Prevoditelj sistem-dinamičkih modela pohranjenih u xmile formatu

Oršulić, Matija

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:137:068315

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-31**

Repository / Repozitorij:

Digital Repository Juraj Dobrila University of Pula

SVEUČILIŠTE U PULI

Matija Oršulić

PREVODITELJ SISTEM-DINAMIČKIH MODELA POHRANJENIH U XMILE FORMATU

(Diplomski rad)

Pula, 2022.

SVEUČILIŠTE U PULI

Fakultet informatike u Puli

Diplomski sveučilišni studij Informatika

Prevoditelj sistem-dinamičkih modela pohranjenih u XMILE formatu

(Diplomski rad)

Mentor: doc. dr. sc. Siniša Sovilj Student: Matija Oršulić

MBS:2422000035/15

Pula, rujan 2022.

Obrazac popuniti TISKANIM slovima

Fakultet/odjel FAKULTET INFORMATIKE U PULI

SINIŠA SOVILJ
Ime i prezime mentora

SVEUČILIŠNI DIPLOMSKI STUDIJ
Studij

SUVREMENE TEHNIKE PROGRAMIRANJA
Kolegij

PRIJAVA TEME ZAVRŠNOG/DIPLOMSKOG RADA

Potvrđujem da sam prihvatio/la temu završnog/diplomskog rada pod naslovom:

PREVODITELJ SISTEM-DINAMIČKIH MODELA POHRANJENIH U XMILE FORMATU

(na hrvatskom jeziku)

TRANSPILER FOR SYSTEM DYNAMICS MODELS STORED IN XMILE FORMAT

(na engleskom jeziku)

(na talijanskom jeziku – samo za studente koji studiraju na talijanskom jeziku)

MATIJA ORŠULIĆ
Ime i prezime studenta/ice

JMBAG 0 2 4 1 0 3 4 2 7 6

Matični broj

Status (zaokružiti): (redoviti) izvanredni

Datum: 2/3/2021

Potpis mentora: Siniša Sovilj

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani MATIJA ORŠULIĆ, kandidat za magistra INFORMATIKE ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljeni način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

Oršulić

U Puli, 28.09.2022.

Sažetak

Ovaj rad se bavi izradom transpilera koji prevodi XMILE format u Kotlin i obratno. Transpiler će se koristiti unutar Kotlin System Dynamics Toolkit-a (ksdtoolkit) razvijenog na Fakultetu informatike u Puli. Opisana je povijest sistem-dinamike i značajne ličnosti koje su se bavile sistem-dinamikom kao što je i njem idejni začetnik Jay Forrester. U radu je obrađen i pojam XMILE formata te se detaljno opisuju oznake i atributi koji su potrebni za prevođenje istog. Pojašnjen je i sam ksdtoolkit i koje su razlike između njega i XMILE formata u smislu kostruiranja samih sistem-dinamičkih modela. Opisane su i razlike između samog traspilera, interpretera i kompajlera kako bi se još više pojasnilo što je to ustvari transpiler. Obrađeni su razni koncepti prevođenja kao što su lexer, parser i gramatika. Rad opisuje i sistem dinamiku, sistem-dinamičke modele te objašnjava kako su se koristile određene tehnologije, kao što su naprimjer programski jezik Kotlin, jedinični (eng. *unit*) testovi, integracijski testovi.

Ključne riječi u radu: Kotlin, XMILE format, sistem dinamika, sistem-dinamički modeli, unit testovi, integracijski testovi, transpiler, lexer, parser.

Sadržaj:

1. Uvod 1
2. Sistem dinamika 3
2.1 Povijest sistem dinamike 3
2.2 Definicija sistem dinamike i sistem-dinamičkog modela 7
3. Transpiler, lexer, parser, tokeni 12
3.1 Transpiler 12
3.2 Lexer (tokenizer) 14
3.3 Parser i gramatika programskih jezika 15
4. XMILE format 18
4.1 Definicija i prednosti XMILE formata 18
4.2 Oznake XMILE formata 19
4.2.1 Oznaka header 21
4.2.2 Oznaka sim_specs 22
4.2.3 Oznaka model_units 22
4.2.4 Oznaka dimensions 23
4.2.5 Oznaka behavior 24
4.2.6 Oznaka data 25
4.2.7 Oznaka model 25
4.2.8 Oznaka variables 27
4.2.9 Oznaka array 28
4.2.10 Oznaka stock 28
4.2.11 Oznaka flow 30
4.2.12 Oznaka aux 31
4.2.13 Oznaka group 32
4.2.14 Oznaka module 33
4.2.15 Oznaka macro 35
4.2.16 Oznaka options 36
4.3. Usklađenost implementacije XMILE formata 36
5. Kotlin System Dynamics Toolkit (ksdtoolkit) 38
5.1. Razlozi kreiranja ksdtoolkit-a 38

5.2. Struktura ksdtoolkit-a 39
5.3. Funkcionalnosti i moguća poboljšanja ksdtoolkit-a 40
5.4. Implementacija transpilera u ksdtoolkit 40
6. Izrada transpilera za ksdtoolkit 41
6.1. Kreiranje funkcija za manipulaciju stringovima i dohvat podataka ... 42
6.2. Provjera podataka 47
6.3. Priprema podataka 48
6.4. Postavljanje podataka 50
6.5. Ispis podataka 51
6.6. Base-Level Conformance 53
7. Testiranje 54
7.1. Jedinični (eng. *Unit*) testovi 54
7.2. Integracijski testovi 55
8. Zaključak 56
Literatura 57
Popis slika 58

1. Uvod

Rad se bavi izradom prevoditelja (eng. *transpilera*); od nadalje prevoditelj; iz XMILE formata u Kotlin i obratno. Prevoditelj se implementira u postojeći toolkit: Kotlin System Dynamics Toolkit (ksdtoolkit) koji je izrađen na Fakultetu informatike u Puli (FIPU). Razlog odabira ove teme je zainteresiranost za lexer i parser koji se koriste prilikom izrade programskih jezika i služe za kompajliranje i interpretiranje teksta. Također, kod odabira ovog rada bila je presudna i sama zanimljivost sistem dinamike i sistem-dinamičkih modela; u daljnjem tekstu SD modeli. Ovom temom povezane su društvene i prirodne znanosti. Pomoću sistem dinamike i SD modela mogu se predviđati društvene pojave. Cilj prevoditelja, a i toolkita je olakšati korištenje SD modela kako bi se napredovalo u kompleksnosti i raznovrsnosti istih. SD modeli koriste se u brojnim područjima društvenih znanosti kao što su medicina, sociologija, promet, ekonomija, ekologija, management i mnoga druga područja. Drugo poglavlje opisuje cjelokupnu povijest sistem dinamike, pojašnjava što su to sistem dinamika i SD modeli. Opisuje i grafičke prikaze koji se koriste u SD modelima. Na kraju tog poglavlja je primjer koji pojašnjava kako funkcioniraju sistem dinamika i SD modeli. Treće poglavlje se bavi transpilerom, lexerom, parserom, tokenima i brojnim drugim pojmovima vezanim uz kreiranje kompajlera, interpretera i transpilera. To poglavlje pobliže definirana sve gore prethodno navedene pojmove. Objašnjena je i razlika između transpilera, kompajlera i interpretera. U tom poglavlju se spominju i razine apstrakcije programskih jezika koje olakšavaju definiranje ovih pojmova. Četvrto poglavlje bavi se XMILE formatom. U njemu je definiran pojam XMILE formata te kako je on nastao i zašto je bio potreban takav format koji će standardizirati modele kreirane sistem dinamikom. To poglavlje bavi se i glavnim oznakama XMILE formata koje imaju svoja pravila i od kojih su neka obavezna da bi XMILE format bio kompatibilan na svim platformama i u svim programima i alatima koji koriste SD modele. Peto poglavlje pobliže objašnjava ksdtoolkit i njegova svojstva i razlog kreiranja ksdtoolkita. U šestom poglavlju glavna tematika je izrada programskog rješenja prevoditelja. Govori se o pet faza prijevod iz XMILE formata u ksdtoolkit.

Pojašnjeno je i značenje Base-Level Conformancea. Sedmo poglavlje bavi se testiranjem koda odnosno jediničnim i integracijskim testovima koji se provode nad isprogramiranim funkcijama prevoditelja.

2. Sistem dinamika

U ovom poglavlju će biti obrađena povijest sistem dinamike i najvažniji trenutci koji su sistem dinamiku doveli do stadija u kojem se nalazi danas. Objasnit će se i kako je sistem dinamika postala popularna u brojnim društvenim sferama i koje su ključne osobe za njen uspjeh i razvoj. Nakon toga ovo poglavlje bazirat će se na definiranje samih pojmova sistem dinamike i SD modela. Oni su važni jer će olakšati razumijevanje XMILE formata koji je ključan za uspješno kreiranje prevoditelja iz XMILE formata u Kotlin model ksdtoolkit-a i obratno. Bit će objašnjeno što je to koncept sustavnog razmišljanja s povratnim vezama. Pojasnit će se od kojih komponenti se sastoji dijagram kauzalnih (uzročnih) veza te dijagram toka koji se također koristi u modeliranju SD modela.

2.1 Povijest sistem dinamike

Sistem dinamiku osmislio je Jay Forrester profesor s MIT-a (eng. *Massachusetts Institute of Technology*) sredinom 50-ih godina prošlog stoljeća. Forrester je radio kao predavač na MIT Sloan School of Management. To je privatni odsjek MIT-a koji se bavi biznisom, managementom i ekonomijom. Na ideju sistem dinamike je došao jer je želio iskoristiti svoja znanstvena i inženjerska znanja i primijeniti ih na području managementa i ekonomije. Sistem dinamika se prvi put koristi u suradnji managera GE-a (eng. *General Electric*) i Forrestera. Manageri su zamolili Forrestera da im pomogne riješiti problem vezan uz zapošljavanje u cikličkim periodima od tri godine unutar kojih bi nekad zaposlenici radili u tri smjene da ispune uvjete, a nekad bi morali otpuštati radnike zbog premalo posla. Forester je prikupio podatke i pomoću SD modela te simulacija vezanih uz poslovanje GE-a uspio utvrditi uzrok tog problema. Ustanovio je da se nestabilnost u zapošljavanju GE-a događa zbog unutrašnje strukture poduzeća i odnosi se na njihovo poslovanje i raspodjelu sredstava i zaliha u poduzeću, a ne zbog utjecaja okoline na poduzeće odnosno da je zbog njihovih unutarnjih odluka došlo do potrebe za

otpuštanjem radnika. Na taj događaj se danas gleda kao na začetak sistem dinamike. (Forrester, 1990.)

1958. godine Forrester je s programerom Richardom Bennettom kreirao SIMPLE (eng. *Simulation of Industrial Management Problems with Lots of Equations*) kompajler koji je Forresteru bio potreban za simulaciju koju je radio za članak u časopisu Harvard Business Review. Kreiranjem SIMPLE kompajlera omogućilo se puno brže širenje sistem dinamike i SD modela. Ubrzo nakon kreacije SIMPLE kompajlera Alexander Pugh III je kreirao još napredniji i proširenij kompajler kojeg naziva DYNAMO kompajler. Naziv je dobio spajanjem prva četiri slova engleske riječi „dynamics" i prva dva slova engleske riječi „model". Novi kompajler je omogućio još brže širenje SD modela prvobitno u korporativne svrhe, a zatim i u nekim novim granama kao što je naprimjer urbanizam. (Forrester, 1990.)

