ovog paketa. Redefinisanje metoda je pogodno ako se ne vrše neke velike promene u prikazu, tj. promene nastale korištenjem već definisanih grafičkih objekata. Dijagram klasa iz ovog paketa prikazan je na Slika 20.



Slika 20 - UML dijagram klasa pakete diagram.edit.part

5.5 MemoryMapParser

MemoryMapParser je klasa koja omogućava punjenje EMF modela na osnovu mem.xml datoteke. U ovoj klasi je iskorišten DOM (Document Object Model) parser jer je struktura mem.xml datoteke i koncept klasa i podklasa jednostavnije predstaviti i razumeti u obliku stabla, DOM stabla u konkretnom slučaju. Pošto je pisan da podrži formiranje modela na osnovu postojećih mem.xml datoteka MemoryMapParser podržava i korišćenje matematičkih izraza kao vrednosti pojedinih atributa klase. Provera ispravnosti kao i određivanje vrste izraza urađeno je korištenjem regularnih izraza, REGEX. Analiziranjem mogućih matematičkih izraza dolazi se do problema redosleda razrešavanja atributa klase, tj. u izrazu se može naći neki podizraz čije izračunavanje tek treba da usledi.

Prvobitna ideja je bila da se razrešavanje vrednosti atributa klasa vrši u dva prolaza. U prvom prolazu bi se razrešile klase čiji atributi su zadati konkretnim vrednostima (heksadecimalne ili decimalne vrednosti za atribute <start>, <size> ili <end>). Drugi prolaz bi razrešavao izraze u kojima bi se moglo pojaviti simboli ili aritmetički izrazi sa simbolima. Mem.xml datoteke koje se sad koriste su ručno pisane i u njima često nije moguće razrešiti atribut svih klase u dva prolaza. Nije postojao odgovarajući mehanizam koji bi omogućio određivanje tačnog broja prolazaka da bi se sve klase razrešile.

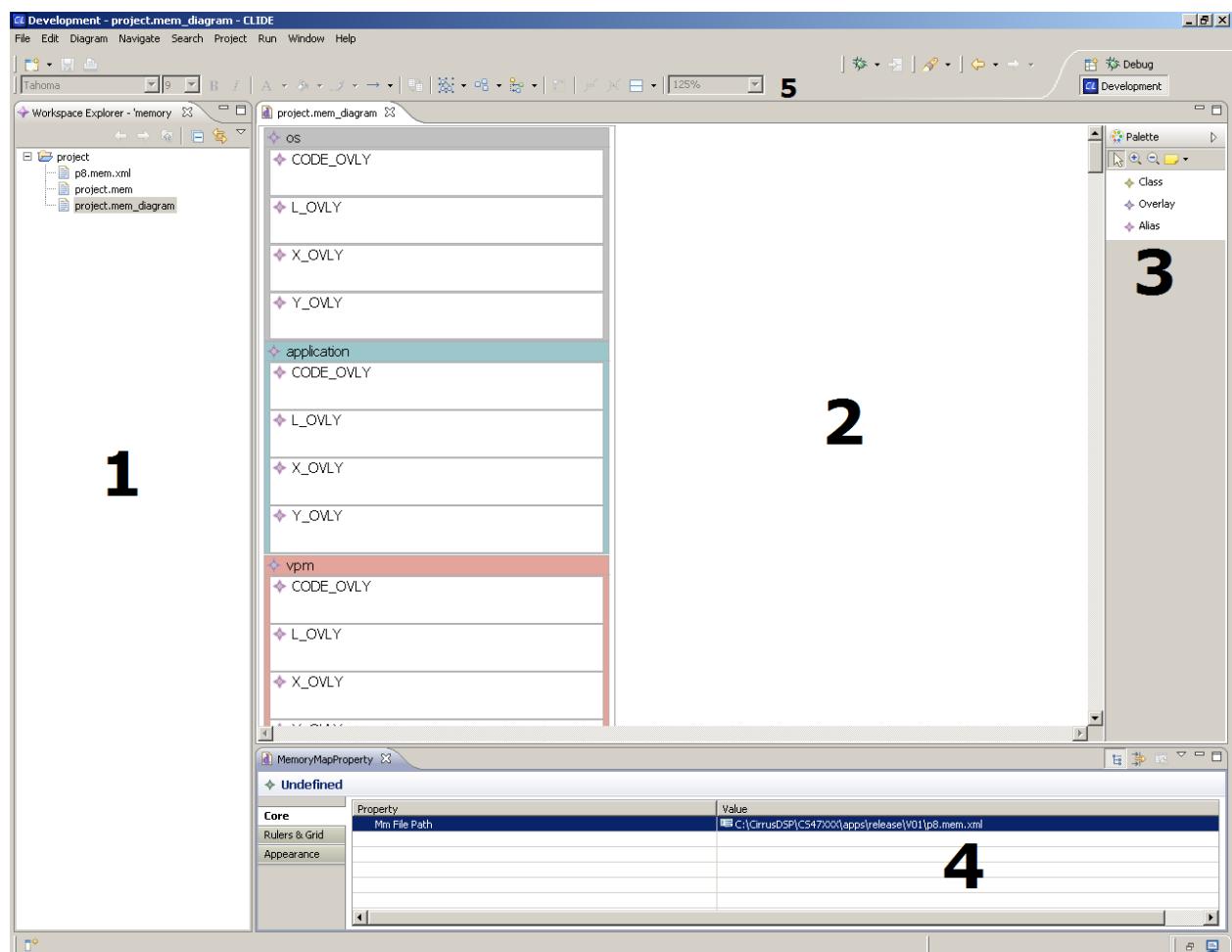
Na osnovu iznesenih problema kao rešenje nameće se rekurzivno razrešavanje atributa klasa. Ako prilikom izračunavanja izraza se pojavi simbol ili izraz koji je nerazrešen trenutno izračunavanje se zaustavlja i vrši se izračunavanje nepoznatog izraza. Prilikom razrešavanja atributa klasa razrešene klase se smeštaju od odgovarajući mapu (HashMap) tako da u nastavku razrešavanja moguće jednostavno proveriti da li je neka klasa razrešena, tj. atributi neke klase.

