УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Немања Ђекић

Интеграција ланаца алата у *Eclipse* развојно окружење, користећи *CDT Managed Build System* проширење

МАСТЕР РАД

Нови Сад, 2017

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**:		Немања Ђекић	
Ментор, **МН**:		др Јелена Ковачевић, доцент	
Наслов рада, **НР**:		Интеграција ланаца алата у *Eclipse* развојно окружење, користећи *CDT Managed Build System* проширење	
Језик публикације, **ЈП**:		Српски / латиница	
Језик извода, **ЈИ**:		Српски	
Земља публиковања, **ЗП**:		Република Србија	
Уже географско подручје, **УГП**:		Војводина	
Година, **ГО**:		2017	
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)		7/34/0/0/10/0/0	
Научна област, **НО**:		Електротехника и рачунарство	
Научна дисциплина, **НД**:		Рачунарска техника	
Предметна одредница/Кqучне речи, **ПО**:		Ланац алата; *Eclipse развојно* окружење; Развој утичних проширења; *Java*; *C* програмски преводилац; Асемблерски преводилац; Повезивач;	
УДК			
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад	
Важна напомена, **ВН**:			
Извод, **ИЗ**:		У овом раду представљен је један од начина интеграције ланаца алата, рађених по наруџби за две различите *DSP* архитектуре, у *Eclipse* развојно окружење помоћу *Managed Build System* проширења. Основни циљ интеграције алата је повећан квалитет и ефикасност развоја апликација за циљну платформу. Ово решење омогућава интеграцију различитих алата који помажу програмерима да побољшају квалитет кода, што побољшава крајњи производ.	
Датум прихватања теме, **ДП**:		15.9.2017.	
Датум одбране, **ДО**:		5.10.2017.	
Чланови комисије, **КО**:	Председник:	др Миодраг Ђукић, доцент	
	Члан:	др Растислав Струхарик, ванред. проф.	Потпис ментора
	Члан, ментор:	др Јелена Ковачевић, доцент	

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

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**:		Nemanja Đekić	
Mentor, **MN**:		Jelena Kovačević, PhD	
Title, **TI**:		Custom Toolchain integration into the Eclipse IDE via CDT Managed Build System Extensibility	
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**:		2017	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Dositeja Obradovića sq. 6	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		7/34/0/0/10/0/0	
Scientific field, **SF**:		Electrical Engineering	
Scientific discipline, **SD**:		Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, **S**/**KW**:		Custom Toolchain; *Eclipse* IDE; Plug-in development; *Java*; *C* Compiler; Assembler; Linker;	
UC			
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:			
Abstract, **AB**:		This paper describes custom Toolchains integration, developed for two different *DSP* architecuters, into the Eclipse IDE via CDT Managed Build System Extensibility. The main goal of the Toolchain integration is increased quality and efficient application development for target platforms. This solution enables integration of various tools that helps software developers to improve quality of the code, which improves the final product.	
Accepted by the Scientific Board on, **ASB**:		15.9.2017.	
Defended on, **DE**:		5.10.2017.	
Defended Board, **DB**:	President:	Ph. D. Miodrag Đukić, docent	
	Member:	Ph. D. Rastislav Struharik, professor	Menthor's sign
	Member, Mentor:	Ph. D. Jelena Kovačević, docent	

Zahvalnost

Zahvaljujem se RT-RK institutu na pruženoj mogućnosti za realizaciju ovog rada, kao i svim kolegama sa kojima sam sarađivao na ovom projektu. Posebno se zahvaljujem saradnicima, Momčilu Kruniću i Marku Rajincu na razumevanju, stručnoj pomoći tokom izrade ovog rada i nesebičnom deljenju znanja.

Veliku zahvalnost dugujem mojim roditeljima i bratu zbog pružene moralne, emotivne i finansijske podrške tokom celokupnog školovanja.

Neveni, koja se relativno skoro pridružila ovom interesantnom putovanju, zahvaljujem se na nesebičnoj, moralnoj i emotivnoj, podršci za moje ideje i vizije.

Takođe bih se zahvalio svima koji su pratili moje ideje, vizije i na bilo koji način pružali podršku.

SADRŽAJ

SPISAK SLIKA

SKRAĆENICE

CPU - *Central Processor Unit*, Centralni procesoor

DSP - *Digital Signal Processor*, Procesor za obradu digitalnih signala

RCP - *Rich Client Platform,* Bogata klijentska platforma

IDE - *Integrated Development Environment*, Integrisano razvojno okruženje

JDT - *Java Development Tool*, *Java* razvojni alat

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

SDK - *Software Development Kit*, Komplet za razvoj programske podrške

API - *Application Interface*, Aplikativna sprega

GUI - *Grafic User Interface*, Grafička korisnička sprega

CDT - *C/C++ Development Tools*, Alati za razvoj *C/C++* aplikacija

MBS - *Managed Build System,* Upravljani gradivni sistem

CMD - *Command Line*, Komandna linija

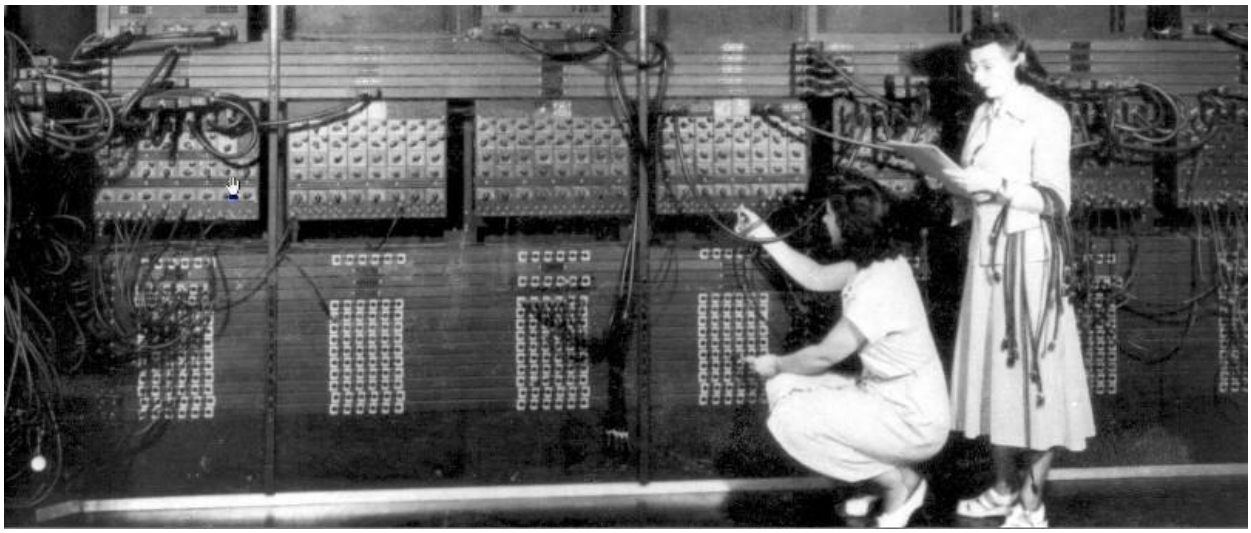
XML - *eXtensible Markup Language*, Proširivi meta jezik, tj. jezik za označavanje tekstualnih dokumenata

1. Uvod

U ovom radu predstavljena je integracija lanaca alata u *SCS* (engl. *Sound Clear Studio*) razvojno okruženje koje je bazirano na *Eclipse* razvojnoj platformi. Tokom izrade ovog rada korišteni su *ADSP2* i *Halo* lanci alata za dve različite arhitekture procesora za digitalnu obradu signala (engl. *Digital Signal Processor – DSP*) [1] [2].

Osnovni cilj integracije lanaca alata je povećan kvalitet i efikasnost razvoja aplikacija za ciljnu platformu [3]. Rešenje, predstavljeno u ovom radu, omogućava integraciju različitih alata koji pomažu programerima da poboljšaju kvalitet koda, što poboljšava krajnji proizvod.

Prvi računar *ENIAC* (engl. *Electronic Numerical Integrator and Computer*) programiran je uz pomoć prekidača i kablova, tačnije postavljanjem skupa prekidača u određena stanja i povezivanjem komponenti kablovima [4] (Slika 1.1).



Slika 1.1 Programiranje *ENIAC* računara pomoću prekidača i kablova (1946-1955)

Na osnovu stanja u koja su postavljeni prekidači *ENIAC* bi izvršavao određene operacije i štampao rezultate. Ovaj računar je obavljao operacije koje danas može da obavi najobičniji kalkulator, a posedovao je oko 6000 prekidača i vreme za dobijanje rezultata je trajalo danima.

U današnjem vremenu ovakav vid programiranja je nezamisliv. Umesto podešavanja stanja prekidača, programeri danas koriste programske jezike visokog nivoa (Python, Java, C++, ...) kako bi predstavili algoritme i opisali strukturu i ponašanje programa.

Detaljnije razumevanje programskih jezika nije moguće bez poznavanja skupa razvojnih alata. Pored sistemskih i korisničkih programa izdvaja se posebna grupa razvojnih programa, poznatija kao ulančani alati (engl. *Tool-chain*). Izraz ulančani alati predstavlja više nezavisnih alata, međusobno povezanih u lanac, gde je izlaz prethodnog ulaz u naredni alat. Razvojni alati su zapravo grupa kojoj pripadaju sledeći programi:

1. Programski prevodilac (engl. *Compiler*)
2. Asemblerski prevodilac (engl. *Assembler*)
3. Povezivač (engl. *Linker*)
4. Alat za kontrolisano izvršavanje programa (engl. *Simulator*)

Programski prevodioci su alati koji imaju ulogu da tumače programe napisane u programskim jezicima višeg nivoa i transformišu ih u ekvivalentne programe poštujući pravila ciljnih programskih jezika. Otkrivanje upozorenja i grešaka prisutnih u izvornom programu i prijavljivanje istih korisniku su neke od funkcija koje programski prevodilac obavlja. Zahvaljujući velikom izboru programskih jezika višeg nivoa, danas je proces pisanja programa uveliko olakšan, stoga je moguće zaključiti da postoji i veliki broj programskih prevodilaca i načina da se program iz izvornog programskog jezika prevede na ciljni programski jezik, tj. mašinski jezik ciljne platforme. Vrlo važno je odrediti kada je jedan programski prevodilac bolji od drugog, za istu ciljnu arhitekturu, jer je u direktnoj sprezi sa efikasnošću izvršavanja programskog rešenja.

Veličina, brzina izvšravanja i brzina prevođenja izvornog koda su karakteristike koje označavaju kvalitet programskog prevodioca. Na osnovu uspešno prevedinih ispitnih kodova utvrđena je ispravnost integrisanih lanaca alata.

U narednom poglavlju biće opisano *Eclipse* razvojno okruženje. Treće poglavlje sadrži koncept rešenja, u kome je objašnjeno šta je sve potrebno za integraciju lanaca alata. Zatim sledi realizacija rešenja, gde se navodi na koji način je izvršena implementacija. Verifikacija programskog rešenja i ispitivanje njegove ispravnosti opisani su u petom poglavlju. Poslednje

poglavlje sadrži zaključak, gde su sumirani postignuti rezultati i sve što je urađeno, sa osvrtom na dalje pravce razvoja i moguća unapređenja.

2. *Eclipse* platforma

Eclipse RCP [5] je razvojno programsko okruženje, koje daje mogućnost proširivanja razvojne platforme. Okruženje je slobodno-dostupnog koda (engl. *Open source*) i zasniva se na *Java* programskom jeziku. Omogućava razvoj alata, samostalnih *RCP* (engl. *Rich Client Platform*) aplikacija, kao i razvoj novih razvojnih okruženja.

Kompanija *IBM* [6] je 1998. godine počela sa *Eclipse* projektom u cilju da napravi zajedničku platformu za sve razvojne timove te kompanije. 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* [7] programskom jeziku. Mnoge kompanije su usvojile *Eclipse* razvojno okruženje, jer je pogodno *Java* integrisano razvojno okruženje i platforma za integrisanje velikog broja različitih alata. *Eclipse* razvojno okruženje unapređuje veliki broj ljudi i kompanija koji unapređuju i proširuju funkcionalnost i doprinose sve većoj upotrebi *Eclipse*-a razvojne platforme u programskom svetu.

Javna licenca (engl. *Eclipse Public Licence*) [8] daje jedinstvenost *Eclipse*-u, jer omogućava prava na sav izvorni kod i besplatnu distribuciju. Pored toga što *Eclipse* trenutno ima ogromnu primenu i uticaj u razvijanju alata, namenskih sistema i ispitivanju programske podrške, mnoge kompanije i dalje pristupaju zajednici otvorenog koda i samim tim učestvuju u konstantnom poboljšanju kvaliteta same razvojne platforme. Za uzvrat, većina kompanija razvija sopstvene alate u komercijalne svrhe i time ostvaruju ogromnu dobit.

Alati i aplikacije koje se razvijaju na *Eclipse* razvojnoj platformi mogu biti napisani u *Java* programskom jeziku, kao i u drugim programskim jezicima kao što su: *C, C++, Cobol, Haskell, Perl, PHP, Python, R, Ruby, Groovy,...* , uz pomoć odgovarajućih utičnih proširenja (engl. *Plug-ins*).