Sljedeći veći događaj u povijesti sistem dinamike događa se 1968. godine. Može se reći da je to bio niz događaja koji su bili vezani uz urbanizam i urbanističke probleme u velikim gradovima. Prvi događaj koji je bio važan za razvoj sistem dinamike u smjeru urbanizma je Forresterovo upoznavanje s novim profesorom na MIT-u, Johnom F. Collinsom. Collins je bivši gradonačelnik Bostona koji je na toj poziciji odradio dva mandata. U razgovoru s Collinsom Forrester je zaključio da je problem s urbanizmom vrlo sličan problemima na kojima je radio u managementu i ekonomiji. Njih dvojica sastavili su tim stručnjaka koji je s njima krenuo razvijati SD modele i raditi na simulacijama koje bi ukazale na probleme urbanizma tog vremena. Na osnovi tih istraživanja Forrester je 1969. godine objavio knjigu Urban Dynamics. Došao je do zaključka da je problem većine velikih gradova u tome što rade nisko budžetni smještaj kojeg ekonomija tih gradova nije mogla održavati odnosno u gradovima nije bilo dovoljno kapitala i radnih mjesta da održe veliki broj jeftinog smještaja. 1969. godine je to bio revolucionarni pogled na urbanizam gradova. (Forrester, 1990.)

Drugi važan događaj za sistem dinamiku tih godina je osnivanje organizacije The Club of Rome. The Club of Rome je organizacija čiji je cilj rješavanje problema koji prijete Zemlji i čovječanstvu. Organizacija se sastoji od brojnih znanstvenika, političara, ekonomista, poduzetnika i mnogih drugih koji imaju znanja za rad na tako važnim problemima društva. Ovaj događaj je važan jer je Aurelio Peccei jedan od osnivača te organizacije pozvao Forrestera na jedno od njihovih okupljanja u Bernu na kojem se raspravljalo o globalnim problemima. Taj događaj smatra se temeljem i motivom za knjigu World Dynamics koju je Forrester napisao 1971. godine. Treba napomenuti da je tamo začeta ideja o jednom od najvažnijih istraživanja vezanim za ekologiju, a to je problem održivosti ljudske vrste na Zemlji. (ClubOfRome, 2021.)

Bila je to knjiga Limits to Growth iz 1972. godine u kojoj su opisana istraživanja provedena na MIT-u. Napisali su ju: Donella Meadows, Dennis Meadows, Jørgen Randers i William W. Behrens III. To istraživanje je i danas vrlo aktualno jer se odnosi na period koji je nama u vrlo bliskoj budućnosti. Predviđa drastičan pad populacije u doglednim godinama. Kao razloge tog pada navode se prenapučenost zemlje i nedostatak hrane. Knjige World Dynamics i Limits of Growth pokazale su da sistem dinamika može biti korištena u raznim područjima i da ima velikih potencijala u rješavanju raznih problema u kojem se suvremeno društvo nalazi, ali i budućih problema koji tek dolaze. (Forrester, 1990.)

Kada se priča o povijesti sistem dinamike važno je spomenuti i prvi grafički programski jezik STELL-u (eng. *Systems Thinking, Experimental Learning Laboratory with Animation*). Njega je osmislio Barry Richmond u svojoj kompaniji 1985. godine. On je zaslužan i za prvu pojavu sistem dinamike u školama jer je STELL-u posudio profesoru biologije Franku Draperu, koji je program implementirao u svoja predavanja o biologiji te tako otkrio prednosti sistem dinamike u predavanju i načinu na koji djeca usvajaju gradivo uporabom STELL-e i sistem dinamike. (Forrester, 1990.)

Slika 1: Jay W. Forrester

Izvor: ETHW, 2021.

Veliku ulogu u povijesti sistem dinamike ima Jay W. Forresster. On je idejni začetnik, ali i protagonist koji se „proteže" cijelom povijesti sistem dinamike. Zato je ovaj zadnji ulomak ovog potpoglavlja posvećen njemu. Forrester je rođen 1918. godine u Anselmo-u, a tijekom svoje karijere dobio je brojne nagrade i bio počasni član brojnih znanstvenih društava. Neka od tih društava su IEEE, Američka Akademija Znanosti i Umjetnost (eng. *American Academy of Arts and Sciences*), Akademije menadžmenta (eng. *Academy of Management*), Nacionalna akademija inženjerstva (eng. *National Academy of Engineering*). Dobio je i tri počasna doktorata na Sveučilištu Nebraska (1954.), Bostonskom Sveučilištu (1969.) te na Fakultetu inženjerstva u Newarku. Dobitnik je i nagrade za Izumitelja godine od Sveučilišta George Washington (1968.) i Zlatne medalje Valdemara Poulsena od Danske akademije tehničkih znanosti (1969.). 1972. godine dobiva jednu od najprestižnijih nagrada u svojoj karijeri Medail of Honor koju dodjeljuje IEEE (eng. *Institute Of Electrical and Electronics Engineers*) za postignuća u digitalnom računarstvu. Preminuo je 2016. s navršenih 98 godina. (ETHW, 2021.)

2.2 Definicija sistem dinamike i sistem-dinamičkog modela

Sistemska dinamika je metodologija koja pomoću matematičkog modeliranja nastoji pronaći optimalno rješenje za neki postavljeni problem. Problemi i sustavi kojima se bavi izuzetno su kompleksni i veliki te se oni najčešće javljaju unutar velikih organizacija ili na globalnoj razini. (Pacandi, 2018.)

Kako bi bolje upoznali sistem dinamiku važno je razumjeti neke koncepte koji se koriste u sistem dinamici, a to su sustavno razmišljanje s povratnim vezama te konceptualni modeli dijagrama kauzalnih (uzročnih) veza i dijagram toka. SD modeli se mogu prikazati pomoću oba dijagrama, katkad se rade i modeli koji imaju grafičke komponente oba dijagrama .

Sustavno razmišljanje s povratnim vezama sastoji se od dva aspekta. Prvi aspekt je sustavno razmišljanje. Ono se odnosi na shvaćanje sustava kao skupa podsustava i nadsustava. Sustav treba promatrati holistički u cjelini, ali i dalje biti svjestan da se on sastoji od manjih cjelina odnosno podsustava. Cilj takvog razmišljanja je podijeliti problem na manje potkomponente. Takva metodologija podjele na manje komponente se u programiranju naziva podjeli pa vladaj (divide and conquer), a služi tome kako bi se smanjila kompleksnost problema i lakše ga se savladalo podjelom na manje dijelove. Tako dolazimo do preciznijeg globalnog sustava koji se sastoji od većeg broja manjih potkomponenti odnosno podsustava. Takvo razmišljanje je bitno za sistem dinamiku jer se ona bavi jako kompleksnim sustavima. Neke od tih sustava smo i naveli u prethodnom poglavlju koje se odnosio na povijest sistem dinamike. Tamo možemo zamijetiti sistem dinamička istraživanja koja se bave strukturom poduzeća, urbanizmom te na kraju krajeva i populacijom cijelog svijeta. Ti sustavi su toliko kompleksni da su bile potrebne knjige kako bi ih se detaljnije opisalo. Što je sustav detaljnije objašnjen to će i sam model biti detaljniji, a onda i bolji ako je model napravljen kvalitetno. Zaključujemo da je jako važno razložiti te sustave na manje podsustave kako bi se pojednostavio primarni problem i bolje uvidjeli detalji sustava.

Drugi aspekt su povratne veze ili povratne petlje kako ih se također naziva. One su važne jer se u sistem dinamici svaki sustav promatra kao ciklički sustav. Razlog tome je što se sistem dinamičko modeliranje bazira na matematici i njegova rješenja su izražena brojevima. Na većinu promatranih varijabli mogu utjecati brojni aspekti, a neke od tih varijabli su naprimjer: populacija, svota novaca na računu ili broj nezaposlenih i naravno svaka od tih varijabli se promatra u određenom periodu vremena. Povratnu vezu možemo prikazati na primjeru populacije i broja novorođenih. Što je populacija veća to utječe na povećanje broja novorođenih samim time povećanje broja novorođenih utječe na povećanje populacije. To je primjer pozitivne povratne veze. Možemo na sličnom primjeru prikazati i primjer negativne povratne veze. U ovom primjeru ćemo ponovno koristiti populaciju, ali umjesto broja rođenih ćemo koristiti broj preminulih. Što je populacija veća to je i broj preminulih veći što je broj preminulih veći to je populacija manja samim time ova veza negativno utječe na populaciju pa ju nazivamo negativna povratna veza. Ovime smo definirali cjelokupni pojam sustavnog razmišljanja s povratnim vezama.

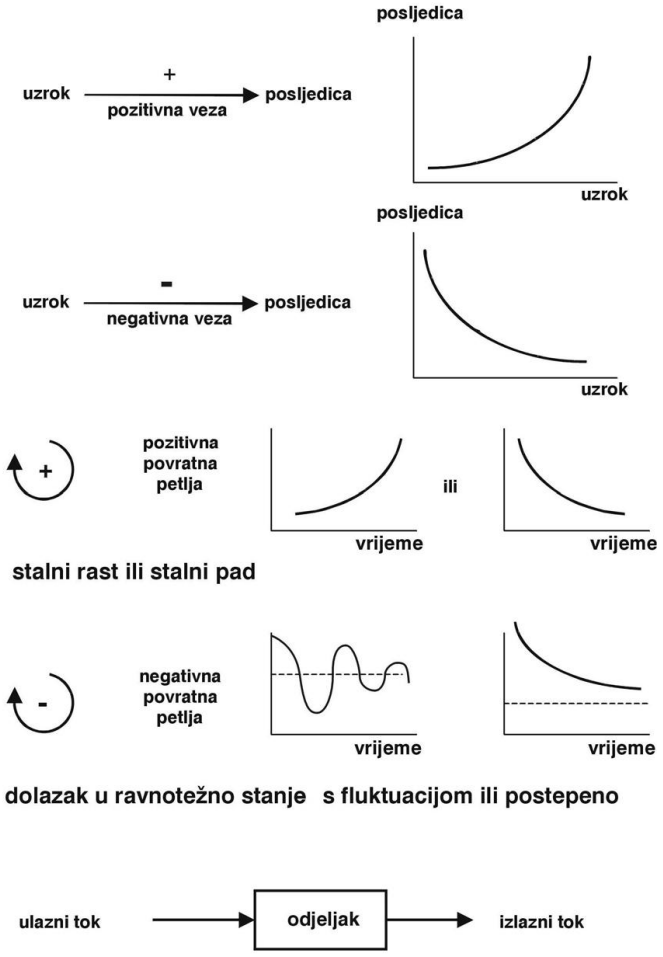
U modelima sistem dinamike agregiraju se entiteti i događaji u odjeljke i tokove kako bi se simuliralo ponašanje sustava sa povratnom vezom (eng. *feedback loop*) za koje se pretpostavlja da su deterministički po svojoj naravi iako uključuju varijable probabilističkih karaktera. Tako se najčešće modeliraju i simuliraju ekonomski, društveni i biološki fenomeni. (Božikov, 2006.)