5.6 MemoryMapWriter

MemoryMapWriter je klasa koja vrši formiranje mem.xml datoteke na osnovu EMF modela. Ona na jednostvan način vrši formiranje mem.xml datoteku u skladu sa specifikacijom mem.xml datoteke (detaljan opis dat je u poglavlju Format mem.xml datoteka).

5.7 Editor i integracija u CLIDE

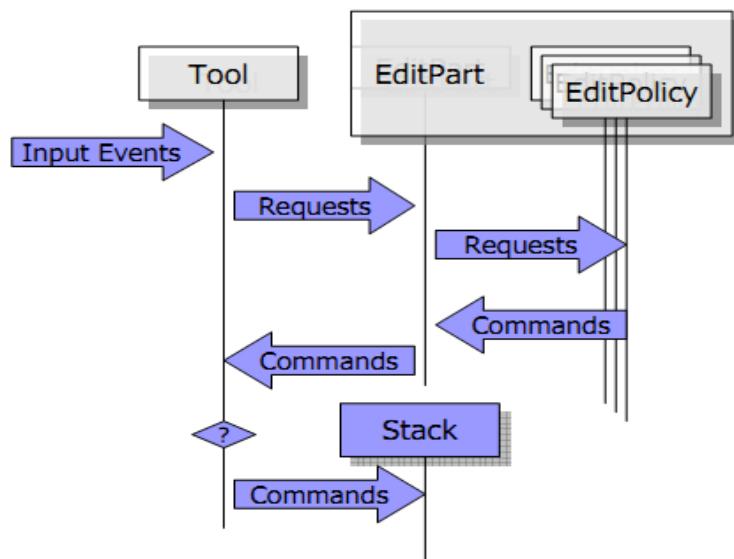
Kreiranje novog projekta koji bi opisivao rad sa memorijskim mapama se vrši korišćenjem odgovarajućeg čarobnjaka (engl. wizard). U okviru čarobnjaka se definiše naziv projekta, naziv radnog prostora kao i naziv konfiguracione datoteku koju želimo da menjamo ili naziv nove konfiguracione datoteke.[4] Prilikom završetka čarobnjaka vrši se formiranje MemoryMap projekta. Klasa MemoryMapProject po uzoru na slične klase iz CLIDE vrši organizaciju i čuvanje informacija vezanih za projekat. Da bi se klasa MemoryMapProject integrisala u CLIDE bilo je potrebno izmeniti i nekoliko postojećih klasa koje se bave CLIDE radnim prostorom a odnose se na parsiranje projektnih datoteka i dodavanje projekata i resursa u radni prostor. Prikaz grafičkog editora je dat na Slika 21.



Slika 21 - Grafički editor

Na Slika 21, broj 1 predstavlja prikaz radnog prostora. CLIDE razvojno okruženje poseduje sopstveni radni prostor (engl. workspace) pa je bilo neophodno implementirati vezu između GMF projekta, koji koristi Eclipse radni prostor kao podrazumevani, i CLIDE projekta.

Broj 2 na Slika 21 predstavlja radnu površinu na koju se dodaju klase, pokrivači ili aliasi. Redefinisanjem metode za formiranje inicijalnog modela pri formiranju novog projekta vrši se dodavanje objekta MemoryMap na radnu površinu. Ovo omogućuje postojanje samo jedne memoriske mape za jednu mem.xml datoteku pa samim tim i projekat. Brisanje objekta koji predstavlja memorisku mapu je onemogućeno u dodavanjem rukovaoca sa zahtevima. U GMF se komunikacija i signalizacija između objekata obavlja putem zahteva (engl. requests), komandi (engl. commands) i obaveštenja (engl. notifications). Dodati rukovaoc vrši proveru svih zahteva i ukoliko dobije zahtev za brisanje objekta vrši formiranje komande koja se ne izvršava, objekat klase UnexecutableCommand. Sistem događaja i komunikacije je dat na Slika 22.