Utično proširenje (engl. *Plug-in*) je najmanja funkcionalna celina *Eclipse* platforme. Složeniji alati se obično sastoje od većeg broja utičnih proširenja. Navedena proširenja se

implementiraju u *Java* programskom jeziku. Utično proširenje se uglavnom sastoji od *Java* koda u *Java* arhivnoj biblioteci i dodatnih resursa kao što su ikonice i slike. Na osnovu opisnog dela svakog utičnog proširenja moguće je definisati veze navedenog proširenja sa ostalim utičnim proširenjima u sistemu. Model povezivanja realizovan je tako da svako utično proširenje može definisati svoje tačke proširenja tako da se druga utična proširenja mogu povezati na iste i menjati im funkcionalnost. Povezivanje može biti ostvareno na jednu ili više drugih tačaka proširenja. Svaka tačka proširenja definiše svoju programsku spregu koju sva utična proširenja koja se povežu sa njom moraju implementirati. Kada se pokrene platforma, jezgro platforme pretražuje dostupna utična proširenja, čita njihove opisne datoteke i popunjava registar utičnih proširenja (engl. *Plug-in register*) ovim informacijama. Na ovaj način vrši se provera kompatibilnosti utičnih proširenja i tačaka na koje se povezuju. Prilikom rada sa platformom dodaju se i uklanjaju utična proširenja.

Datoteke *manifest.mf* i *plugin.xml* predstavljaju opisani deo utičnih proširenja. Datoteka *manifest.mf* napisana je po *OSGi* [9] specifikaciji i ona opisuje zavisnost utičnog proširenja od drugih utičnih proširenja. Za razliku od *manifest.mf* datoteke, *plugin.xml* 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. Svim potrebnim informacijama o tačkama proširenja moguće je pristupiti iz registara utičnih proširenja bez učitavanja bilo kakvog koda ili aktiviranja samih utičnih proširenja. Na ovaj način platforma može da sadrži veliku bazu utičnih proširenja od kojih se učitavaju samo ona koja su potrebna.

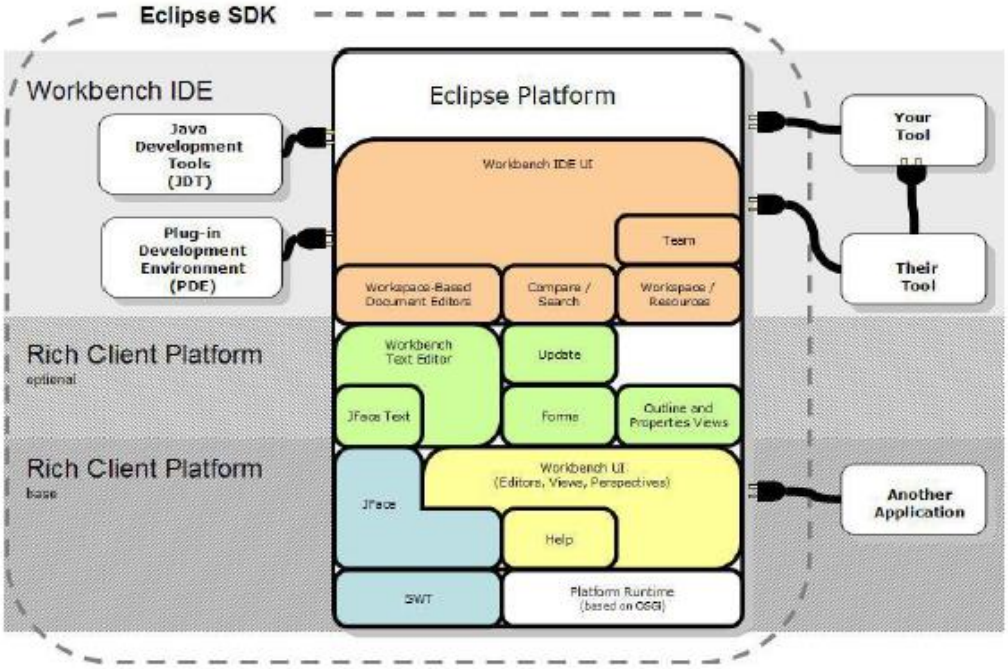
Utična proširenja se aktiviraju po potrebi. Kada se pokrenu ona koriste registar utičnih proširenja da otkriju i pristupe utičnim proširenjima povezanim na njihove tačke proširenja. Utično proširenje će ostati aktivirano sve dok ne bude eksplicitno deaktivirano ili platforma bude ugašena. Utična proširenja sadrže fascikle u kojima se čuvaju stanja i vrednosti promenljivih iz jedne sesije, koja mogu da se koriste u narednoj sesiji. Poznavanje svih dostupnih utičnih proširenja i mogućnost razmene informacije o njima, pre njihovog pokretanja, skraćuje vreme potrebno za njihovo pokretanje, memorijske zahteve kao i uklanjanje problema vezanih za redosled pokretanja utičnih proširenja.

Java virtualna mašina (engl. *Java Virtual Machine*) pokreće *Eclipse* platformu. Svako utično proširenje ima svoj *Java* pokretač klasa (engl. *Java Class Loader*) koji je zadužen za učitavanje njegovih klasa. Svako utično proširenje eksplicitno deklariše svoju zavisnost od drugih utičnih proširenja i kontroliše vidljivost javnih (engl. *Public*) klasa i programskih sprega svojih biblioteka. Navedene informacije, kao i kojim klasama će svako utično proširenje

pristupati, nalaze se u opisanim datotekama, a pravila vidljivosti su podržana tokom izvršavanja od strane pokretača klasa.

Jezgro *Eclipse* platforme (Slika 2.1) [10] se sastoji iz nekoliko osnovnih podsistema:

1. Radno okruženje (engl. *Workspace*)
2. Radni sto (engl. *Workbench*)
3. Podsistem za pomoć korisniku (engl. *Help*) – mehanizam preko kog *Eclipse* alati mogu definisati i dodati svoju dokumentaciju u nezavisne knjige dokumentacija. 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 2.1 Najvažnije komponente *Eclipse* platforme

2.1 Radno okruženje

Radno okruženje je sačinjeno od jednog ili više projekata koji predstavljaju odgovarajući direktorijum u direktorijumskoj strukturi. Projekti se mogu nalaziti u bilo kom direktorijumu, ali je nepisano pravilo da se nalaze u odgovarajućim poddirektorijumima, direktorijuma radnog

okruženja. Mehanizam svojstva (engl. *Nature*) projekta opisuje zapravo tip projekta. Utična proširenja imaju mogućnost da definišu nove tipove svojstava projekta i načine na koji se oni konfigurišu. Jedan projekat može imati više svojstava, stoga može biti deljen između većeg broja utičnih proširenja i na taj način razvojno okruženje poseduje podršku za više različitih tipova projekata.

Korisnik u potpunosti upravlja datotekama koje su sadržane unutar projekta. *Eclipse* razvojna platforma definiše programsku podršku za rad sa resursima radnog okruženja. Pomoću prilagodljivih objekata predstavljaju se resursi i definiše njihova funkcionalnost.Radno okruženje nudi i jednostavan mehanizam koji prati kratku istoriju izmena nad datotekama, tako da one mogu biti vraćene u neko od prethodnih stanja.

Marker za obeležavanje resursa je takođe integrisan u razvojno okruženje. Pomoću njih se obeležavaju delovi u datotekama kao što su: greške prevodioca, delovi koda koje treba doraditi, rezultati pretrage (engl. *Search*), obeleživači pozicije (engl. *Bookmarks*), radni zadaci (engl. *Tasks*) i prekidne tačke (engl. *Breakpoints*). Ovaj mehanizam je otvoren tako da utična proširenja mogu definisati nove tipove markera i njihove načine korišćenja.

Snimanje sadržaja razvojnog okruženja je otvoren proces pomoću kojeg je moguće uvezati utična 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. Prilikom sledećeg pokretanja 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 i odreagovati na promene koje su se desile dok je bilo neaktivno.

2.2 Radni sto

Radni sto ima centralnu ulogu unutar *Eclipse* platforme i ujedno je sinomim za korisničku spregu iste. Korisnička sprega *Eclipse* platforme definiše utična proširenja radnog stola koja sadrže strukture preko kojih alati komuniciraju sa korisnikom. 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 uređivač datoteka (engl. *Editor*), pogledi (engl. *Views*) i perspektive (engl. *Perspective*).

Pomoću uređivača datoteka korisnik može 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. Kada su u aktivnom režimu uređivači mogu dodati svoje akcije u menije ili u paletu sa alatkama. Standardni uređivač za tekstualne datoteke

definisan je platformom. Uređivači za posebne tipove datoteka moraju biti prilagođeni od strane utičnih proširenja.

Podaci o nekom objektu sa kojima korisnik radi prikazani su pomoću pogleda. Na primer, dok korisnik unosi programski kod u nekom tekstualnom uređivaču, pogled može biti upotrebljen da prikaže informacije o tom delu koda (funkcijama, metodama i promenljivim definisanim u tekućem programskom bloku i slično). Pogled takođe može da bude izvor sadržaja koji će biti prikazan u drugom pogledu (npr. jedan pogled može prikazivati detalje objekta izabranog u drugom pogledu). Pogledi imaju jednostavan životni ciklus, jer su izmene u njima momentalno sačuvane i njihove posledice odmah prikazane u ostalim delovima korisničke sprege (engl. *User Interface*).

Jedna perspektiva *UI* platforme može biti prikazana u datom trenutku, iako je poznato da ih ima više. Perspektiva može da sadrži uređivače teksta i pregleda. Njihov izbor i raspored zavise od svrhe konkretne perspektive. Perspektive za navigaciju kroz resurse, pregledanje dokumenata za pomoć korisniku (engl. *Help*) i podrška za rad u timu unapred su definisane unutar same platforme.

Utična proširenja se integrišu sa *UI* platformom na sledeće načine:

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

Elementi perspektive se učitavaju po potrebi i zatvaraju kada više nisu neophodni. U opisanim datotekama utičnih proširenja nalaze se ikone i nazivi njihovih akcija, tako da stavke u meniju mogu biti iscrtane bez učitavanja samih elemenata.

2.3 *Eclipse* alati za razvoj programske podrške

Kombinacija nekoliko *Eclipse* projekata, uključujući *Eclipse*, *Java* razvojne alate (engl. *Java Development Tool – JDT*) i okruženja za razvoj utičnih proširenja (engl. *Plug In Development – PDE*), čini *Eclipse* alate za razvoj programske podrške (engl. *Eclipse Software Development Kit – SDK*). *Eclipse SDK* je integrisano razvojno okruženje za razvoj *Java* aplikacija i izgradnju proizvoda baziranih na *Eclipse* platformi. *Eclipse SDK* je nezavisna platforma, a sastoji se od jezgra (engl. *Core*) i raznih priključaka (engl. *Plug-ins*). Jezgro pruža usluge za upravljanje priključcima, a priključci pružaju potpunu funkcionalnost.

2.4 *Eclipse* integrisano razvojno okruženje

Eclipse integrisano razvojno okruženje (engl. *Integrated Development Environment*) je programska podrška koja pruža širok skup objekata za razvoj programskih aplikacija. Uglavnom se sastoji od: uređivača izvornog koda, programskog prevodioca (engl. *Compiler*), alata za građenje koda i pronalaženje grešaka.

2.5 *Eclipse* okruženje za razvoj priključaka

Skup alata dizajniranih da pomažu programeru u razvoju, ispitivanju, izgradnji izvršnih arhiva i isporuci (engl. *Deployment*) *Eclipse* dodataka, fragmenata, mogućnosti (engl. *Features*) i *RCP* aplikacija, naziva se *Eclipse* okruženje za razvoj priključaka (engl. *Plug-in Development Environment – PDE*).

PDE je sastavljen od tri glavne komponente:

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

Svaka od navedenih komponenti funkcioniše samostalno. Postoje i dve dodatne komponente:

1. Dokumentacija (engl. *Documentation*) – omogućava pomoćnu dokumentaciju korisniku
2. Inkubator – služi za razvoj alata i nadogradnji koje nije moguće smestiti u *Eclipse SDK*

2.6 *Eclipse* aplikativna programska sprega

Aplikativna programska sprega radnog stola i njegova implementacija zasnivaju se na *SWT JFace* bibliotekama. Pomoću navedenih biblioteka vrši se razvoj grafičke korisničke sprege (engl. *Graphic User Interface – GUI*) sa korisnikom u *Eclipse* razvojnom okruženju.

2.6.1 *SWT* grafička biblioteka

SWT biblioteka je skup grafičkih elemenata koji su integrisani sa grafičkim delom operativnog sistema. Kao zamenu za *AWT* i *SWING* biblioteke, u sklopu *Eclipse* projekta, *IBM*-ov tim je napisao novu biblioteku u *Java* programskom jeziku. *SWT* omogućava razvoj prenosivih aplikacija koje direktno pristupaju grafičkom delu bez obzira koji operativni sistem je u pitanju. Glavni nedostatak *AWT* (engl. *Abstract Window Toolkit*) grafičke biblioteke je integracija i prenosivost iste na operativnim sistemima. Iz navedenih razloga *Eclipse* platforma i njeni alati koriste *SWT* biblioteku za predstavljanje grafičke implementacije korisniku. *AWT* je

bila u direktnoj interakciji sa grafičkim elementima operativnog sistema, ali operativnog sistema na kome je razvijen određeni alat ili system. Za razliku od *AWT* biblioteke, koja je podržavala samo elemente nižeg nivoa, *SWING* je imao problem sa performansama i prenosivošću zbog velikih memorijskih zahteva i iz navedenog razloga javila se potreba za novim rešenjem. *IBM*-ov tim je odlučio da razvije novu biblioteku.