Prilikom modeliranja koristi se jedan od prethodno navedenih konceptualnih modela dijagram kauzalnih (uzročnih) petlji ili dijagram toka. Dijagram kauzalnih uzročnih petlji prikazuje uzročne i posljedične veze između entiteta. Veze se označavaju strelicama koje mogu biti pozitivne i negativne. Pozitivne su ako rast uzroka rezultira rastom posljedice ili ako pad uzorka dovodi do pada posljedice. Tada je ta strelica pozitivna (+). Strelica je negativna ako rast uzroka rezultira padom posljedice na koji utječe ili ako pad uzroka rezultira rastom posljedice na koji utječe tada možemo reći da je strelica negativna (-). Dvije ili više veza koje međusobno povezuju uzrok i posljedicu bilo lančano, bilo direktno između uzroka i posljedice možemo nazvati i povratnom vezom ili povratnom petljom. Ona se

zapisuje kružnom strelicom koja u sredini ima plus (+) ili minus (-) ovisno o tome kako uzrok utječe na posljedicu koju povezuju. Ako su sve uzročno posljedične veze unutar petlje pozitivne tada je i sama povratna veza pozitivna. Ako povratna veza sadrži negativan uzrok ili posljedicu onda brojimo koliko negativnih veza ima u povratnoj petlji. Ako je broj negativnih veza paran tada je povratna petlja pozitivna i označava se plusom, a ako broj negativnih veza neparan tada je povratna petlja negativna i označava se minusom. Ovakva analiza povratne petlje je moguća jer svaka negativna veza mijenja smjer promjene u povratnoj petlji iz tog razloga se parnim brojem negativnih veza dolazi na početno stanje odnosno negativne veze se međusobno poništavaju. (Božikov, 2006.)

Na slici 2 na sljedećoj stranici možemo vidjeti grafički prikaz svih gore navedenih elemenata koji se nalaze u dijagramu uzročnih petlji te još možemo zamijetiti i ulazni tok, odijeljak i izlazni tok. Ulazni tok, odijeljak i izlazni tok možemo naći i u dijagramu toka. Ulazni tok je ustvari uzrok koji utječe na neki entitet. Odijeljak je taj entitet, a izlazni tok je ustvari posljedični tok koji spaja entitet s nekim drugim entitetom kao uzročni tok tog drugog entiteta. S desne strane svakog od prikaza možemo vidjeti kako će se linijski graf kretati u odnosu na prolazak vremena odnosno što se više udaljavamo od osi y to smo više u budućnosti za neki specifični entitet kojeg promatramo.

Slika 2: Primjeri grafičkih simbola korištenih na dijagramima uzročnih petlji

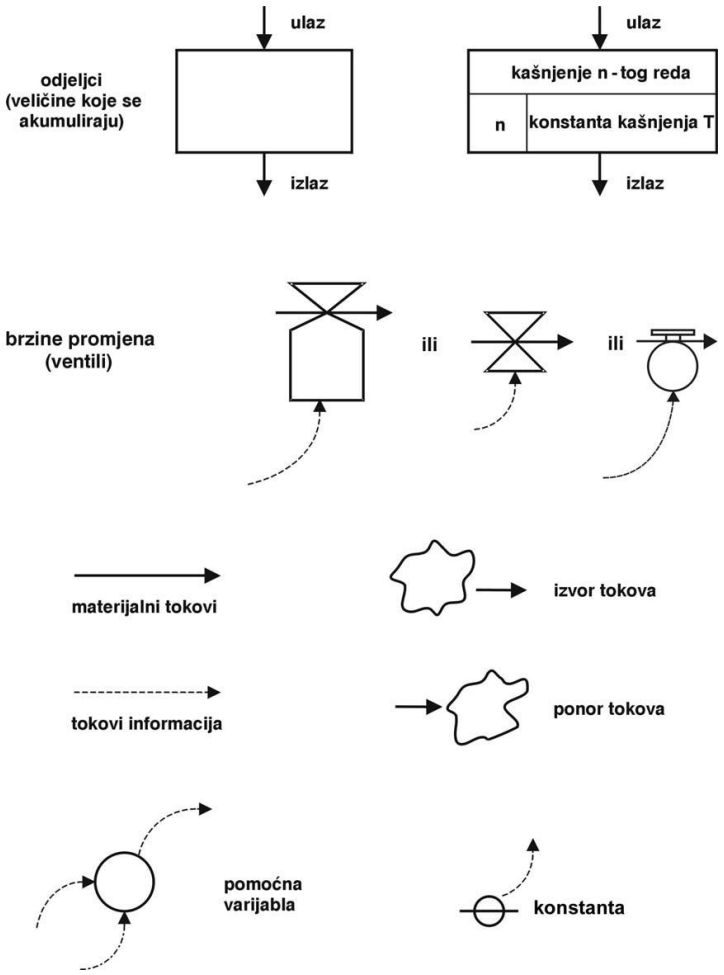


Izvor: Božikov, 2006.

Dijagram toka sadrži odjeljke koji su označeni pravokutnikom. Odjeljci su veličine koje se akumuliraju odnosno ulazni tok povećava količinu materijala u odjeljku, a izlazni tok smanjuje količinu materijala u odjeljku. Tokovi na sebi sadrže prikaz ventila koji označava kolika se količina određenog materijala propušta ili ispušta kroz taj tok u određenoj jedinici vremena. Tokovi mogu biti označeni punim i iscrtkanim strelicama. Pune strelice označavaju materijal, a iscrtkane strelice označavaju informacije. Krugom se u dijagramu toka označavaju pomoćne varijable. Postoji i poseban tip pomoćne varijable koji se naziva konstanta on se prikazuje s krugom koji je precrtan jednom horizontalnom linijom. Nepravilnim oblicima prikazani su izvor i ponor koji ustvari služe kao budući spoj s nekim vanjskim sustavom iz okoline. Postoje još i eksponencijalna kašnjenja koja se

također prikazuju pravokutnicima, ali ti pravokutnici sadrže još i raspored kašnjenja te vremensku konstantu kašnjenja. (Božikov, 2006.)

Slika 3: Primjeri grafičkih simbola korištenih na dijagramima toka



Izvor: Božikov, 2006.

3. Transpiler, lexer, parser, tokeni

Ovo poglavlje pojašnjava terminologiju vezanu uz transpiler. Cilj je objasniti samo značenje pojmova kao što su transpiler, lexer, parser, gramatika, tokeni. Bavi se i razlikom između transpilera, kompajlera i interpretera kako bismo znali razlikovati ta tri koncepta prevođenja. Da se što bolje objasne svi navedeni pojmovi i koncepti raščlaniti će se i razine apstrakcije programskih jezika te podjela programskih jezika po generacijama.

3.1 Transpiler

Transpiler (transcompiler) je prevoditelj koji prevodi jedan programski jezik u drugi programski jezik. Važno je naglasiti da ti programski jezici moraju biti iste ili približne razine apstrakcije. Transpiler se sastoji od raznih komponenti kao što su Lexer, Parser i Tokeni. U ovom radu prevodi se XMILE format u Ksdtoolkit odnosno Ksdtoolkit u XMILE format. Ta su dva sustava; XMILE je jezik oznaka (eng. *markup language*), a ksdtoolkit je DSL jezik koji se koristi za specifičnu domenu uporabe (eng. *domain-specific language*); približno iste razine apstrakcije. Kotlin u kojem je kreiran ksdtoolkit je objektno orijentirani programski jezik koji podržava i mogućnost funkcijskog programiranja. XMILE format (XML Modeling Interchange Language) je standard koji se koristi kao ulaz (eng. *input*) ili izlaz (eng. *output*) za Sistem dinamičke modele. XMILE format je ustvari jezik oznaka kao što smo to gore naveli i upitno je da li je on ustvari uopće klasični programski jezik. Programski jezici se danas klasificiraju u pet generacija ovisno o razini apstrakcije.

U programskim jezicima prve generacije, programski kod se piše direktno u strojnome (binarnome) obliku. Kod ovih jezika nije potreban kompajler ili interpreter već ih računalo odmah izvodi. Programski jezici druge generacije koriste asemblerske jezike kod pisanja programskog koda. Asemblerski kod se zatim pretvara u strojni kod. Kao i kod strojnog koda, asemblerski kod je vezan uz određenu vrstu procesora. Programski jezici treće generacije omogućuju pisanje programskoga koda u programskome jeziku koji se pretvara u strojni kod pomoću

kompajlera. Programiranje nije ovisno o procesoru. Računalu se opisuje kako se nešto rješava (imperativno programiranje). Programiranje je isprva bilo samo proceduralno, a od 1980. postaje i objektno orijentirano. Primjeri programskih jezika treće generacije: Java, PHP, C, C++, C#, Pascal, Visual Basic, Fortran, Cobol i tako dalje. Programski jezici četvrte generacije su programski jezici kojima se računalu navodi što treba riješiti, ali ne i kako (deklarativno programiranje). Najčešće se koriste kod pristupa bazama podataka naprimjer SQL. Programski jezici pete generacije namijenjeni su programiranju umjetne inteligencije (ekspertni sustavi, neuronske mreže itd.). Primjer programskih jezika pete generacije su: Prolog, OPS5 i Mercury. (Jakupović, Šuman, 2014.)

Iz gornje podijele može se primijetiti da su Kotlin i XMILE ustvari programski jezici treće generacije. Oni su na istoj razini apstrakcije odnosno oba koriste imperativno programiranje te se i jedan i drugi mogu koristiti na više računala neovisno o procesoru. Razlika je u tome što je XMILE jezik oznaka, dok se Kotlin kao i Java bazira na JVM-u (eng. *Java Virtual Machine*) i njegovom interpreteru za bytecode.

Interpreter učitava računalni program, zatim čita instrukciju po instrukciju, tumači ju i izvršava. Budući da tumačenje instrukcija troši vrijeme, interpreteri su spori. Računalni program nije u izvršnome obliku (strojnome kodu). On može biti u izvornome obliku (izvornome kodu) ili u nekome posebnom obliku potrebnom interpreteru (npr. Bytecode kod JVM – Java Virtual Machine). Pri izvođenju izvršne verzije računalnoga programa (ekstenzija EXE), on se treba iz izvornoga koda preoblikovati u strojni kod. Ovo preoblikovanje izvodi posebni računalni program - kompajler. On učita cijeli računalni program te ga pretvori u poseban oblik koji računalo može izravno izvesti, a da ne treba dodatni računalni program. Računalni program u strojnome kodu nije čitljiv ljudima, već računalima. (Jakupović, Šuman, 2014.)

Na kraju ovog potpoglavlja možemo sumirati neke osnovne razlike između transpilera, interpretera i kompajlera. Transpiler je prevoditelj koji prevodi programske jezike iste razine apstrakcije i nema nikakav doticaj sa strojnim kodom.

Kompajler je prevoditelj koji iz bilo koje više razine apstrakcije prevodi kod u strojni kod razumljiv računalu. To čini tako da cijeli kod prvo prevede, a nakon toga ga izvrši u strojnom kodu. Interpreter interpretira liniju po liniju koda i tako ju izvrši. Također, on ne mora nužno pretvarati kod veće razine apstrakcije u strojni kod, ali mora imati neki oblik koji će mu služiti za izvršavanje tog koda, a to iziskuje dodatne instalacije na računalo kako bi ono moglo razumjeti i interpretirati željeni kod.

