



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У
НОВОМ САДУ



Саша Карталија

**Једно проширење интегрисаног
развојног окружења са подршком за
развој асемблерских апликација**

МАСТЕР РАД

Нови Сад, Децембар 2013.



УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, РБР :	
Идентификациони број, ИБР :	
Тип документације, ТД :	Монографска документација
Тип записа, ТЗ :	Текстуални штампани материјал
Врста рада, ВР :	Дипломски – мастер рад
Аутор, АУ :	Саша Карталија
Ментор, МН :	Др Јелена Ковачевић
Наслов рада, НР :	Једно проширење интегрисаног развојног окружења са подршком за развој асемблерских апликација
Језик публикације, ЈП :	Српски / латиница
Језик извода, ЈИ :	Српски
Земља публикавања, ЗП :	Република Србија
Уже географско подручје, УГП :	Војводина
Година, ГО :	2013
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)	8/47/10/4/11/3/0
Научна област, НО :	Електротехника и рачунарство
Научна дисциплина, НД :	Рачунарска техника
Предметна одредница/Кључне речи, ПО :	Наменски систем, интегрисано развојно окружење, уређивач асемблерског кода, вишепроцесорска платформа
УДК	
Чува се, ЧУ :	У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, ВН :	
Извод, ИЗ :	У овом раду реализована је имплементација уређивача текста за развој асемблерског кода на више-језгарној платформи, коришћењем оквира за развој уско-наменских програмских језика.
Датум прихватања теме, ДП :	
Датум одбране, ДО :	
Чланови комисије, КО :	Председник: Проф. др Растислав Струхарик
	Члан: Др Илија Башичевић
	Члан, ментор: Др Јелена Ковачевић
	Потпис ментора



KEY WORDS DOCUMENTATION

Accession number, ANO :			
Identification number, INO :			
Document type, DT :	Monographic publication		
Type of record, TR :	Textual printed material		
Contents code, CC :	Master Thesis		
Author, AU :	Saša Kartalija		
Mentor, MN :	PhD Jelena Kovačević		
Title, TI :	One extension of an integrated development environment with support for assembler applications development		
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 :	2013		
Publisher, PB :	Author's reprint		
Publication place, PP :	Novi Sad, Dositeja Obradovica sq. 6		
Physical description, PD : (chapters/pages/ref./tables/pictures/graphs/appendixes)	8/47/10/4/11/3/0		
Scientific field, SF :	Electrical Engineering		
Scientific discipline, SD :	Computer Engineering, Engineering of Computer Based Systems		
Subject/Key words, S/KW :	Integrated development environment, embedded system, assembly editor, multiprocessor platform		
UC			
Holding data, HD :	The Library of Faculty of Technical Sciences, Novi Sad, Serbia		
Note, N :			
Abstract, AB :	This paper contains implementation of an integrated development environment with support for assembler applications development on multiprocessor platform with Xtext domain specific language.		
Accepted by the Scientific Board on, ASB :			
Defended on, DE :			
Defended Board, DB :	President:	PhD Rastislav Struharik	Menthor's sign
	Member:	PhD Ilija Bašičević	
	Member, Mentor:	PhD Jelena Kovačević	

Zahvalnost

Zahvaljujem se mojim roditeljima na безусловnoj podršci, pomoći i strpljenju tokom mog školovanja.

Od srca se zahvaljujem mojoj devojci na ukazanoj pažnji i pomoći u prethodnom periodu, jer bez nje i sam ovaj čin bi imao manji značaj.

Srdačno se zahvaljujem svom mentoru Dr Jeleni Kovačević koja mi je omogućila izradu ovog master rada u prijatnom okruženju sa prijatnim ljudima.

Na kraju se zahvaljujem Ivanu Považanu i svim ostalim kolegama koji su na bilo koji način pružili pomoć i podršku tokom istraživanja kako teorijskog, tako i praktičnog dela ovog master rada.

SADRŽAJ

1. Uvod.....	2
2. Teorijske osnove	4
2.1 Sonnet razvojna platforma	4
2.1.1 Sonnet SIDCore DSP procesor	5
2.1.2 Sonnet SNAPCore DSP procesor	6
2.2 Koncept i struktura Eclipse platforme.....	7
2.2.1 Radno okruženje	10
2.2.2 Radni sto	10
2.2.3 Eclipse alati za razvoj programske podrške.....	11
2.2.4 Eclipse integrisano razvojno okruženje	12
2.2.5 Eclipse okruženje za razvoj priključaka	12
2.2.6 Eclipse aplikativna programska sprega.....	12
2.2.6.1 Java grafička biblioteka - SWT	12
2.2.6.2 Java grafička biblioteka - Jface	13
2.3 Programski jezici.....	14
2.4 Xtext okvir.....	14
2.4.1 Karakteristike Xtext okvira.....	15
3. Koncept rešenja.....	17
3.1 Xtext gramatika	17
3.2 Uređivač asemblerskog koda	20
3.2.1 Bojenje ključnih reči	21
3.2.2 Formatiranje sadržaja uređivača	23
3.2.3 Automatski prediktivni asistent	23

3.3	Pogledi obeleživača resursa	26
3.3.1	Pogled problema i upozorenja	27
3.3.2	Pogled zadatka i lokacija koda	27
3.4	Prekidne tačke i kontrolisano izvršavanje	29
3.5	Višestruko pojavljivanje elementa	30
3.6	Pogled sadržaja uređivača	31
3.7	Informacioni prozor – “lebdeći tekst”	32
4.	Programsko rešenje	35
4.1	Paket Xtext	37
4.1.1	Datoteka Asm.xtext	37
4.1.2	Datoteka GenerateAsm	38
4.2	Paket XtextFormatting	40
4.2.1	Klasa AsmFormatter	40
4.3	Paket XtextLinking	40
4.3.1	Klasa AsmLinkingMessages	40
4.4	Paket XtextScoping	40
4.4.1	Klasa AsmScopeProvider	40
4.5	Paket XtextValidation	40
4.5.1	Klasa AsmJavaValidator	40
4.6	Paket XtextUi	41
4.6.1	Klasa AsmUiActivator	41
4.6.2	Klasa AsmUiModul	41
4.7	Paket XtextUiContentAssist	41
4.7.1	Klasa AsmProposalProvider	41
4.8	Paket XtextUiEditorCallback	42
4.8.1	Klasa EditorCallBack	42
4.9	Paket XtextUiHighlighting	42
4.9.1	Klasa SemanticHighlightingCalculator	42
4.9.2	Klasa SemanticHighlightingConfiguration	42
4.9.3	Klasa TokenToIdMapper	42
4.10	Paket XtextUiHover	43
4.10.1	Klasa AsmEObjectHoverProvider	43
4.10.2	Klasa AsmEObjectDocumentationProvider	43
4.11	Paket XtextUiLabeling	43
4.11.1	Klasa AsmLabelProvider	43

4.12	Paket XtextUiMarkOccurrences	43
4.12.1	Klasa SonnetOccurenceComputer	44
4.13	Paket XtextUiNavigation	44
4.13.1	Klasa AsmHyperlinkHelper	44
4.14	Paket XtextUiOutline	44
4.14.1	Klasa AsmOutlineTreeProvider	44
4.14.2	Klasa DefinesFilterContribution	44
4.14.3	Klasa IncludesFilterContribution	44
4.14.4	Klasa LabelsFilterContribution	45
4.15	Paket XtextUiPreferences	45
4.15.1	Klasa HoverPagePreference	45
4.16	Paket XtextUiRefactoring	45
4.16.1	SonnetRenameRefactoringProvider	45
4.17	Paket XtextUiTask	45
4.17.1	Klasa AsmTaskCalculator	45
4.17.2	Klasa AsmTaskElementChecker	45
4.18	Paket XtextUiTemplates	46
4.18.1	Klasa SonnetTemplateProposalProvider	46
5.	Ispitivanje i verifikacija	47
5.1	JUnit ispitivanje	48
5.2	Squish ispitivanje	51
6.	Zaključak	53
7.	Dodatak A	55
8.	Literatura	57

SPISAK SLIKA

Slika 1. – Najvažnije komponente fizičke arhitekture Sonnet platforme.....	5
Slika 2. - Najvažnije komponente Eclipse platforme.....	9
Slika 3. – Blok šema Xtext modela.....	15
Slika 4. – Proces punjenja uređivača i provere sadržaja	20
Slika 5. – Stranica za podešavanje bojenja ključnih reči	22
Slika 6. – Građenje asemblerske instrukcije pomoću prediktivnog asistenta	24
Slika 7. – Korišćenje definisanih elemenata pomoću asistenta.....	24
Slika 8. – Stranica za definisanje šablona asemblerskog koda	25
Slika 9. – Definisani šablon unutar liste prediktivnog asistenta	26
Slika 10. – Izgled definisanog šablona unutar uređivača.....	26
Slika 11. – Grafički predstavljena greška	27
Slika 12. – Pogled zadatka i obeleživača lokacije koda	28
Slika 13. – Perspektiva kontrolisanog izvršavanja aplikacije	29
Slika 14. – Višestruko pojavljivanje asemblerske instrukcije.....	31
Slika 15. – Pogled sadržaja uređivača.....	32
Slika 16. – Primeri informacionih prozora.....	33
Slika 17. – Informacioni prozor u režimu kontrolisanog izvršavanja.....	34

SPISAK TABELA

Tabela 1. – Spisak modula programskog rešenja.....	36
Tabela 2. – Rezultati Junit ispitnih slučajeva.....	49
Tabela 3. – Rezultati Squish ispitnih slučajeva.....	51

SKRAĆENICE

IDE - *Integrated Development Environment*, Integrisano razvojno okruženje

RCP - *Rich Client Platform*, Alat za integraciju komponenata programske podrške

API - *Application Programming Interface*, Aplikativna programska sprega

DSP - *Digital Signal Proccesor*, Digitalni signal procesor

XML - *Extensible Markup Language*, Proširiv jezik za opisivanje

SIDE – *Sonnet Integrated Development Enviroment*, Sonnet integrisano razvojno okruženje

GPL – *General Purpose Language*, Programski jezik opšte namene

DSL – *Domain Specific Language*, Usko namenski programski jezik

SWT – *Standard widget toolkit*, Standardna grafička biblioteka

AWT – *Abstract Windowing Toolkit*, Standardna grafička biblioteka

PDE – *Plug-in Development Environment*, Okruženje za razvoj utičnih proširenja

SDK – *Software Development Kit*, Alati za razvoj programske podrške

UI – *User Interface*, Korisnička sprega

1. Uvod

U ovom radu realizovan je uređivač teksta za razvoj asemblerskog koda na Sonnet više-jezgarnoj platformi, korišćenjem Xtext okvira [1] za razvoj usko-namenskih programskih jezika [2], koji obezbeđuje sledeće funkcionalnosti:

1. Bojenje sintakse asemblerskog jezika (engl. *Syntax coloring*)
2. Automatski prediktivni asistent (engl. *Code completion*)
3. Sadržaj uređivača sintakse (engl. *Outline view*)
4. Putanja kroz sadržaj uređivača (engl. *Source-code navigation*)
5. Indeksiranje (engl. *Indexing*)
6. Poređenje sadržaja (engl. *Compare view*)
7. Izmena imena u celokupnom sadržaju uređivača (engl. *Rename refactoring*)
8. Režim kontrolisanog izvršavanja (engl. *Debug*)
9. Pogled sadržaja uređivača, grešaka, upozorenja (engl. *Views*)
10. Obeleživači resursa (engl. *Markers*)

Uređivač je grafička komponenta razvojnog okruženja koja se koristi u svrhu uređivanja teksta određenog formata ili dijagrama. Zbog postojanja različitih ciljnih platformi i različitih tipova procesora javljaju se različiti tipovi sintakse asemblerskog jezika prilagođeni određenoj arhitekturi. Iz tog razloga razvijen je uređivač teksta predodređen za razvoj nove sintakse asemblerskog jezika. Uređivač asemblerskog koda je realizovan u okviru Sonnet integrisanog razvojnog okruženja (engl. *Sonnet Integrated Development Environment-SIDE*), razvijenog na katedri za Računarsku Tehniku i Računarske Komunikacije za potrebe više-jezgarne platforme Sonnet, kompanije Starkey [3]. SIDE razvojno okruženje zasnovano je na Eclipse platformi [4]. Obezbeđuje sve alate neophodne za pisanje, uređivanje, prevođenje, ispitivanje i otklanjanje

grešaka, programskih kodova namenjenih za različite tipove Sonnet familije procesora. SIDE je nezavisan od Eclipse platforme i koristi se u bilo kom Java okruženju.

Za razvoj alata za uređivanje asemblerskog koda izabran je usko-namenski jezik Xtext. Usko-namenski jezici pružaju kompaktan i jednostavan način da se formulišu svi aspekti sistema koji se razvija, a u našem slučaju alata za uređivanje asemblerskog koda. Pored jednostavnosti, Xtext obezbeđuje nezavisnost od platforme na kojoj se razvija sistem, kao i odličnu korisničku spregu (engl. *Application Programmer Interface-API*).

Uređivač asemblerskog koda podržava konstrukcije asemblerskog jezika predodređene za Sonnet platformu. Pored samog uređivača realizovan je i skup dodatnih alata koji u znatnoj meri olakšavaju programeru korišćenje uređivača i sam razvoj programske podrške za ciljnu platformu [5]. Korisniku SIDE okruženja su na raspolaganju dodatna podešavanja samog uređivača, praćenje sadržaja uređivača, prepoznavanje istih instrukcija kao i automatski asistent koji na bazi predviđanja, korisniku nudi određene opcije pri pisanju programskog koda.

Rad je strukturiran na sledeći način: u drugom poglavlju su date teorijske osnove Eclipse platforme, kao i osnovni koncepti Xtext jezika i fizičke arhitekture Sonnet platforme i Sonnet familije procesora; u trećem poglavlju opisan je primer razvijene sintakse asemblerskog jezika i samog uređivača asemblerskog jezika uz dodatne alate, dok je u četvrtom poglavlju dato programsko rešenje zadatka; peto poglavlje opisuje proces ispitivanja i verifikacije rešenja sistema; šesto poglavlje sadrži zaključak, kao i moguće tačke proširenja, sedmo poglavlje sadrži dodatak, dok je u osmom poglavlju navedena korišćena literatura.

2. Teorijske osnove

U ovom poglavlju analizirane su neke od programskih i fizičkih komponenti koje se koriste u realizaciji uređivača asemblerskog koda kao sastavnog dela SIDE sistema. Od fizičkih komponenti opisana je arhitektura Sonnet familije procesora kompanije Starkey. Zbog složenosti pomenute fizičke arhitekture u nastavku su ukratko opisane komponente koji su od interesa za sistem.

Sa druge strane, od programskih komponenti sistema kao najvažnija u procesu realizacije sistema opisana je Eclipse platforma sa svojim neophodnim alatima, kao i grafičkim bibliotekama ključnim za razvoj uređivača asemblerskog koda i uopšte grafičke sprege sa korisnikom.

Na kraju poglavlja data je podela programskih jezika i opis Xtext jezika, kao predstavnika usko-namenskih programskih jezika.

2.1 Sonnet razvojna platforma

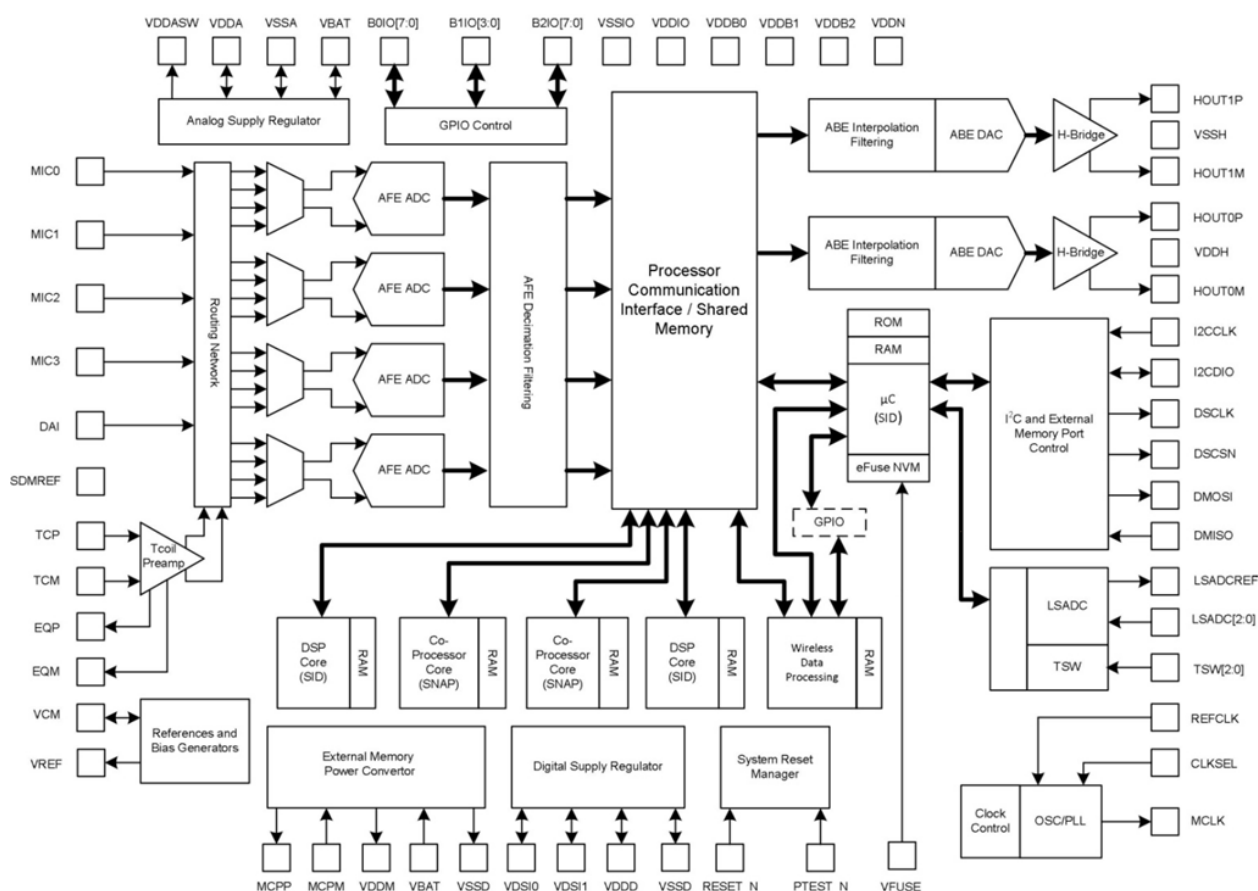
Sonnet platforma, kompanije Starkey, namenjena je za razvoj slušnih aparata niske potrošnje električne energije. Platforma se sastoji od pet DSP procesora čija međusobna razlika se ogleda u količini integrisane memorije, kao i u programskoj podršci.

DSP procesore iz Sonnet familije procesora, između ostalog, čine odvojene memorijske zone za podatke, kao i programska memorijska zona. Memorija unutar DSP procesora služi za čuvanje podataka, kao i programa koji se izvršava na procesoru. Svaka memorijska zona ima posebnu ulogu. U zavisnosti od potrebe, adresiranjem se vrši odabir memorijske zone. Pored čitanja memorijske lokacije memorija mora da obezbedi i upis. Ovo se obično izvodi pomoću određene kontrolne linije, kojoj se zadaju kontrole upisa (engl. *Write*) ili čitanja (engl. *Read*).

Memorijske zone za smeštanje podataka su X i Y, dok memorijska zona P predstavlja programsku memorijsku zonu. Kako je ovakva memorija razvijena u FLASH tehnologiji, ima mogućnost višestrukog brisanja i programiranja te je pogodna za razvoj uređaja.

Tri procesorska jezgra, od kojih jedno ima ulogu mikro-kontrolera, učestvuju u obradi i kontroli audio signala unutar slušnog aparata. Preostala dva procesora su numerički procesori posebno usmereni na obradu numeričkih algoritama. Podržavaju širok spektar algoritama, uključujući Furijeove transformacije, vektorske operacije, logičke operacije, algoritme filtriranja, razne analize.