SWT koristi elemente grafičkog podsistema operativnog sistema kad god je to moguće, a za elemente koje ne podržava operativni sistem vrši se njihova simulacija. *SWT* teži da održi funkcionalnost i verodostojan izgled operativnog sistema na kome se koristi.

2.6.2 *Jface* biblioteka

*JF*ace je biblioteka koja sadrži klase koje služe za programiranje ulazno izlaznih zadataka korisničke sprege. Radi sa *SWT* bibliotekom i potpuno je nezavisna od operativnog sistema na kom se izvršava.

JFace poseduje 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 predstavljaju prozore koji prikazuju liste, stabla i table i rade sa istim. Pogled se smatra jednim od najvažnijih elemenata *JFace* biblioteke. Dodatna obrada sadržaja koji se prikazuje u prozorima, omogućena je u vidu sortiranja, nazubljivanja, prevlačenja, obeležavanja iz razloga što pogled podržava akcije kao što su pritisak tastera na mišu, vraćanje pogleda na poslednju verziju, kao i kretanje kroz sadržaj pogleda.

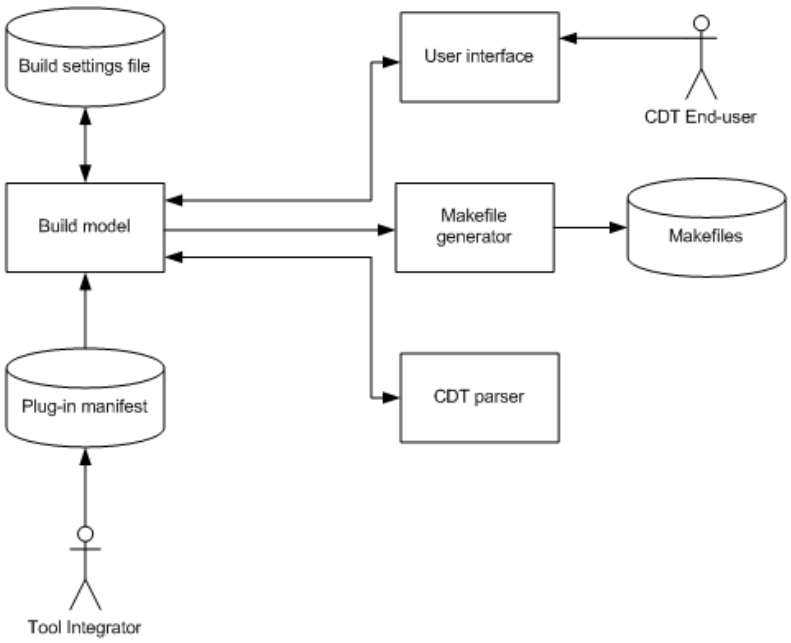
Akcije su još jedan od važnijih elemenata *JFace* biblioteke i one omogućavaju da korisničke komande mogu biti definisane nezavisno od toga gde se nalaze na korisničkoj sprezi. Akcija je komanda koja može biti pokrenuta pritiskom na dugme, opciju menija, polje u paleti alata.

JFace omogućava rad sa fontovima i slikama, kao i njihovo međusobno deljenje među korisničkim alatima.

Dijalozi i čarobnjaci omogućavaju korisniku da kroz jedan ili više uzastopnih koraka izvrši određene akcije, a sami koraci su propraćeni odgovarajućim pomoćnim asistencijama.

3. Koncept rešenja

Glavni zadatak ovog rada je bila integracija lanaca alata (engl. *Custom Toolchains*) za dve različite *DSP* arhitekture, u *Eclipse* razvojno okruženje. Integracija lanaca alata uz pomoć upravljanog sistema gradnje (engl. *Managed Build System – MBS*) [*11*] u *CDT*[1]-u [12] delovala je kao najjednostavniji pristup za rešavanje navedenog zadatka. U narednim paragrafima biće ukratko opisan koncept samog upravljanog sistema gradnje.



Slika 3.1 Prikaz upravljanog sistema gradnje

[1] *Eclipse* projekat za gradnju *C/C++* alata.

3.1 Upravljani sistem gradnje u *CDT-u*

Gradnja datoteka u *CDT* projektu moguća je na dva načina:

1. Standardna gradnja – Korisnik sam piše gradivne datoteke.
2. Upravljana gradnja – *CDT* Sistem za upravljanje gradnje koda automatski generiše gradivne datoteke.

Postoji nekoliko prednosti *MBS* sistema:

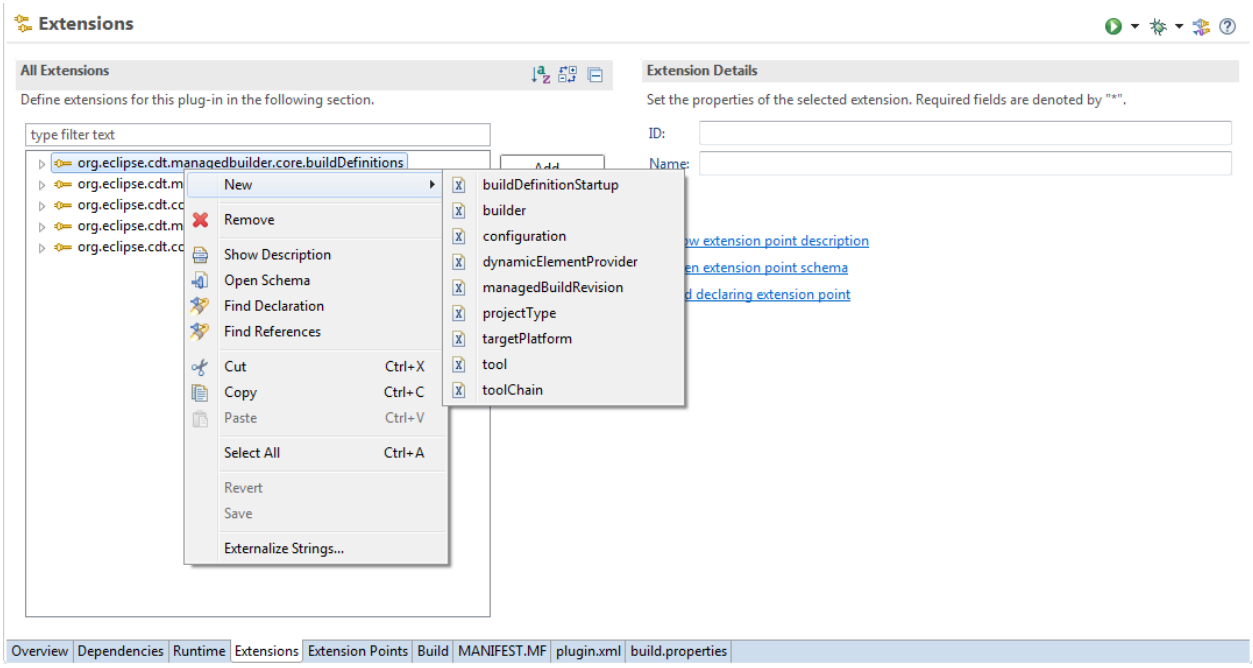
1. Automatsko generisanje datoteka – *MBS* u potpunosti automatizuje generisanje gradivnih datoteka, tako da korisnik ne mora da radi sa sintaksom gradivnih datoteka i postavke gradnje se kontrolišu pomoću korisničke sprege. Pored svega navedenog korisnik ne mora da menja gradivne datoteke svaki put kada se u projektu dese neke promene (npr. brisanje i dodavanje datoteka). *MBS* regeneriše navedene datoteke po potrebi.
2. Snabdevanje informacijama za gradnju koda – Neretko se dešava da naznačene informacije alata u komandnoj liniji (engl. *Command Line – CMD*) budu potrebne i u drugim delovima *CDT* projekta. Npr. programski prevodilac može da koristi ugrađene skupine uključenih direktorijuma i makroa koji su takođe potrebni parseru. *MBS* može automatski da izračuna ove informacije na osnovu trenutnih postavki sistema za gradnju i da obezbedi ove informacije ostatku *CDT* projekta.
3. *MBS* omogućava programerima da definišu grupu veoma korisnih *call-back* funkcija. *Call-back* je mehanizam koji omogućava da se funkcija prosledi kao parameter, da bi kasnije bila pozvana po potrebi. Na primer, programeri mogu pomoću integrisanog *ITool* interfejsa, na osnovu *ID*-a (engl. *Indetification number*) alata, da menjaju komande za pozivanje istih. Pomoću *setToolCommand* metode *ITool* interfejsa, postavlja se određena komanda za pozivanje datog alata. Osim komande, ovoj metodi se prosleđuje i binarna promenljiva, koja sadrži putanju do navedenog alata.

   Primer:

```
ITool compilerTool = configuration.getToolBySuperClassID("scs.toolchain.compiler")[0];
compilerTool.setToolCommand("\"$(ADSP2_BIN)/chesscc\" + w chessroot + r" );
```

MBS generiše datoteke za gradnju i automatski dobija informacije o gradnji na osnovu:

1. definicija lanaca alata, alata i graditelja – Programer definiše lance alata, alate i graditelje pomoću *org.eclipse.cdt.managedbuilder.core.buildDefinition* tačke proširenja. (Slika 3.2),



Slika 3.2 Definisanje alata pomoću *org.eclipse.cdt.managedbuilder.core.buildDefinition* tačke proširenja

2. korisnički prilagođenih postavki, kao što su: tipovi ulaznih i izlaznih datoteka, alati koji se koriste za gradnju određenih projekata i druge,
3. projektnih resursa, kao što su: gradivne datoteke, biblioteke koje koriste navedene datoteke, zaglavlja itd.

3.1.1 Definicija lanaca alata

Integracija lanaca alata u *Eclipse* razvojno okruženje moguća je uz pomoć tačke proširenja *org.eclipse.cdt.managedbuilder.core.buildDefinition*. Ova tačka proširenja obezbeđuje mehanizme za definiciju i opisuje posebnom sintaksom spoljašnje alate (engl. *External Tools*) koji se koriste za gradnju projekta. Kroz *MBS* definiše se pojam lanac alata. Lanac alata je skup alata koji su potrebni za prevođenje i gradnju projekata. *MBS* definicija alata određuje alate i definicije graditelja koji se koriste za igradnju. Ova definicija alata predstavlja spošljanji alat i način na koji se koriste prilikom izgradnje. Definicija alata sadrži:

1. Alatnu komandu (engl. *Tool command*) – Koristi se za pokretanja alata

2. Skup definicija opcija alata. - Svaka definicija opcije određuje tip vrednosti koji predstavlja opciju, načina na koji je vrednost predstavljena prilikom pozivanja alata, uobičajenu vrednost opcije i ime opcije.
3. Skup ulaza alata – Opisuje koje tipove resursa može da gradi navedeni alat
4. Skup izlaza alata – Opisuje koje tipove resursa generiše navedeni alat

Definicija gradivnih alata predstavlja komandu i argumente koji se koriste za pokretanje graditelja, kao i generator koji generiše gradivne datoteke zasnovane na projektnim informacijama.

Tip projekta definiše grupe skupova konfiguracija (npr. Otklanjanje grešaka (engl. *Debug*)) koje se koriste za sam projekat. Svaka konfiguracija određuje artifakt koji nastaje kada je konfiguracija izgrađena, alat koji je korišten za gradnju, specifična podešavanja za alatne opcije i druge konfiguracione informacije.

3.1.2 Izmene podešavanja gradnje

MBS obezbeđuje korisničku spregu za izmene podešavanja gradnje datoteka. *MBS* takođe implementira projekat i datoteke koji su potrebni za izmene podešavanja redom.

MBS generiše prikladnu korisničku spregu zasnovanu na informacijama o lancu alata koje obezbeđuje programer. Korisnik može da izmeni sledeća podešavanja:

1. Alatna podešavanja – Komande za pozivanje alata i opcije za svaki alat koji će se koristiti prilikom gradnje projekta.
2. Podešavanja gradnje – Komande za gradnju i argumenti gradnje.
3. Podešavanja specifičnih koraka gradnje – Omogućavaju korisniku da definiše skup komandi pre ili posle gradnje projekta i obezbeđuju mogućnost definisanje i drugih alata, pored onih koje je programer definisao.
4. Procesno okruženje za gradnju
5. Druga podešavanja za gradnju – Makro podešavanja, podešavanja parsiranja greške, podešavanja binarnog parsera i dr.

3.1.3 Gradnja projekta

Kada se pozove instrukcija za gradnju projekta, poziva se generator gradivnih datoteka koji generiše prikladne gradivne datoteke za projektne resurse, zasnovane na projektnim gradivnim informacijama. *MBS* obezbeđuje generator gradivnih datoteka koji generiše gradivne datoteke za

GNU Make [2]alat [13]. Ne postoji ograničenje koji graditelj će biti korišten, tako da programeri mogu obezbediti generator gradivnih datoteka napravljen prema korisničkim potrebama, što znači da može da se koristi bilo koji graditelj prilagođen korisničkim potrebama.

Kada postoji skup projektnih resursa i povezanih gradivnih postavki, generator gradivnih datoteka je spreman za generisanje gradivnih datoteka. Generator otkriva alatnu komandu koja je korištena za gradnju datog projekta, na osnovu ulaznih i izlaznih informacija i postavki opcija alata.