3.2 Lexer (tokenizer)

Cilj lexera je pretvoriti tekst u niz manjih razumljivih cjelina koji se nazivaju tokeni. Zato se u nekim literaturama lexer naziva i tokenizer. Lexer ustvari radi leksičku analizu teksta. Leksička analiza teksta je naziv za proces kreiranja tokena dijeljenjem teksta na manje podstrukture. Lexeri se koriste i u govornim jezicima kako bismo lakše promatrali strukturu jezika. Tokeni su manja cjelina koja sadrži tip podataka te njegovu vrijednosti ako je potrebno. Tokeni se mogu razlikovati ovisno o tome za koji se jezik koriste. Na osnovu tog kreiranog tokena i njegove pripadnosti određenoj skupini podataka transpiler, kompajler ili interpreter znaju kako trebaju postupati s tim podatkom. U klasičnim primjerima programskih jezika tokeni se obično dijele na određene skupine kao što su naprimjer: podatak, funkcija, klasa i brojne druge. Svaka od tih struktura može imati i podskupinu. Naprimjer podatak se može podijeliti na različite tipove podataka kao što su broj, riječ, slovo... Nakon što se odradi leksička analiza sljedeći korak je raščlanjivanje dobivenih podataka i kreiranje parse stabla. Dolazimo do zaključka da se lexer bavi samo podjelom podataka u određene skupine, a ne i samom sintaksom tih podataka.

Posao lexera je iščitati tokene iz ulaznih podataka i proslijediti ih parseru. U većini slučajeva je ulazni podatak tekstualnog tipa i ne pregledava se znak po znak nego se razlomi na manje dijelove koji se pohranjuju u listu. Lexer se bavi riječima koje su globalno rezervirane za određeni programski jezik i riječima koje identificiraju neki podatak te njihovim vrijednostima. Nakon toga je na parseru red

da odredi da li su te riječi rezervirane za određeni programski jezik ili se samo koriste kao identifikator određene varijable. (Farrell, 1995.)

3.3 Parser i gramatika programskih jezika

Posao parsera je pronaći greške u sintaksi i kreirati parse stablo od dobivene liste tokena. Obično se za to koristi i gramatika koja olakšava kreiranje parse stabla jer sadrži pravila po kojima podatci moraju biti posloženi. Ako podatci nisu složeni po gramatičkim pravilima tada možemo reći da u tom programskom kodu postoji sintaktička greška. Postoje dvije normalne forme koje se koriste prilikom opisa gramatike nekog jezika. Noam Chomsky je prvi krenuo s razvijanjem formalne metode s kojom bi prikazao strukturu jezika. To mu je i pošlo za rukom kad je kreirao CNF (eng. *Chomsky Normal Form*). CNF je i prva forma kojom se može pojasniti gramatika nekog jezika. CNF se sastoji od tokena koji prvo definira naziv tipa podatka, a nakon toga strelicom „pokazuje" od čega se sve može sastojati taj tip podatka. Ako tip podatka ima više podopcija one se dijele znakom |. Taj znak se u logici naziva ili (eng. *or*). Podopcije mogu biti simboli ili neki drugi tip podatka. Također, je važno napomenuti da svaka CNF gramatika počinje s tipom podatka S. On obuhvaća sve moguće tipove podatka i može ga se promatrati kao jedna cjelina odnosno rečenica tog jezika. (Farrell, 1995.)

Slika 4: Primjer CNF gramatike

```
S -> EXPRESSION
EXPRESSION -> TERM | TERM + EXPRESSION | TERM - EXPRESSION
TERM -> FACTOR | FACTOR * EXPRESSION | FACTOR / EXPRESSION
FACTOR -> NUMBER | ( EXPRESSION )
NUMBER -> 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 |
          1 NUMBER | 2 NUMBER | 3 NUMBER | 4 NUMBER |
          5 NUMBER | 6 NUMBER | 7 NUMBER | 8 NUMBER |
          9 NUMBER | 0 NUMBER
```

Izvor: Farrell, 1995.

Slika 4 prikazuje primjer CNF gramatike koja je ujedno jednostavna, ali i kompleksna. Jednostavan je njen prikaz koji ima svega pet tipova podataka od kojih se može sastojati neka rečenica u tom jeziku. Doduše kompleksan je jer se tih pet tipova podataka može kombinirati na više načina i to koristeći rekurziju.

Rekurzija se događa kada dva tipa podataka s lijeve strane sadrže jedan drugoga kao opciju na desnoj strani. Valja napomenuti da su tipovi podataka na lijevoj strani više razine nego ti isti tipovi podataka na desnoj strani. Iz tog razloga ovakav tip jezika nazivamo desno rekurzivnim jezikom. Primjer toga u ovoj gramatici su pojmovi EXPRESSION i TERM koji sadrže jedan drugog na desnoj strani.

Druga forma kojom se može pojasniti gramatika nekog jezika je EBNF (eng. *Extended Backus Naur Form*). Tu formu su kreirali Johan Backus i Peter Naur, a ona je nadogradnja CNF-a. Njena je prednost što neke jezike može bolje pojasniti nego CNF forma. Doduše obje gramatike se mogu konvergirati u kompajler, ali većinom se danas kod kreiranja kompajlera koristi EBNF. EBNF ima brojne sličnosti CNF-u, ali sadrži i neke razlike. Umjesto strelice se u EBNF-u koristi znak :== koji ima identičnu ulogu kao i strelica u CNF-u. Podjela opcija na desnoj strani se i dalje radi pomoću znaka |, ali sad se simboli odvajaju jednostrukim navodnicima dok se tipovi podatka zapisuju jednako kao i u CNF-u. U EBNF-u su uvedena i dva nova pojma radi se o uglatim i vitičastim zagradama. Uglate zagrade služe za označavanje nijedne ili jedne pojave tog simbola dok vitičaste zagrade služe za označavanje barem jedne ili više pojava tog simbola. Katkad je korisno označiti i nulu kod n pojava nekog simbola. To radimo tako da vitičastim zagradama dodamo zvjezdicu {}*. (Farrell, 1995.)

Slika 5: Primjer EBNF gramatike

```
S :== EXPRESSION
EXPRESSION :== TERM | TERM { [+,-] TERM] }
TERM :== FACTOR | FACTOR { [*,/] FACTOR] }
FACTOR :== NUMBER | '(' EXPRESSION ')'
NUMBER :== '1' | '2' | '3' | '4' | '5' |
           '6' | '7' | '8' | '9' | '0' |
           '1' NUMBER | '2' NUMBER | '3' NUMBER |
           '4' NUMBER | '5' NUMBER | '6' NUMBER |
           '7' NUMBER | '8' NUMBER | '9' NUMBER | '0' NUMBER
```

Izvor: Farrell, 1995.

Na slici 5 možemo vidjeti EBNF primjer za istu gramatiku kao i kod CNF primjera sa slike 4. Razlike su minorne, a jedna od najvažnijih razlika je ta da se u ovoj gramatici pojavljuje samo jedna rekurzija. Rekurzija se događa u tipu podatka

FACTOR koji u svojim opcijama s desne strane ima tip podatka EXPRESSION. Kada se EBNF gramatika odnosno jezik prevodi u kompajler rekurzivna pravila prevode se u rekurzivne pozive procedura. Pravila koja sadrže vitičaste zagrade prevode se u petlje, a pravila s uglatim zagradama u if-ove.

4. XMILE format

Poglavlje XMILE format će pojasniti specifikacije samog XMILE formata odnosno njegove dijelove i strukturu. Kao što smo prethodno spomenuli XMILE format služi za pohranu i distribuciju SD modela. Zato ćemo pojasniti razloge zbog kojih je došlo do kreiranje ovog formata. Navest ćemo i neke programe koji koriste taj format i opisati kako se dogodio prijelaz iz nekih prethodnih formata u XMILE. Također, ćemo opisati od kojih se oznaka sastoji XMILE format. Oznake XMILE formata su jako važne za kreaciju transpilera jer će one odrediti tokene te kako treba kreirati lexer i parser. U zadnjem podpoglavlju ovog poglavlja navest ćemo sve što nam je potrebno kako bismo kreirali transpiler koji sadrži sve nužno za implementaciju bazičnog odnosno osnovnog levela XMILE formata.

4.1 Definicija i prednosti XMILE formata

XMILE (eng. *XML Interchange Language for System Dynamics*) format je javno dostupan i besplatan XML protokol koji služi za dijeljenje, interoperabilnost i ponovnu uporabu sistem dinamike i SD modela. Kreirala ga je organizacija za standardizaciju OASIS (eng. *Organization for the Advancement of Structured Information Standards*). OASIS je neprofitna internacionalna organizacija koja je do sad kreirala brojne standardizirane formate. Neki od tih formata uz XMILE su: SGML (eng. *Standard Generalized Markup Language*), XML (eng. *Extensible Markup Language*), HTML (eng. *Hypertext Markup Language*) i brojni drugi. XMILE je primjer na koji način radi OASIS i koji je njegov cilj. Kreiranjem XMILE formata sistem dinamika je dobila brojne prednosti koje prethodno nije imala. Naprimjer ovim formatom je omogućeno brže dijeljenje modela između istraživača. Modeli se mogu koristiti kao podmodeli nekih kompleksnijih i opširnijih modela. XMILE pruža mogućnost kreiranja velikih online repozitorija koji će sadržavati brojne SD modele. Također, njegova glavna prednost je što će primorati korporacije i programere da svoje programe ili alate; koji se bave sistem dinamikom; prilagode tako da mogu koristiti XMILE format jer inače više neće biti konkurentni na tržištu. Krajnji benefit će na kraju biti šira uporaba sistem dinamike.

Općenito će se povećati dostupnost alata, ali i modela sistem dinamike. (Chichakly, 2013.)

Neki programi koji prethodno nisu imali u svojoj implementaciji mogućnost korištenja XMILE formata su se prilagodili tom standardu i omogućili ga svojim korisnicima. Neki od njih su programi iThink, Vensim i STELL-a. XMILE format je ustvari tekstualnog oblika i ne koristi binarni prikaz u svojoj implementaciji. Samim time XMILE korisniku pruža mogućnost da lako sam unutar datoteke; pomoću nekog tekstualnog editora; izmjeni ili pretraži neke pojmove i vrijednosti. Zato je XMILE smanjio i mogućnost greške prilikom slanja datoteke jer kod tekstualnih oblika teže dolazi do greške ili ne kompatibilnosti na različitim sustavima. Omogućio je i lakše načine verzioniranja pomoću Gita i SVN-a (eng. subversion) koji su također puno bolje prilagođeni tekstualnim oblicima. (Chichakly, 2013.)