Na slici 2. prikazana je fizička arhitektura Sonnet familije procesora.



Slika 1. – Najvažnije komponente fizičke arhitekture Sonnet platforme

2.1.1 Sonnet SIDCore DSP processor

Sonnet SIDCore (engl. *Starkey Ideal Digital Signal Processor*) je potpuno opremljeno DSP jezgro sa dodatnim mikro-konrolerom, projektovano za relativno nisku potrošnju električne energije pri obradi audio signala unutar slušnih aparata.

Neke od važnijih karakteristika ovog DSP-a:

1. 56-bitni akumulatorski registri
2. Podrška za instrukcije tipa množenje-akumuliranje (engl. *Dual Multiply and Accumulate – DualMAC*)
3. Do četiri instrukcije po ciklusu
4. Harvard arhitektura memorije
5. 32-bitna programska memorija, sa kombinacijom 16-bitnim i 32-bitnim instrukcijama, što dovodi do povećanja gustine koda i smanjenja potrošnje energije
6. Do 64k memorijskih reči na svakoj od magistrala
7. 56-bitni pomerač (engl. *Shifter*)
8. Dupla preciznost (engl. *Double precision*)
9. Manipulacija bitima, pomeranje, rotacija, maskiranje, uslovna izvršavanja
10. Stek memorija namenjena za programske petlje
11. Kompatibilnost sa programskim jezikom C

Jedan od glavnih ciljeva SIDCore dizajna, ako ne i glavni cilj, je upravo postizanje minimalne potrošnje električne energije prilikom obrade audio signala unutar slušnog aparata. Minimalna potrošnja je glavni zahtev koji se može postići smanjenjem broja potrebnih ciklusa za izvršavanje zadataka, ubrzanjem pristupa memoriji, poboljšanje sistemskih komponenti uključujući dekodirajuću instrukciju, memorijske magistrale, operativne jedinice.

2.1.2 Sonnet SNAPCore DSP procesor

Sonnet SNAPCore (engl. *Starkey Numerical Accelerator Processor*) je programabilno DSP jezgro, predodređeno za veoma zahtevne numeričke obrade i kompleksne algoritme. Ovo jezgro podržava obrade nad širokim spektrom algoritama između ostalih i brze Furijeove transformacije, vremensko-frekventne analize, operacije nad vektorima kao i numeričke operacije.

Od glavnih zahteva koje SNAP jezgro treba da obezbedi, svakako glavni je minimalna potrošnja električne energije, da bi bio pogodan za kompleksne obrade unutar slušnog aparata. Ekvivalentno SID jezgru, glavni zahtevi se postižu smanjenjem broja potrebnih ciklusa za izvršavanje instrukcija, ubrzan pristup memoriji, usavršavanje sistemskih komponenti samog jezgra. Dizajn SNAP jezgra se ističe propusnim opsegom i paralelizmom na nivou izvršavanja instrukcija.

Neke od važnijih karakteristika ovog DSP-a:

1. Kompleksno množenje unutar jednog ciklusa
2. 56-bitni akumulatori predodređeni za kompleksne obrade
3. Podrška za filtre tipa FIR, IIR i vektorske operacije
4. Četiri memorijska porta podataka: XA, YA, XB, YB
5. Svaki port podržava 24-bitni i 48-bitni prenos
6. Podrška za instrukcije tipa množenje-akumuliranje (engl. *Quad Multiply and Accumulate – QuadMAC*)
7. Harvard arhitektura memorije
8. 32-bitna programska memorija
9. Visok nivo paralelizma prilikom izvršavanja instrukcija
10. 16-bitno adresiranje do 64k reči na svakoj magistrali podataka
11. 56-bitni pomerač
12. Dupla preciznost
13. Visok propusni opseg između procesora da bi se smanjio broj ciklusa i potrošnja energije.

2.2 Koncept i struktura Eclipse platforme

Eclipse je višezjezično programsko razvojno okruženje slobodno-dostupnog koda (engl. *Open source*), koje pruža proširivu razvojnu platformu. Zasniva se na Java programskom jeziku i omogućava razvoj alata, aplikacijskih okruženja kao i samostalnih aplikacija (engl. *Rich Client Platform – RCP*) [6].

Kako bi napravila zajedničku razvojnu platformu za svoje alate kompanija IBM 1998. godine počinje sa projektom Eclipse.

Konačan cilj je bio da se obezbedi određen broj korisnika, proizvođača i programera, koji bi koristili proširivu platformu za integraciju alata i sve to u Java programskom jeziku.

Eclipse razvojno okruženje mnoge kompanije su usvojile kao pogodno Java integrisano razvojno okruženje i platformu za integrisanje velikog broja različitih alata [7].

Proširujućom arhitekturom, odličnom funkcionalnošću i kvalitetom sam, Eclipse kao razvojno okruženje je okupio korisnike i proizvođače da se pridruže daljem unapređivanju, proširivanju funkcionalnosti i sve većoj upotrebi u programerskom svetu.

Pečat jedinstvenosti Eclipse dobija preko javne licence (engl. *Eclipse Public Licence*) koja je omogućila prava na sav izvorni kod i besplatnu distribuciju. Iako trenutno Eclipse ima ogromnu primenu i uticaj u razvijanju alata, namenskih sistema, ispitivanju programske podrške, mnoge kompanije i dalje pristupaju zajednici otvorenog koda i samim tim učestvuju u

konstantnom poboljšanju kvaliteta samog razvojnog sistema. Za uzvrat, kompanije razvijaju sopstvene alate u komercijalne svrhe i ostvaruju ogromnu dobit.

Programi i alati koji se razvijaju na Eclipse platformi mogu biti pisani u Java programskom jeziku, kao i u drugim programskim jezicima kao što su Ada, C, C++, COBOL, Fortran, Haskell, JavaScript, Perl, PHP, Python, R, Ruby, Scala, Clojure, Groovy, Scheme i Erlang, uz pomoć raznih priključaka.

Najmanja funkcionalna celina Eclipse platforme naziva se utično proširenje odnosno priključak (engl. *Plug-in*). Složeniji alati obično se sastoje od većeg broja utičnih proširenja. Utična proširenja se implementiraju u Java programskom jeziku. Najčešće se utično proširenje sastoji od Java koda u Java arhiviranoj biblioteci, dodatnih resursa kao što su slike, ikone, katalozi poruka, biblioteka sa funkcijama izvornog programskog jezika (ako je u pitanju integrisano razvojno okruženje). Svako utično proširenje ima svoj opisni deo koji definiše veze tog utičnog proširenja sa ostalim utičnim proširenjima u sistemu. Model povezivanja je takav da svako utično proširenje može definisati svoje tačke proširenja na koje se druga utična proširenja mogu povezati i sam se povezati na jedno ili više drugih utičnih proširenja. Svaka tačka proširenja definiše svoju programsku spregu koju sva utična proširenja koja se povežu sa njom moraju implementirati. Pri pokretanju platforme, jezgro platforme pretražuje dostupna utična proširenja, čita njihove opisne datoteke i popunjava registar utičnih proširenja (engl. *Plugin register*) ovim informacijama. Pri tome se vrši provera kompatibilnosti utičnih proširenja i tačaka na koje se povezuju. Utična proširenja se dodaju i uklanjaju u toku rada sa platformom.

Opisni deo utičnih proširenja se sastoji od dve datoteke. Prva je manifest.mf datoteka koja je napisana po OSGi standardu i opisuje zavisnost utičnog proširenja od drugih utičnih proširenja. Druga je plugin.xml koja u XML jeziku opisuje tačke proširenja utičnih proširenja (one koje definiše i one na koje se povezuje). Tačka proširenja (engl. *Extension Point*) može definisati i svoja dodatna polja u XML jeziku koja koriste utična proširenja koja se povezuju na nju. Na ovaj način utično proširenje koje koristi tačku proširenja može dati potrebne informacije utičnom proširenju koje je definisalo tu tačku. Što je još važnije, svim informacijama o tačkama proširenja se može pristupiti iz registra utičnih proširenja bez aktiviranja samih utičnih proširenja i učitavanja bilo kakvog koda. Ovako se omogućava da platforma sadrži veliku bazu utičnih proširenja od kojih će biti učitani samo oni koji su neophodni.

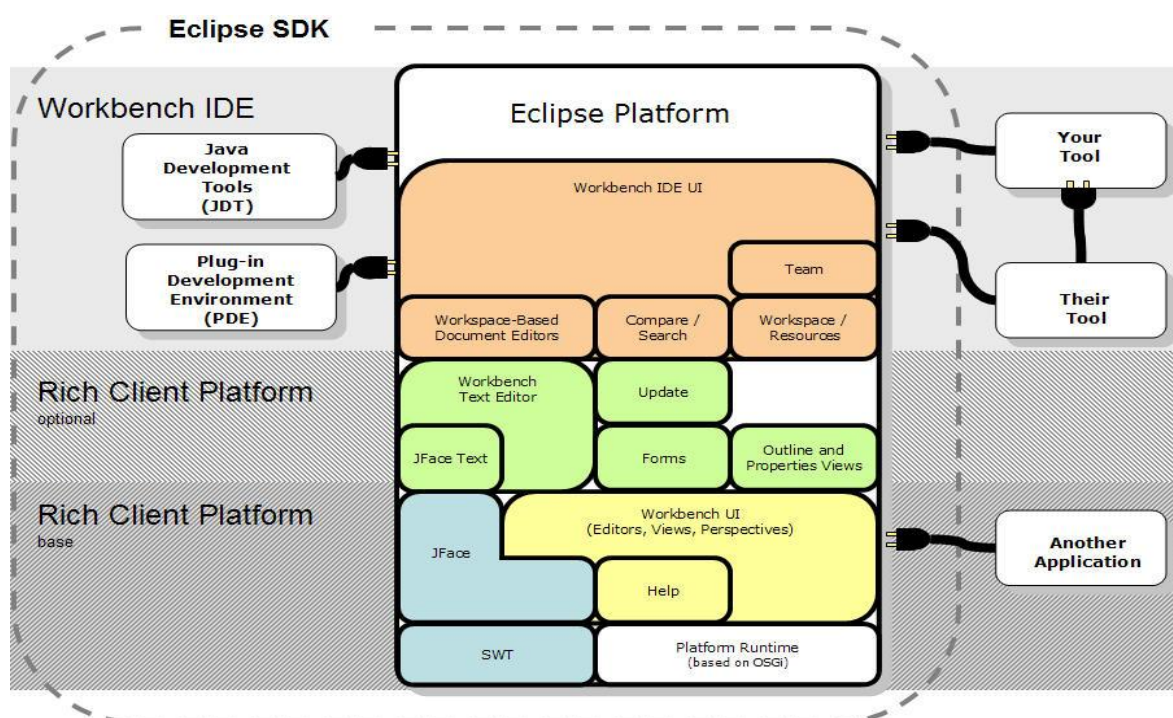
Utična proširenja se aktiviraju tek kad to bude potrebno. Kad se pokrenu ona koriste registar utičnih proširenja da otkriju i pristupe utičnim proširenjima povezanim na njihove tačke proširenja. Utično proširenje će ostati aktivirano sve dok ne bude eksplicitno deaktivirano ili dok platforma ne bude ugašena. Svako utično proširenje je opremljeno direktorijumom u kome čuva svoje podatke tako da stanja i vrednosti promenljivih iz jedne sesije, mogu ako je potrebno biti

sačuvani za sledeću sesiju. Poznavanje svih dostupnih utičnih proširenja pre pokretanja i mogućnost razmene informacija o njima pre njihovog pokretanja skraćuje vreme potrebno za njihovo pokretanje, memorijske zahteve aplikacije i uklanja probleme vezane za redosled pokretanja utičnih proširenja.

Platformu pokreće Java virtuelna mašina. Svako utično proširenje ima svoj Java pokretač klasa (engl. *Java class loader*) koji je zadužen za učitavanje njegovih klasa. Svako utično proširenje eksplicitno deklarira svoju zavisnost od drugih utičnih proširenja čijim klasama će pristupati i kontroliše vidljivost javnih (engl. *Public*) klasa i programskih sprega svojih biblioteka. Ove informacije se nalaze u opisnim datotekama, a pravila vidljivosti su podržana tokom izvršavanja od strane pokretača klasa.

Jezgro Eclipse platforme se sastoji iz nekoliko osnovnih podsistema [8]:

1. Radno okruženje (engl. *Workspace*)
2. Radni sto (engl. *Workbench*)
3. Podsistem za pomoć korisniku (engl. *Help*) - mehanizam preko kojeg Eclipse alati mogu definisati i dodati svoju dokumentaciju u nezavisne knjige dokumentacije. Sadržaj dokumentacije se dodaje u vidu HTML datoteka. Organizovanje sadržaja u nezavisne knjige i definisanje struktura za navigaciju opisuje se u posebnim XML datotekama.
4. Podsistem za timski rad – Eclipse platforma omogućava da projekat u radnom okruženju bude uključen u sistem za rukovanje verzijama i timskim radom. Na taj način više programera je u mogućnosti da radi na istom projektu.



Slika 2. - Najvažnije komponente Eclipse platforme

2.2.1 Radno okruženje

Radno okruženje se sastoji od jednog ili više projekata koji zapravo predstavljaju odgovarajući direktorijum u direktorijumskoj strukturi. Projekti se mogu nalaziti u bilo kom direktorijumu mada se po pravilu nalaze u odgovarajućim poddirektorijumima, direktorijuma radnog okruženja. Da bi se opisao tip projekta koristi se mehanizam svojstva (engl. *Nature*) projekta. Utična proširenja mogu definisati nove tipove svojstava projekata i definisati način na koji se oni konfigurišu. Takođe jedan projekat može imati i više svojstava tako da može biti deljen između većeg broja utičnih proširenja. Na taj način SIDE sistem poseduje podršku za više različitih tipova projekata od kojih su od interesa Sid Core Image, Snap Core Image i Assembly Executable, jer izvorni programski kodovi projekata sa ovim svojstvima se uređuju unutar uređivača asemblerskog koda.

Projekti sadrže datoteke koje stvara i kojima u potpunosti upravlja korisnik. Eclipse razvojna platforma definiše programsku podršku za rad sa resursima radnog okruženja. Sami resursi su predstavljeni prilagodljivim objektima preko kojih im se može definisati funkcionalnost. Kao zaštitu od slučajnog gubitka određenog resursa, radno okruženje nudi i jednostavan mehanizam koji prati kratku istoriju izmena na datotekama tako da one mogu biti vraćene u neko od prethodnih stanja.

Radno okruženje nudi mehanizam markera za obeležavanje resursa. Oni služe za obeležavanje delova u datotekama kao što su: greške prevodioca, delovi koda koje treba doraditi, rezultati pretraga (engl. *Search hits*), obeleživači pozicije (engl. *Bookmarks*), radni zadatci (engl. *Tasks*) i prekidne tačke (engl. *Breakpoints*). Ovaj mehanizam je otvoren tako da utična proširenja mogu definisati nove tipove markera i načine njihovog korišćenja.

Proces snimanja sadržaja radnog okruženja je takođe otvoren za povezivanje utičnih proširenja koja žele da ostanu sinhronizovana sa okruženjem u različitim sesijama. Potrebni elementi utičnih proširenja koji se prijave biće sačuvani u direktorijumskoj strukturi memorije računara do sledeće sesije. Po sledećem pokretanju konkretnog utičnog proširenja, ono prima izveštaj o izmenama u mreži resursa u odnosu na svoje poslednje snimljeno stanje. Na ovaj način utično proširenje može preneti svoje stanje u sledeću sesiju, a ipak odreagovati na promene koje su se desile dok je bilo neaktivno.

2.2.2 Radni sto

Korisnička sprega Eclipse platforme definiše utična proširenja radnog stola koja sadrže strukture preko kojih alati komuniciraju sa korisnikom.

Zbog ove svoje centralne uloge radni sto je ujedno i sinonim za korisničku spregu Eclipse platforme. Programska sprega radnog stola se oslanja na programsku spregu dve grafičke

biblioteke opšte namene: SWT (engl. *Standard widget toolkit*), i u manjoj meri JFace. Osnovni elementi korisničke sprege Eclipse platforme su uređivači teksta (engl. *Editors*), pogledi (engl. *Views*) i perspektive (engl. *Perspectives*).

Uređivači teksta omogućavaju korisniku da otvori, izmeni i sačuva datoteke. U tom smislu su slični utičnim proširenjima koja rade sa resursima u direktorijumskoj strukturi, ali su mnogo bliže povezani sa radnim stolom. Kad su aktivni, uređivači teksta mogu dodati svoje akcije u menije ili paletu sa alatkama. Platforma definiše i standardni uređivač teksta za tekstualne datoteke. Uređivači za posebne tipove datoteka moraju biti prilagođeni od strane utičnih proširenja.

Pogledi prikazuju podatke o nekom objektu sa kojim korisnik radi. Na primer, dok korisnik unosi programski kod u nekom uređivaču teksta, pogled može biti upotrebljen da prikaže informacije o tom delu koda (funkcijama i promenljivama) definisanim u tekućem programskom bloku i slično). I pogled može biti izvor sadržaja koji će biti prikazan u drugom pogledu (na primer: jedan pogled može prikazivati detalje objekta izabranog u drugom pogledu). Pogledi imaju jednostavniji životni ciklus od uređivača teksta – najčešće su izmene u njima momentalno sačuvane i njihove posledice odmah prikazane u ostalim delovima korisničke sprege – platforme (engl. *User Interface - UI*).

UI platforma može imati veći broj perspektiva, ali u datom momentu samo jedna može biti prikazana. Perspektiva može sadržati više različitih uređivača teksta i pogleda. Njihov izbor i raspored zavise od svrhe konkretne perspektive. Platforma ima unapred definisane perspektive za navigaciju kroz resurse, pregledanje dokumenata za pomoć korisniku (engl. *Help*) i podršku za rad u timu.

Utična proširenja se integrišu sa UI platformom na jasno definisane načine. Glavne tačke proširenja omogućavaju proširenje radnog stola na sledeće načine:

1. Dodavanjem novih uređivača teksta
2. Dodavanjem novih tipova pogleda
3. Dodavanjem novih perspektiva (koje raspoređuju stare i nove uređivače i poglede)

Platforma vodi računa o elementima perspektive, učitava ih po potrebi i zatvara kad više ne budu neophodni. Ikone i nazivi njihovih akcija se nalaze u opisnim datotekama utičnih proširenja tako da stavke u meniju mogu biti iscrtane bez učitavanja samih elemenata.

2.2.3 Eclipse alati za razvoj programske podrške

Eclipse alati za razvoj programske podrške (engl. *Eclipse Software Development Kit-SDK*) je kombinacija nekoliko Eclipse projekata, uključujući Eclipse platformu, Java razvojne alate (engl. *Java Development Tools-JDT*) i okruženja za razvoj utičnih proširenja (engl. *Plug-in*

Development Environment-PDE). Eclipse SDK je integrisano razvojno okruženje za razvoj Java aplikacija i izgradnju proizvoda baziranih na Eclipse platformi. Ova platforma, nezavisna od proizvođača, sastoji se od jezgra (engl. *Core*) i raznih priključaka (engl. *Plugin*). Jezgro pruža usluge za upravljanje priključcima, a priključci pružaju potpunu funkcionalnost.

2.2.4 Eclipse integrisano razvojno okruženje

Integrisano razvojno okruženje (engl. *Integrated Development Environment-IDE*) je programska podrška koja programerima pruža širok skup objekata za razvoj programskih aplikacija. Uobičajeno se sastoji od: uređivača izvornog koda, kompajlera (interpretera), alata za građenje koda i pronalaženje grešaka.

2.2.5 Eclipse okruženje za razvoj priključaka

Eclipse okruženje za razvoj priključaka (engl. *Plugin Development Environment-PDE*) predstavlja skup alata dizajniranih da pomažu Eclipse programeru u razvoju, ispitivanju, izgradnji izvršnih arhiva i isporuci (engl. *Deployment*) Eclipse dodataka, fragmenata, mogućnosti (engl. *Features*) i RCP aplikacija.