Kada se izgenerišu sve potrebne gradivne datoteke, poziva se spoljašnji graditelj da gradi projekat. Korisnički definisana i programerski definisana okruženja su uključena u skup promenljivih (engl. *Environment variables*) koje opisuju razvojno okruženje, a koje se koriste za gradivni proces.

Pored navedenih, *MBS* definiše i neke specijalne tipove opcija koje mogu da se koriste za specifične opcije koje predstavljaju biblioteke, objekte i preprocesorske simbole alata. Na osnovu vrednosti ovih opcija, *MBS* automatski računa potrebne informacije za izgradnju i prosleđuje ih ostatku *CDT*-a.

[2] *GNU Make* – Alat koji kontroliše generisanje izvršnih datoteka i drugih ne-izvornih datoteka programa iz izvornih datoteka programa

4. Realizacija rešenja

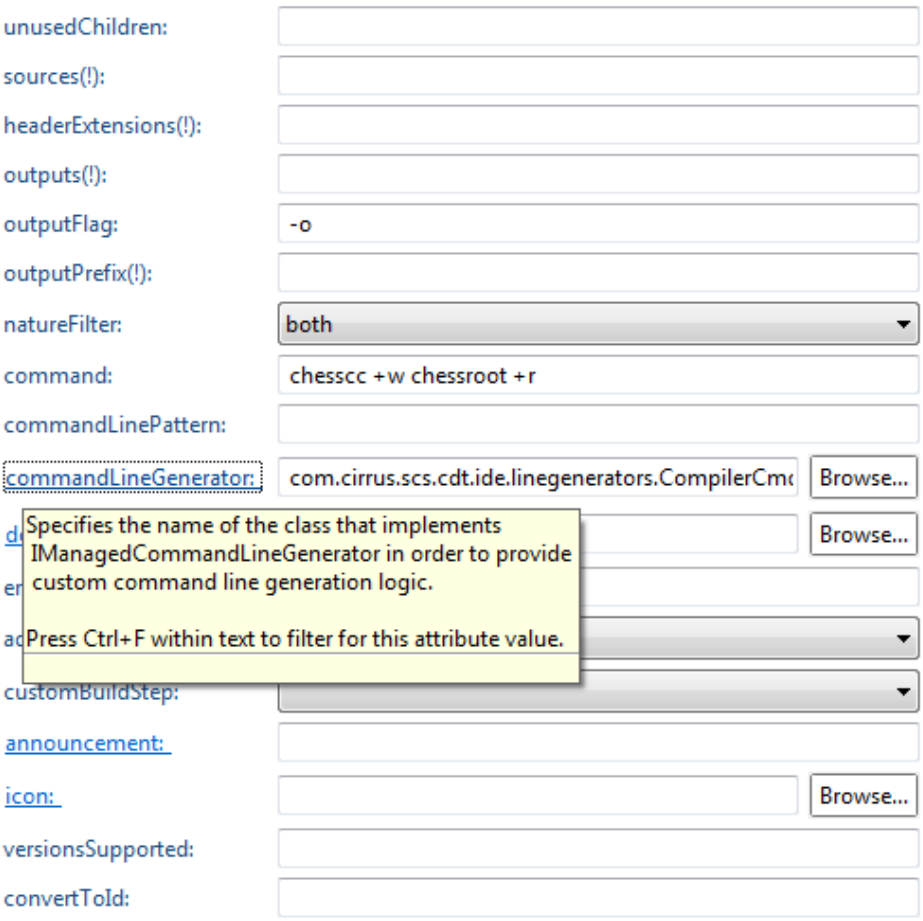
Jedna od mogućnosti koju nudi *MBS* je integracija projekata i alata prilagođenim korisniku u *Eclipse* razvojno okruženje. U ovom poglavlju biće detaljno opisan postupak integracije lanaca alata u *Eclipse* razvojno okruženje. Integracija alata realizovana je pomoću utičnih proširenja i *Java* programskog jezika.

4.1 Podešavanje razvojnog okruženja

Integracija alata vrši se pomoću tačke proširenja *buildDefinitions,* [14] koja je definisana u *org.eclipse.cdt.managedbuilder.core* utičnom proširenju. Potrebno je imati pristup tački proširenja i šemi koja je opisuje, da bi se vršila bilo kakva integracija. Navedena tačka proširenja i šema nalaze se u *SDK* verziji *CDT*-a. Da bi bilo moguće integrisati lanac alata, potrebno je instalirati isti u razvojno okruženje.

4.2 Utična proširenja

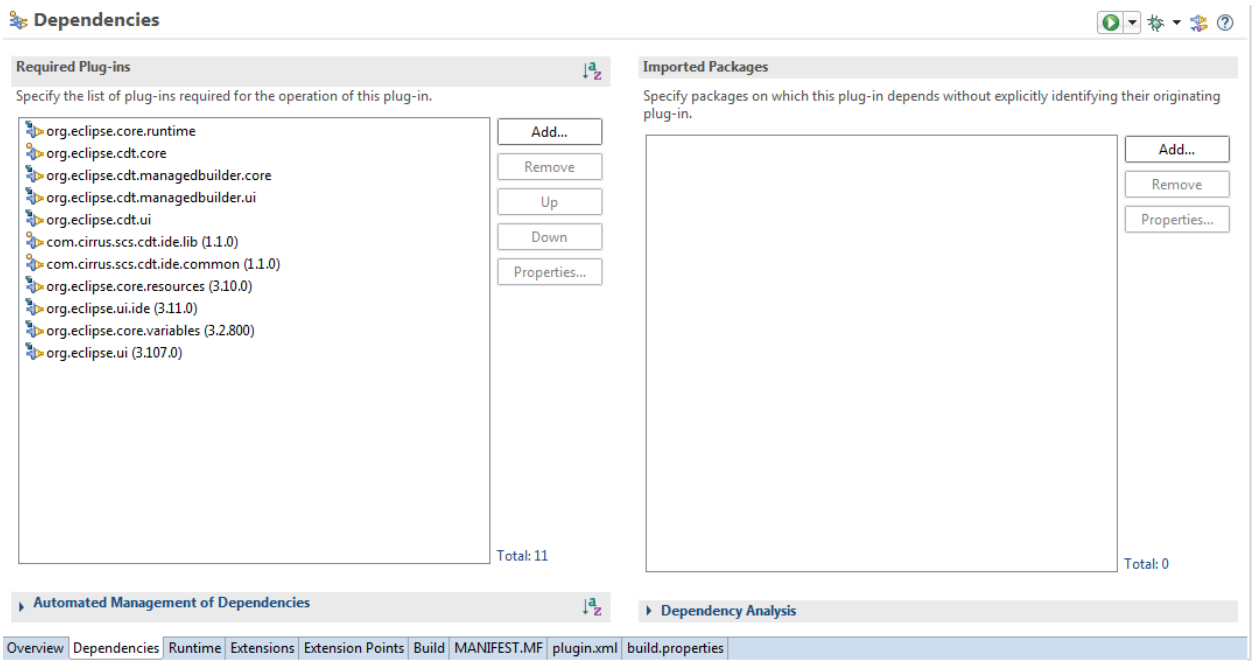
Tačke proširenja mogu biti definisane u bilo kom utičnom proširenju, ali je za potrebe rada napravljen novi projekat, odnosno novo utično proširenje sa praznom *MF* (engl. *Manifest file*) datotekom. U ovom slučaju, novi projekat je dobio ime *com.scs.cdt.ide.toolchain,* ali naravno ono može da bude proizvoljno. U sadržaju utičnog proširenja moguće je videti da je čarobnjak postavio indetifikacioni broj *ID,* npr. *adsp2.scs.toolchain.toolchain* , ali korisnik može da menja *ID* prema sopstvenim potrebama. Unutar *MF* datoteke definiše se integracija alata bez pisanja bilo kakavog koda i pomoću polja „*Generate Java Class that...“* se generiše java klasa u kojoj je moguće definisati komande alata, koje se pozivaju kroz komandnu liniju. Ako je klasa već izgenerisana, dovoljno je pritisnuti dugme *Browse* i izabrati željenu klasu.



Slika 4.1 Generisanje *Java* klase za određeni alat

4.3 Proširenja

U *com.cirrus.scs.cdt.ide* utičnom proširenju unutar *MF* datoteke, definisani su alati pomoću *MBS*-a proširivanjem *buildDefinition* tačke proširenja. Osim definicije alata, dodata je i zavisnost između *com.cirrus.scs.cdt.ide* utičnog proširenja i *org.eclipse.cdt.managedbuilder.core* utičnog proširenja na mestu gde je tačka proširenja definisana. U polju *Dependencies* (Slika 4.2) dodata su zahtevana utična proširenja (engl. *Required Plug-ins*) od kojih je jedno *org.eclipse.cdt.managedbuilder.core* utično proširenje. Pristup samim proširenjima moguć je pomoću *Java* klasa koje su vezane za ista. Pomoću *Java* klasa korisnik može da podešava proširenja prema sopstvenim potrebama.

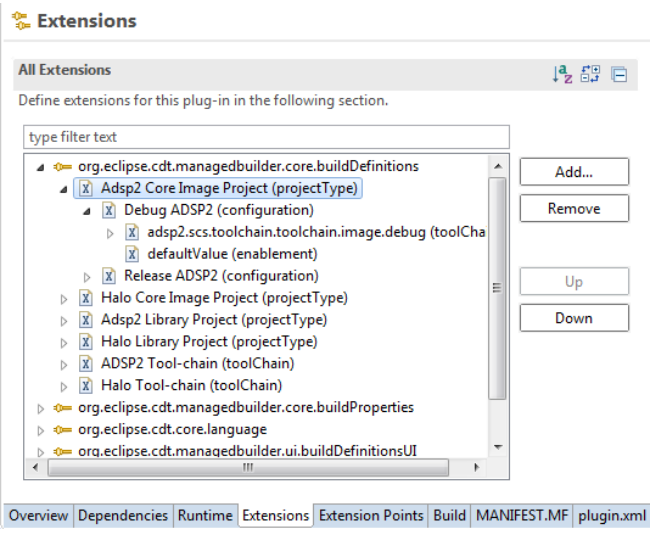


Slika 4.2 Primer zahtevanih utičnih proširenja koja su dodana u polje *Dependencies*

4.4 Tip projekta

Unutar proširenja definisan je tip projekta koji se gradi. Osim tipa projekta, definisani su i konfiguracija projekta, lanac alata, ciljna platforma i primeri alata.(Slika 4.3)

ID tipa novog projekta je *scs.toolchain.executable*. Ovaj *ID* se koristi prilikom kreiranja projekta da bi bilo poznato koji lanac alata će se koristiti za gradnju navedenog projekta.



Slika 4.3 Primer tipa projekta

4.4.1 Konfiguracija projekta

Za svaki definisani tip projekta, definišu se i *Debug* i *Release* konfiguracije (slika 4.3), odnosno konfiguracije za otklanjanje grešaka, konfiguracija za finalni proizvod i ostale. *Debug* konfiguraciji dodeljen je *scs.toolchain.configuration.image.debug ID,* a *Release* konfiguraciji *scs.toolchain.configuration.image.release ID.*

4.4.2 Pareseri grešaka

Unutar *CDT SDK-a* postoji nekoliko parsera grešaka, a to su:

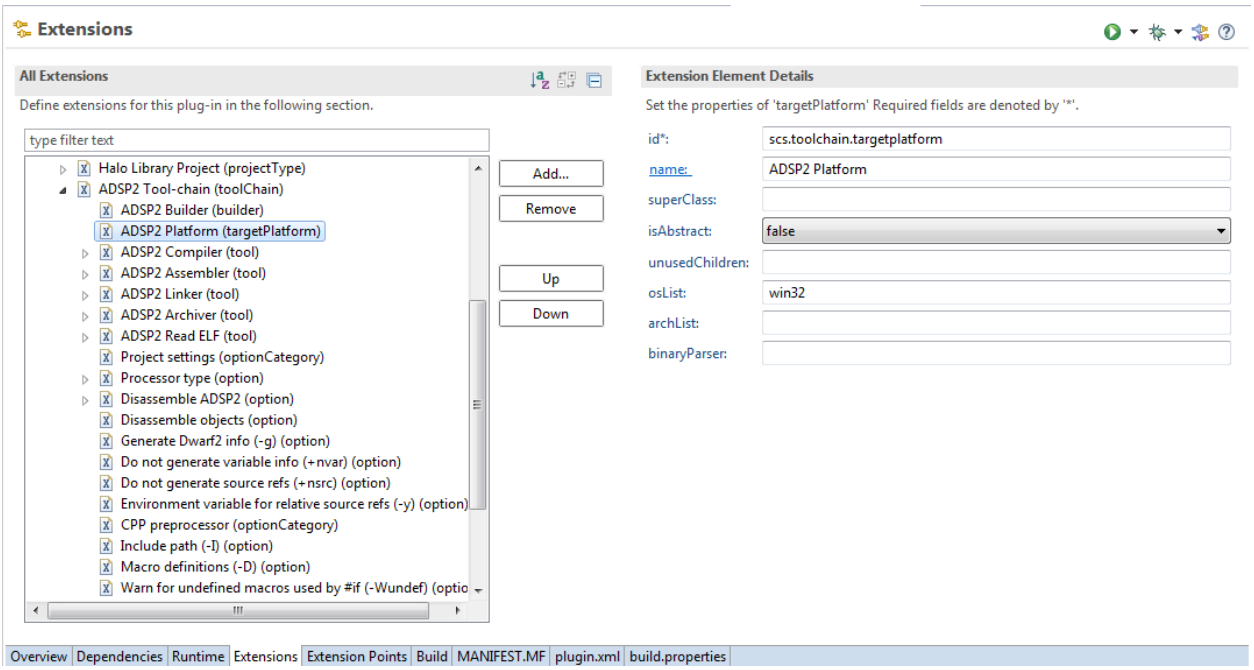
- *org.eclipse.cdt.core.MakeErrorParser*;
- *org.eclipse.cdt.core.GCCErrorParser*;
- *org.eclipse.cdt.core.GLDErrorParser*;
- *org.eclipse.cdt.core.GASErrorParser*;
- *org.eclipse.cdt.core.VCErrorParser*;

Parseri grešaka pomažu u otkrivanju i otklanjanju grešaka u kodu, tako što na izlazu ispisuju lokaciju i tip greške. Navedene datoteke su već definisane samim *SDK*-om, ali za svaki lanac alata prave se posebni parseri grešaka.