4.2 Oznake XMILE formata

XMILE format je inkomponiran u drugi format koji se naziva XML (eng. *Extensible Markup Language*). XML (kao što i samo ime govori) je jezik koji se bazira na oznakama (eng. *markup*). XML je jezik koji je tekstualnog oblika te nema nikakvu potrebu za klasičnim kompajlerom. On ne ovisi niti o operacijskom sustavu niti o platformi odnosno hardveru na kojem se koristi. Za njegovo izvršavanje potrebno je imati neku vrstu prevoditelja ili interpretera koji će znati prepoznati njegova svojstva i prevesti ga u neki drugi programski jezik. Samim time i XMILE format ima ista svojstva. U XML-u i XMILE formatu jako važnu ulogu imaju oznake koje određuju kako interpretirati odnosno prevesti pojedini dio koda. Oznake definiraju značenje sadržaja koji se nalazi unutar oznaka. Svaka oznaka može sadržavati određene pod oznake i svaka oznaka ima svoj zapis za početak i kraj bloka naredbi. Sve što se nalazi unutar tog bloka naredbi vezano je za oznaku koja obuhvaća taj blok.

Oznaka za početak se kreira tako da se ime oznake ogradi znakovima < i >. Primjer jednog početka oznake je <XMILE>. Kraj oznake se prikazuje vrlo slično samo pomoću znaka / dodatno naglasimo da se radi o kraju oznake. Primjer kraja

oznake je </XMILE>. Kraj oznake ne može sadržavati nikakve dodatne informacije dok početak oznake može imati uz ime same oznake i neke dodatne informacije o oznaci koje nazivamo atributima oznake. Naprimjer to izgleda ovako <xmile version="1.0" xmlns="http://docs.oasis-open.org/xmile/ns/XMILE/v1.0">. Pomoću znaka = smo odredili da je verzija ove XMILE datoteke jednaka 1. Također, treba napomenuti da je ono čemu je verzija jednaka omeđeno navodnim znacima (" "). Ovaj dio nam je važan iz razloga jer smo tako utvrdili neke interpunkcijske znakove. Ovi interpunkcijski znakovi će nam pomoći prilikom kreiranja transpilera odnosno liste tokena. Na tim znakovima ćemo u kodu podijeliti niz stringova kako bi dobili manje cjeline. (OASIS, 2021.)

Nakon što je pojašnjen dio o interpunkcijskim znakovima i kako oni spajaju i odvajaju tekst XMILE datoteke, prelazimo na glavne oznake koje se nalaze u jednoj XMILE datoteci. Važno je napomenuti da postoji više razina oznaka ovisno o tome jesu li oznake obavezne ili nisu i nalaze li se unutar drugih nadoznaka. Glavna oznaka cijelog dokumenta XMILE formata je naravno oznaka XMILE. Svaki sistem-dinamički model zapisan u XMILE formatu na početku dokumenta mora imati početnu XMILE oznaku. Jedina oznaka koja se nalazi izvan početne i krajnje XMILE oznake je oznaka XML koja „obavija" XMILE format i ukazuje da je XMILE struktura koja se koristi unutar XML-a, kako je gore već prethodno navedeno. Svaki dokument pisan u XMILE formatu mora biti pisan standardom UTF-8 te mora biti navedena verzija XMILE formata koja se koristi. Trenutno jedina verzija XMILE formata je verzija 1.0. (OASIS, 2021.)

U XMILE formatu postoji devet oznaka najviše razine (eng. *top-level*). Svaka od tih oznaka je podoznaka oznake XMILE. Oznake najviše razine se dijele na obavezne, opcionalne, oznake koje se pojavljuju nula ili više puta, te oznake koje se pojavljuju barem jednom. Ove oznake mogu se unutar oznake XMILE pojavljivati bilo kojim redoslijedom, ali preporučeni redoslijed je: header, sim_specs, model_units, dimensions, behavior, style, data, model, macro. Oznake koje se pojavljuju prije u nabrajanju nalaze se iznad onih koje su poslije nabrojane, takav redoslijed je preporuka organizacije OASIS. Jedina obavezna oznaka na najvišoj razini oznaka je header i ona mora obavezno biti sadržana kao podoznaka

XMILE oznake. Oznaka model je oznaka koja pripada oznakama koje se moraju pojaviti barem jednom što znači da je i ona obavezna oznaka, ali za razliku od header oznake ona se može pojaviti više puta tako da ju ne svrstavamo u kategoriju obaveznih oznaka nego u kategoriju oznaka koje se pojavljuju barem jednom. Oznaka macro je oznaka koja se ne mora pojaviti niti jednom, ali se može pojaviti i više puta tako da ju svrstavamo u oznake koje se pojavljuju nula ili više puta. Sve preostale oznake pripadaju kategoriji opcionalnih oznaka. (OASIS, 2021.)

4.2.1 Oznaka header

Oznaka header (hrv. *zaglavlje*) je jedna od oznaka koja je obavezna u svakoj datoteci XMILE formata. Ona sadrži i svoje podoznake koje se također dijele na obavezne, opcionalne, oznake koje se pojavljuju nula ili više puta te oznake koje se pojavljuju barem jednom. Zato što postoji jako velik broj oznaka u ovom i sljedećim potpoglavljima vezanim uz oznake bavit ću se samo s oznakama koje su obavezne te oznakama koje smatram važnima prilikom prevođenja. Te oznake i njihovu funkciju ću detaljnije opisati. Prilikom implementacije prevoditelja će vjerojatno biti uključene i neke oznake koje u ovim potpoglavljima nisu navedene. Oznaka header je oznaka koja sadrži sve važne informacije o podrijetlu dokumenta pohranjenog u XMILE formatu. Ta oznaka se koristi i u drugim markup jezicima koje je kreirao OASIS. Header sadrži dvije obavezne oznake vendor i product. Oznaka vendor (hrv. *prodavač*) je oznaka koja „obavija” ime kompanije koja je izradila taj dokument. Oznaka product (hrv. *proizvod*) unutar svoje početne oznake sadrži varijablu version (hrv. *verzija*). Version je obavezan dio unutar početne oznake product. Također oznaka product može sadržati i varijablu lang skraćeno od riječi language (hrv. *jezik*). Varijabla lang označava u kojem je jeziku pisan dokument. Ako u oznaci product varijabla lang nije naznačena XMILE format će uzeti engleski kao zadani jezik. Neke od opcionalnih podoznaka oznake header su: options, name, version, caption, image, author,client, affilation, copyright, contact, crated, modified, uuid i includes. Neke od tih oznaka imaju i svoje podoznake kao naprimjer options, contact. (OASIS, 2021.)

4.2.2 Oznaka sim_specs

Oznaka sim_specs skraćeno od Simulation Specification Section (hrv. *odjeljak specifikacijske simulacije*) je jedna od opcionalnih oznaka XMILE dokumenta, ali to nije u potpunosti točno. Svaki XMILE dokument mora sadržavati barem jednu sim_specs oznaku bilo kao oznaku najviše razine koja je podoznaka XMILE oznake ili kao podoznaku korijenske (eng. root) oznake model. Također, treba napomenuti da XMILE dokument može sadržavati više sim_specs oznaka, a u tom slučaju sim_specs oznaka koja se nalazi unutar model oznake ima prednost prilikom simulacije modela u odnosu na oznaku koja se pojavljuje kao oznaka najviše razine. Zato možemo reći da je ovaj odjeljak odnosno oznaka vrlo važna i u našoj implementaciji u ksdtoolkit-u jer možda nije obavezna kao podoznaka XMILE oznake, ali je globalno gledano obavezna. Sim_specs oznaka se sastoji od dvije obavezne oznake start i stop. Oznaka start označava vrijeme početka, a oznaka stop označava koliko vremena mora proći da se simulacija modela zaustavi. Oznaka sim_specs također ima i jednu opcionalnu podoznaku koja je korisna i često korištena u SD modelima. Ta oznaka je dt ona služi da bi korisnik mogao odrediti step (hrv. *korak*). Tako korisnik ustvari određuje koliki će se vremenski interval provjeravati kao zasebna cjelina u odnosu na neki drugi vremenski interval. Oznaka dt sadrži varijablu reciprocal (hrv. *recipročan*) koja može sadržavati vrijednosti true (hrv. *istina*) i false (hrv. *laž*). Treba napomenuti da i oznaka sim_specs također sadrži neke varijable kao što su method (hrv. *metoda*), time_units (hrv. *vremenski jedinica*), i pause (hrv. *pauza*). (OASIS, 2021.)

4.2.3 Oznaka model_units

Oznaka model_units je također opcionalna oznaka. Ona omogućava korisniku da napravi XMILE dokument koji se sastoji od većeg broja modela. Oznaka model_units se sastoji od oznake unit koja ima atribut name koji je obavezan za svaki zasebni unit koji je korisnik kreirao. Podoznake oznake unit su oznake alias i eqn. Oznaka eqn označava equation (hrv. *jednadžbu*) koja se koristi unutar tog modela, eqn je opcionalna podoznaka koja se može pojaviti samo jednom.

Oznaka alias se može pojavljivati nula ili više puta odnosno ovisno o tome koliko je unit-a (hrv. *jedinica*) aliasa potrebno unutar svake oznake unit-a. Kod oznake model_units je važno napomenuti da postoji i atribut disabled (hrv. *onesposobi*) koji u slučaju da mu je vrijednost postavljena na true (hrv. *istina*) se ne uključuje u simulaciju i provedbu tog XMILE dokumenta odnosno njegovih modela. (OASIS, 2021.)

Slika 6: Primjer model_units oznake XMILE formata

```
<model_units>
   <unit name="models_per_person_per_year">
     <eqn>models/person/year</eqn> <!-- name, equation -->
   </unit>
   <unit name="Rabbits">
     <alias>Rabbit</alias>   <!-- name, alias -->
   </unit>
   <unit name="models_per_year">
     <eqn>models/year</eqn>       <!-- name, eqn, alias -->
     <alias>model_per_year</alias>
     <alias>mpy</alias>
   </unit>
   <unit name="Joules" disabled="true"> <!-- disabled unit -->
     <alias>J</alias>
   </unit>
</model_units>
```

Izvor: OASIS, 2021.

4.2.4 Oznaka dimensions

Oznaka dimensions je obavezna kada unutar oznake options koja sadrži listu funkcionalnosti dodamo odnosno postavimo podoznaku uses_arrays. Oznaku options, njene podoznake i funkcionalnosti ćemo objasniti u zadnjem potpoglavlju poglavlja 4.2. Set dimenzija koji se postavi unutar bloka oznaka dimensions moraju biti konzistentne kroz cijeli XMILE dokument odnosno kroz cijeli model koji je implementiran tim dokumentom. Svaka oznaka dimensions mora sadržavati podoznaku dim koja ima atribute name i size. Atribut size (hrv. *veličina*) mora biti veći ili jednak jedan te on obavezno mora biti dodijeljen osim u slučaju koji je naveden u nastavku. Ako elementi array-a (hrv. *liste*) imaju imena, onda se oni navode redom unutar oznake elem. U slučaju da oznaka dim sadrži podoznaku

elem tada se ne navodi atribut size unutar oznake dim jer se sama veličina liste može zaključiti iz broja oznaka elem. (OASIS, 2021.)

Slika 7: Primjer oznake dimensions XMILE formata

```
<dimensions>
   <dim name="N" size="5"/>    <!-- numbered indices -->
   <dim name="Location">       <!-- named indices -->
     <elem name="Boston"/>    <!-- name of 1st index -->
     <elem name="Chicago"/>   <!-- name of 2nd index -->
     <elem name="LA"/>        <!-- name of 3rd index -->
   </dim>
</dimensions>
```

Izvor: OASIS, 2021.