PDE se sastoji od tri glavne komponente:

1. Građenje koda (engl. *Build*)
2. Aplikativna programska sprega (engl. *Application Programmer Interface-API*)
3. Korisnička UI sprega

Svaka od ovih komponenti funkcioniše samostalno. Postoje takođe dve dodatne komponente. Dokumentacija (engl. *Documentation*) koja omogućava pomoćnu dokumentaciju korisniku i inkubator (engl. *Incubator*) koja služi za razvoj alata i nadogradnji koje nije moguće smestiti u Eclipse SDK.

2.2.6 Eclipse aplikativna programska sprega

Aplikativna programska sprega radnog stola i njegova implementacija zasnivaju se na dve grupe alata, SWT i JFace biblioteke. Ove dve biblioteke se koriste za razvijanje grafičke sprege sa korisnikom u Eclipse okruženju.

2.2.6.1 Java grafička biblioteka - SWT

SWT je biblioteka grafičkih elemenata integrisana sa grafičkim delom operativnog sistema, pisana u Java programskom jeziku. Razvijena je kao deo Eclipse projekta od strane IBM tima, kao zamena za AWT i SWING. Omogućava razvoj prenosivih aplikacija koje direktno pristupaju grafičkom delu bez obzira koji operativni sistem je u pitanju. Celokupna Eclipse platforma i njeni alati koriste SWT za predstavljanje grafičkih informacija korisniku.

Glavni nedostatak AWT (engl. *Abstract Windowing Toolkit*) grafičke biblioteke je njena integracija i prenosivost na operativnim sistemima. AWT je bila direktno u interakciji sa grafičkim elementima operativnog sistema, ali onog na kojem je razvijan određen alat ili sistem. Dok je AWT biblioteka podržavala samo elemente nižeg nivoa, SWING je imao problema sa performansama i prenosivošću zbog velikih memorijskih zahteva i iz tog razloga javila se potreba da se ovaj nedostatak odstrani [9]. Tako je IBM tim rešio da razvije potpuno novu biblioteku.

SWT unapređuje AWT tako što koristi elemente grafičkog podsistema operativnog sistema kad god je to moguće, a za elemente koji nisu podržani od operativnog sistema vrši se njihova emulacija. SWT teži da održi funkcionalnost i verodostojan izgled operativnog sistema na kome se koristi.

2.2.6.2 Java grafička biblioteka - Jface

JFace je biblioteka sa klasama koje služe za programiranje ulazno-izlaznih zadataka korisničke sprege, radi sa SWT bibliotekom i potpuno je nezavisna od operativnog sistema na kojem se izvršava.

Sadrži klase koje omogućavaju rukovanje sa:

1. Pogledima (engl. *Views*)
2. Akcijama (engl. *Actions*)
3. Slikama (engl. *Images*)
4. Fontovima (engl. *Fonts*)
5. Dijalozima (engl. *Dialogs*)
6. Čarobnjacima (engl. *Wizards*)

Pogledi kao jedan od važnijih elemenata Jface biblioteke predstavljaju prozore koji prikazuju i rukuju sa listama, stablima i tabelama. Sadržaj koji se prikazuje u pogledima može se dodatno obrađivati u vidu sortiranja, filtriranja, nazublјivanja, prevlačenja, obeležavanja, iz razloga što pogled podržava akcije kao što su pritisak tastera na mišu, vraćanje pogleda na poslednju poznatu verziju, kretanje kroz sadržaj pogleda.

Akcije kao još jedan od važnijih elemenata ove biblioteke omogućavaju da korisničke komande mogu biti definisane nezavisno od toga gde se nalaze u korisničkoj sprezi. Akcija je komanda koja može biti pokrenuta pritiskom na dugme, opciju menija, polje u paleti sa alatima.

Jface biblioteka omogućava rad sa fontovima i slikama kao i njihovo međusobno deljenje među korisničkim alatima.

Dijalozi i čarobnjaci definišu okvir koji služi za kompleksnije interakcije sa korisnikom. Omogućavaju korisniku da kroz jedan ili više uzastopnih koraka izvrši određene akcije. Koraci su praćeni odgovarajućom pomoćnom asistencijom.

2.3 Programski jezici

Programski jezici se prema širini oblasti primene mogu podeliti na dve osnovne kategorije:

1. Jezici opšte namene (engl. *General Purpose Language - GPL*), kao što su: JAVA, C i C++.
2. Usko namenski jezici (engl. *Domain Specific Language - DSL*), kao što su: Xtext, SQL, HTML, Verilog, VHDL i drugi.

Jezici opšte namene se koriste za razvoj i održavanje širokog spektra algoritama na računarskim sistemima u čemu se i ogleda njihova prednost u odnosu na usko namenske jezike.

Pored široke primene GPL jezika, određeni računarski sistemi i algoritmi zahtevaju drugačiji pristup razvoju programske podrške. DSL jezici, kao specijalizovani jezici za rešavanje zahteva sa specifičnim ciljevima u dizajnu i implementaciji iz određene oblasti, obezbeđuju lakoću korišćenja, jednostavan način da se formulišu svi aspekti sistema koji se razvija, i na kraju rešenje koje u potpunosti ispunjava zahteve sistema. SQL je najpoznatiji primer DSL jezika na kojem nije moguće napisati web aplikaciju, ali zato je za manipulaciju velikom količinom podataka izuzetno efikasan.

Oblast primene jednog DSL jezika je praktično svedena na rešavanje jednog problema. DSL jezik se od samog početka dizajnira i razvija za konkretan problem i upravo iz tog razloga ga nazivamo usko namenski jezik. Tako da za svaki DSL jezik koji je razvijen postoji tačno predodređena namena.

2.4 Xtext okvir

U domenu razvoja jezika sa specifičnom sintaksom, namenjenom za konkretan problem, značajno mesto zauzima Xtext usko namenski jezik. Ovaj DSL jezik je nastao kao deo oAW3 (engl. *OpenArchitectureWare*) projekta. Domen Xtext DSL jezika je kreiranje DSL jezika tako da možemo reći da Xtext predstavlja meta-jezik.

Implementacija je zasnovana na kreiranju odgovarajućeg parsera i objektnog meta-modela na osnovu opisa zadatah na meta-jeziku.

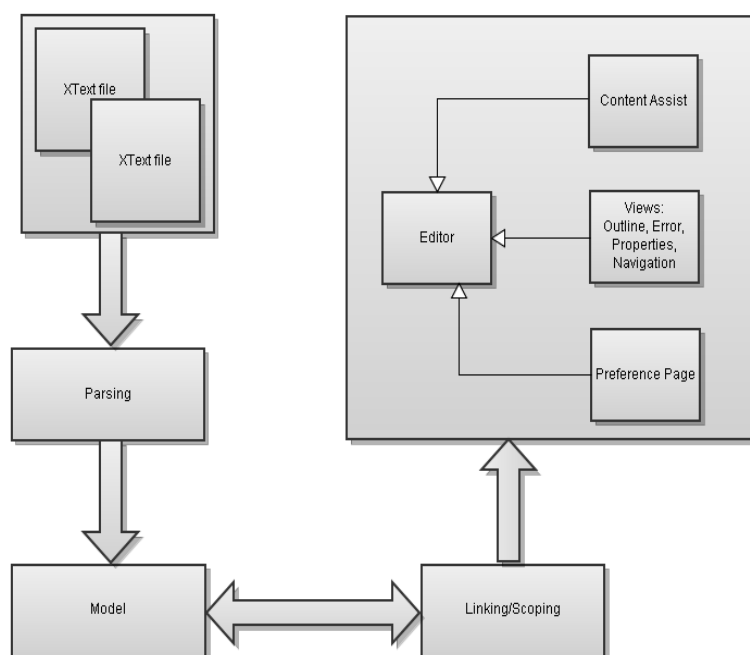
Rezultat parsiranja Xtext gramatike u ovom radu je asemblerski uređivač koji podržava asemblerske instrukcije.

Pošto su DSL jezici predodređeni za manju ciljnu oblast, u tom duhu je i u ovom radu upotrebljen Xtext okvir za razvoj uređivača čija je namena razvijanje programske podrške koja omogućava razvoj asemblerskih aplikacija za ciljnu Sonnet platform.

2.4.1 Karakteristike Xtext okvira

Xtext definiše sledeće:

1. Skup AST klasa predstavljenih kao EMF (engl. *Eclipse Modeling Framework*) meta-model.
2. Parser koji je u mogućnosti čitati tekstualnu sintaksu, nakon čega vraća generisani meta-model.
3. Eclipse uređivač koji nam omogućava predstavljanje pravila sintakse jezika u više boja, prediktivnog asistenta dopune programskog koda, konfigurisanu listu koju čine elementi sintakse, kao i proveru grešaka u sintaksi.



Slika 3. – Blok šema Xtext modela

Na slici 3. prikazana je blok šema modela koja predstavlja smer razvoja uređivača sa podrškom za razvoj asemblerskih aplikacija.

Nakon što je na pravi način sagledan problem pravi se model koji će dovesti do rešenja. Na početku se pišu pravila jezika odnosno u ovom slučaju specifična asemblerska gramatika. Od toga kako je definisana gramatika, u daljem procesu razvijanja nam zavisi funkcionalnost automatskog prediktivnog asistenta, sam sadržaj pogleda koju sačinjavaju elementi sintakse, kao i mogućnost verifikacije grešaka.

Nakon ispravno definisane gramatike vrši se parsiranje iste i kao rezultat dobija se model sa pravilima novog jezika. Međusobna veza unutrašnjih pravila zavisi od načina definicije istih. Pravila se pišu u formi stabla sa početkom na vrhu pa kroz propadanje i grananje dolazi se do najsitnijih detalja.

U sledećem koraku vrši se povezivanje kao i automatsko generisanje pojedinih delova modela. Sve ovo omogućava Xtext okvir, kao i samu izmenu svih generisanih klasa. Pravilnim definisanjem sintakse dobija se pogled sa sadržajem uređivača asemblerskog koda, automatski prediktivni asistent, spisak grešaka koje je korisnik napravio, kao i stranica sa mogućim dodatnim podešavanjima koja su vezana za rezultujući uređivač.

3. Koncept rešenja

U ovom poglavlju dat je koncept rešenja uređivača asemblerskog koda koji predstavlja proširenje Sonnet integrisanog razvojnog okruženja, sa podrškom za razvoj asemblerskih aplikacija na ciljnoj više-jezgarnoj Sonnet platformi. Da bi se prvo razvila asemblerska aplikacija, a potom i našla na ciljnoj platformi, potrebna je odgovarajuća grafička sprega unutar Sonnet razvojnog okruženja, kao i sprega između SIDE okruženja i Sonnet platforme. Realizovano proširenje Sonnet okruženja, u vidu uređivača asemblerskog koda predstavlja grafičku spregu i prvi korak u procesu razvoja asemblerske aplikacije.

3.1 Xtext gramatika

U ovom primeru predstavljen je deo procesa razvijanja uređivača sa podrškom za specifičnu asemblersku sintaksu. Priložen je jednostavan primer koji za zadatak ima da pokaže na koji način se može definisati gramatika u Xtext okviru da bi se u asemblerskom uređivaču mogle koristiti imenovane sekcije koda (engl. *Label*) kao i celobrojne konstante (engl. *Integer constant*). Ispod je prikazan primer Xtext gramatike, odnosno opis programske podrške koja mora da obezbedi korišćenje asemblerskog jezika unutar uređivača koda.

```

grammar sonnet.ide.xtext.Asm with org.eclipse.xtext.common.Terminals
generate asm "http://www.ide.sonnet.xtext/Asm"
import "http://www.eclipse.org/emf/2002/Ecore" as ecore
Model: (instr+=Instruction)*;
Instruction: Label | Dot | Preprocessor_commands;
Label: LabelWithPunkt | LabelWithEndl;
LabelWithPunkt: FirstOfLabel command+=Commands+ ;
FirstOfLabel: (name=ID | name1=INT) punkt=Punkt;
Punkt: punkt=(':', '::');
...
Numbers: Integer_constant | Floating_point_constant;
Integer_constant: Binary | Octal | Decimal | Hexadecimal;
Decimal: INT | INTDECIMAL;
Hexadecimal: HEXA;
Binary: BINARY;
Octal: OCTAL;
...
terminal BINARY: '-'? '0b' ('0' | '1')+;
terminal OCTAL: '-'? '0' ('0'..'7')+;
terminal HEXA: '-'? '0x' ('a'..'f' | 'A'..'F' | '0'..'9')+;
terminal INT returns ecore::EInt: ('0'..'9')+;
terminal INTDECIMAL: '-'? ('0'..'9') ('0'..'9')*;
terminal ID: '^'? ('a'..'z' | 'A'..'Z' | '_') ('a'..'z' | 'A'..'Z' | '_' | '0'..'9' | '.')*;

```

Osnovno parsiranje gramatike može biti odvojeno u sledeće faze:

1. Leksička analiza (engl. *Lexing*)
2. Parsiranje (engl. *Parsing*)
3. Povezivanje (engl. *Linking*)
2. Potvrđivanje (engl. *Validation*)

Xtext definiše određena semantička pravila na nivou gramatike. Ova pravila omogućavaju izvođenje semantičke analize nad stablom parsiranja samo na osnovu tekstualnog opisa jezika.

Neka od semantičkih pravila koja se navode na nivou Xtext gramatike su:

1. Dodele vrednosti – iskaz $(instr \ += \ Instruction)^*$; definiše gramatičko pravilo prepoznavanja većeg broja iskaza koji su tipa *Instruction*. Ovaj iskaz istovremeno opisuje semantiku koja kaže da će se kreirati atribut *instr* koji će predstavljati listu objekata tipa *Instruction* a sve to zbog kvantifikatora zvezde (*).
2. Akcije koje se definišu unutar zagrada $\{ \}$. Akcije u Xtext-u omogućavaju redefinisane povratnog tipa.
3. Reference predstavljene sa $instr = [Instruction]$; predstavljaju gramatičko pravilo koje obezbeđuje kasnija referenciranja, nakon definisanja. Na taj način obezbeđuje se kretanje kroz asemblerski kod unutar uređivača.

Xtext implementacija koja je deo EMF-a pored kreiranja parsera i odgovarajućeg meta - modela kreira i odgovarajući uređivač koji se u potpunosti integriše u Eclipse okruženje.

Prilikom parsiranja, u prvoj fazi niz karaktera, odnosno tekstualni ulaz se deli na tokene (engl. *Tokens*). Token je kratak deo ulaznog niza karaktera i sadrži jedan ili više karaktera. Token je pokazivač na neko drugo pravilo ili na neko od ugrađenih terminal pravila (ID, INT itd.) Terminal pravila su takođe tokeni. Postoji konvencija o imenovanju terminal pravila velikim slovom. Xtext automatski pravi meta model iz definisane gramatike.

Pravila parsiranja u primeru su: *Model, Instruction, Label, LabelWithPunkt, FirstOfLabel, Punkt, Numbers, Integer_constant, Binary, Hexadecimal, Octal, Decimal*, dok su terminal pravila: *BINARY, OCTAL, HEXA, INT, INTDECIMAL, ID*. Telo pravila se nalazi iza dvotačke i sastoji se od definicije terminala i neterminala. Terminali se navode kao konstante karaktera (npr. ključna reč '*0b*' kod pravila *BINARY*) dok se neterminali kreiraju uvezivanjem na druga pravila parsiranja. Operatorima dodele je omogućeno definisanje vrednosti atributa instanci klasa odgovarajućeg meta modela. Takođe je moguća upotreba standardnih kvantifikatora:

1. ? - nula ili jedan broj pojavljivanja u sintaksi
2. * - nula ili više puta je moguće pojavljivanje u sintaksi
3. + - jedan ili više ponavljanja u sintaksi
4. kao i predikata: ! - ne i & - i

Gramatika počinje sa određenim zaglavljem koje definiše neke karakteristike gramatike. Prvi deo zaglavlja predstavlja ime gramatike dok drugi deo definiše drugu postojeću gramatiku koja može da se uvezuje i iskoristi.

Prvo pravilo u primeru (*Model*) se naziva ulazno pravilo. Ono predstavlja skup svih instrukcija obuhvaćenih gramatikom kojom se realizuje podrška asemblerskog jezika. Pravilo koje omogućava podršku za pisanje imena počinje ID ili INT pravilom (a-z, A-Z, 0-9, _) koje obuhvata navedene karaktere, cifre, kao i donju crtu. Nakon ovog pravila stavlja se dvotačka iza koje sledi bilo koja instrukcija iz seta instrukcija *Commands*.

Druga instrukcija je nekoliko celobrojnih konstanti:

1. Binarna (engl. *Binary*)
2. Decimalna (engl. *Decimal*)
3. Oktalna (engl. *Octal*)
4. Heksadecimalna (engl. *Hexadecimal*)

Svaka od ovih konstanti osim decimalnog formata zapisa počinje sa prefiksom na koji se nastavlja niz cifara. Pa tako za binarni format prefix je *0b*, za oktalni *0* a za heksadecimalni *0x*. Na prefikse se nastavlja niz cifara dok za heksadecimalni format pored niza cifara imamo i karaktere A-F, a-f.

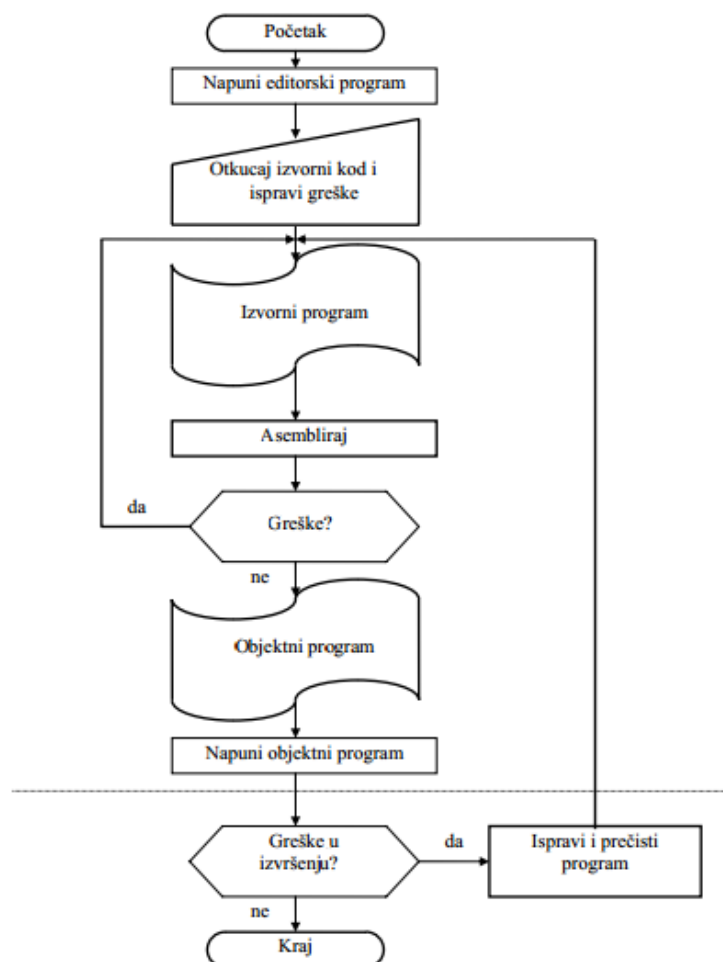
3.2 Uređivač asemblerskog koda

Uređivač asemblerskog koda se koristi za razvoj i naknadne izmene izvornog koda koji je potrebno prevesti od strane asemblera. Kada u izvornom asemblerskom kodu ne postoje greške, generiše se objektni kod koji se puni u memoriju na ciljnoj platformi radi izvršavanja.

Izvršavanje koda na fizičkoj arhitekturi se postiže konfigurisanjem ploče i slanjem izvršnog koda na razvojnu ploču putem kontrolera za kontrolisano izvršavanje. Komunikacija između razvojne ploče i računara obavlja se slanjem odgovarajućih komandnih reči. Ovim putem je moguća kontrola izvršenja programa, izvršavanje koda instrukciju po instrukciju, prikazivanje neposrednih rezultata nakon izvršenja svake instrukcije i pristup sadržajima elemenata procesora potrebnih za otkrivanje i otklanjanje grešaka. Greške u toku razvoja asemblerske aplikacije uočavaju se u ovom stepenu. Da bi se greške korigovale izvorni kod mora da se ponovo uređuje unutar uređivača i potom asemblira.