Slika 22 - Sistem događaja, GEF i GMF

Paleta sa alatima je prikazana na Slika 21 pod brojem 3. Navedena paleta sa alatima je definisana u GMF modelima.

CLIDE razvojno okruženje je onemogućilo posmatranje atributa objekata iz radnog okruženja. Sistemski pregled atributa bi ispisivao informacije bilo kog objekta u razvojnog okruženju. Da bi se omogućio specifičan pregled atributa (engl. property view) bilo je potrebno napisati odgovarajuće klase koje bi omogućile prikaz pregleda atributa samo za objekte sa dijagrama. Prilagođeni prikaz atributa je prikazan brojem 4 na Slika 21.

6. Verifikacija

U ovom radu je opisan grafički editor za generisanje konfiguracionih datoteka za povezivač sa apsolutnim punjačem. Pošto se konfiguracione datoteke već koriste grafički editor je morao da podrži editovanje postojećih konfiguracionih datoteka. Verifikacija grafičkog editora je urađena kroz dva vrste testova.

Prvi grupa testova vrši učitavanje postojeće konfiguracione (*.mem.xml) datoteke, prikaz dijagrama i izmena konfiguracione datoteke prilikom čuvanja dijagrama. Konfiguracione datoteke koristi povezivač sa Crystal platforme i verifikacija prvog koraka je urađena poređenjem izlaznih datoteka povezivača u kojima su navedene informacije o početnoj i krajnjoj adresi memorijskog segmenta. Datoteke su identične.

Druga grupa testova vrši učitavanje konfiguracione datoteke, prikaz dijagrama, izmena pojedinih klasa na dijagramu i čuvanje dijagrama. Verifikacija je izvršena poređenjem izlaznih datoteka povezivača.

7. Zaključak

U ovom radu je objašnjeno jedno rešenje implementacije grafičkog editor memorijskih mapa. Grafički editor je realizovan korištenje Eclipse paketa za modelovanje grafičkog okruženja, GMF. Pošto grafički editori generisan ovom komponentom sadrži samo dijagram i editor koji se veže za neki projekat bilo je neophodno izvršiti integraciju editora u neko postojeće razvojno okruženje. Editor memorijskih mapa je integriran u CLIDE razvojno okruženje jer je editor namenjen za izmenu konfiguracionih datoteka za familije procesora kompanije *Cirrus Logic*.

Povezivač koji proizvodi sliku programa za apsolutni punjač ima složen zadatak. Potrebno je omogućiti punjenje više različitih modula za obradu signala u isto vreme. Ovaj problem se rešava podelom memorije DSP-a na zone koje su organizovane hijerarhijski. Svaka zona može primiti jedan modul koji vrši tip obrade za koju je zona namenjena. Na osnovu datoteke koja opisuje zone povezivač proizvodi sliku modula koju je moguće puniti zajedno sa drugim modulima koji vrše drugi tip obrade, i upisuju se u drugu zonu.

Datoteke sa opisom zona postaju komplikovanije sa porastom broja modula, pa se javlja potreba za grafičkim alatom za proizvodnju i prilagođavanje ovih datoteka. Ovaj grafički alat daje sliku podele memorije.

Prednosti razvoja alata korištenjem GMF se prvenstveno ogledaju u formiranju modela podataka koji će biti korišteni kao i formiranju prikaza. Kao što je ranije navedeno razvijanje softvera na osnovu modela kao i generisanje koda na osnovu modela dovodi do povećanja kvaliteta koda, smanjenja programerskih grešaka pa samim tim i povećenjem produktivnosti. Jedan od nedostataka GMF paketa je to što je vrlo nezgodno izvršiti njegovu integraciju u neko već postojeće rešenje. Postoji veliki broj mehanizama koji su sakriveni od korisnika i nalaze se u

unutar GMF tako da je integracija u značajnoj meri otežana. Prilikom razvoja nekog novog alata bilo bi zgodno iskoristiti GMF za formiranje osnove celokupnog sistema prvenstveno zbog generisanja koda na osnovu modela.

8. Literatura

- [1] Bill More, David Dean: *Eclipse development using Graphical Editing Framework and Eclipse Modeling Framework*, IBM RedBooks, 2004
- [2] Vladimir Kovačević, Miroslav Popović: *Sistemska programska podrška u realnom vremenu*, Univerzitet u Novom Sadu, Fakultet Tehničkih Nauka, 2002
- [3] Richard C. Grönback: *A Domain-Specific Language Toolkit*, 2009
- [4] Eric Clayberg, Dan Rubel: *Building Commercial-Quality Plug-ins*