4.2.5 Oznaka behavior

Svaki XMILE format može uključiti behavior oznaku kao opcionalnu oznaku. Pomoću oznake behavior želimo utjecati na entitete koji se koriste u modelu kojeg kreiramo. Ako unutar svog XMILE dokumenta koristimo oznaku behavior moramo obavezno iskoristiti i neku od drugih naredbi kako bi zadali željeno ponašanje određenog aspekta modela. Zato se u implementaciji oznaka behavior najčešće koristi s macro naredbama koje zadaju neka uvjetovana ponašanja kao naprimjer non_negativ oznaka koja zadaje da neki stock ili flow ne mogu biti negativni. Macro naredbe su dodatno objašnjene u poglavlju 4.2.15. Na slici osam vidimo primjer jednog takvog bloka naredbi behavior koji prikazuje restrikciju non_negativ na sve flow-ove XMILE dokumenta. Svaka restrikcija postavljena unutar behavior oznake može biti promijenjena prilikom lokalnog definiranja flow-a ili stock-a. (OASIS, 2021.)

Slika 8: Primjer oznake behavior XMILE formata

```
<behavior>
   <flow>
     <non_negative/>
   </flow>
</behavior>
```

Izvor: OASIS, 2021.

4.2.6 Oznaka data

Data je jedna od opcionalnih oznaka koje nisu obavezne prilikom izrade SD modela u XMILE formatu. Data tag sadrži dvije glavne import i export oznake koje određuju dali se određeni podatci importaju ili eksportiraju. Obje pod oznake sadrže atribute type, enabled, frequency, orientation, resource i worksheet. Prva četiri atributa su opcionalna te ih iz tog razloga neću dodatno pojašnjavati. Atribut resource je obavezan. Resource može biti relativna putanja datoteke, apsolutna putanja datoteke ili URL. Worksheet je obavezan atribut ako se radi o datoteci iz Excel-a i predstavlja ime datoteke koju koristimo u SD modelu. Oznaka export još može sadržavati i dodatne podoznake interval ili jedan od dva izvora podataka. (OASIS, 2021.)

4.2.7 Oznaka model

Oznaka model je najvažnija oznaka cijelog XMILE formata u njoj se nalaze sve najvažnije informacije koje su potrebne za kreiranje SD modela. Oznaka model može unutar svog bloka naredbi sadržavati i neke od prethodno navedenih glavnih oznaka kao što su naprimjer sim_specs i behavior. Jedna od opcionalnih oznaka je i podoznaka views. Podoznaka views sadrži sve grafičke komponente koje služe za grafički prikaz SD modela korisniku. Pod oznaku views nećemo detaljnije opisivati u zasebnom potpoglavlju jer nam ona nije ključna kada su u pitanju podaci i sama semantika SD modela. Također, treba naglasiti da ksdtoolkit ima svoj način kreiranja grafičkih komponenti tako da prilikom prevođenja XMILE formata u ksdtoolkit odnosno Kotlin neće biti potrebno prevoditi i grafičke

komponente. Prilikom prevođenja iz ksdtoolkita u XMILE format možemo kreirati grafičku šablonu koja će se koristiti za sve SD modele kreirane u ksdtoolkitu. Tako ćemo ubrzati i olakšati prevođenje u oba smjera. Na slici 9 prikazan je jedan primjer bloka naredbi oznake model. Podoznake unutar oznake model moraju uvijek biti poredane kao što je prikazano u bloku na slici. Ako neke od opcionalnih podoznaka nema tada se ostale oznake koje su zastupljene u modelu moraju pridržavati redoslijeda koji je prikazan izuzimajući tu oznaku.

Slika 9: Primjer oznake model XMILE formata

```
<model>
   <sim_specs>                <!-- OPTIONAL - see Chapter 2 -->
     ...
   </sim_specs>
   <behavior>            <!-- OPTIONAL - see Chapter 2 -->
     ...
   </behavior>
   <variables>                <!-- REQUIRED -->
     ...
   </variables>
   <views>               <!-- OPTIONAL - see Chapters 5 & 6 -->
     ...
   </views>
</model>
```

Izvor: OASIS, 2021.

XMILE format može sadržavati velik broj oznaka model unutar jednog XMILE dokumenta. Primarni ili korijenski model nema ime, ali svi ostali modeli ako su zastupljeni moraju sadržavati atribut name (hrv. *ime*) oni predstavljaju podmodele korijenskog modela. Oznaka model ima opcionalne atribute: name, run, namespace i resource. Kao što smo već naglasili atribut name se ne dodjeljuje jedino kod korijenskog modela jer on preuzima ime cijelog XMILE dokumenta. Atribut run je tipa boolean odnosno true or false i koristi se samo prilikom pokretanja SD modela odnosno daje naznaku koji se model treba koristiti, a koji model je isključen prilikom pokretanja simulacije. Atribut namespace se koristi kada želimo odabrati neki drugi identifikator osim zadanog identifikatora unutar XMILE oznake. Naprimjer možemo odrediti da namespace bude zadan ovako: namespace = "std, isee". U ovom slučaju prvo se znakovi uspoređuju s

identifikatorima namespace-a std, a nakon toga se uspoređuju s identifikatorima namespace-a isee. Posljednji atribut oznake model je resource. On je opcionalni atribut koji može sadržavati relativnu ili apsolutnu putanju do datoteke ili URL. Atribut resource se najčešće koristi kada unutar XMILE datoteke ne koristimo oznaku modules i onda umjesto toga pomoću atributa resource importiramo vanjske modele koji nam pomažu u kreiranju našeg modela. Prilikom izrade transpilera bit će važno napraviti razliku između atributa i oznaka te odrediti kako oni zaista funkcioniraju. (OASIS, 2021.)

Treba napomenuti kako određeni modeli unutar XMILE formata mogu biti dekriptirani kako bi se zaštitili povjerljivi podatci tog modela. Tada se modelu može pristupiti samo ako se unese prethodno definirana lozinka. Ovaj transpiler neće imati takve mogućnosti odnosno dekriptirani modeli se neće moći prevesti u ksdtoolki. (OASIS, 2021.)

4.2.8 Oznaka variables

Najvažnija podoznaka oznake model je oznaka variables. Oznaka variables sadrži sve podatke koji su potrebni za izvršavanje ili utječu na izvršavanje SD modela. Oznaka variables sadrži obavezne atribute name te obavezne podoznake dimensions i element. Atribut name je ime variable koju opisujemo unutar bloka variables. Podoznaka dimensions je obavezna kod oznaka koje sadrže dimenzije. Ako varijabla sadrži više dimenzija onda se unutar bloka naredbi oznake dimensions kao podoznaka koristi oznaka dim koja također sadrži obavezni atribut name. Oznaka variables sadrži još jednu obaveznu podoznaku element koja je obavezna samo u slučaju kada se nad listom podataka mora napraviti promjena koja se ne odnosi na sve podatke. Postoje dvije moguće implementacije oznake element. Prva se odnosi na listu podataka, a druga implementacija se odnosi na grafičke funkcije. Ako se element odnosi na listu podataka tada se unutar bloka naredbi element enkapsulira podoznaka eqn, a ako se odnosi na grafičke komponente tada se isto radi s podoznakom gf. Oznaka variables sadrži i opcionalne atribute: access i autoexport. Također, sadrži i opcionalne podoznake: mathml, units, doc, event_posters, range, scale i format. Njih nećemo detaljnije

pojasniti jer su opcionalne i pitanje je koje od njih će na kraju biti obuhvaćene u implementaciji. Tako da će te oznake biti detaljnije objašnjene u implementaciji ako bude potrebe za tim. (OASIS, 2021.)

4.2.9 Oznaka array

Jedna od mogućih podoznaka oznake variables je i oznaka array. Svaka varijabla može biti kreirana kao lista, ali unutar XMILE oznake header moramo postaviti oznaku uses_arrays. Također, smo oznaku uses_arrays mogli postaviti i u oznaci options. Svaka varijabla tipa reda mora sadržavati jednu i isključivo jednu oznaku dimensions koja definira tu listu. Detaljniji izgled oznake dimensions možete vidjeti u poglavlju 4.2.4 Oznaka dimensions. Postoje tri različite vrste lista u XMILE formatu. Prva vrsta liste odnosno arraya je apply-to-all arrays (hrv. *primijenjen na sve liste*). U tom slučaju se jedna varijabla koja je vlasnik ponaša kao surogat prema svim ostalim elementima liste. Sljedeća vrsta liste je non-apply-to-all arrays (hrv. *neprimijenjen na sve liste*). Kada imamo ovu vrstu arraya onda se varijabla koja je vlasnik ponaša kao kontejner koji sadrži sve elemente i na njih nužno prenosi određene informacije koje su jednake kod svih tih elemenata. Treća i najkompliciranija vrsta liste je apply-to-all arrays with non-apply-to-all graphical functions (hrv. *primijenjen na sve liste i neprimijenjen na sve grafičke funkcije*). Kod ove vrste liste varijabla vlasnik se prema svim ostalim elementima ponaša kao surogat osim u slučaju grafičkih elemenata. (OASIS, 2021.)

4.2.10 Oznaka stock

Oznaka stocks je jedna od podoznaka oznake variables. Stock (hrv. *zaliha*); oznaka stock odnosi se na element koji u poglavlju 2.2 nazivamo odjeljak. Direktan prijevod riječi stock je kao što smo naveli zaliha, ali u većini slučajeva se ta zaliha odnosi na neku brojčanu vrijednost koja može biti materijalna ili informacijska te samim time pohranjena u odjeljak. Zato se prilikom grafičkog prikaza koristi takva terminologija. Svaka oznaka stock zahtjeva da atribut name bude unesen. Kada želimo pokrenuti simulaciju svaka varijabla koja je definirana oznakom stocks mora sadržavati početnu vrijednost koja se definira unutar podoznake eqn. Ako

želimo da se početna vrijednost zalihe promjeni moramo implementirati barem jedan flow (hrv. *tok*) bilo da se radi o ulaznom ili izlaznom toku. Na slici koja se nalazi ispod prikazan je blok naredbi unutar oznake stock. (OASIS, 2021.)

Slika 10: Primjer oznake stock XMILE formata

```
<stock name="Motivation">
   <eqn>100</eqn>
   <inflow>increasing</inflow>
   <outflow>decreasing</outflow>
</stock>
```

Izvor: OASIS, 2021.