4.4.3 Lanac alata

Za svaki tip projekta postoji definisana konfiguracija, koja zahteva skup lanaca alata koji su integrisani u razvojno okruženje. *Debug* i *Release* konfiguracije projekta u ovom radu koriste *adsp2.toolchain.toolchain.debug* i *adsp2.toolchain.toolchain.release* podskupove lanaca alata. U ovim skupovima nalaze se osnovni alati koje navedene konfiguracije koriste za gradnju programskih rešenja.

Pored skupa lanaca alata, definisan je i operativni sistem na kome je vidljiv dati lanac alata, a u ovom slučaju to je *Windows* platforma. U podešavanjima konfiguracije, u polje *osList*, uneta je komanda *win32* za *Windows* operativni sistem. Pored *Windows* operativnog sistema, podržani su i drugi kao što su *Linux* i *Solaris*. Prilikom korištenja neke od *Linux* distribucija u polje *osList* unosi se vrednost *Linux*, a za *Solaris* distribucije *solaris, hpux ili aix* i za ostale operativne sisteme *other*.(Slika 4.4)



Slika 4.4 Operativni sistem za platformu na kojoj je vidljiv integrisani lanac alata

4.4.4 Graditelj projekta

Svaki lanac alata može da ima i podskup graditelja koji opisuju koji uslužni program je lanac alata koristio da napravi gradivne artifakte konfiguracije projekta. Osnovna podela graditelja u ovom slučaju je:

1. Integrisani *CDT* graditelj (engl. *CDT Internal builder*)
2. Spošljanji graditelj (engl. *External builder*) – Prilagođeni su korisničkim potrebama. Za potrebe ovog rada implementiran je prilagođeni graditelj (*adsp2.toolchain.toolchain.builder*) projekta.

4.4.5 Ciljna platforma

Ciljna platforma opisuje operativni sistem i arhitekturu platforme na kojoj će se izvršavati artifakti koje je napravio navedeni lanac alata. Za potrebe ovog rada *primitive.toolchain.targetPlatform* ciljna platforma povezana je sa tačkom proširenja lanca alata.

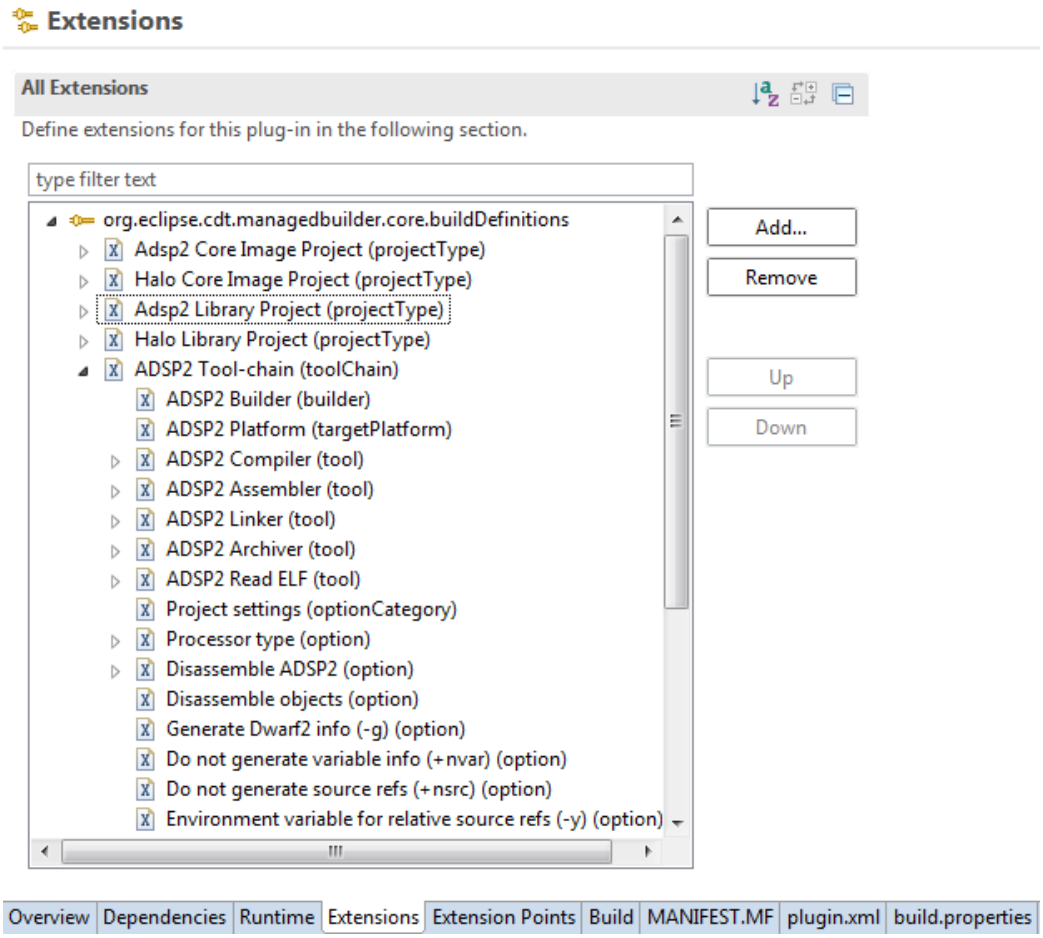
Ovde je slučaj da su platforma na kojoj se razvija razvojno okruženje koristi *Windows* operativni sistem, a ciljna platforma koristi *Linux* operativni sistem. Na osnovu ciljne platforme unosi se vrednost binarnih parsera, a u ovom slučaju vrednost binarnih parsera je *org.eclipse.cdt.core.ELF*, jer je to vrednost koja se unosi za *Linux* i *Solaris* binarne parsere. U

slučaju da je ciljna platforma koristi *Windows* operativni sistem, vrednost binarnog parsera bi bila *org.eclipse.cdt.core.PE* .

4.4.6 Alati

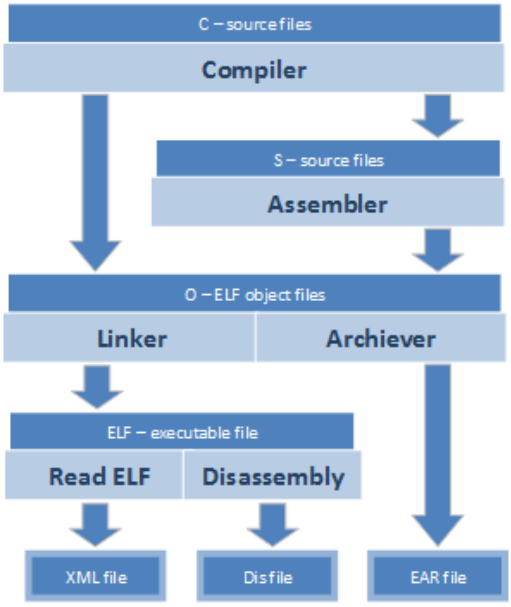
Svaki lanac alata (slika 4.5) opisuje skup alata koje graditelj koristi da bi napravio gradivni artifakt konfiguracije. Osnovni alati u skupu lanca alata su:

1. Programski prevodilac (engl. *Compiler*)
2. Asemblerski prevodilac (eng. *Assembler*)
3. Programski povezivač (engl. *Linker*)
4. Programski arhiver (engl. *Archiver*)
5. *Read ELF* alat
6. Programski demonter (engl. *Disassembly*)



Slika 4.5 Adsp2 lanac alata

Programski prevodilac je alat koji prevodi ulazne datoteke napisane pomoću visokog programskog jezika *C* u asemblerske izlazne datoteke ili u izlazne *ELF* objektne datoteke, zavisno od prosleđenih parametara. Kompajler je uglavnom prva karika u lancu alata, koja se poziva za gradnju, ukoliko je programsko rešenje pisano u *C* programskom jeziku Na slici ispod moguće je videti primer prevođenja programskog rešenja napisanog u *C* programskom jeziku. (Slika 4.6.)



Slika 4.6 Tok prevođenja izvornog koda.

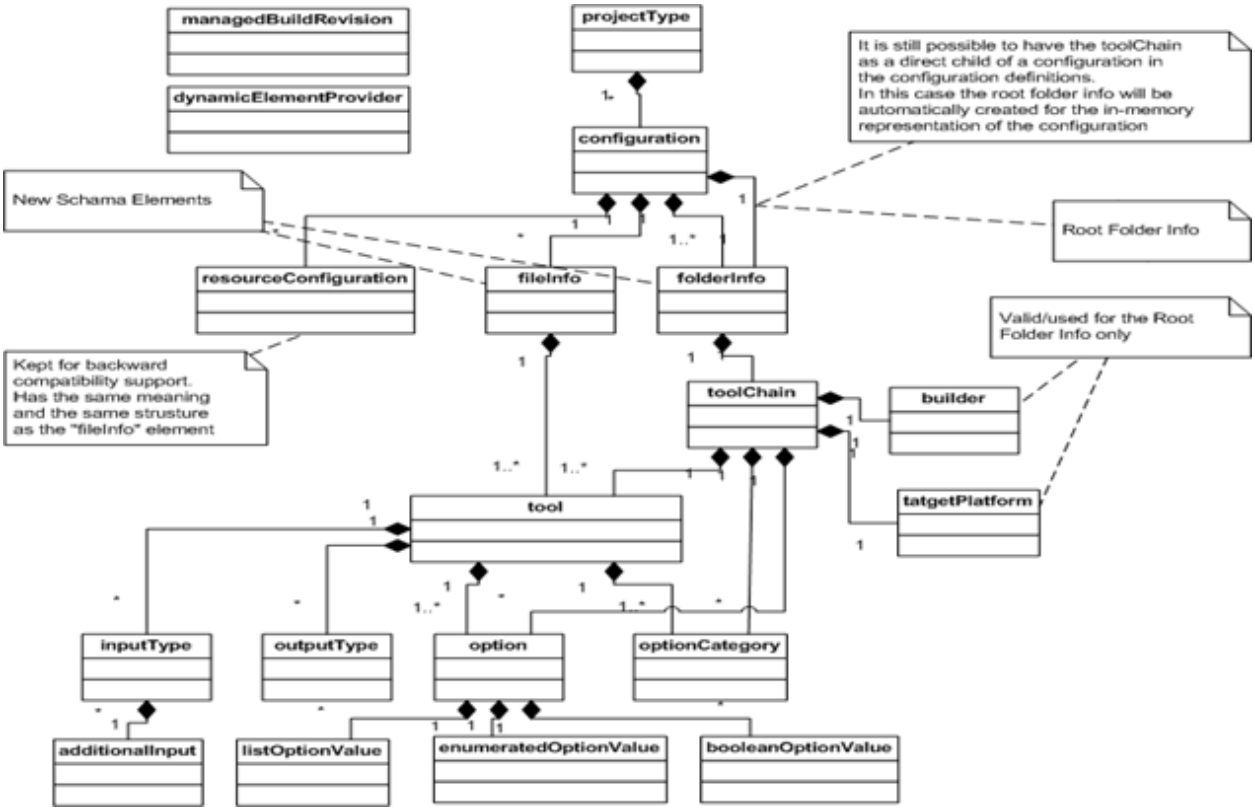
Sledeća karika u lancu alata je asemblerski prevodilac. Asemblerski prevodilac je alat koji prevodi datoteke aplikativnog programa ciljnog procesora napisane pomoću asemblerskog koda u *ELF* objektne datoteke. Datoteke napisane u asemblerskom kodu moraju biti *ASCII* tekstualne datoteke koje sadrže asemblerski kod koji je saglasan sa asemblerskom sintaksom za ciljani procesor.

Programski povezivač je program koji uzima jedan ili više objekata stvorenih uz pomoć programskih prevodilaca i sastavlja ih u izvršnu datoteku (engl. *ELF-executable file*). Ulazne datoteke programskog povezivača (*ELF* objektne datoteke) prate *ELF* standard, sa proširenom podrškom za različite memorijske zone. Izlazne datoteke programskog povezivača su *ELF* izvršne datoteke, koje mogu da se učitaju u program procesora ili memoriju za podatke. Određeni aspekti procesa povezivanja mogu se navesti kroz direktive u gradivnoj konfiguracionoj datoteci (engl. *BCF – Build Configuration File*).