Da bi razvijeni programski kod nakon asembliranja i povezivanja mogao da se izvršava na ciljnoj fizičkoj arhitekturi potrebno je da se takav kod putem modula za komunikaciju sa ciljnom platformom smesti u neizbrisivu memoriju na samoj platformi.

Proces punjenja uređivača a potom asembliranja i provere grešaka prikazan je na slici 4.



Slika 4. – Proces punjenja uređivača i provere sadržaja

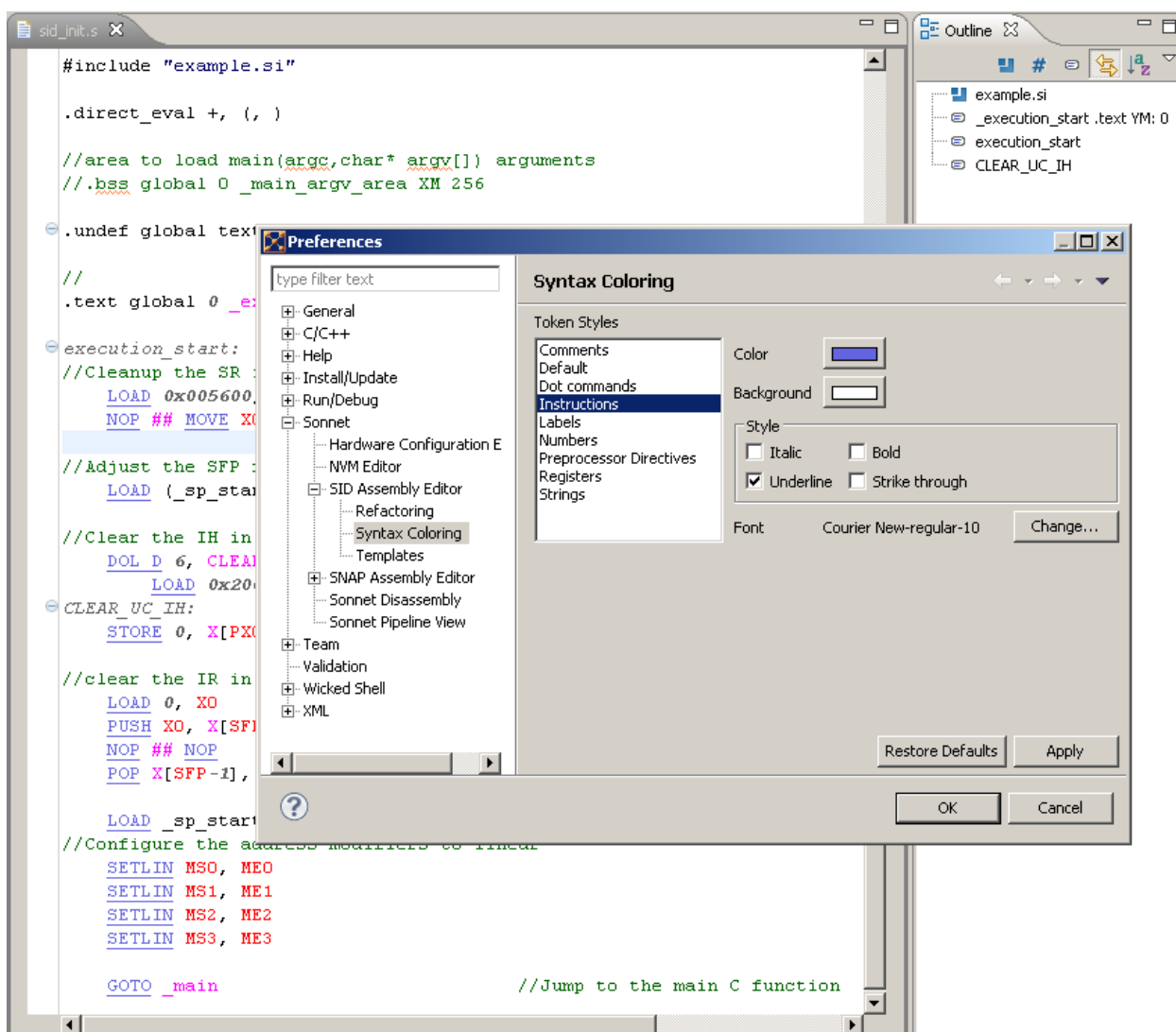
U uređivaču asemblerskog koda implementirane su sledeće funkcionalnosti:

1. Prezentacija teksta i mogućnost njegove izmene
2. Standardne operacije sa tekстом: kopiranje, premeštanje i dodavanje (*engl. Cut, Copy, Paste*)
3. Funkcije za pretragu i automatsku izmenu teksta (*engl. Find-Replace*)
4. Ugrađeni meniji
5. Vizuelna prezentacija beleški u tekstu na površinama uz levu ivicu površine za unos teksta - marginama (*engl. Ruler bar*)
6. Prikaz dodatnih informacija kao što su brojevi linija
7. Mogućnost bojenja ključnih reči (*engl. Syntax highlighting*)
8. Mogućnost definisanja posebnog ponašanja u određenim kontekstima
9. Mogućnost obeležavanja teksta u prozoru njegovim podvlačenjem
10. Automatsko osvežavanje svih pogleda koji imaju veze sa elementima uređivača, dok korisnik unosi tekst
11. Dodavanje lebdećeg teksta (*engl. Hover*) iznad margine i samog teksta
12. Automatsko („pametno“) uvlačenje teksta

3.2.1 Bojenje ključnih reči

Da bi se prikaz u prozoru uređivača teksta formatirao na željeni način odmah nakon što korisnik unese tekst, koristi se koncept oštećenja i popravke (*engl. Damage and repair*). Svaki znak koji unese korisnik potencijalno zahteva da se deo programskog koda oko tog znaka drugačije formatira. Određivanje dela teksta koji je potrebno ponovo formatirati zove se utvrđivanje oštećenja. Klase koje ovo rade implementiraju programsku spregu `IPresentationDamager`. Izračunavanje novog formata oštećenog teksta zove se popravka i ovo rade klase koje implementiraju programsku spregu `IPresentationRepairer`. Celokupan proces održavanja odgovarajućeg formata prikaza, dok se kod u uređivaču menja, naziva se poravnavanje (*engl. Reconciling*). Klase koje opisuju celokupan proces implementiraju programsku spregu `IPresentationReconciler`. Objekti ove klase dele dokumente u regione na osnovu sadržaja i vode računa da pozvane klase za utvrđivanje i popravku oštećenja budu odgovarajuće za dati tip sadržaja. Svaka klasa za poravnavanje dolazi sa parom klasa za utvrđivanje i popravku oštećenja. Platforma nudi i generičke klase za utvrđivanje i popravku oštećenja, koje se konfigurišu setom pravila u vidu klase `DefaultDamagerRepairer`. Pri instanciranju objekta ove klase ona se konfiguriše zadavanjem odgovarajućeg objekta tipa `RuleBasedScanner` koji sadrži pravila na osnovu kojih se formatira tekst. Pravila za formatiranje teksta u asemblerskom kodu za implementirani simulator opisana su u klasi `CLScanner`. Ova klasa sadrži niz pravila za formatiranje (bojenje) teksta izvornog koda. Određene grupe simbola se prikazuju u određenoj boji i stilu slova. Definisana su sledeća pravila (koja su implementirana podešavanjem ponuđenih generičkih pravila):

1. Pravilo za nizove znakova (engl. *String*). Obuhvata sadržaje između znakova navoda.
 2. Pravilo za prazna mesta (tabulatore, znakove razmaka i kraja linije).
 3. Pravilo za komentar (sadržaj linije iza simbola #).
 4. Pravilo za labele u kodu (labele u asemblerskom kodu moraju se nalaziti uz levu ivicu koda).
 5. Pravilo za bojenje ključnih reči – ovde je definisano nekoliko grupa:
 - Instrukcije (ADD, SUB, LOAD, STORE, itd.).
 - Tipovi podataka – za inicijalizaciju promenljivih u memoriji (.text, .bss, .data, .rodata, itd.).
 - Direktive – (#define, #include, itd.).
 - Simboli promenljivih – nazivi registara (A0, A1, B0, B1, itd.).
 6. Pravilo za brojeve (decimalni, heksadecimalni, oktalni, itd.).
- Boje i stilovi koji će biti primenjeni su definisani u posebnoj klasi `CLColorConstants`.



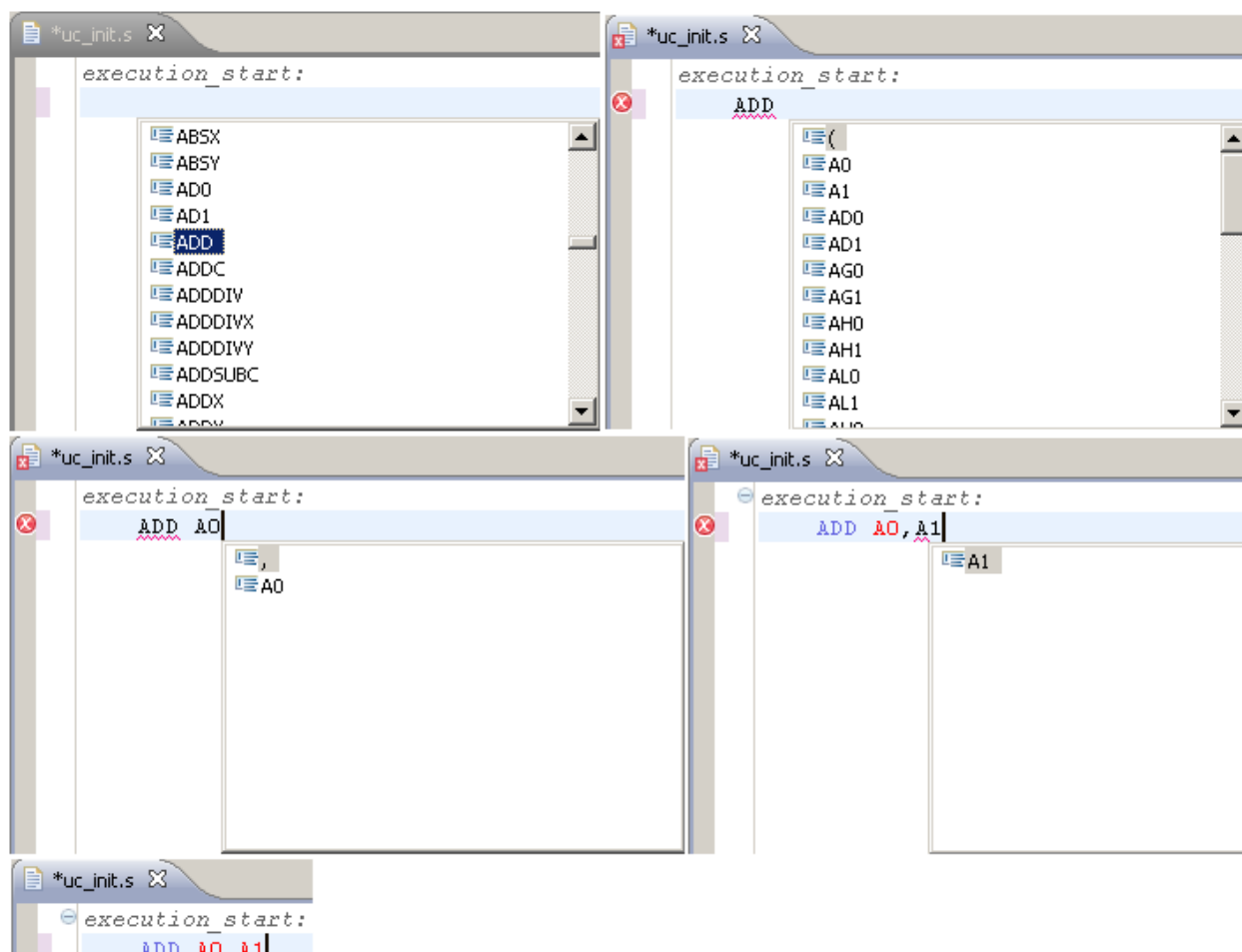
Slika 5. – Stranica za podešavanje bojenja ključnih reči

3.2.2 Formatiranje sadržaja uređivača

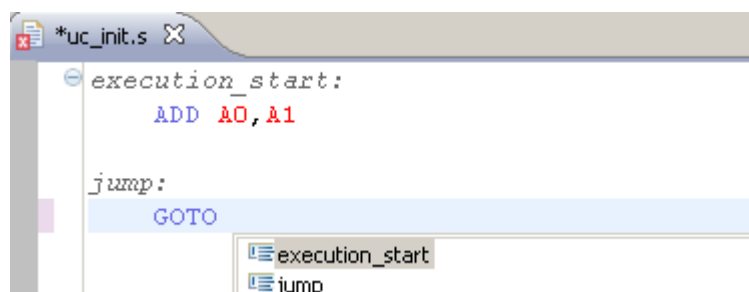
Uvlačenje predstavlja način formatiranja koji omogućava da se nakon pritiska tastera za prelazak u novi red, kursor za unos koda nađe na onoj udaljenosti od početka linije na kojoj je počeo unos koda u prethodnoj liniji. Ovo relativno olakšava pisanje programskog koda, kod kojeg se strukture lakše prepoznaju kad su komande istog bloka u istoj vertikalnoj liniji. Takođe sa ovim mehanizmom je podržano i lakše restrukturiranje koda jer je moguće označiti više linija i sve ih uvući za isti broj praznih mesta. Da bi se ovo postiglo dodata je klasa koja opisuje strategiju za ovaj način automatskog formatiranja teksta – `CLAutoIdentStrategy` koja je realizovana po ugledu na sličnu klasu iz JDT utičnog proširenja. Ova klasa se dodaje u listu strategija automatskog formatiranja teksta, kojoj uređivač teksta pristupa preko konfiguracione klase `CLSourceViewerConfiguration`.

3.2.3 Automatski prediktivni asistent

Savremena integrisana razvojna okruženja imaju sposobnost da posredstvom uređivača programskog koda, programeru omoguće veću produktivnost i manje napora prilikom razvijanja programske podrške. Proširenje integrisano u uređivač programskog koda poznato kao automatski prediktivni asistent (engl. *Content assist* ili *Code completion*) nudi korisniku uređivača listu mogućih dopuna trenutne linije programskog koda u zavisnosti od trenutne pozicije kursora miša. Na taj način uređivač asemblerskog koda akcijom na tastere tastature `Ctrl+Space` aktivira grafički objekat koji sadrži listu mogućih instrukcija, promenljivih, registara i drugih elemenata na trenutnoj poziciji. Na slici 6. ilustrativno je prikazano građenje instrukcije pomoću prediktivnog asistenta, dok na slici 7. je prikazano korišćenje već definisanih elemenata.



Slika 6. – Građenje asemblerske instrukcije pomoću prediktivnog asistenta



Slika 7. – Korišćenje definisanih elemenata pomoću asistenta

Sam gramatički prikaz ADD instrukcije prikazuje mogućnosti u zavisnosti od pozicije kursora miša.

```

InstructionAdd:
    'ADD' (REG_ACC | REG_DATA) ','
    (REG_ACC | REG_DATA | Numbers)
;
REG_ACC:
    ('A0' | 'A1' | 'B0' | 'B1')
;
REG_DATA:
    ('X0' | 'X1' | 'Y0' | 'Y1')
;
Numbers:
    ('0'..'9')+
;

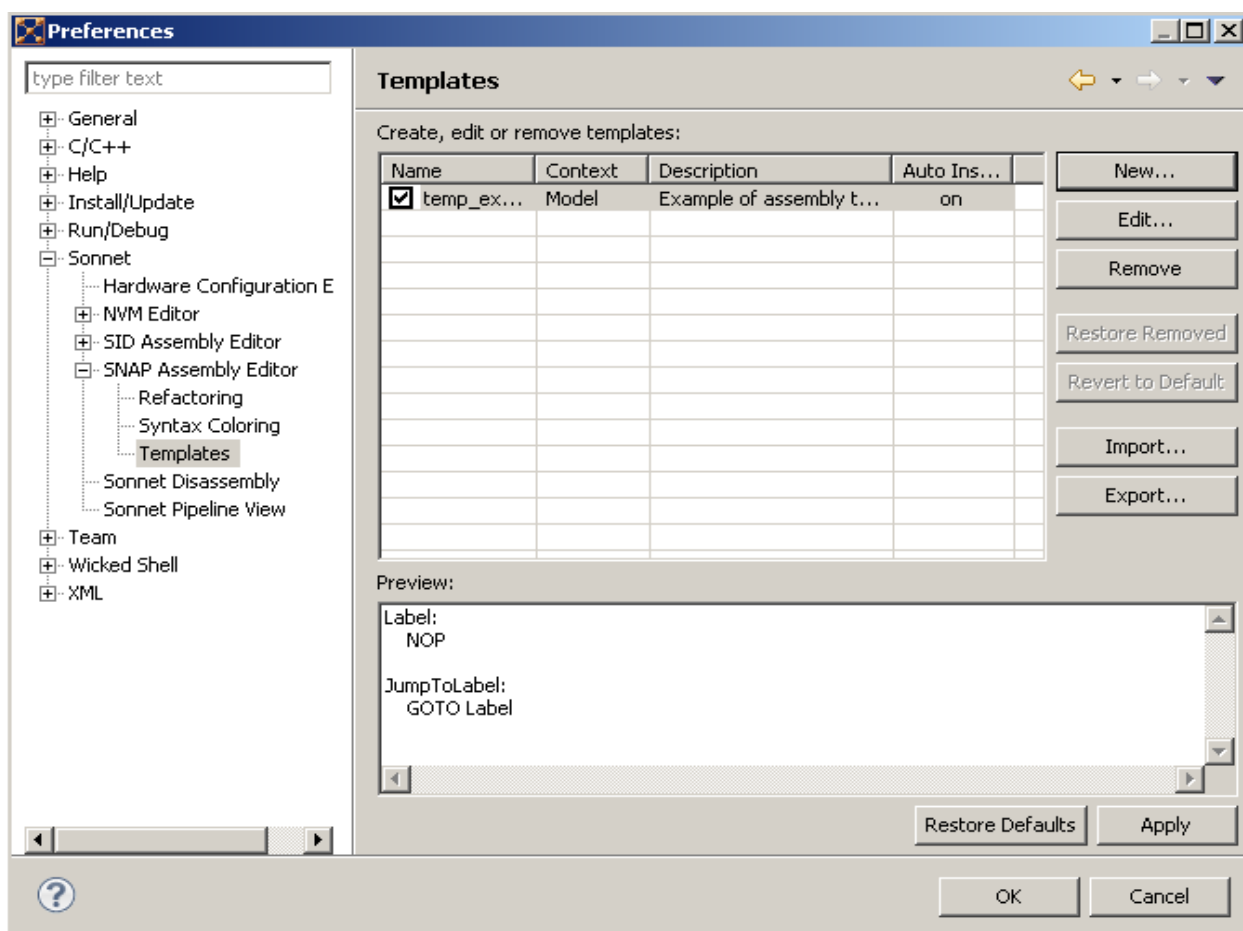
```

Uređivač asemblerskog koda koristi princip pozadinske analize (engl. *Background parsing process*). Ovaj princip sadrži tri ključna koraka:

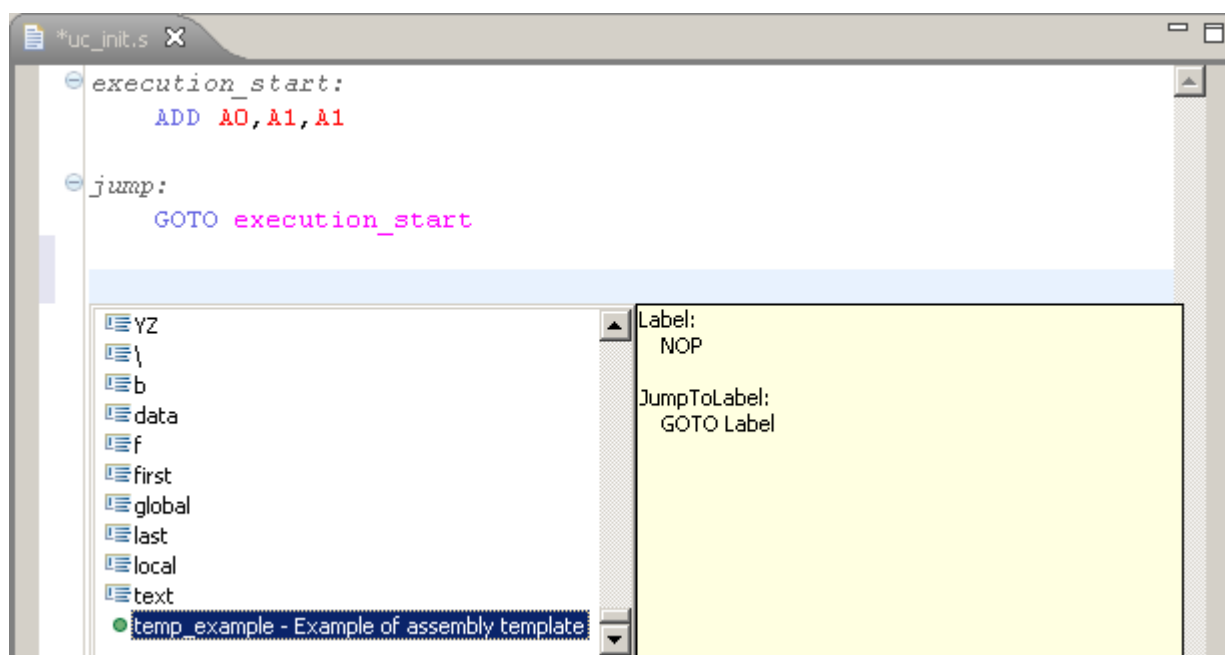
1. Korisnik vrši izmene unutar uređivača programskog koda.
2. Sadržaj uređivača se pozadinski analizira prema gramatici asemblerskog jezika za koju je razvijen uređivač.
3. Na osnovu analiziranog sadržaja uređivača pravi se model u obliku stabla. Ova tri koraka se ponavljaju iznova svaki put kada se izmeni sadržaj uređivača asemblerskog koda.