Slika 10 unutar bloka naredbi stock sadrži oznake eqn, inflow i outflow. Oznaka eqn označava početnu vrijednost zalihe odnosno stock-a. Oznaka inflow označava ulazni tok, a oznaka outflow označava izlazni tok. Zaliha ne mora sadržavati niti jedan ulazni ili izlazni tok, ali može ih sadržavati više ako je slučaj da određena zaliha zahtjeva takve uvijete. Ako zaliha sadrži više ulaznih tokova, a njihov redoslijed je bitan tada moramo napomenuti da će se u obzir uzimati redoslijed zapisa oznaka ulaznog toka. Oznake ulaznog toka koje se nalaze iznad u našem bloku naredbi smatrat će se oznakama koje se prije promatraju prilikom pregleda bloka naredbi stock. Isti princip će se primijeniti i kod izlaznih tokova. Kod ulaznih tokova redoslijed je važan jedino ako se koriste queues (hrv. r*edovi)* ili capacity-constrained conveyors (hrv. *transporteri s ograničenim kapacitetom*). Kod izlaznih tokova ovisno o količini zalihe prvo će se puniti prvi izlazni tok, a ako ostane zalihe onda i svaki sljedeći izlazni tok. Naravno kod izlaznog toka ovaj princip se može primjenjivati jedino ako zaliha nije neka vrijednost koja može poprimiti negativan predznak. Unutar bloka naredbi stock mogu se pojaviti još tri oznake: conveyor, queue i non_negative. Oznaka conveyor uzrokuje da se zaliha ponaša kao transporter sa željenim transportnim parametrima. Oznaka queue uzrokuje da se zaliha ponaša kao red, a oznaka non_negative je boolean koja određuje smije li zaliha biti negativna ili ne. Važno je napomenuti da svaka non_negative oznaka unutar stocka nadjačava istu tu oznaku unutar oznake behavior. Oznaka queue nema nikakve podoznake. Doduše njezini izlazni tokovi mogu imati podoznaku overflow (hrv. *prelijevanje*). Oznaka conveyor sadrži obaveznu podoznaku len koja

označava dužinu transporta u jedinici vremena. Ta vrijednost se bilježi od trenutka kad je materijal ušao u transporter do trenutka kad je izašao. Također, valja napomenuti da oznaka conveyor ima i četiri opcionalne podoznake: capacity, in_limit, sample, arrest te četiri opcionalna atributa: discrete, batch_integrity, one_at_a_time i exponential_leak. Opcionalne oznake i atribute nećemo dodatno pojašnjavati, ali neki od njih će vjerojatno biti uključeni prilikom implementacije. (OASIS, 2021.)

Slika 11: Primjer oznake stock s podoznakom conveyor XMILE formata

```
<stock name="Students">
   <eqn>1000</eqn>
   <inflow>matriculating</inflow>
   <outflow>graduating</outflow>
   <conveyor>
     <len>4</len>
     <capacity>1200</capacity>
   </conveyor>
</stock>
```

Izvor: OASIS, 2021.

Slika 11 prikazuje zalihu imena studenti koja ima početnu vrijednost 1000. Ulazni tok joj se naziva maturanti, a izlazni tok su studenti koji su diplomirali. Transporter je dužine četiri. Iz tog možemo zaključiti da taj studij traje četiri godine, a maksimalni kapacitet tijekom te četiri godine je 1200 studenata.

4.2.11 Oznaka flow

Oznaka flow (hrv. *tok*) dijeli se na inflow (hrv. *ulazni tok*) i outflow (hrv. *izlazni tok*). Svaki tok mora sadržavati atribut name koji određuje naziv tog toka. Većina tokova sadrži i oznaku eqn koja označava operaciju koja se koristi u trenutku uporabe tog toka. Jedini tokovi koji nemaju nikakvu zadanu operaciju su tokovi koji imaju redove ili izlazne transportere koji su već prethodno određeni. (OASIS, 2021.)

Slika 12: Primjer oznake flow XMILE formata

```
<flow name="increasing">
   <eqn>rewards*reward_multiplier</eqn>
</flow>
```

Izvor: OASIS, 2021.

Oznaka flow ima i opcionalne podoznake. Prva pod oznaka je multiplier. Multiplier se koristi kad dvije zalihe odnosno dva stock-a nemaju istu mjernu jedinicu ili mjerilo. Tada koristimo multiplier kako bi napravili pretvorbu tih jedinica. Postoje još tri podoznake oznake flow. Te tri oznake su non_negative, overflow i leak i one se međusobno isključuju. Oznaku non_negative smo već objasnili u jednom od prethodnih poglavlja i ona prevenira tok da pređe u negativno stanje neovisno o operaciji koju koristimo za njen izračun. Overflow je oznaka koja u slučaju da koristimo red ili transporter služi kako bi se višak materija koji se šalje tokom mogao prebaciti iz popunjenog ili blokiranog elementa u neki drugi element koji je predodređen kao zamjenski. Može se dogoditi da više elemenata reda bude popunjeno ili blokirano, tada tok utječe u sljedeći element u nizu. Ovakva pojava može se pojaviti samo kod izlaznog toka reda. Oznaka leak je posebna oznaka koja kod overflow transportera označava onaj transporter koji ima leak (hrv. *curenje*). To se u modelu događa kada u nekom trenutku dođe do brisanja ili privremenog odvajanja toka. (OASIS, 2021.)

4.2.12 Oznaka aux

Aux je skraćenica od riječi auxiliaries koja u prijevodu na hrvatski znači pomoćna tvar, dodatak. Oznaka aux također zahtjeva unos atributa name. Ako aux želimo koristiti u simulaciji bit će nam potrebna i njegova podoznaka eqn. Na slici 13 koja se nalazi na sljedećoj stranici možete vidjeti primjer osnovnog oblika bloka naredbi aux. Oznaka aux u slučaju s ove slike je konstanta vrijednost i ona se ne može promijeniti. (OASIS, 2021.)

Slika 13: Primjer oznake aux XMILE formata

```
<aux name="reward_multiplier">
   <eqn>0.15</eqn>
</aux>
```

Izvor: OASIS, 2021.

Kod oznake aux važno je još napomenuti i jedan opcionalan atribut flow_concept. Flow_concept je tipa boolean odnosno može biti istinit ili neistinit. Zadana vrijednost ovog atributa je neistinita jer većinu aux odnosno dodataka ne želimo odmah deklarirati kao tok. (OASIS, 2021.)

4.2.13 Oznaka group

Oznaka group (hrv. *grupa*) kao što i sama riječ govori označava grupu ili sektor povezanih modela koji odgovaraju određenom sektoru kao što su naprimjer financije. Group također zahtjeva obavezan unos atributa name te može sadržavati oznaku doc koja označava dokumentaciju te grupe odnosno sektora. Treba napomenuti i podoznaku entity (hrv. *entitet*) koja također ima obavezni atribut name i služi kako bi se naveli svi entiteti koji pripadaju toj grupi. Oznaka group je podoznaka oznake variables. (OASIS, 2021.)

Slika 14: Primjer oznake group XMILE formata

```
<group name="Financial Sector">
   <doc>
     The operation of the Finance department is modeled in
     this sector.
   </doc>
   <entity name="Cumulative Revenue"/>
   <entity name="revenue"/>
   ...
</group>
```

Izvor: OASIS, 2021.

Group ima i jedan opcionalni atribut run koji je tipa boolean. On služi za definiranje da li je taj model na koji se oznaka run odnosi poželjno simulirati ili nije.

Zadana vrijednost booleana run je neistinita odnosno ne možemo pokrenuti simulaciju tog modela. (OASIS, 2021.)

4.2.14 Oznaka module

Oznaka module (hrv. *modul*) služi kakao bi se unutar bloka naredbi koje omeđuje oznaka variable rezervirala mjesta za submodele. Submodel (hrv. *podmodel*) se nalazi unutar svoje oznake modela. Podmodeli se nalaze unutar iste datoteke u kojoj se nalazi i korijenski model ili drugi podmodeli koji su nadređeni tom podmodelu. Podmodel mora uključivati atribut name, osim ako se ne nalazi u zasebnoj datoteci. Tada podmodel ne mora sadržavati atribut name iz razloga jer se taj podmodel može promatrati kao zasebni model. Ako podmodel želimo koristiti u bilo kojem nadređenom modelu ili korijenskom modelu mora biti prezentiran u bloku naredbi variables pomoću oznake module. Oznaka module sadrži dva atributa i jedno svojstvo. Oznaka name je obavezna ako nije korištena opcionalna oznaka resource. Oznaka name mora biti jednaka oznaci koja je korištena kod atributa name podmodela kojeg želimo koristiti u modelu. Ako se ne može pronaći identični podmodel koji bi odgovarao atributu name tada se ignorira traženje podmodela, a njegovu ulogu preuzima ne imenovani model odnosno korijenski model unutar referencirane datoteke. Atribut resource je ne opcionalan atribut koji se koristi samo ako je podmodel koji se traži unutar neke druge datoteke. U tom slučaju se unutar atributa resource navodi potpuna ili relativna putanja do datoteke ovisno o načinu pohrane datoteka. Postoji i treća opcija u kojoj se unosi URL putanja. Svojstvo koje se koristi unutar oznake module je oznaka <connect to="…" from="…"/>. Atribut to predstavlja ime inputa modula unutar podmodela ili njegovih podmodela. Atribut from predstavlja ime outputa podmodela koje se dodjeljuje inputu zadanom oznakom connect. Također, kod modula je važno napomenuti da je svaki modul jedan level hijerarhije iznad podmodela na kojeg se referira. Slike na sljedećoj stranici prikazuju primjer koda u kojem se koriste oznake module i submodel. Slika 15. pokazuje kod koji se odnosi na poziv podmodela iz iste datoteke, a slika 16 prikazuje poziv podmodela iz druge XMILE datoteke. (OASIS, 2021.)

Slika 15: Primjer oznake module koji se referencira na submodel unutar iste datoteke

```
<?xml version="1.0" encoding="utf-8" ?>
<xmile version="1.0" xmlns="http://docs.oasis-open.org/xmile/ns/XMILE/v1.0">
   <header>
     <product version="1.0">Product</product>
     <vendor>sample</vendor>
     <options>
        <uses_submodels/>
     </options>
   </header>
   <sim_specs method="Euler" time_units="Months">
     <start>1</start>
     <stop>13</stop>
     <dt>0.25</dt>
   </sim_specs>
   <model>
     <variables>
        <module name="Sub_Model">
           <connect to="Input" from=".Root_Model_Output"/>
           <connect to=".Root_Model_Input" from="Sub_Model.Output"/>
        </module>
        <aux name="Root_Model_Output" access="output"/>
        <aux name="Root_Model_Input" access="input"/>
     </variables>
   </model>
   <model name="Sub_Model">
     <variables>
        <aux name="Input" access="input"/>
        <aux name="Output" access="output"/>
     </variables>
   </model>
</xmile>
```

Izvor: OASIS, 2021.

Slika 16:Primjer oznake module koja se referencira na podmodel unutar druge datoteke

```
<?xml version="1.0" encoding="utf-8" ?>
<xmile version="1.0" xmlns="http://docs.oasis-open.org/xmile/ns/XMILE/v1.0">
   <header>
     <product version="1.0">Product</product>
     <vendor>sample</vendor>
     <options>
        <uses_submodels/>
     </options>
   </header>
   <sim_specs method="Euler" time_units="Months">
     <start>1</start>
     <stop>13</stop>
     <dt>0.25</dt>
   </sim_specs>
   <model>
     <variables>
        <module name="Sub_Model" resource="module.xml">
           <connect to="Input" from=".Root_Model_Output"/>
           <connect to=".Root_Model_Input" from="Sub_Model.Output"/>
        </module>
        <aux name="Root_Model_Output" access="output"/>
        <aux name="Root_Model_Input" access="input"/>
     </variables>
   </model>
</xmile>
```

Izvor: OASIS, 2021.