Read ELF alat parsira i izvodi informacije za otklanjanje grešaka iz *ELF* izvršnih datoteka u *debugInfo.xml* datoteku, koju razvojno okruženje koristi u svrhu otklanjanja grešaka.

Programski demonter je alat koji izvodi informacije iz *ELF* izvršnih datoteka u *Dis* datoteke koje se koriste za prezentaciju *disassembly* pogleda u razvojnom okruženju.

Programski arhiver je alat koji kreira i upravlja *ELF* objektnim datotekama. Više objektnih datoteka može biti spojeno u jednu arhivsku datoteku (engl. *Archive file*). Individualne objektne datoteke su ubačene u arhivu bez konverzije. Programski povezivač može da koristi rezultujuće arhivske datoteke.



Slika 4.7 Šema gradivnih definicija modela

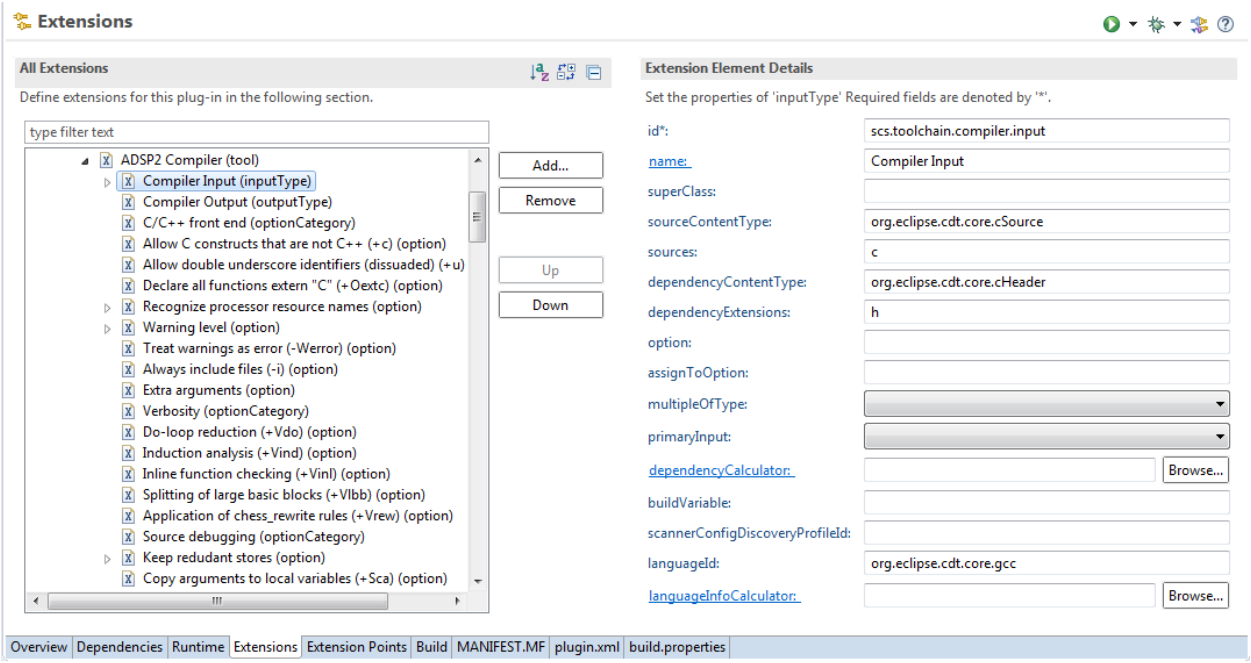
Opisani alati se koriste za gradnju projekata koji su zasnovani na *C/C++* programskom jeziku. Njihova svrha je gradnja funkcionalnih aplikacija za ciljane ugrađene sisteme (engl. *Embedded systems*).

Model upravljačkog gradivnog objekta (Slika 4.7) korišten je za definiciju lanaca alata, tipova projekata, konfiguracija i drugih postavki koje su bitne za *MBS*.

4.4.6.1 Ulazni i izlazni tipovi

Svaki alat opisuje tipove ulaza i izlaza u poljima *inputType* i *outputType*.(Slika 4.8) Na primer, za programski prevodilac dodaje se ulazni tip sa *ID*-em *scs.toolchain.compiler.input*.

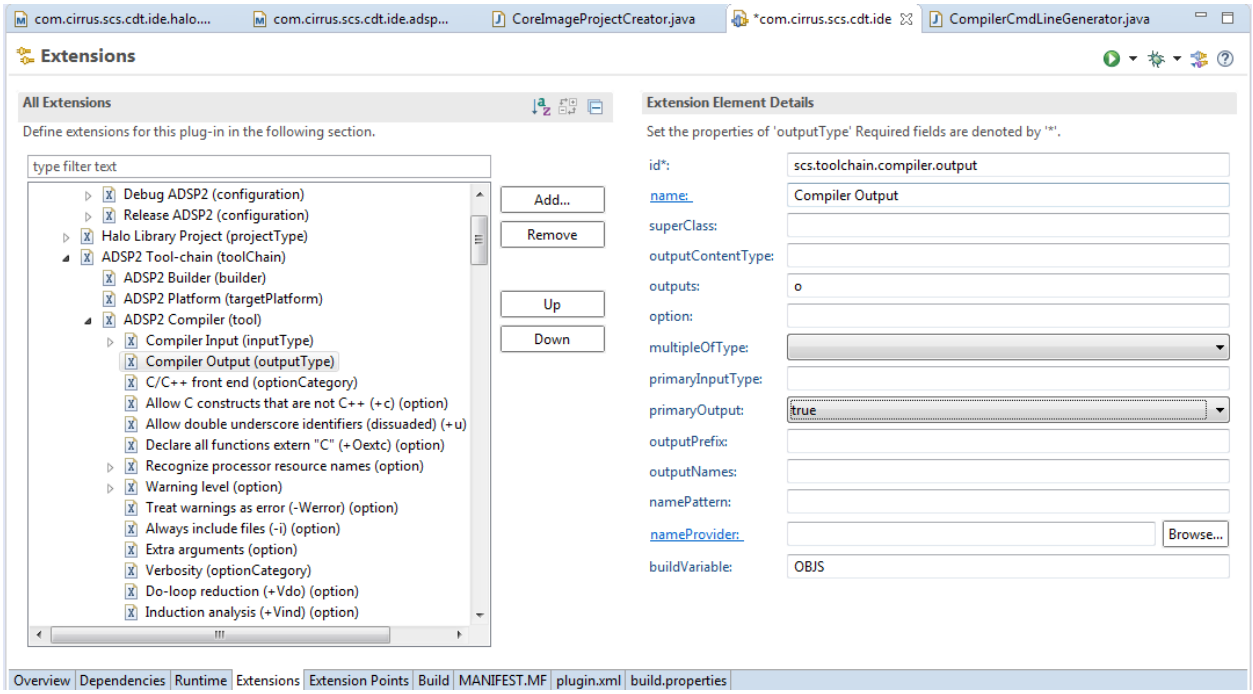
Navedeni programski prevodilac radi sa fajlovima napisanim u *C* programskom jeziku. U polju *sourceContentType* upisan je *ID* utičnog proširenja na kome su zasnovane *C* datoteke. Ako ne postoji neko posebno utično proširenje, moguće je koristiti već integrisano utično proširenje *org.eclipse.cdt.core.cSource*.



Slika 4.8 Ulazni i izlazni tipovi alata

Model za gradnju projekta mora da zna da li postoje neka posebna proširenja datoteka koja ukazuju na to da je datoteka zaglavlje (engl. *Header*). U te svrhe u polju *dependencyContentType* upisana je vrednost *org.eclipse.cdt.core.cHeader* i *primaryInput* polje je podešeno na vrednost *True*.

Izlaznom tipu (slika 4.9) programskog prevodioca dodeljen je *ID* *scs.toolchain.compiler.output*. Pretpostavka je da je izlaz programskog prevodioca moduo objekta koji ima proširenje *.o*. Iz tog razloga upisana je vrednost *o* u polje *outputs* i *primaryOutput* polje je podešeno na vrednost *True*. Kreirani moduo objekta koriste i drugi alati, kao što je na primer programski povezivač, te je stoga u *buildVariable* polju upisana vrednost *OBJS*. Preporučljivo je koristi istu vrednost u *buildVariable* polju u ostalim alatima koji koriste moduo objekta.



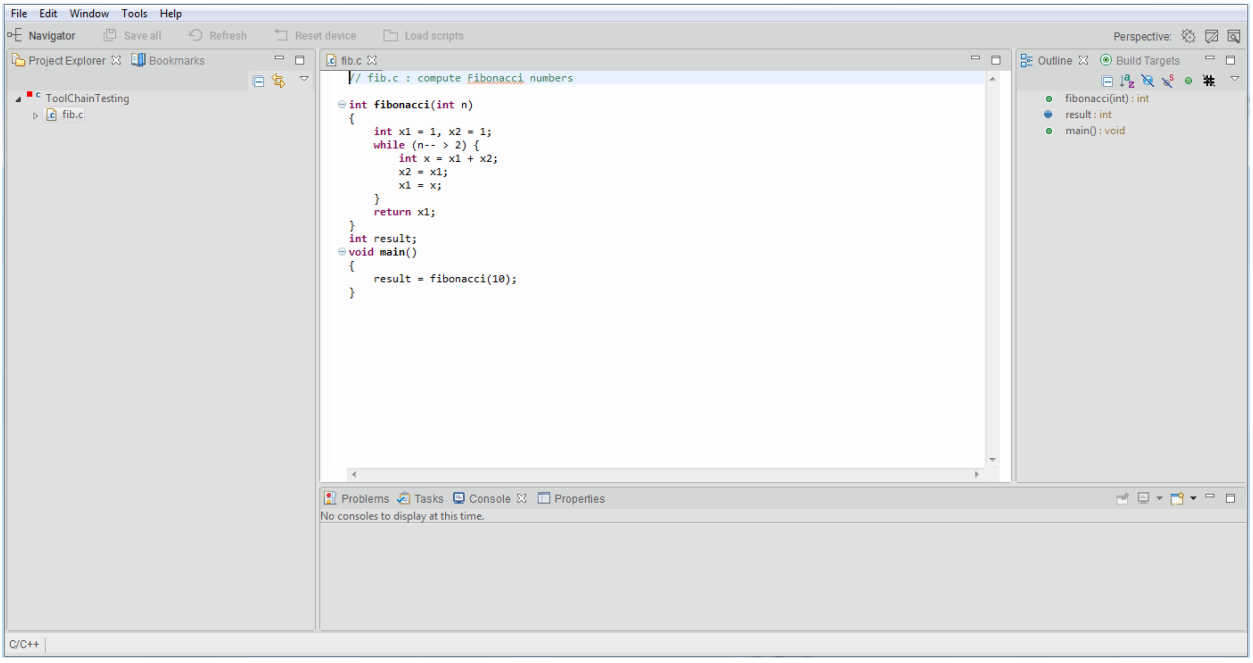
Slika 4.9 Izlazni tip programskog prevodioca

5. Rezultati

U ovom poglavlju opisan je proces verifikacije i validacije realizovanog rešenja. Postoji nekoliko ispitnih faza kroz koje realizovano rešenje treba da prođe da bi se utvrdilo da li zadovoljava zadate kriterijume.

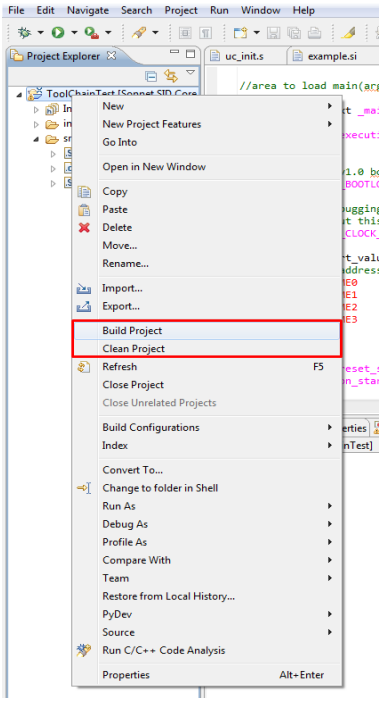
Prva faza ispitivanja

U *Eclipse* razvojnom okruženju kreiran je novi projekat sa *fib.c* datotekom. Navedena datoteka sadrži kod koji opisuje Fibonačijev niz i upravo je Fibonačijev niz poslužio u svrhu ispitivanja pravilne integracije lanaca alata.