Kada korisnik koji unutar uređivača razvija programsku podršku zatraži pomoć koristeći Ctrl+Space, u zavisnosti od trenutne pozicije unutar uređivača, dobija ponudu mogućih aktivnih tipova od asistenta, odnosno dobija listu sugestija na koji način i kojim ključnim rečima je moguće završiti trenutnu liniju koda, koje su na trenutnoj poziciji podržane. U zavisnosti od teksta koji se menja u datom trenutku, modela gramatike, ranije verzije sadržaja uređivača i pozicije pokazivača miša, određeni tipovi se aktiviraju ili ne. Razlikujemo dve vrste sugestija:

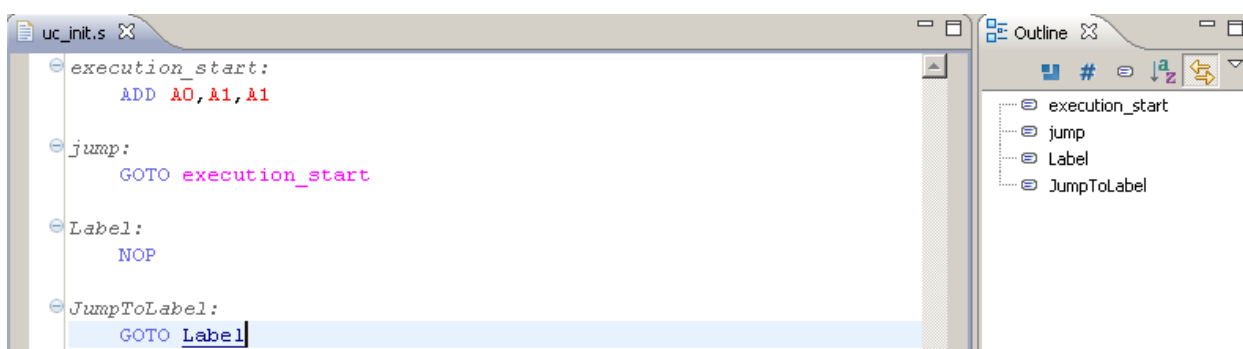
1. Dopuna ključnim rečima – u zavisnosti od trenutne pozicije i programskog koda koji se menja, uređivač nudi moguće ključne reči za završetak linije.
2. Ručno definisani šabloni – šablon je struktura smislenog koda.



Slika 8. – Stranica za definisanje šablona asemblerskog koda



Slika 9. – Definisani šablon unutar liste prediktivnog asistenta



Slika 10. – Izgled definisanog šablona unutar uređivača

3.3 Pogledi obeleživača resursa

Ovaj tip pogleda predstavlja dinamičko-osvežavajući pogled što znači da prati i reaguje na promene koje su mu bitne, a koje su u direktnoj interakciji sa uređivačem asemblerskog koda.

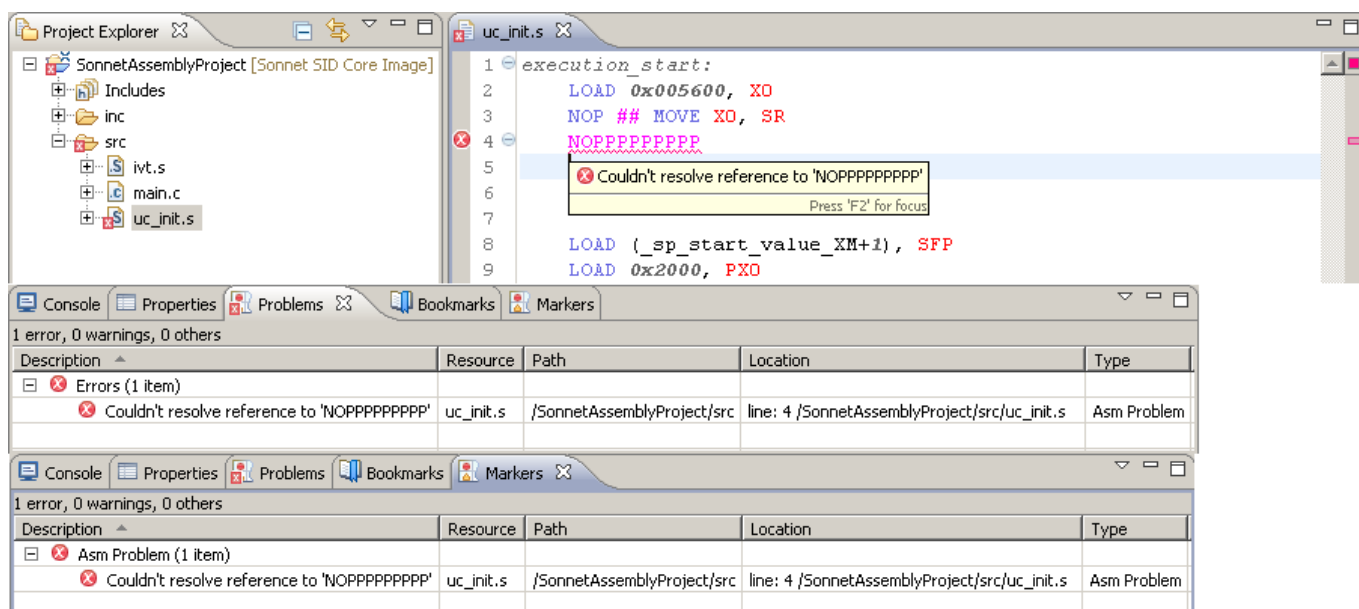
Radno okruženje nudi mehanizam markera za obeležavanje resursa. Markeri su mali pomoćni grafički objekti kojih u projektu može biti na stotine. Oni služe za obeležavanje delova u datotekama kao što su: greške prevodioca (engl. *Errors*), upozorenja prevodioca (engl. *Warnings*), delovi koda koje treba doraditi (TODO lista), rezultati pretraga (engl. *Search hits*), obeleživači pozicije (engl. *Bookmarks*) i prekidne tačke (engl. *Breakpoints*).

Akcijom za čuvanje promena unutar uređivača vrši se inkrementalno građenje programskog koda, nakon kojeg se automatski osvežavaju pogledi. Na taj način se i sadržaj ovog pogleda osvežava, ukoliko je pronađena neka vrsta greške, upozorenja, prekidne tačke, definisanog zadatka.

3.3.1 Pogled problema i upozorenja

Ukoliko prilikom razvijanja programske podrške programer napravi grešku, ili se pojavi neka vrsta upozorenja od strane prevodioca, pojavljuje se marker na levoj margini uređivača. Marker se automatski beleži unutar pogleda koji je zadužen za skladištenje markera (engl. *Markers View*). Ukoliko se radi o markeru greške ili upozorenja on se automatski skladišti i unutar pogleda koji je zadužen za skladištenje nastalih grešaka i upozorenja (engl. *Problems View*). Na taj način u bilo kom trenutku se mogu pregledati sve greške ili upozorenja u svim projektima koji se nalaze unutar radne površine.

Duplim pritiskom miša na grešku ili upozorenje otvara se uređivač koji je zadužen za datoteku unutar koje se javlja greška, pa na taj način preko markera korisnik se može brzo pozicionirati na odgovarajuće mesto u kodu. Na slici 11. prikazan je primer.



Slika 11. – Grafički predstavljena greška

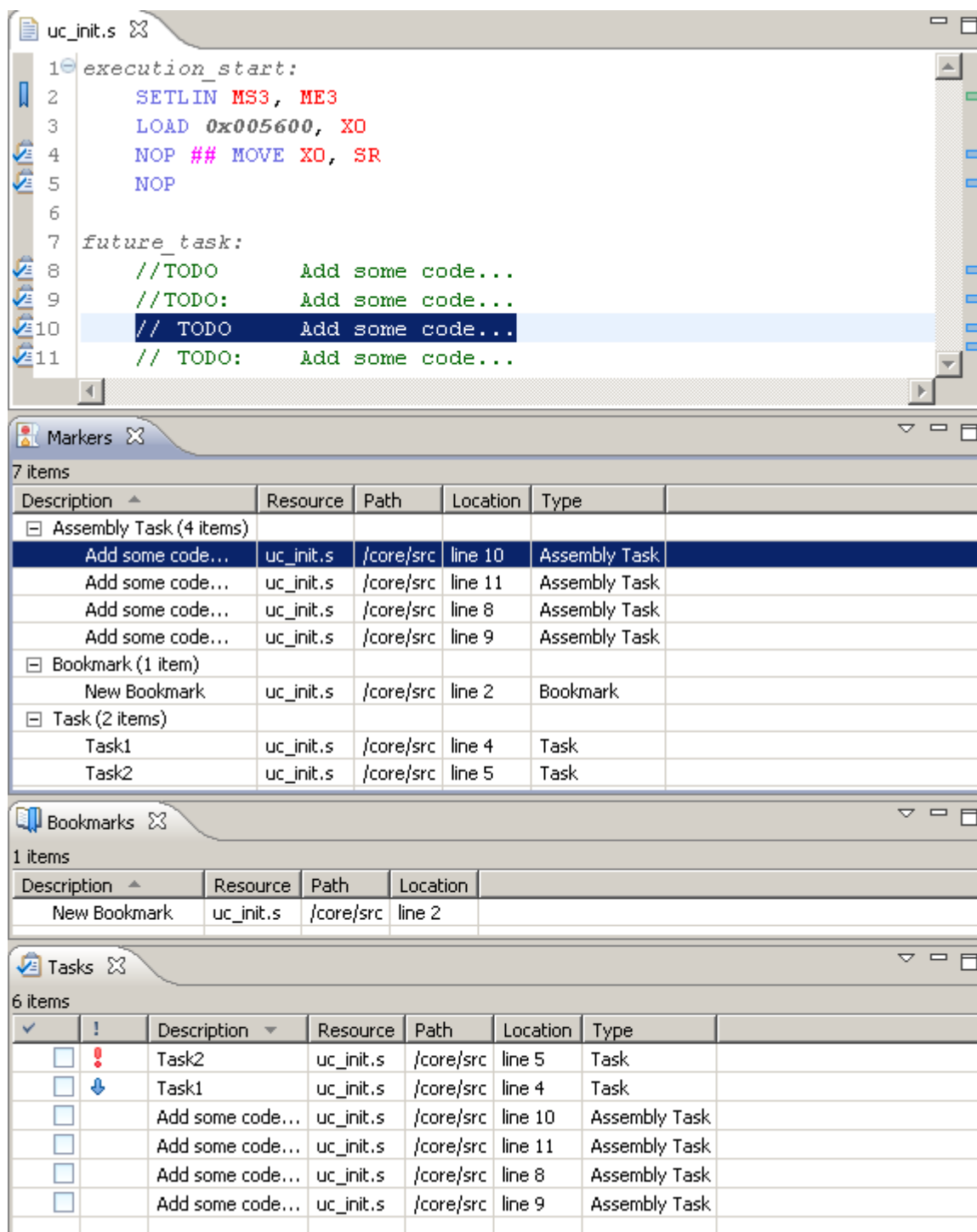
3.3.2 Pogled zadataka i lokacija koda

Uređivač asemblerskog koda omogućava programeru da označi lokaciju od interesa unutar koda, bez direktne izmene samog programskog koda. Lokacije od interesa se čuvaju u radnom prostoru i grupisane su unutar pogleda zaduženog za lokacije koda od interesa (engl. *Bookmark View*). Klikom na oznaku otvara se uređivač i locira se označena lokacija. Kada je uređivač aktivan, označena lokacija je prikazana na margini sa leve strane uređivača asemblerskog koda.

Uređivač omogućava programeru da kreira beleške zadataka direktno unutar uređivača, uz mogućnost podešavanja prioriteta, komandom iz menija ili specijalnim ključnim rečima kroz komentar programskog koda. Ključne reči će od strane parsera biti predefinisane kao beleške zadataka. Ključne reči koje se koriste unutar asemblerskog uređivača su: `//TODO` i `//TODO:`.

Beleške zadataka se skladište u pogledu zaduženom za popis definisanih zadataka (engl. *Task View*). Klikom na belešku otvara se uređivač i pozicionira na lokaciju beleške. Sa leve strane uređivača nalazi se i grafička predstava beleške.

Pošto je i kod beleženja zadataka, i kod beleženja lokacije od interesa, upotrebljen mehanizam markera, pored sopstvenih pogleda u kojima se skladište, ovi markeri su svi zajedno grupisani u pogledu zaduženom za sve vrste markera (engl. *Markers View*). Na slici 12. se može videti primer uređivača asemblerskog koda sa definisanim zadatcima i pogledima.

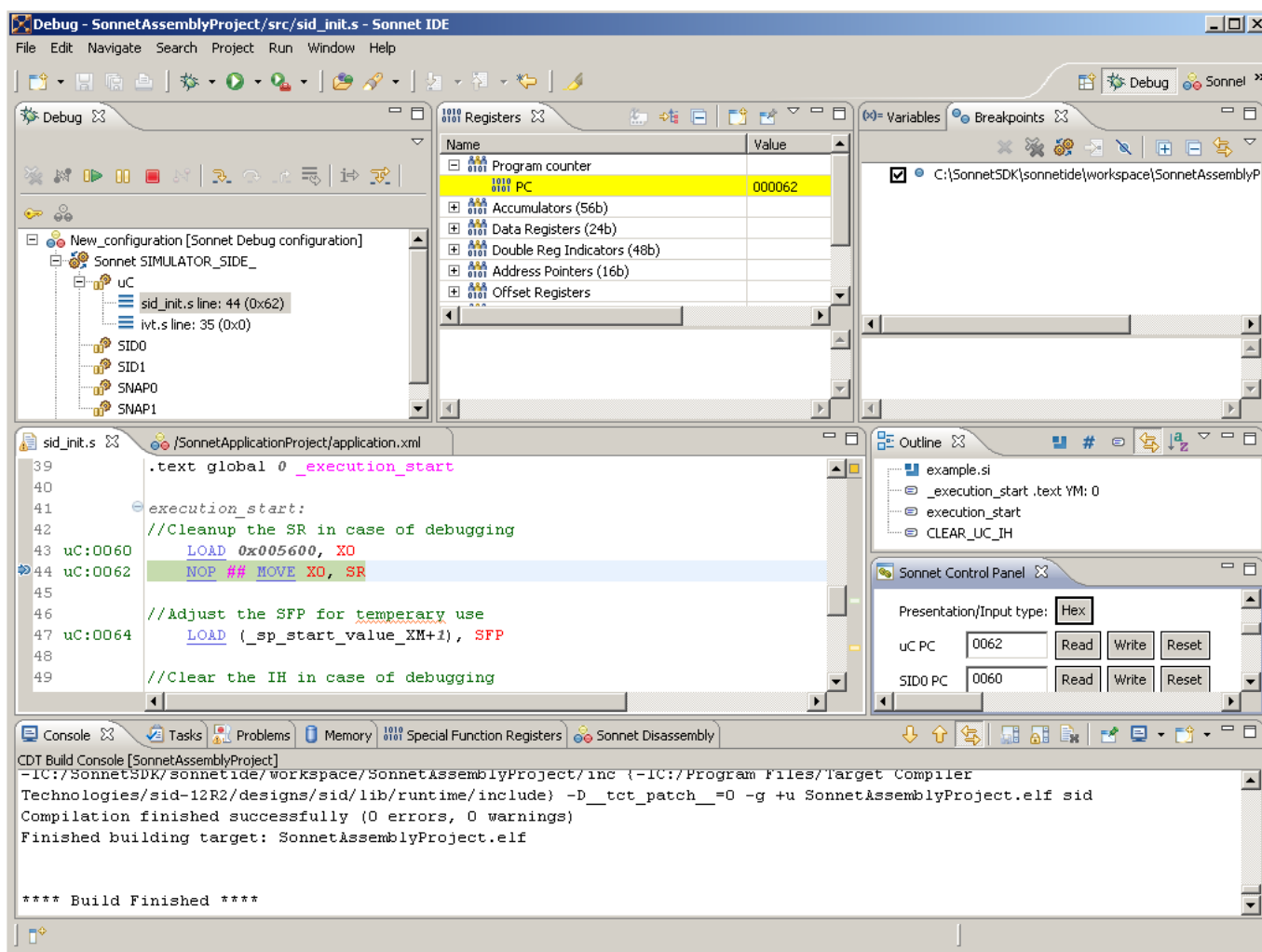


Slika 12. – Pogled zadataka i obeleživača lokacije koda

3.4 Prekidne tačke i kontrolisano izvršavanje

Prekidne tačke su uglavnom prikazane na levoj margini uređivača koji prikazuje izvršni asemblerski kod. Kada asemblerska aplikacija tokom izvršavanja naiđe na prekidnu tačku, emituje se DebugEvent događaj tipa SUSPEND sa razlogom BREAKPOINT. Platforma omogućava rukovanje prekidnim tačkama iz korisničke sprege preko klasa IBreakpoint i IBreakpointManager. IBreakpointManager predstavlja centralni autoritet, on dodaje i uklanja prekidne tačke, aktivira ih ili deaktivira i informiše sve zainteresovane elemente o događajima vezanim za njih.

SIDE okruženje omogućava podršku uklanjanju programskih grešaka (engl. *Debugging*) kroz uređivač asemblerskog koda i korisničku spregu koja omogućava korisniku da komunicira sa programom tokom njegovog izvršavanja na fizičkoj arhitekturi odnosno Sonnet platformi.



Slika 13. – Perspektiva kontrolisanog izvršavanja aplikacije

Perspektiva namenjena režimu za kontrolisano izvršavanje asemblerske aplikacije sadrži paletu sa alatima korisnim prilikom kontrolisanog izvršavanja, pogled za pregled svih tipova procesora Sonnet platforme (uC, SID0, SID1, SNAP0, SNAP1), pogled za prikaz konzole koji omogućuje pregled informacija iz programa, pogled u kojem su prikazane vrednosti svih dostupnih asemblerskih registara sa mogućnošću direktnog menjanja vrednosti, pogled aktivnih i ne aktivnih prekidnih tačaka, pogled koji prikazuje trenutne vrednosti određenih sekcija aktivnog asemblerskog koda, pogled sadržaja uređivača. Sam izgled perspektive, njenih vidljivih pogleda i uređivača, raspored prikaza elemenata može da se menja i prilagodi programeru koji razvija asemblersku aplikaciju.