Slika 5.1 Datoteka sa Fibonačijevim nizom

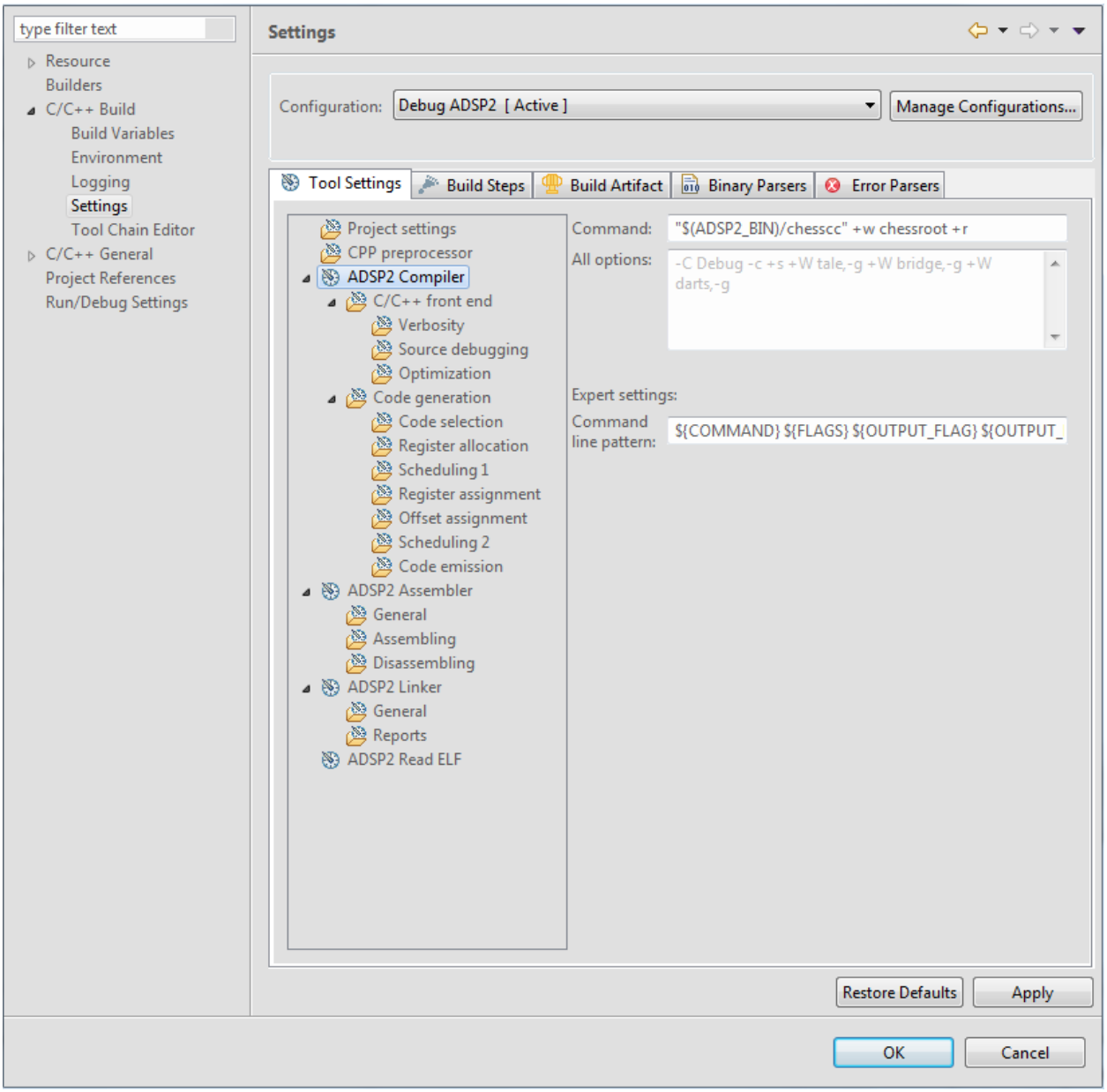
Kada je projekat kreiran, desni klik mišem na isti i proverava se da li postoji *Build* opcija, ako postoji, to znači da razvojno okruženje prepoznaje lanac alata i da je moguće graditi isti projekat. (Slika 5.2)



Slika 5.2 Verifikacija postojanja lanaca alata

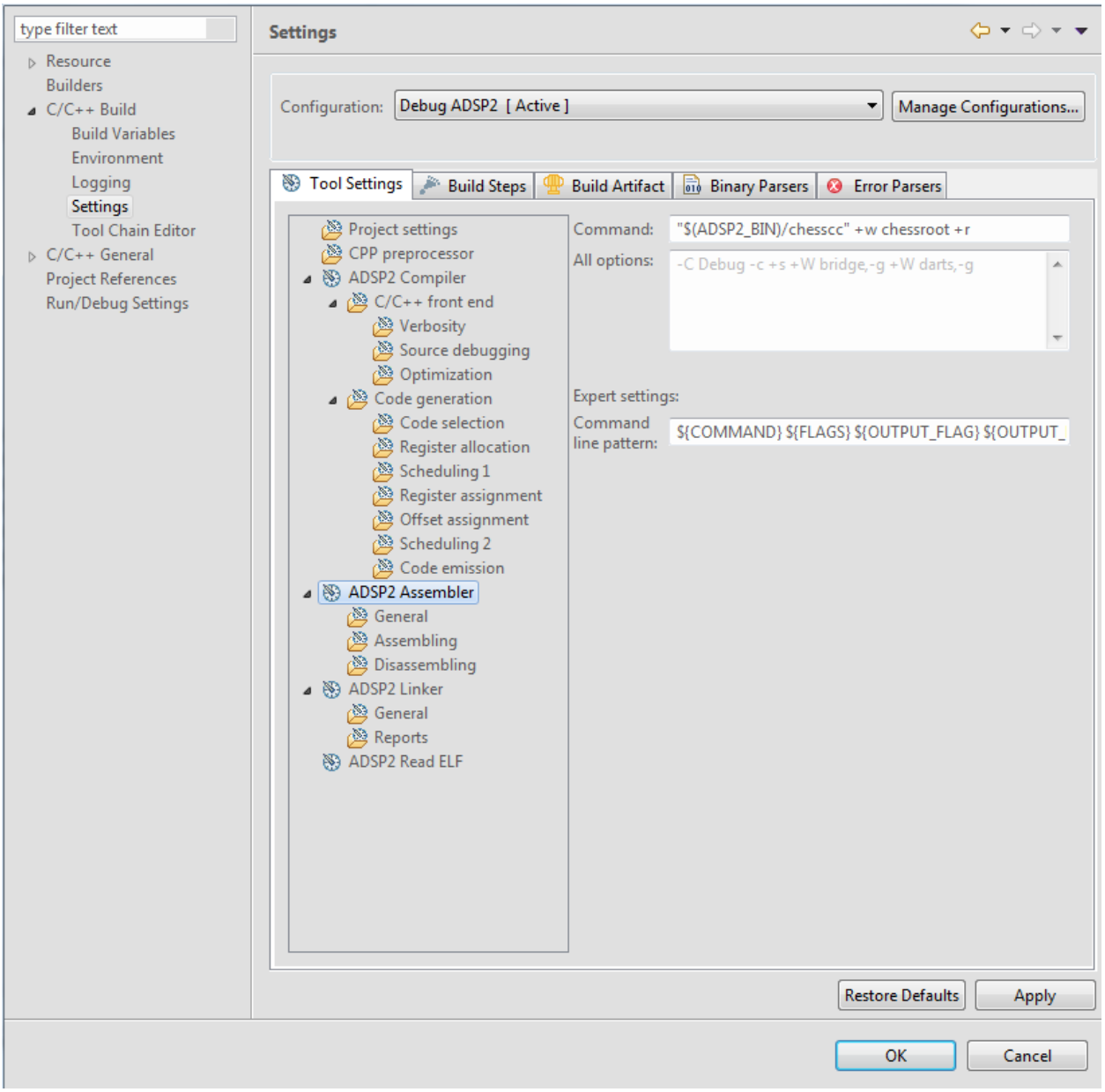
Druga faza ispitivanja

U ovom koraku provereno je koji se sve alati pozivaju prilikom gradnje projekta i to je učinjeno na sledeći način: desni klik na projekat koji se gradi i prati se sledeća putanja *Properties* > *C/C++ Build* > *Settings* i u *ToolSettings* polju moguće je videti koji se sve alati pozivaju. Osnovni alati koji su potrebni za gradnju projekta su programski prevodilac (slika 5.3), asemblerski prevodilac (slika 5.4), programski povezivač (slika 5.5) i *Read ELF* alat (slika 5.6). Na fotografijama ispod moguće je videti posebno svaki alat i njegove opcije. U polju *Command* moguće je videti binarnu promenljivu alata i komandu za pozivanje istog. Polje *All options* je polje u kome su vidljive sve prethodno definisane opcije alata. *Command line pattern* polje rezervisano je za skup definisanih promenljivih koje sadrže komande koje se prosleđuju komandnoj liniji za dati alat.



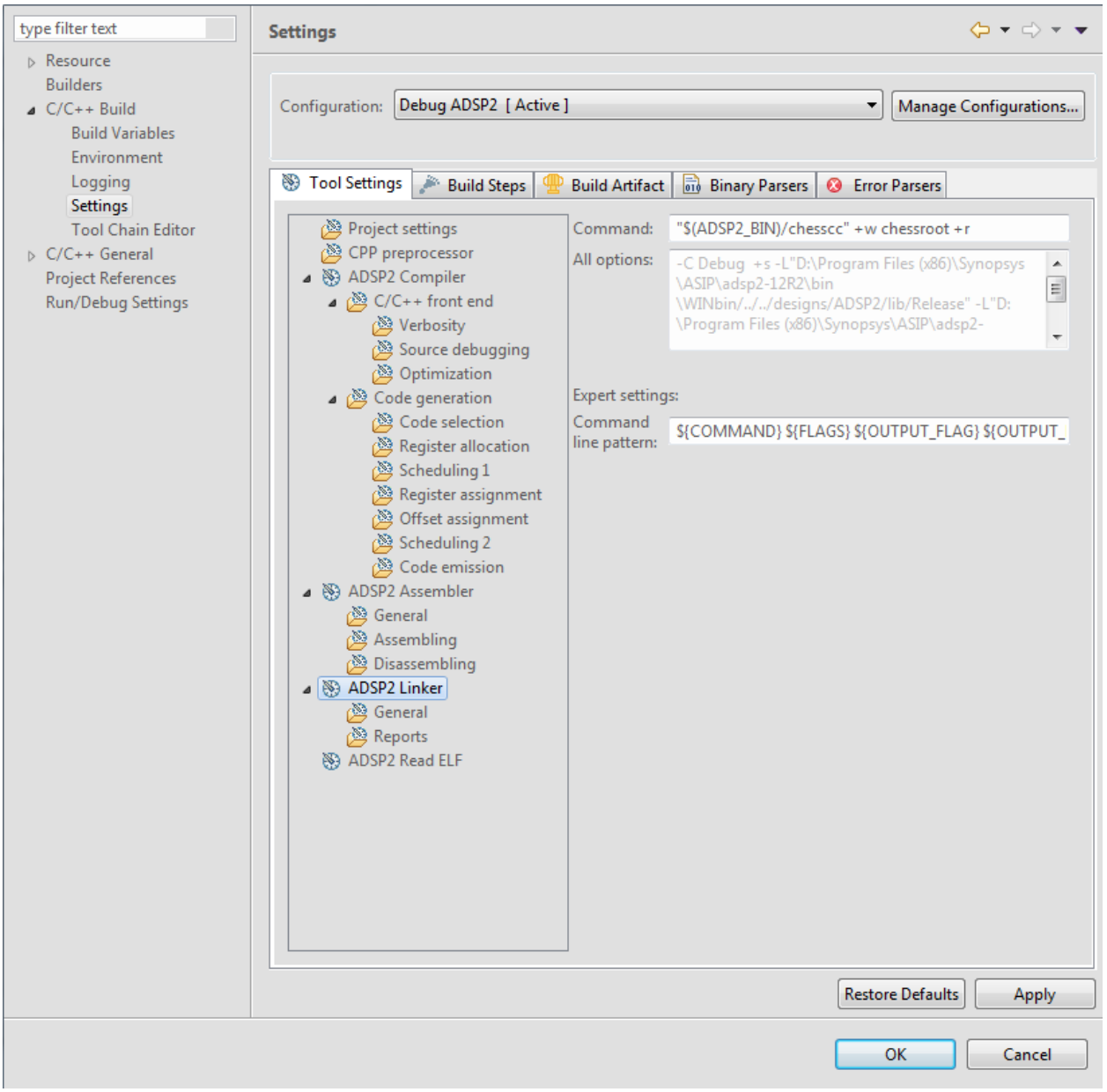
Slika 5.3 Programski prevodilac i njegove opcije

Na slici 5.3 moguće je videti primer programskog prevodioca i njegovih opcija. U polju *Command* nalazi se binarna promenljiva i komanda za pozivanje programskog prevodioca. Polje *All options* prikazuje sve definisane opcije navedenog programskog prevodioca.