Podržane akcije u režimu za kontrolisano izvršavanje asemblerskog programa:

1. Prekid sesije kontrolisanog izvršavanja i nastavak izvršavanja programa u režimu bez kontrole (engl. *Disconnect*).
2. Izvršavanje programa korak po korak unutar uređivača asemblerskog koda, od trenutne tačke izvršavanja (engl. *Step into, Step over*).
3. Prekid i nastavak izvršavanja programa (engl. *Suspend, Resume*).
4. Završetak sesije kontrolisanog izvršavanja programa (engl. *Terminate*).
5. Promena vrednosti registara i određenih sekcija aktivnog koda od strane korisnika.

3.5 Višestruko pojavljivanje elementa

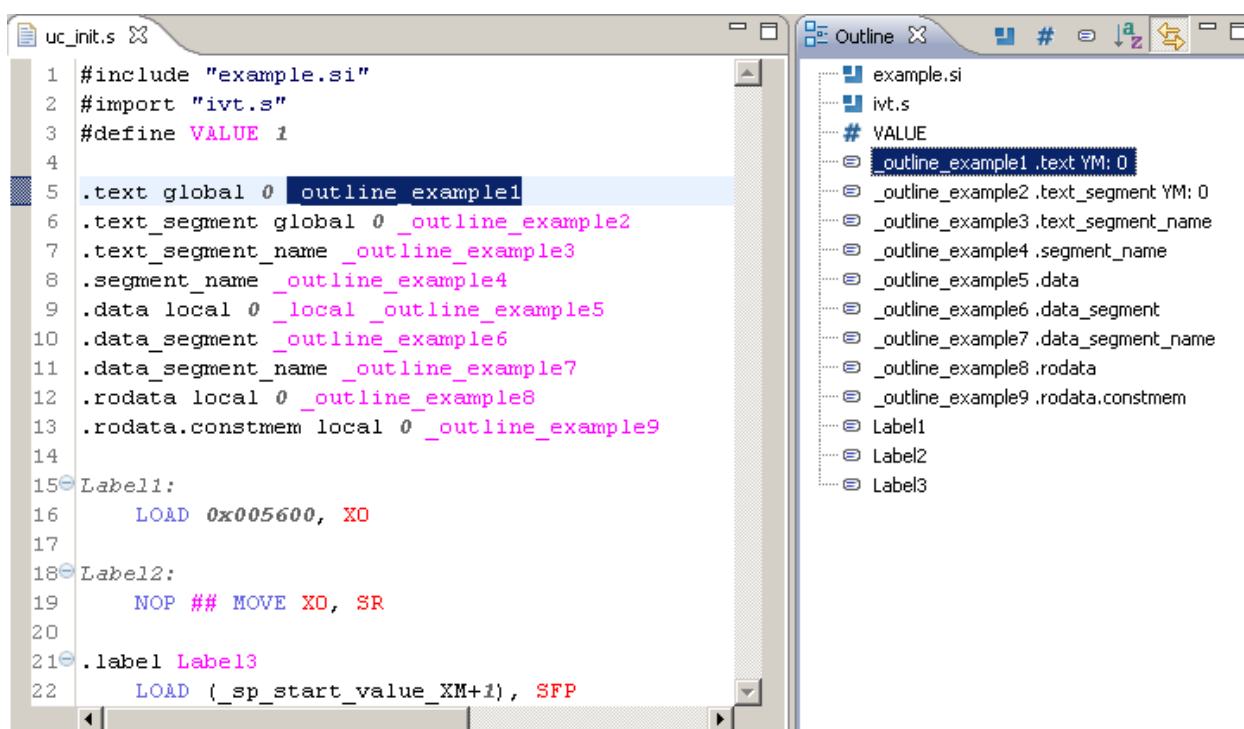
Pronalaženje višestrukog pojavljivanja (engl. *Mark occurrences*) istog elementa je jednostavna i korisna opcija uređivača asemblerskog koda koji na osnovu gramatike, modela jezika i sadržaja uređivača, korisniku javlja sve lokacije pojavljivanja elementa unutar aktivnog uređivača.

Pritiskom na dugme koje se nalazi na paleti alata, aktivira se funkcionalnost. Nakon toga pritiskom na ime instrukcije, registar, makro definiciju, unutar uređivača asemblerskog koda, pronalaze se sve lokacije unutar uređivača gde se pojavljuju isti ti elementi. Nakon što se pronađu, korisniku uređivača se metodom zatamnjenja označe. Klikom na neki drugi element, oznaka se skida sa prethodnog i označava se pojavljivanje novog elementa. Dugme ove funkcionalnosti (engl. *Toggle Mark Occurrences*) može biti u stanju on ili off. Na slici 13. prikazan je primer višestrukog pojavljivanja instrukcije unutar uređivača asemblerskog koda.

3.6 Pogled sadržaja uređivača

1. Nazivi sekcija koda (engl. *Label*).
2. Makro definicije (engl. *Defines*).
3. Text sekcije koda - `.text`.
4. Data sekcije koda - `.data`.

Postoji mogućnost filtriranja sadržaja pogleda. Na taj način mogu se prikazati sve navedene sekcije ili pojedinačne kombinacije. Sadržaj pogleda se osvežava na svaku promenu unutar sadržaja uređivača i na taj način sadržaj pogleda prati sadržaj uređivača. Kada se promeni sadržaj uređivača pogled se osvežava pomoću dobavljača sadržaja (engl. *Content provider*). Svaki put kada pogled dobije novi element za prikaz, on zatraži od dobavljača imena (engl. *Label provider*) samo ime elementa kao i grafičku prezentaciju-ikonu pozivajući dve glavne metode `getText()` i `getImage()`. Pogled sadržaja uređivača prikazan je na slici 15. na kojoj se vidi i veza između pogleda i samog uređivača, pa samim tim u slučaju obimnog izvornog koda, uz pomoć pogleda se lako locira određena sekcija.

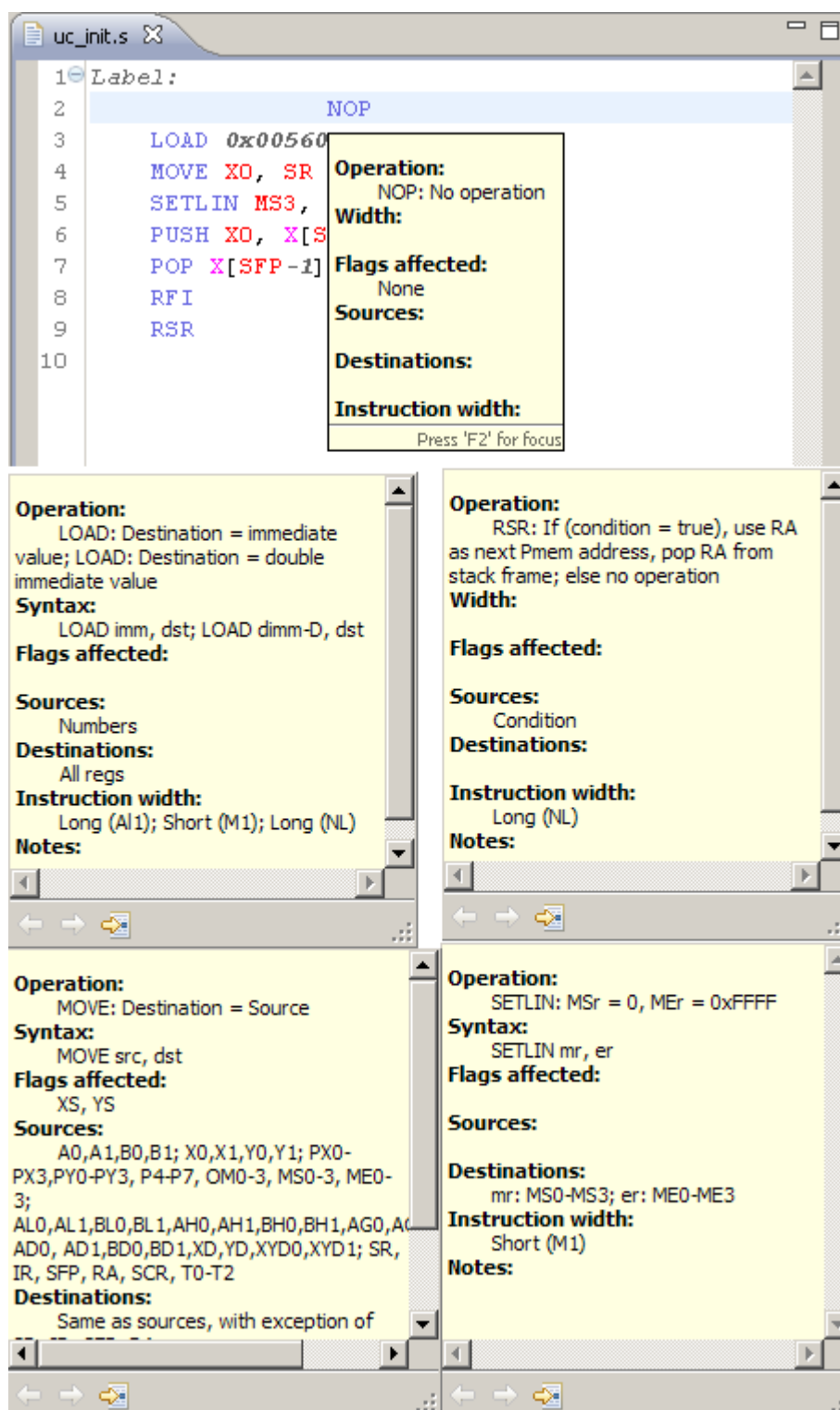


Slika 15. – Pogled sadržaja uređivača

3.7 Informacioni prozor – “lebdeći tekst”

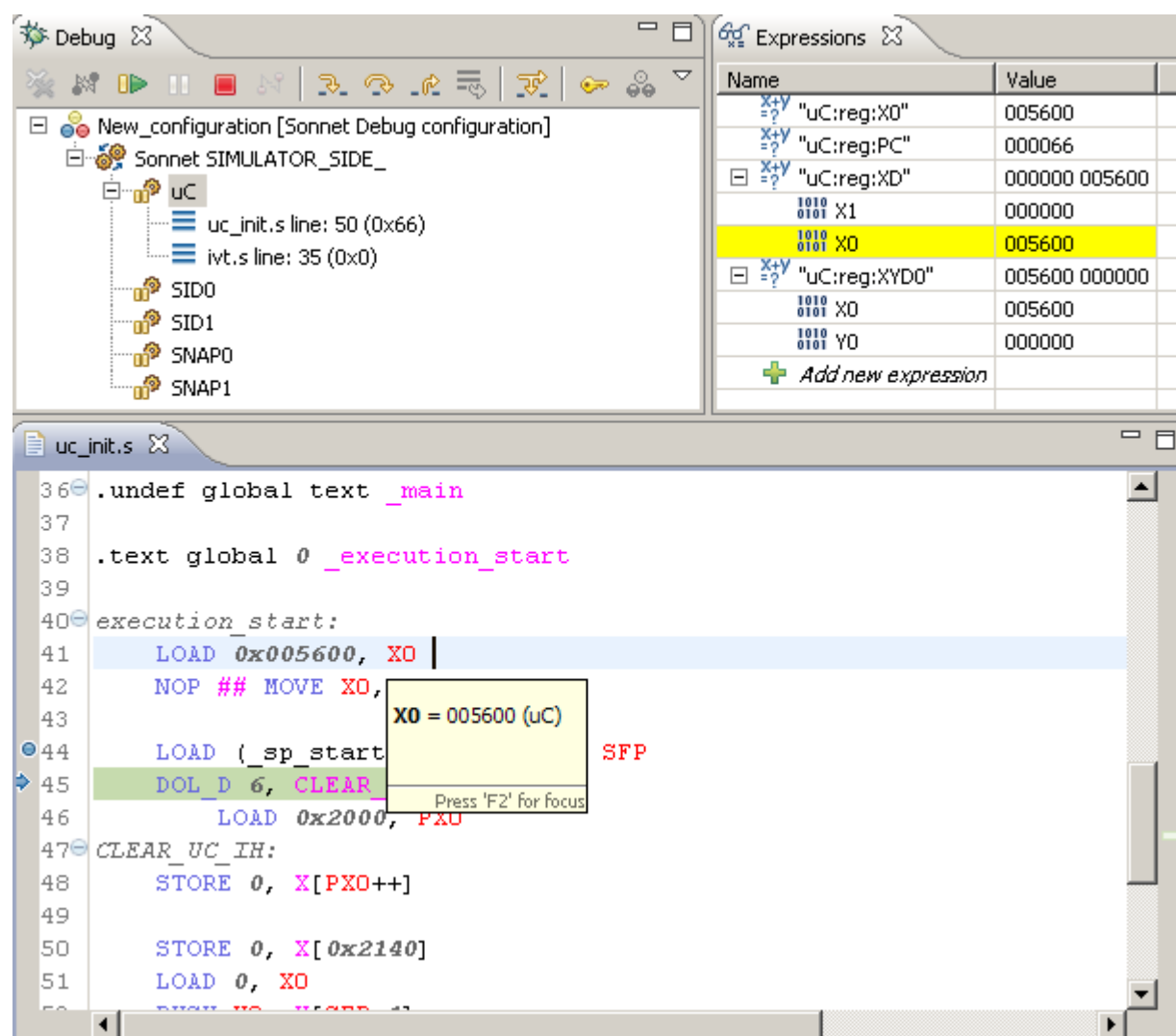
Veoma korisna funkcionalnost uređivača asemblerskog koda jeste informacioni prozorčić sa dodatnim informacijama o elementu unutar uređivača koji se našao u fokusu pokazivača miša.

Kada korisnik koji razvija programsku podršku želi nešto više da zna o nekoj asemblerskoj instrukciji ili pak da vidi način korišćenja pa čak i sam primer korišćenja instrukcije, sam format instrukcije, ako želi da zna nešto više o registru koji koristi, potrebno je da se pokazivačem miša nađe iznad elementa za kojeg su mu potrebne dodatne informacije odnosno dokumentovana saznanja. Primer je prikazan na slici 16.



Slika 16. – Primeri informacionih prozora

Ako je IDE u režimu kontrolisanog izvršavanja u slučaju da se registar nađe u fokusu miša ove dodatne informacije će biti zamenjene konkretnim vrednostima koje su dodeljene samim izvršavanjem. Samim tim programer na lak i jednostavan način može da prati samo izvršavanje aplikacije i da na vreme uoči ukoliko dolazi do greške prilikom izvršavanja i da je ispravi. Na slici 17. je prikazan režim kontrolisanog izvršavanja sa trenutnom vrednošću registra.



Slika 17. – Informacioni prozor u režimu kontrolisanog izvršavanja

4. Programsko rešenje

U ovom poglavlju je detaljno objašnjena sprega između ključnih modula-java paketa, programske podrške, njihova međusobna komunikacija i sama realizacija programske podrške.

Programska realizacija projektnog zadatka napisana je u programskom jeziku Java. Korišćena je Eclipse razvojna platforma.

U tabeli 1. su pobrojani svi paketi koji se nalaze u programskom rešenju kao i detaljniji opis najbitnijih Java klasa u svakom paketu ponaosob. Uz svaki paket dat je kratak opis njegove funkcionalnosti.

Naziv paketa	Datoteke-klase paketa	Kratak opis paketa
Xtext	Asm.xtext GenerateAsm.mwe2	Podršavanje Xtext okvira i generatora Java klasa. Datoteka sa opisom asemblerskog jezika.
XtextFormatting	AsmFormatter.java	Formatiranje sadržaja uređivača asemblerskog koda.
XtextLinking	AsmLinkingMessage.java	Širi opis grešaka programskog koda.
XtextScoping	AsmScopeProvider.java	Podrška za kretanje kroz programski kod unutar uređivača.
XtextValidation	AsmJavaValidator.java	Provera ispravnosti asemblerskog koda unutar uređivača.
XtextUi	AsmUiActivator.java AsmUiModule.java	Aktiviranje grafičke sprege sa korisnikom.
XtextUiContentAssist	AsmProposalProvider.java	Implementacija automatskog prediktivnog asistenta.
XtextUiEditorCallback	EditorCallBack.java	Inicijalna provera izvornog koda pri aktiviranju uređivača.

XtextUiHighlighting	SemanticHighlightingDocumentationProvider.java SemanticHighlightingDocumentationCalculator.java SemanticHighlightingDocumentationConfiguration.java	Bojenje ključnih reči asemblerkog jezika unutar uređivača.
XtextUiHover	DocumentationProvider.java HoverProvider.java	Dodatne informacije o delu asemblerkog koda unutar info (lebdećeg) prozora.
XtextUiLabeling	AsmDescriptionLabelProvider.java AsmLabelProvider.java	Obezbeđuje imena elemenata asemblerkog jezika koje je potrebno prikazati u pogledu sadržaja uređivača.
XtextUiMarkOccurrences	OccurenceComputer.java	Označava višestruko pojavljivanje određenog dela koda unutar uređivača asemblerkog koda.
XtextUiNavigation	AsmHyperlinkHelper.java	Omogućava međusobno uvezivanje definisanog elementa i lokacije gde se element koristi.
XtextUiOutline	AsmOutlineTreeProvider.java DefinesFilterContribution.java IncludesFilterContribution.java LabelsFilterContribution.java	Realizacija pogleda sadržaja uređivača asemblerkog koda. Popunjavanje pogleda, filtriranje sadržaja pogleda.
XtextUiPreferences	HoverPagePreference.java	Stranica za podešavanje dodatnih informacija, nad delom koda koji se našao u fokusu kursora miša.
XtextUiRefactoring	RenameRefactoringProvider.java	Modul omogućava da se kroz par koraka preimenuje svako pojavljivanje elementa asemblerkog koda.
XtextUiTask	AsmTaskCalculator.java AsmTaskElementChecker.java	Definisanje zadataka koje je potrebno naknadno odraditi. Pogled zadataka, i grafička prezentacija definisanih zadataka.
XtextUiTemplates	TemplateProposalProvider.java	Podrška za pravljenje šablona od asemblerkog koda koji se često koristi unutar uređivača.

Tabela 1. – Spisak modula programskog rešenja

4.1 Paket Xtext

4.1.1 Datoteka Asm.xtext

Datoteka `Asm.xtext` u ovom paketu predstavlja ključnu datoteku rešenja sistema. Ona sadrži sam opis asemblerskog jezika. Za njen sadržaj kaže se da je gramatika jezika. Njena ekstenzija je `.xtext`, a programski jezik na kojem je napisana je Xtext usko-namenski jezik odnosno već pomenuti DSL jezik.

U ovoj datoteci su opisana sva pravila jezika koji se formira. Sam početak datoteke mora da počinje modelom koji u našem slučaju izgleda ovako:

```

Model:
    NewLine*
    Instruction+
;

Instruction:
    Labels | DotCommands | PreprocessorCommands
;

```

Prethodni primer prikazuje nekoliko pravila (engl. *Rule*). Početno pravilo mora da se zove model. Model nam kaže da datoteka koju uređuje korisnik uređivača asemblerskog koda, može da počne praznim redovima, a ne mora, nakon čega sledi set instrukcija. Pravilo `Instruction` sadrži pravilo za imenovane sekcije asemblerskog koda - labele, makro definicije ili ključne reči koje počinju tačkom.

Primer imenovane sekcije koda razvijenog asemblerskog jezika je:

```

Labela:
    NOP

```

Primer makro definicije je:

```

#define FLAG 1

```

Primer ključne reči koja počinje tačkom je:

```

.text global XM 0 _text

```

Na isti način se formira podrška za korišćenje brojeva. Ispod je prikazan primer opisa jezika koji omogućava korišćenje različitih tipova brojeva odnosno konstanti.

```

Numbers:
    Integer_constant | Floating_point_constant
;

Integer_constant:
    Binary_integer_constant | Octal_integer_constant
    | Decimal | Hexadecimal_integer_constant
;

Floating_point_constant:
    Binary_floating_point_constant | Octal_floating_point_constant
    | Decimal_floating_point_constant | Hexadecimal_floating_point_constant
;

Decimal:
    INT | INTDECIMAL
;

Hexadecimal_integer_constant:
    HEXA
;

Binary_integer_constant:
    BINARY
;

Octal_integer_constant:
    OCTAL
;

Binary_floating_point_constant:
    BINARYFLOAT | BINARYFLOAT2
;

Octal_floating_point_constant:
    OCTALFLOAT | OCTALFLOAT2
;

Decimal_floating_point_constant:
    INTFLOAT | INTFLOAT2
;

Hexadecimal_floating_point_constant:
    HEXAFLOAT | HEXAFLOAT2
;

```

4.1.2 Datoteka GenerateAsm

Datoteka zadužena za razna podešavanja koja se primenjuju prilikom generisanja Java klase iz ulaznog opisa jezika koji se razvija.

Prilikom svakog generisanja prethodno se vrši brisanje prethodne iteracije. Na sledeći način zadaju se putanje koje je potrebno očistiti.

```

component = DirectoryCleaner {
    directory = "${runtimeProject}/src-gen"
}

component = DirectoryCleaner {
    directory = "${runtimeProject}.ui/src-gen"
}

```

Prilikom generisanja modela jezika vrše se sledeća podešavanja:

```

component = Generator {
    pathRtProject = runtimeProject – putanja za module zadužene za funkcionalnost
    pathUiProject = "${runtimeProject}.ui" – putanja za module zadužene za izgled
    pathTestProject = "${runtimeProject}.tests" – putanja Junit testova
    projectNameRt = projectName – ime modula
    projectNameUi = "${projectName}.ui" – ime modula zaduženog za izgled
    language = {
        uri = grammarURI
        fileExtensions = file.extensions – podešavanje extenzije za uređivač jezika

        fragment = parser.antlr.ex.rt.AntlrGeneratorFragment {
            options = {
                ignoreCase = true
                classSplitting = true – generisanje više Java klasa
                fieldsPerClass = "1000" – broj polja po klasi
                methodsPerClass = "1100" – broj metoda po klasi
            }
        }

        // formatter API
        fragment = formatting.FormatterFragment {} - podrška formatiranja sadržaja
        uređivača

        // outline API
        fragment = outline.OutlineTreeProviderFragment {} - podrška za pogled na sadržaj
        uređivača
        fragment = outline.QuickOutlineFragment {}

        // content assist API
        fragment = contentAssist.JavaBasedContentAssistFragment {} - podrška automatskog
        prediktivnog asistenta
    }
}

```

4.2 Paket XtextFormatting

4.2.1 Klasa AsmFormatter

Programeri veruju da oblikovanje koda može kod da „učini lepšim“. Bez obzira na izgled koda, oblikovanje definitivno čini kod znatno čitljivijim i lakšim za razumevanje i održavanje.

U tu svrhu ova klasa na akciju programera oblikuje kod i čini ga preglednijim. Osnovne tehnike pri oblikovanju koda su beli prostor i uvlačenje. Na taj način srodni delovi koda se grupišu. Stil oblikovanja koda je unapred definisan za asemblerski jezik. Tako se vidno odvajaju labela, makro definicije, instrukcije i ostale sekcije u kodu.

4.3 Paket XtextLinking

4.3.1 Klasa AsmLinkingMessages

Prilikom razvoja asemblerskog koda unutar uređivača često dođe do nastanka greške od strane programera, ne poznavajući u potpunosti sintaksu programskog jezika. Da bi bio upućen na postojanje greške, kao i na tip greške, putem ove klase obezbeđuju se osnovne informacije o nastaloj grešci, a u nekim situacijama i sugestija kako bi se se greška najlakše mogla otkloniti.

4.4 Paket XtextScoping

4.4.1 Klasa AsmScopeProvider

Prilikom razvoja programske podrške za ciljnu DSP platformu unutar uređivača asemblerskog koda, često se koriste spoljni resursi koji se uključuju u projekat. U njima može da se nalazi određeni deo programskog koda, ili samo određene definicije koje mogu da se iskoriste u aktivnoj datoteci unutar uređivača. Kada se iskoriste, pomoću ove klase akcijom na Ctrl+Click skače se na njihovu definiciju koja je u istoj datoteci ili u nekoj drugoj datoteci čija je putanja prethodno uvezana pomoću #import ili #include direktiva. Ista funkcionalnost važi i za sve elemente programskog koda koji su definisani u istoj datoteci.

4.5 Paket XtextValidation

4.5.1 Klasa AsmJavaValidator

Jedna od najvažnijih klasa iz skupa klasa zaduženih za provere ispravnosti sadržaja uređivača asemblerskog koda. Vodi računa da se u programskom kodu ne pojavi više sekcija sa istim imenom i o tome obaveštava programera. Imenovane sekcije programskog koda unutar jedne datoteke moraju imati jedinstvena imena.

Nakon svake izmene sadržaja uređivača, popunjava, za svaki asemblerski element koji je predodređen za neka kasnija uvezivanja, posebnu listu. Na taj način popunjava se lista definicija makroa, određenih sekcija na koje su omogućeni kasniji skokovi sa druge lokacije unutar koda, višestrukih pojavljivanja jednog elementa unutar aktivnog uređivača, kao i lista uvezanih putanja. Pored toga tu je i lista svih korišćenih registara unutar uređivača.

Da bi se mogle koristiti određene sekcije iz uvezane datoteke, potrebno je uvezati njenu putanju, proveriti da li je putanja ispravna, odnosno da li na lokaciji postoji tražena datoteka, proveriti njen sadržaj i pokupiti sve elemente koji se na neki način mogu referencirati. Tako da nije dovoljno samo uvezati lokaciju datoteke, već ona mora biti tačna, u suprotnom korisnik će biti obavešten greškom.

Definisanje makroa, zahteva jedinstveno ime ukoliko se ne koriste dodatni parametri. Ukoliko se koriste može se definisati više makroa ali sa različitim brojem parametara. Pored broja parametara, zadatak ove klase je i da u svakom trenutku zna za imena definicija kao i za broj parametara i da na zahtev programera unutar liste prediktivnog asistenta ponudi sve moguće opcije. Kada se pak iskoristi definisani makro, prilikom akcije na Ctrl+Click, klasa mora znati na koju makro definiciju je potrebno skočiti jer postoji više definicija istog imena, promenljivog broja parametara.

4.6 Paket XtextUi

4.6.1 Klasa AsmUiActivator

Klasa zadužena za aktiviranje priključka koji omogućava grafičku spregu sa korisnikom.

4.6.2 Klasa AsmUiModul

Klasa u kojoj se registruju komponente grafičke sprege sa korisnikom. Neke od važnijih komponenti koje se aktiviraju su šabloni, dodatne informacije unutar lebdećeg prozora, pogledi, bojenje ključnih reči asemblerskog jezika...

4.7 Paket XtextUiContentAssist

4.7.1 Klasa AsmProposalProvider

Automatski prediktivni pomoćnik je funkcionalnost koju obezbeđuju pojedina razvojna okruženja. Omogućava programeru brži razvoj i bolju efikasnost prilikom razvoja programske podrške. U zavisnosti od dela koda, kao i lokacije unutar uređivača teksta, automatski pomoćnik omogućava programeru listu trenutno dostupnih ključnih reči.

4.8 Paket XtextUiEditorCallback

4.8.1 Klasa EditorCallBack

Ova klasa omogućava da se pri svakom aktiviranju uređivača asemblerskog koda, odnosno pri otvaranju datoteke čiji je sadržaj asemblerski kod automatski proveriti eventualno postojanje grešaka i upozorenja. Pored toga postavljaju se svi obeleživači resursa, počev od obeleživača grešaka, upozorenja, zadataka, tačaka itd. Razlog ovakvog pristupa je u tome što postoji mogućnost da je u spomenutim datotekama napravljena neka greška ili upozorenje unutar nekog drugog tipa uređivača ili čak razvojnog okruženja, i na taj način korisnik bi pri otvaranju datoteke unutar uređivača dobio pogrešne informacije o obeleživačima resursa koje su usnimljene poslednji put prilikom zatvaranja uređivača asemblerskog koda. Informacije bi se osvežile tek kada bi korisnik uređivača napravio neku izmenu unutar programskog koda, nakon koje bi se automatski pokrenulo osvežavanje svih obeleživača resursa i njihovih pogleda.

4.9 Paket XtextUiHighlighting

Bojenje ključnih asemblerskih reči je mogućnost uređivača da prikazuje izvorni kod u različitim bojama i različitog stila i fonta u zavisnosti od tipa ključne reči. Mogućnost promene boje ne utiče na funkcionalnost samog sadržaja uređivača, već je poenta na preglednosti i čitljivosti sadržaja uređivača asemblerskog programskog koda.

4.9.1 Klasa SemanticHighlightingCalculator

Klasa koja je zadužena za podešavanje boje, stila i fonta za sve ključne reči proistekle od Xtext semantičkih pravila izuzev terminal pravila.

4.9.2 Klasa SemanticHighlightingConfiguration

Obezbeđuje podešavanje podrazumevanog stila boje i fonta za sve ključne reči prilikom inicijalnog aktiviranja uređivača asemblerskog koda sve dok ne dođe do premene podešavanja od strane korisnika uređivača.

4.9.3 Klasa TokenToIdMapper

Klasa koja svojim metodama omogućava podešavanja boje, stila i fonta za sve ključne reči asemblerskog jezika koje su nastale upotrebom Xtext terminal pravila kao što su ID, NUMBERS, STRING, COMMENT.

4.10 Paket XtextUiHover

4.10.1 Klasa AsmEObjectHoverProvider

Info prozorčić je grafički-korisnički objekat. Pojavljuje se uz pomoć kursora miša kada korisnik uređivača asemblerskog koda pređe iznad ključne reči za koju su dostupne dodatne informacije. Objašnjenje, način upotrebe, praktični primer upotrebe, ili bilo koja druga bitna informacija o elementu jezika se pojavljuje u lebdećem prozoru.

Ovom klasom pobrajani su svi tipovi objekata odnosno ključne reči asemblerskog jezika za koje su omogućene dodatne informacije od koristi.

4.10.2 Klasa AsmEObjectDocumentationProvider

Klasa koja obezbeđuje dokumentaciju odnosno sadržaj koji će se naći unutar prozora koji obezbeđuju dodatne informacije.

4.11 Paket XtextUiLabeling

Pogledi su zaduženi za prikazivanje sadržaja uređivača asemblerskog koda. Iz tog razloga potrebno im je obezbediti odgovarajući sadržaj koji će biti prikazan unutar pogleda. Nakon što se obezbedi sadržaj iz uređivača, za prikaz u pogledu potrebno je obezbediti dodatne informacije za pojedine elemente modela. Te dodatne informacije se odnose na imena elemenata modela kao i njihovu grafičku prezentaciju da bi se na slikovit način odvojili različiti tipovi elemenata iz uređivača asemblerskog koda. Klasa ovog paketa obezbeđuje pogledu upravo te dodatne informacije.

4.11.1 Klasa AsmLabelProvider

Klasa obezbeđuje imenovanja elemenata iz uređivača asemblerskog koda kao i grafičku prezentaciju za sledeće elemente: naziv sekcije koda (engl. *Label*), naziv sekcije podataka (.data), tekstualne sekcije (.text), segment sekcije (.segment), definicija makroa (#define), spoljnih resursa uvezanih u aktivnu datoteku pomoću putanje (#import, #include). Na ovaj način pogled sadržaja uređivača zna za imena svojih elemenata koje mora prikazati korisniku uređivača asemblerskog koda.

4.12 Paket XtextUiMarkOccurrences

Paket koji sadrži funkcionalnost koja omogućava programeru prikaz svih ponavljanja određene sekcije koda unutar uređivača asemblerskog koda. Označavanje ponavljanja (engl. *Mark Occurrence*) je sposobnost uređivača da unutar svog sadržaja pronade sva ponavljanja elementa sintakse koji se našao u fokusu miša.

4.12.1 Klasa SonnetOccurenceComputer

Nakon akcije programera na dugme koje se nalazi na paleti alata za rad sa uređivačem asemblerskog koda, ili putem akcije Ctrl+Shift+O klasa je zadužena za punjenje mape sa elementima koji se ponavljaju unutar uređivača i zatim je potrebno da grafičkim putem predstavi korisniku broj ponavljanja, tačnu lokaciju, odnosno broj linije unutar uređivača na kojem se pojavljuje i mogućnost da pritiskom miša direktno se skoči na liniju, kao i da sva pojavljivanja unutar uređivača označi bojom.

4.13 Paket XtextUiNavigation

4.13.1 Klasa AsmHyperlinkHelper

Klasa omogućava vezu između definicija i određenih sekcija asemblerskog koda radi kasnijeg uvezivanja i kretanja kroz programski kod. Na taj način se omogućava programeru da na lak način iz jednog dela koda gde se koristi definisani makro skoči na lokaciju unutar programskog koda gde se nalazi sama definicija.

4.14 Paket XtextUiOutline

4.14.1 Klasa AsmOutlineTreeProvider

Klasa zadužena za preuzimanje sadržaja uređivača asemblerskog koda, filtriranja elemenata i odabira elemenata koji su potrebni za prikazivanje u pogledu sadržaju uređivača. Na osnovu odabranih elemenata metode klase prikazuje strukturu sadržaja uređivača asemblerskog koda u obliku stabla. Klasa je zadužena i za osvežavanje sadržaja pogleda na svaku promenu programskog koda unutar uređivača asemblerskog koda.

4.14.2 Klasa DefinesFilterContribution

Klasa predstavlja grafičku spregu sa korisnikom, namenjenu za filtriranje sadržaja pogleda strukture uređivača asemblerskog koda. Rezultat rada ove klase je dugme čija funkcionalnost omogućava korisniku da sakrije sve #define definicije koje se pojavljuju u programskom kodu, a samim tim u ovoj vrsti pogleda.

4.14.3 Klasa IncludesFilterContribution

Klasa predstavlja grafičku spregu sa korisnikom, namenjenu za filtriranje sadržaja pogleda strukture uređivača asemblerskog koda. Rezultat rada ove klase je dugme čija funkcionalnost omogućava korisniku da sakrije sve #include putanje koje su uvezane unutar programskog koda, da se ne bi prikazivale u pogledu.

4.14.4 Klasa LabelsFilterContribution

Klasa predstavlja grafičku spregu sa korisnikom, namenjenu za filtriranje sadržaja pogleda strukture uređivača asemblerskog koda. Rezultat rada ove klase je dugme čija funkcionalnost omogućava korisniku da sakrije sve nazive pojedinih sekcija asemblerskog koda. Na taj način u pogledu sadržaja uređivača asemblerskog koda neće se pojaviti asemblerske labelle odnosno imenovani delovi koda.

Prethodne tri klase zadužene za filtriranje pogleda, omogućavaju prikazivanje svih elemenata asemblerskog koda predodređenih za ovaj tip pogleda, pojedinačne kombinacije pojedinih elemenata, ili čak mogućnost ne prikazivanja ni jednog elementa unutar pogleda.

4.15 Paket XtextUiPreferences

4.15.1 Klasa HoverPagePreference

Klasa koja gradi stranicu za podešavanje (engl. *Preference*) u sekciji Window-Preferences-Hover Tooltip Provider. Na stranici je moguće uključiti podršku prozora sa dodatnim informacijama o elementu asemblerskog jezika koji se našao u fokusu miša, ili pak isključiti.

4.16 Paket XtextUiRefactoring

4.16.1 SonnetRenameRefactoringProvider

Klasa obezbeđuje mogućnost da se na lak način preimenuje željeni element iz strukture uređivača asemblerskog koda. Na taj način se obezbeđuje preimenovanje istog elementa na svakom mestu pojavljivanja unutar uređivača.

4.17 Paket XtextUiTask

4.17.1 Klasa AsmTaskCalculator

Klasa zadužena za kreiranje zadataka, njihovo popunjavanje unutar pogleda zadataka, osvežavanje nakon izmena datoteke sa izvornim kodom unutar uređivača asemblerskog koda, kao i za kreiranje obeleživača lokacije unutar programskog koda nakon kreiranja zadatka.

4.17.2 Klasa AsmTaskElementChecker

Klasa zadužena za kreiranje zadataka na osnovu sadržaja uređivača asemblerskog koda. Ukoliko se unutar programskog koda pronade ključna reč rezervisana za definisanje zadataka, na osnovu obeleživača resursa označava se lokacija unutar uređivača gde je potrebno odraditi zadatak, a samim tim se zadatak pojavljuje u pogledu zaduženom za spisak zadataka.

4.18 Paket XtextUiTemplates

4.18.1 Klasa SonnetTemplateProposalProvider

Ako se određeni deo koda prilikom razvijanja programske podrške u uređivaču konstantno ponavlja, taj deo koda je odličan kandidat za integrisanje u šablon. Šablon je deo koda koji programer često koristi i koji se može automatski ubaciti na željeno mesto u kod nad kojim se radi u uređivaču asemblerskog koda. Korišćenjem šablona postiže se ušteda vremena pri razvijanju aplikacija.

5. Ispitivanje i verifikacija

Razvojem tehnologije današnji softverski sistemi su često velike i kompleksne aplikacije, a samim tim i verifikacija ispravnosti odnosno funkcionalnosti nije lak posao i oduzima znatnu količinu vremena.

Provera funkcionalnosti opisane programske podrške je temeljno ispitana i verifikovana upotrebom Junit alata za funkcionalno ispitivanje Java klasa [10]. Za ispitivanje objekata grafičke sprege korišćen je alat Squish [11]. U nastavku sledi opis ispitivanja pomoću ova dva alata. Automatizam prilikom izvršavanja ispitnih slučajeva obezbeđuje alat Hudson.

Hudson je web-baziran alat pisan u Java programskom jeziku, namenjen automatizovanoj izgradnji razvojnog okruženja uz konstantno praćenje promena, ispitivanje i slanje obaveštenja o stanju sistema [12].

Kontinualna izgradnja sistema (engl. *Continuous Integration-CI*), je programerska praksa na osnovu koje, veći broj članova tima veoma često integrišu delove sistema na kojima rade u jednu grupu. Na taj način dolazi do finalizacije sistema. Svaka integracija dela sistema je verifikovana i ispitana sa ciljem detekcije nastalih problema i grešaka.

Nakon svake izmene programskog koda prilikom razvijanja programske podrške, Hudson objedinjuje i automatski pokreće sve definisane JUnit i Squish ispitne slučajeve, a nakon toga prikazuje rezultate izvršavanja ispitnih slučajeva, timu koji je zadužen za razvoj i ispitivanje programske podrške. Ovakav pristup ispitivanju funkcionalnosti omogućava pravovremenu reakciju na nastale probleme i pristupa se rešavanju grešaka unutar sistema koji se razvija.

U nastavku je prikazano na koji način je ispitana funkcionalnost uređivača asemblerskog koda, kao i postignuti rezultati.

5.1 JUnit ispitivanje

JUnit je okvir za funkcionalno ispitivanje sistema, napisan za Java programski jezik, jednostavan za korišćenje i lak za učenje. Postao je toliko popularan da je preveden i prilagođen za korišćenje u drugim programskim jezicima. Omogućava automatizovano izvršavanje ispitnih slučajeva iz Eclipse razvojnog okruženja za bilo koju Java klasu, odnosno za bilo koju njenu metodu.

JUnit može da se koristi sve vreme tokom razvoja programske podrške, jer se po pravilu prvo piše ispitni slučaj, a posle toga piše programski kod, odnosno pišu se ispitni slučajevi kojima se ispituje svaka metoda. Ovakav princip ispitivanja sistema stvara sigurnost da sve što je implementirano je potpuno funkcionalno.