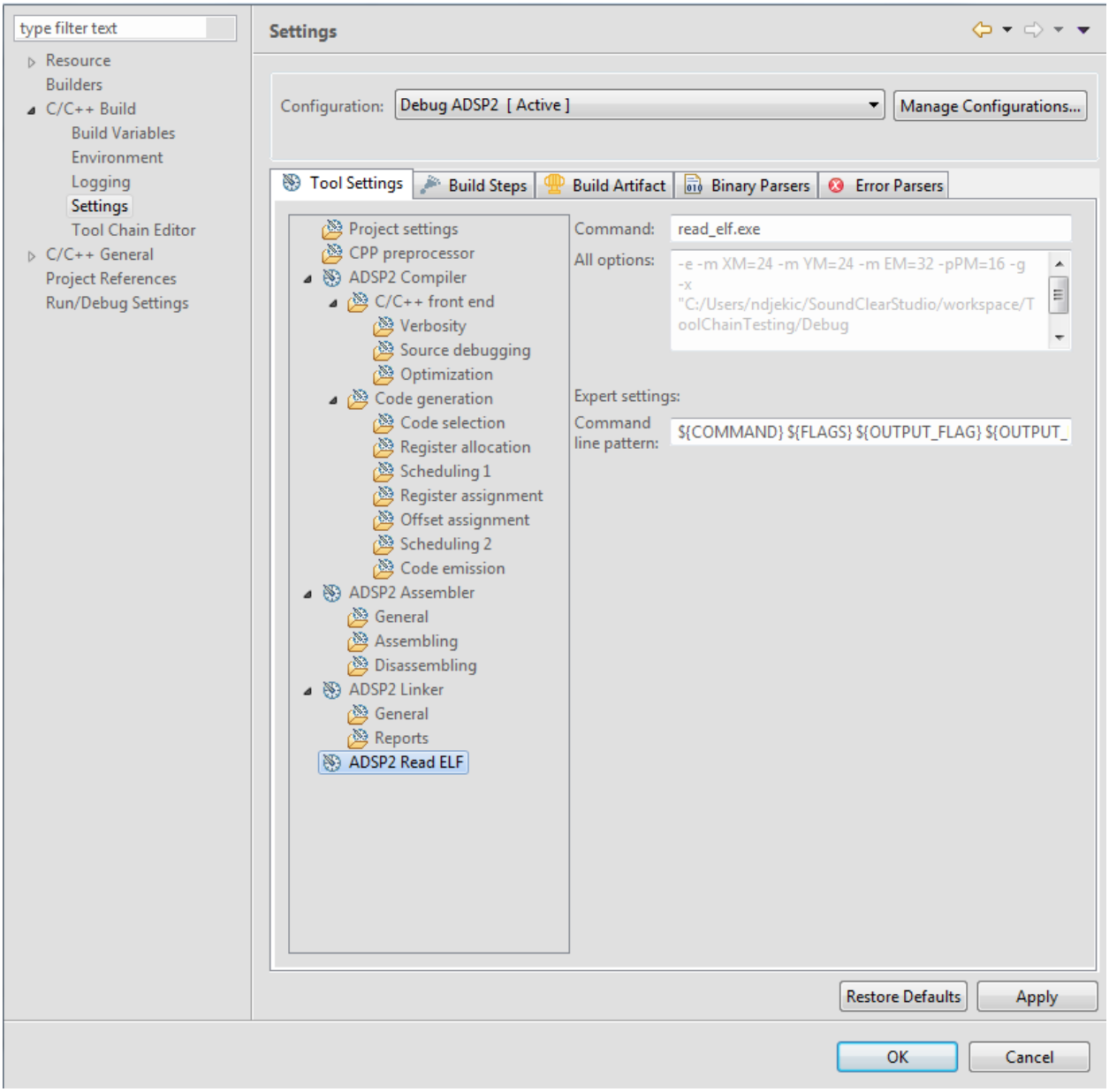
Slika 5.4 Asemblerski prevodilac i njegove opcije

Slika 5.4 prikazuje primer asemblerskog prevodioca i njegovih opcija. U polju *Command* nalazi se binarna promenljiva i komanda za pozivanje asemblerskog prevodioca. Polje *All options* prikazuje sve definisane opcije navedenog asemblerskog prevodioca.



Slika 5.5 Programski povezivač i njegove opcije

Slika 5.5 prikazuje primer programskog povezivača i njegovih opcija. U polju *Command* nalazi se binarna promenljiva i komanda za pozivanje programskog povezivača. Polje *All options* prikazuje sve definisane opcije navedenog programskog povezivača.



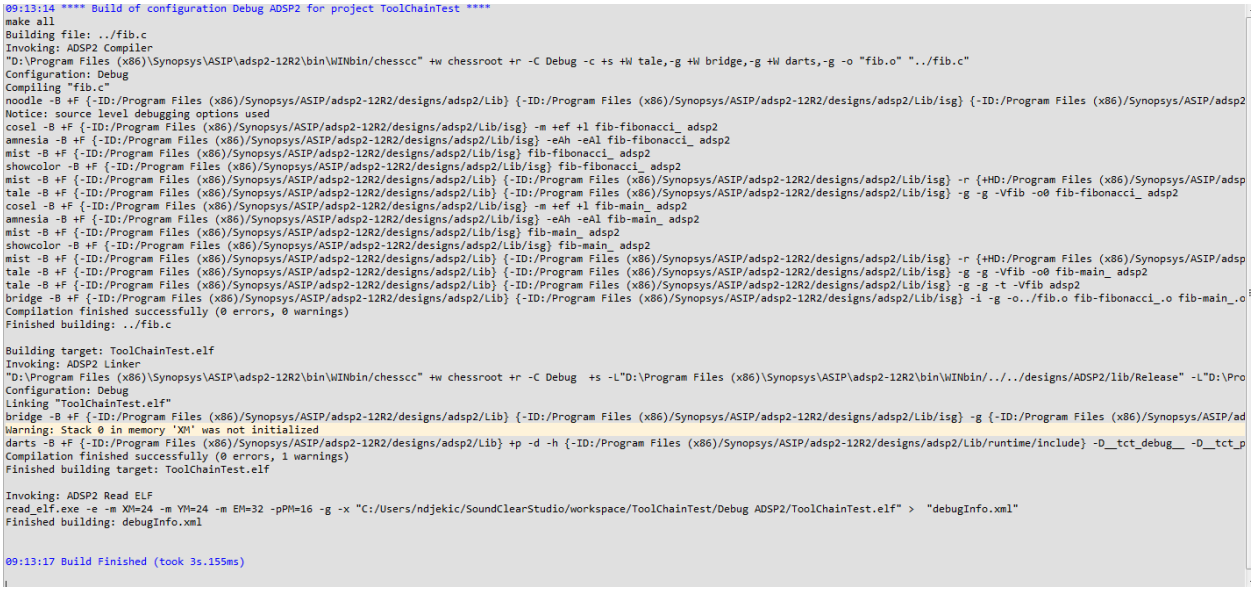
Slika 5.6 *Read ELF* alat i njegove opcije

Na slici 5.6 moguće je videti primer *Read ELF* alata i njegovih opcija. U polju *Command* nalazi se komanda za pozivanje *Read ELF* alata. Polje *All options* prikazuje sve definisane opcije *Read ELF* alata. *Command line pattern* polje pokazuje redosled pozivanja naredbi za pozivanje alata u komandnoj liniji.

Kada je utvrđeno da se željeni alati nalaze u podešavnjima i da su vidljive sve definisane opcije, prelazi se na sledeći korak ispitivanja opisanog resenja.

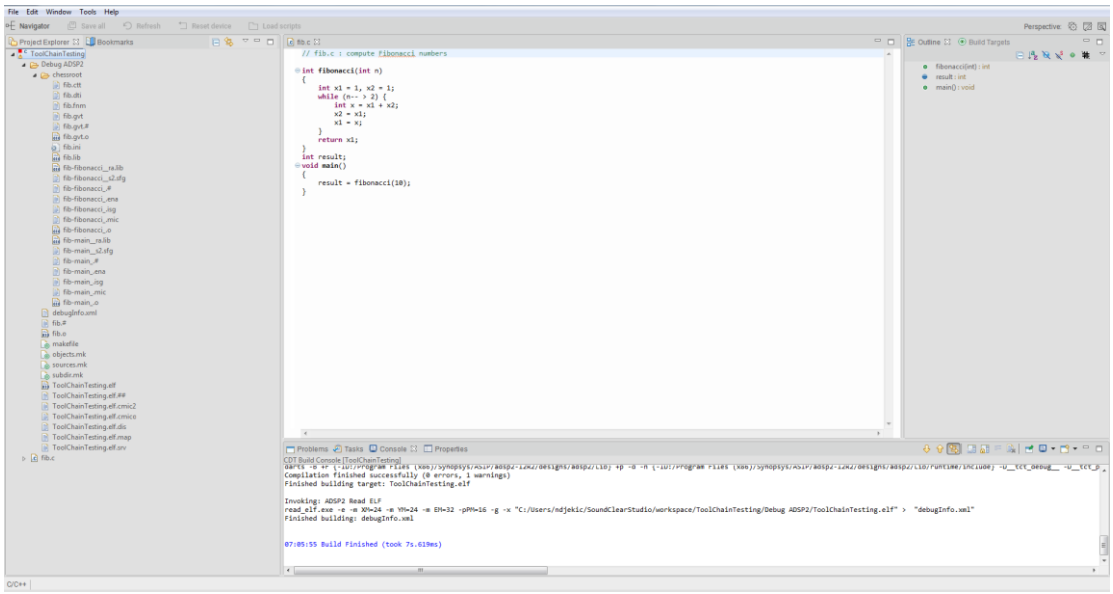
Treća faza ispitivanja

Kada je izvršena provera da li su svi potrebni alati integrisani, prelazi se na korak gradnje projekta. Kada se projekat izgradi, proverava se ispis, da li su pozvani svi potrebni alati i da li su na izlazu dobijene željene *ELF* izvršne datoteke, na osnovu ulaznih datoteka. (Slika 5.7)



Slika 5.7 Proces gradnje projekta

Kada je utvrđeno da su na izlazu dobijene željene *ELF* izvršne datoteke, koje mogu da se vide u fascikli *Debug* ADSP2 (slika 5.8), moguće je potvrditi da je lanac alata uspešno integrisan u *Eclipse* razvojno okruženje.



Slika 5.8 Verifikacija izlaznih datoteka

6. Zaključak

U okviru ovog rada integrisan je lanac alata, koji je prilagođen korisničkim potrebama, u *Eclipse* razvojno okruženje. Rad je baziran na *CDT Managed Build System* proširenju kroz koje je integrisan lanac alata. U *Eclipse RCP* razvojnom okruženju, razvijena su sledeća utična proširenja: *com.cirrus.scs.cdt.ide* (u ovom utičnom proširenju definisani su lanci alata i tipovi projekata sa svim pratećim opcijama), *com.cirrus.scs.cdt.adsp2.coreimage.model* (u ovom utičnom proširenju kreirani su projekti koji su realizovani za spomenute ciljne platforme i verifikaciju lanaca alata) i *com.cirrus.scs.cdt.adsp2.coreimage.ui* (u ovom proširenju definisan je korisnički interfejs prilikom za kreiranje ispitnog projekta). Navedena utična proširenja kreirana su u svrhu ispitivanja i verifikacije lanaca alata. Pomoću navedenih utičnih proširenja kreiran je projekat sa *C* gradivnom datotekom (programsko rešenje napisano u *C* programskom jeziku), koja je prevedena u *ELF* izvršnu datoteku za određeni tip *DSP*-a. Grafička sprega, koja pruža korisniku vizuelnu interakciju sa navedenim projektom, realizovana je uz pomoć *SWT* i *JFace* grafičkih biblioteka.

Verifikacija izvršena u nekoliko faza i utvrđeno je da je lanac alata uspešno integrisan u razvojno okruženje i da je pomoću istog moguće prevesti *C* gradivne datoteke i izvršne *ELF* datoteke za ciljnu DSP arhitekturu.

U nastavku, ukartko će biti navedene mogućnosti proširenja ovog rešenja.

S obzirom da je lanac alata integrisan u *Eclipse* razvojno okruženje, pored ovog jednog, moguće je integrisati i više lanaca alata, koji mogu da služe za različite projekte, tako da korisnik ne mora svaki put kada želi da razvija projekat za drugu ciljnu platformu, da traži novo razvojno okruženje. Dovoljno je da ima jedno razvojno okruženje, koje nudi podršku za više različitih ciljnih platformi.

7. Literatura

[1] V. Kovačević, M. Popović, M. Temerinac, N. Teslić: Arhitektura i algoritmi digitalnih signal procesora 1, 2005

[2] M. Temerinac, S. Berber, Ž. Lukač: *Osnovi algoritama i struktura DSP 1, 2013*

[3] Momčilo Krunić, Nenad Četić, Ivan Považan, Miroslav Popović: Custom Toolchain Integration into the Eclipse based IDE, 2016

[4] Internet adresa: http://www.computerhistory.org/revolution/birth-of-the-computer/4/78, učitana 26.9.2017.

[5] Jeff McAffer, Jean-Michel Lemieux,Chris Aniszczyuk: Eclipse Rich Client Platform second edition, 2010

[6] IBM. (2015, March) IBM. Internet adresa: http://www.ibm.com/us/en/

[7] Kathy Sierra, Bert Bates: Head First Java 2nd Edition, 2005

[8] Eclipse foundation. (2015, March) Eclipse Public License - v 1.0. Internet adresa: http://www.eclipse.org/org/documents/epl-v10.php

[9] OSGi. (2015, March) OSGi. Internet adresa: http://www.osgi.org/Technology/WhatIsOSGi

[10] Arhitektura Eclipse platforme, Internet adresa: https://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.papyrus.uml.doc%2Ftarget%2Fgenerated-eclipse-help%2FPapyrusStarterGuide.html, učitana 26.9.2017.

[11] Mikhail Sennikovsky: CDT Managed Build System, Eclipse Languages Symposium, October 2005

[12] CDT. (2015, March) Eclipse CDT (C/C++ Development Tooling). Internet adresa: https://eclipse.org/cdt/

[13] Internet adresa: https://www.gnu.org/software/make/, učitana 26.9.2017.

[14] Eclipse foundation (2016, June) Eclipse documentation – Eclipse Luna. Internet adresa: http://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.cdt.doc.isv%2Fguide%2Fmbs%2FextensibilityGuide%2FManaged_Build_Extensibility.html, učitana 26.9.2017.