Postupak ispitivanja pomoću JUnit ispitnih slučajeva, od samog početka razvoja sistema, obuhvata sledeće korake:

1. Detaljna analiza problema
2. Pravljenje idejnog modela sistema koji se razvija
3. Pisanje ispitnih slučajeva na osnovu prethodnog modela
4. Razvijanje programske podrške na osnovu ispitnih slučajeva
5. Prilagođavanje i izmene u toku realizacije programske podrške, JUnit ispitnim slučajevima
6. Provera rezultata ispitnih slučajeva
7. Ukoliko su svi ispitni slučajevi uspešno izvršeni sistem funkcioniše prema specifikaciji, a ukoliko neki od ispitnih slučajeva nije uspešno izvršen, dolazi do dodatne dorade na sistemu sve dok se ne dostigne očekivano stanje
8. Postupak od koraka 4 do koraka 7 se ponavlja tokom celokupnog razvoja sistema, sve dok sistem nije u potpunosti realizovan i svi ispitni slučajevi uspešno izvršeni.

U tabeli 2. prikazani su rezultati JUnit ispitnih slučajeva koji se odnose na verifikaciju asemblerskih konstrukcija, odnosno proveru podrške instrukcija od strane uređivača. Ukoliko neka od instrukcija nije podržana, ispitni slučaj nebi bio uspešno izvršen. Na početku razvoja uređivača asemblerskog koda, svi ispitni slučajevi su neuspešno izvršavani jer je na osnovu njih pisana programska podrška. Vremenom broj neuspešno izvršavanih ispitnih slučajeva je opadao.

Naziv ispitne grupe	Vreme izvršavanja[ms]	Broj ispitnih slučajeva	Uspešno izvršeni	Neuspešno izvršeni
AbsTest	62	1	✓	
AddTest	63	2	✓	
AddcTest	93	2	✓	
BssRuleTest	62	1	✓	
BssSegmentRuleTest	63	1	✓	
CmpTest	78	2	✓	
CmpyTest	62	1	✓	
DataRuleTest	78	1	✓	
DefineRuleTest	46	4	✓	
DelimiterRuleTest	62	2	✓	
DotCommandsTest	1.2 sec	5	✓	
DoubleColonTest	94	1	✓	
LabelRuleTest	94	7	✓	
NegTest	63	1	✓	
NumberTests	110	8	✓	
ParenthesisRuleTest	78	2	✓	
ParseTestFilesTest	26 sec	1	✓	
RodataSegmentRuleTest	78	1	✓	
SegmentNameTest	78	3	✓	
SpaceTest	62	1	✓	
StackFrameSizeTest	78	1	✓	
StackTest	78	1	✓	
SubTest	78	1	✓	
SubcTest	78	1	✓	
TerminalRuleTest	78	2	✓	
TextRuleTest	78	3	✓	
UcmpTest	10	2	✓	
UcmpyTest	62	1	✓	
UndefRuleTest	62	1	✓	
Ukupno	40 sec	60	60	0

Tabela 2. – Rezultati Junit ispitnih slučajeva

Ukupno vreme izvršavanja Junit ispitnih grupa koje sadrže ispitne slučajeve za ispitivanje asemblerskih konstrukcija iznosi oko 40 sekundi u zavisnosti od specifikacije računara na kome se izvršavaju. Broj ispitnih slučajeva je 60. U tabeli se može videti da većini ispitnih grupa je potrebno između 60 i 80 milisekundi, dok pojedine ispitne grupe se izvršavaju i po par sekundi,

opet u zavisnosti od specifikacije računara, broja ispitnih slučajeva, kao i samog tipa odnosno namene ispitnog slučaja.

Ispitna grupa NumberTests namenjena je ispitivanju podrške različitih tipova brojeva unutar uređivača asemblerskog koda. Objedinjuje ukupno osam ispitnih slučajeva. Jedan od ispitnih slučajeva je prikazan ispod namenjen ispitivanju decimalnih brojeva, kako pozitivnih tako i negativnih.

```

@Test
def void testDecimal() {
    myAssertNoError ("Labela: 5 \n");
    myAssertNoError ("Labela: 0.25 \n");
    myAssertNoError ("Labela: 25E-2 \n");
    myAssertNoError ("Label: -5 \n");
    myAssertNoError ("Label: -0.25 \n");
    myAssertNoError ("Label: -25E-2 \n");
    myAssertError ("Label: -25Q-2 \n");
    myAssertError ("Labela: b0.25 \n");
    myAssertError ("Labela: 0.b25 \n");
}

```

Da bi se ispitni slučaj izvršio, Junit automatski aktivira uređivač i proverava podršku za zadati kriterijum. Na isti način koncipirani su svi Junit ispitni slučajevi namenjeni ispitivanju podrške asemblerskih konstrukcija od strane uređivača. U samom ispitnom slučaju navodi se unapred očekivani rezultat, odnosno da li uređivač podržava konstrukciju – myAssertNoError, ili ne podržava – myAssertError. Navedeni ispitni slučaj će biti uspešno izvršen samo ako uređivač asemblerskog koda podržava prvih šest zadanih konstrukcija i ne podržava poslednje tri konstrukcije iz datog primera.

Ispitna grupa koja se ističe od ostalih, kao što se vidi u tabeli je ParseTestFilesTest. Vreme potrebno za njeno izvršavanje je oko 26 sekundi. Sadrži samo jedan ispitni slučaj koji je karakterističan po tome što radi sa unapred definisanom grupom ispitnih datoteka, čiji sadržaj je asemblerski kod. Broj datoteka je 26 i one sadrže veliki broj linija smišljenog asemblerskog koda, koje poseduju određenu funkcionalnost. Prilikom ispitivanja, Junit svaku datoteku otvara unutar uređivača asemblerskog koda i ispituje svaku liniju programskog koda. Ukoliko je svaka konstrukcija iz svake ispitne datoteke podržana unutar uređivača ispitni slučaj je uspešno izvršen.

5.2 Squish ispitivanje

Squish pripada grupi komercijalnih alata, namenjenih za automatizovano ispitivanje i verifikaciju funkcionalnosti grafičke korisničke sprege. Razvijen je od strane Froglogic kompanije. Ima mogućnost ispitivanja grafičke funkcionalnosti Java aplikacija koje se oslanjaju na SWT, AWT, Jface i SWING grafičke biblioteke.

Squish omogućava pisanje ispitnih slučajeva kroz nekoliko programskih jezika: JavaScript, Perl, Python, Ruby ili Tcl. U realizaciji uređivača assemblerskog koda, ispitivanje je vršeno ispitnim slučajevima pisanim Python programskim jezikom.

Za razliku od Junit ispitivanja gde se programska podrška razvija na osnovu ispitnih slučajeva, Squish je namenjen ispitivanju programske podrške koja je već razvijena. Ispitni slučaj nastaje na dva načina:

1. Ručno pisanje ispitnog slučaja u jednom od podržanih programskih jezika.
2. Snimanjem unapred definisanih postupaka korišćenja grafičkog okruženja.

U tabeli 3. prikazani su rezultati Squish ispitnih slučajeva koji se odnose na verifikaciju grafičke korisničke sprege.

Naziv ispitnog slučaja	Uspešno izvršenih	Neuspešno izvršenih
tst_AsmBookmark	✓	
tst_AsmEditorDragAndDrop	✓	
tst_AsmEditorWithError	✓	
tst_AsmEditorWithoutError	✓	
tst_AsmMarkOccurence	✓	
tst_AsmTask-TODO	✓	
tst_AssemblyProjectVerify	✓	
tst_AssemblyTemplate	✓	
tst_CheckContentAssist	✓	
tst_CheckIncludePath	✓	
tst_CheckInitialValidation	✓	
tst_CheckRegisterLabels	✓	
tst_CoreImgProjectVerify	✓	
tst_En/DisableBreakpoint	✓	
tst_InvalidBreakPoint	✓	
tst_NewCoreImgProject	✓	
tst_PreferencesVerify	✓	
tst_RemoveBreakpoint	✓	
Ukupno	18	0

Tabela 3. – Rezultati Squish ispitnih slučajeva

Ispitni slučajevi navedeni u tabeli obuhvataju ispitivanje funkcionalnosti uređivača, kao i ostalih grafičkih objekata koji su u sprezi sa uređivačem:

- Obeleživači resursa kao što su: prekidne tačke (dodavanje novih, uklanjanje, aktivacija, deaktivacija, provera da li su postavljene na dozvoljene asemblerske instrukcije), pozicije od interesa unutar programskog koda, obeleživači definisanih zadataka, osvežavanje obeleživača nakon izmena vezanih za uređivač.
- Automatski prediktivni asistent (pojavljivanje na zahtev korisnika, lista mogućih dopuna linije programskog koda)
- Pogled sadržaja uređivača (pojavljivanje elemenata, filtriranje sadržaja, osvežavanje nakon izmena unutar sadržaja uređivača).
- Definisanje šablona kroz stranicu za podešavanje, korišćenje definisanih šablona, uklanjanje šablona.
- Pravljenje i rad sa tipom projekta koji koristi uređivač asemblerskog koda za razvoj programske podrške.
- Prikaz informacija korisniku o višestrukome pojavljivanju elementa uređivača.
- Inicijalna provera sadržaja uređivača, kao i provera postojanja grešaka i upozorenja nakon određenih izmena sadržaja uređivača.

Svi ispitni slučajevi su uspešno izvršeni. Squish je omogućio temeljno, automatizovano ispitivanje funkcionalnosti grafičkih objekata, i samim tim poboljšao kvalitet ovog proširenja, SIDE okruženja, i znatno skratio potrebno vreme verifikacije funkcionalnosti.

U dodatku je prikazana sintaksa jednog Squish ispitnog slučaja pisanog u Python programskom jeziku. Njegov zadatak je ispitivanje funkcionalnosti stranice za podešavanja vezana za uređivač, kroz promene boje i stila ključnih reči, dodavanjem novog šablona, koji se na zahtev pojavljuje kao opcija prediktivnog asistenta, i na kraju vraćanje inicijalnih podešavanja. Ostali Squish ispitni slučajevi, takođe su napisani u prikazanoj formi.

6. Zaključak

U radu je opisano jedno od proširenja Sonnet integrisanog razvojnog okruženja sa podrškom za razvoj asemblerskih aplikacija namenjenih za Sonnet više-procesorsku platformu. Proširenje SIDE razvojnog okruženja obuhvata uređivač asemblerskog koda sa svim standardnim funkcionalnostima jednog uređivača teksta, kao što su: unos i izmena teksta, obeležavanje, kopiranje, premeštanje i dodavanje teksta, pretraga teksta, ugrađeni meniji, broj linija uređivača. Pored standardnih tu su i funkcionalnosti direktno vezane za asemblerski jezik: bojenje ključnih reči, automatski prediktivni asistent, poravnanje asemblerskog koda, pogled sadržaja uređivača, zadataka i problema, kontrolisano izvršavanje i tačke prekida.

Uređivač asemblerskog koda je realizovan pomoću Eclipse platforme za razvoj samostalnih aplikacija-RCP, koje izdvajaju samo neophodnu funkcionalnost Eclipse platforme i samim tim se izbegava velika složenost i nepotrebno opterećenje resursa, a za cilj se dobija SIDE razvojno okruženje za ciljnu više-procesorsku DSP platformu.

Razvijeno proširenje SIDE razvojnog okruženja je ispitano sa 68 obimnih i temeljno napisanih ispitnih slučajeva. Funkcionalno ispitivanje Java klase urađeno je Junit alatom, dok je korisnička grafička sprega verifikovana upotrebom Squish alata.

Navedeni postupci ispitivanja su pouzdan način za proveru rada sistema. Međutim ukoliko se u obzir uzme činjenica da nije moguće obuhvatiti ceo sistem zbog složenosti, činjenicu da trenutna verzija sistema nije konačna, i na kraju činjenicu da subjekat koji razvija ispitne slučajeve i celokupan sistem, zbog svoje prirode je podložan povremenim greškama, ovaj način provere se može smatrati sasvim zadovoljavajućim ali ne i savršenim. Ako izuzmemo prethodne činjenice, može se reći da su svi ispitni slučajevi uspešno izvršeni, čime je potvrđena ispravnost i funkcionalnost proširenja SIDE okruženja.

Eclipse platforma nudi generičku podršku za razvoj integrisanih razvojnih okruženja u vidu posebnih utičnih proširenja, samostalnih aplikacija i alata, kao i niz drugih pogodnosti:

- Brz razvoj aplikacije, jer je veliki deo funkcionalnosti već ponuđen od strane Eclipse platforme.
- Manji broj problema prilikom razvoja programske podrške zbog iskorišćenja gotovog koda Eclipse platforme.
- Java programski jezik, kao osnovni jezik razvoja SIDE okruženja, omogućava prenosivost između različitih platformi.
- Mogućnost proširenja funkcionalnosti u svim pravcima, dodavanjem novih utičnih proširenja.

Xtext DSL okvir je korišćen za opis apektata asemblerskog jezika, koji uređivaču asemblerskog koda omogućavaju podršku za razvoj asemblerskih aplikacija za Sonnet platformu.

Da bi se stekla šira slika o obimu uloženog truda u razvoj uređivača za ciljnu platformu, navodi se da je napisano oko hiljadu linija programskog koda gramatike asemblerskog jezika, iz kojih je Xtext okvir izgenerisao mnoštvo klasa i metoda (par desetina hiljada linija koda), zatim su proširene postojeće klase namenjene proveriti grešaka, formatiranju sadržaja uređivača, bojenju sintakse, listi sadržaja uređivača itd. Ova proširenja za cilj su imala poboljšanje prvobitne funkcionalnosti (dobijene Xtext okvirom).

Pravci daljeg razvoja su brojni. Zasigurno, bitnija poboljšanja idu u smeru još bolje optimizacije asemblerskog jezika, kao i grafičkih komponenti uređivača, prvenstveno dela koji ukazuje na sadržaj samog uređivača, kao i automatskog prediktivnog asistenta, kako bi mu se poboljšala osobina predikcije, bez obzira na trenutnu poziciju unutar uređivača asemblerskog koda.

7. Dodatak A

```
#####
# Testing Assembly editor
#
# New Core Image project for testing Assembly editor features.
#   • Action: open "Sonnet/Assembly Editor" preference page
# Change color, style, font of assembly syntax group (Labels, Instructions, Comments...).
# Make new assembly template and verify with action ctrl+space under assembly editor area.
# At the end in "Sonnet/Assembly Editor" preference page click button Restore Defaults.
# Exit Application
#####

def main():

    source(findFile("scripts", "common.py"))
    source(findFile("scripts", "globals.py"))
    startAppAndClearWorkspace()
    newProject()

    mouseClicked(waitForObjectItem(":New_Tree", "Core Image Project"), 48, 11, 0, Button.Button1)
    clickButton(waitForObject(":New.Next >_Button"))
    type(waitForObject(":New Core Image Firmware Project.Project name:_Text"), "assemblyProj")
    clickButton(waitForObject(":New Core Image Firmware Project.Next >_Button"))
    clickButton(waitForObject(":New Core Image Firmware Project.uC based project_Button"))
    clickButton(waitForObject(":New Core Image Firmware Project.Finish_Button"))

    activateItem(waitForObjectItem(":Sonnet Development - Sonnet IDE_Menu", "Window"))
    activateItem(waitForObjectItem(":Window_Menu_2", "Preferences"))
    expand(waitForObjectItem(":Preferences_Tree", "Sonnet"))
    expand(waitForObjectItem(":Preferences_Tree", "SID Assembly Editor"))

    mouseClicked(waitForObjectItem(":Preferences_Tree", "Syntax Coloring"), 25, 6, 0, Button.Button1)
    mouseClicked(waitForObjectItem(":Preferences.Token Styles_List", "Numbers"), 20, 6, 0,
    Button.Button1)
    mouseClicked(waitForObjectItem(":Preferences.Token Styles_List", "Preprocessor Directives"), 23, 12,
    0, Button.Button1)
    mouseClicked(waitForObjectItem(":Preferences.Token Styles_List", "Registers"), 23, 8, 0,
    Button.Button1)
    mouseClicked(waitForObjectItem(":Preferences.Token Styles_List", "Instructions"), 30, 8, 0,
    Button.Button1)
```

```

clickButton(waitForObject(":Preferences.Color_Button"))
chooseColor(waitForObject(":SWT"), 255, 0, 0)
clickButton(waitForObject(":Style.Italic_Button"))
clickButton(waitForObject(":Style.Underline_Button"))
clickButton(waitForObject(":Preferences.Apply_Button"))
clickButton(waitForObject(":Preferences.OK_Button"))

activateItem(waitForObjectItem(":Sonnet Development - Sonnet IDE_Menu", "Window"))
activateItem(waitForObjectItem(":Window_Menu_2", "Preferences"))
expand(waitForObjectItem(":Preferences_Tree", "Sonnet"))
expand(waitForObjectItem(":Preferences_Tree", "SID Assembly Editor"))
mouseClick(waitForObjectItem(":Preferences_Tree", "Templates"), 39, 8, 0, Button.Button1)
clickButton(waitForObject(":Preferences.New..._Button"))
type(waitForObject(":New_Template_Text"), "newTemp")
mouseClick(waitForObject(":New_Template.Description:_Text"), 141, 8, 0, Button.Button1)
type(waitForObject(":New_Template.Description:_Text"), "New Assembly Template")
type(waitForObject(":New_Template.Description:_Text"), "<Tab>")
type(waitForObject(":New_Template.Pattern:_StyledText"), "templateLabel: NOP")
clickButton(waitForObject(":New_Template.OK_Button"))
clickButton(waitForObject(":Preferences.OK_Button"))

for x in range(0, 44):
    type(waitForObject(":sid_init.s_StyledText_3"), "<Down>")

type(waitForObject(":sid_init.s_StyledText_3"), "<Ctrl+ >")
doubleClick(waitForObjectItem(":_Table", "302/0"), 106, 3, 0, Button.Button1)
mouseClick(waitForObject(":*sid_init.s_StyledText"), 259, 184, 0, Button.Button1)

type(waitForObject(":*sid_init.s_StyledText"), "<Ctrl+s>")
mouseClick(waitForObject(":sid_init.s_StyledText_3"), 172, 220, 0, Button.Button1)
mouseClick(waitForObject(":sid_init.s_StyledText_3"), 198, 215, 0, Button.Button1)

waitFor("object.exists(':templateLabel_TreeItem')", 20000)
test.compare(findObject(":templateLabel_TreeItem").text, "templateLabel")

activateItem(waitForObjectItem(":Sonnet Development - Sonnet IDE_Menu", "Window"))
activateItem(waitForObjectItem(":Window_Menu_2", "Preferences"))
mouseClick(waitForObjectItem(":Preferences_Tree", "Syntax Coloring"), 29, 8, 0, Button.Button1)
clickButton(waitForObject(":Preferences.Restore Defaults_Button"))
clickButton(waitForObject(":Preferences.OK_Button"))

activateItem(waitForObjectItem(":Sonnet Development - Sonnet IDE_Menu", "File"))
activateItem(waitForObjectItem(":File_Menu", "Exit"))

```

8. Literatura

- [1] oAW Xtext: *A framework for textual DSLs*,
<http://www.voelter.de/data/workshops/EfftingeVoelterEclipseSummit>
- [2] Volter. M., *DSLs for Product Lines: Approaches, Tools, Experiences*, Software Product Line Conference (SPLC), 2011 15th International Conference of the IEEE, Stuttgart, Germany, August 2011.
- [3] Starkey, Official web page Available: <http://www.starkey.com>
- [4] Lars Vogel: *Eclipse IDE based on Eclipse 4.2 and 4.3*, 2013.
- [5] Prof. Dr Vladimir Kovačević, Prof. Dr Miroslav Popović: *Sistemska programska podrška u realnom vremenu I*, Novi Sad, 2011.
- [6] Jeff McAffer, Jean-Michel Lemieux, Chris Aniszczyuk: *Eclipse Rich Client Platform second edition*, 2010.
- [7] Zhihui Yang, Michael Jiang: *Using Eclipse as a Tool-Integration Platform for Software Development*, 2007.
- [8] Eclipse Platform Technical Overview, International Business Machines Corp., 2006.
- [9] Eric Clayberg, Dan Rubel: *Building Commercial – Quality Plug-ins*, Addison – Wesley Professional, 2004.
- [10] P.Tahchiev, F.Leme, V.Massol, G.Gregory: *JUnit in Action, Second Edition*, 2010.
- [11] Squish Manual, <http://doc.froglogic.com>
- [12] Hudson Extensible continuous integration server, <http://hudson-ci.org/docs/>