4.2.15 Oznaka macro

Oznaka macro (hrv. *makro*) služi kako bi se korisniku XMILE formata omogućilo kreiranje svojih naredbi unutar XMILE formata. Također, pomoću oznake macro mogu se kreirati i nove oznake XMILE formata koje naprimjer mogu služiti prilikom prevođenja XMILE formata u neki drugi paket za obradu SD modela kao što je i ksdtoolkit. Mi nećemo koristiti makro naredbe u tu svrhu, ali ćemo implementirati neka svojstva te oznake u našem prijevodu u ksdtoolkit. Oznaka macro nalazi se van svih standardnih blokova naredbi i iste je razine kao i ostale glavne naredbe te može stajati zasebno u XMILE formatu s oznakom header. To ustvari znači da makro može biti zasebna XMILE datoteka kojom se šalju samo određene odnosno željene funkcije koje proširuju mogućnosti XMILE formata. Oznaka macro mora obavezno sadržavati atribut name i oznaku eqn. Atribut name označava ime makro naredbe, a oznaka eqn označava neki željeni XMILE izraz ili funkciju. Oznaka macro sadrži i neke opcionalne oznake kao što su: parm, format, doc, sim_spec, dt, stop, variables, view. Kao što vidite neke od tih oznaka smo objasnili u prethodnim poglavljima tako da se ne ćemo sada detaljnije baviti njima. Oznaka macro ima i jedan opcionalni atribut namespace koji smo također objasnili u prethodnim poglavljima. Moramo još samo napomenuti da oznaka macro može sadržavati i podmodele. (OASIS, 2021.)

Slika 17: Primjer oznake macro XMILE formata

```
<macro name="FIND">
   <parm>A</parm>
   <parm>value</parm>
   <parm default="1">i</parm>
   <eqn>IF i > SIZE(A) THEN 0 ELSE
     IF A[i] = value THEN i ELSE FIND(A, value, i + 1)</eqn>
   <format>FIND(<array>, <value>[, <starting index>])</format>
</macro>
```

Izvor: OASIS, 2021.

4.2.16 Oznaka options

Kao što smo naveli u poglavlju 4.2.4 kod oznake dimensions ovo je zadnje poglavlje o oznakama XMILE formata. Oznaka options (hrv. *opcije*) je oznaka koja sadrži listu funkcionalnosti koje se koriste u toj XMILE datoteci. Ova oznaka ne mora biti uključena u svim XMILE datotekama i može sadržavati proizvoljan broj funkcionalnosti koje su korisniku potrebne. Ako se neka od tih funkcionalnosti koristi u XMILE formatu nužno ju je navesti unutar bloka naredbi oznake options. Oznaka options sadrži sljedeće funkcionalnosti odnosno podoznake: uses_conveyor, uses_queue, uses_arrays, uses_submodela, uses_macros, uses_event_posters, has_model_view, uses_outputs, uses_inputs, uses_annotation. Kao što vidimo iz većine naziva ovih oznaka može se zaključiti čemu služe i koje su njihove funkcionalnosti tako da ih ne ćemo dodatno pojašnjavati. Na slici 18 koja se nalazi odmah ispod ovog teksta je prikazan jedan primjer bloka naredbi oznake options. (OASIS, 2021.)

Slika 18: Primjer oznake options XMILE formata

```
<options namespace="std, isee">
   <uses_conveyors leak="true"/>        <!-- has conveyors, some leak -->
   <uses_arrays maximum_dimensions="2"/>   <!-- has 2D arrays -->
   <has_model_view/>                    <!-- has diagram of model -->
</options>
```

Izvor: OASIS, 2021.

4.3. Usklađenost implementacije XMILE formata

Ovo poglavlje opisuje koje oznake i koje funkcionalnosti moraju biti obuhvaćene u implementaciji prevoditelja kako bi prevedeni XMILE format bio usklađen sa svim preostalim paketima i programima koji koriste SD modele i XMILE format. Mi ćemo se koncentrirati na bazični odnosno osnovni level interpretacije. To će nam biti polazna točka za eventualni daljnji razvoj prevoditelja. Da bi udovoljili osnovnom levelu usklađenosti morat ćemo u naš prevoditelj implementirati sve sljedeće oznake i funkcionalnosti XMILE formata. Sve važnije oznake smo detaljno objasnili u prethodnom poglavlju sada ćemo nabrojati samo one oznake koje su

potrebne za implementiranje osnovnog levela prevoditelja i njihova osnovna svojstva koja su potrebna. Prva oznaka koja mora biti implementirana u prevoditelju je oznaka XMILE. Ona mora sadržavati atribute version i atribut namespace koji označava koji se namespace XML-a koristi. Sljedeća oznaka koju moramo implementirati je oznaka header koja mora sadržavati podoznake vendor i product te broj verzije product-a odnosno verzije proizvoda. Nakon oznake header, implementacija mora sadržavati prevoditelj za oznaku model odnosno XMILE dokument mora sadržavati barem jednu oznaku model koja odgovara svim pravilima definiranim u poglavlju 4.2.7. U tom poglavlju je dodatno pojašnjeno sve vezano uz samu oznaku model tako da to ne ćemo dodatno pojašnjavati. Unutar naše implementacije prevoditelja XMILE formata trebat će implementirati mogućnost imenovanja ostalih modela koji nisu korijenski model. Naš prevoditelj morat će biti sposoban prilikom čitanja XMILE formata rješavati nekonzistentnosti između više datoteka XMILE formata. Morat će poštovati pravila imenovanja odnosno pravila namespace-a koji se koristi. Sadržavat će i opcionalne mogućnosti koje se koriste unutar oznake options. Naravno to će vrijediti samo kada će se neka od tih opcionalnih mogućnosti koristiti unutar te XMILE datoteke. Poštovat će pravilo da svaka XMILE datoteka mora imati barem jednu sim_specs oznaku sa svim svojim obaveznim pod oznakama. Ova oznaka pojašnjena je u poglavlju 4.2.2. Podržavat će mogućnosti oznake behavior dodatno pojašnjene u poglavlju 4.2.5. Morat će poštovati mogućnost includa-a odnosno povezivanje većeg broja XMILE datoteka. Morat će poštivati sve bazične objekte funkcionalnosti tako da će svi bazični objekti funkcionalnosti moći biti korišteni i prevedeni u ksdtoolki i ponovo natrag u XMILE format. Prevoditelj će morati poštivati gramatiku za brojeve, varijable i izraze XMILE formata. I na kraju morat će sadržavati uobičajena sredstva varijabli, posebice imena, liste dimenzija i mogućnost ne apliciranja određene funkcionalnosti na sve elemente. Kao što se može zamjetiti unutar poglavlja 4.2. objasnili smo i neke oznake koje ovdje nisu navedene. Neke od tih oznaka možda i budu uključene u naš prevoditelj ovisno o vremenu koje bude utrošeno na samu implementaciju i kodiranje prevoditelja. (OASIS, 2021.)

5. Kotlin System Dynamics Toolkit (ksdtoolkit)

Ovo poglavlje će pojasniti strukturu samog ksdtoolkit-a, njegove funkcionalnosti i mogućnosti, moguće prostore za poboljšanja te koji su razlozi i kako se uopće došlo na samu ideju izrade ksdtoolkita. Nakon toga ćemo još pojasniti kako će i na koji način unutar tog postojećeg koda i sustava ksdtoolkit-a biti implementiran prevoditelj odnosno transipler.

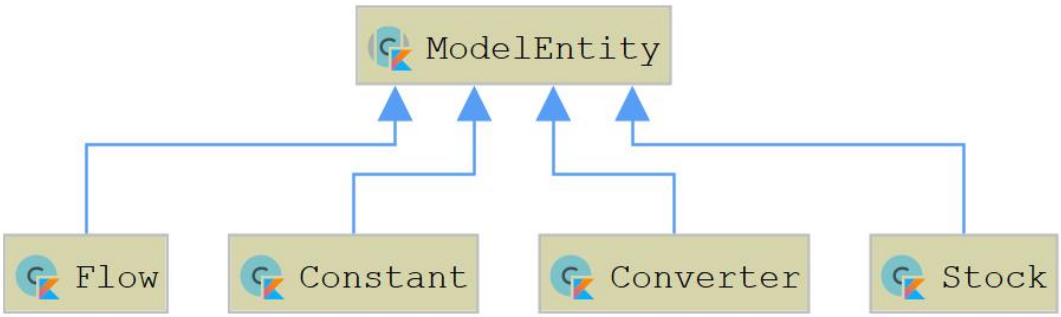
5.1. Razlozi kreiranja ksdtoolkit-a

Ksdtoolkit je kreiran zato što su trenutno ponuđena rješenja u brojnim programskim jezicima prikazala limitiranost u pojedinim segmentima. Primjer nekih jezika u kojima su napravljene implementacije SD modela su: C, C++, Java, Python i JavaScript. Kotlin se činio kao dobro rješenje iz razloga jer je dobro balansiran programski jezik koji sadrži brojne mogućnosti kao naprimjer: objektno-orijentirano programiranje, funkcijsko programiranje. Sadrži deklariranje tipova varijabli, a tako je olakšan pronalazak grešaka unutar programskog koda te ima funkcionalnost deklariranja posebne null varijable kojoj se uz ime tipa varijable dodaje upitnik kako bi se naznačilo da ta varijabla može biti jednaka null vrijednosti. Ovo su samo neke od prednosti koje nude mogućnost da Kotlin bude bolje rješenje u odnosu na gore navedene programske jezike. Također, jedan od važnih faktora je i taj što je Kotlin jedan od novijih programskih jezika. Autori ksdtoolkit-a nisu pronašli implementaciju SD modela u nekim od novih i modernijih programskih jezika koji imaju implementirane sve benefite koje sadrži Kotlin. Samim time potreba za ksdtoolkit-om je bila nužna. Treba istaknuti da je jedan od glavnih ciljeva bio i pokrivenost svih platforma odnosno tri najkorištenije platforme. To su android, web sučelje i desktop aplikacija. Tu mogućnost prema saznanju autora imaju samo tri implementacije SD modela: 1) System Dynamics JavaFramework, 2) sd.js, and 3) BPTK-Py. (Sovilj, Etinger, Sirotić, Pripužić, 2021.)

5.2. Struktura ksdtoolkit-a

Kao što smo već naveli u prethodnom poglavlju, ksdtoolkit je napisan u Kotlinu. To je programski jezik koji je nastao kao svojevrsno poboljšanje Jave, programskog jezika koji se u današnje vrijeme poglavito koristi u izradi aplikacija za android sustave. Za pisanje koda korišten je program IntelliJ IDEA Ultimate edition, verzija 2019.1. Trenutačno nije podržano korištenje GUI-a (eng. *Graphic User Interface*) tako da je za prikaz SD modela korišten Simantics System Dynamics 1.9.1 workbench. Unutar ksdtoolkit-a postoje četiri tipa entiteta modela: Stock, Flow, Constant, Converter i svi oni nasljeđuju apstraktnu klasu ModelEntity. ModelEntity sadrži atribute name (hrv. *ime*), current value (hrv. *trenutna vrijednost*) i previous value (hrv. *prethodna vrijednost*). Najvažnije svojstvo ove apstraktne klase je svojstvo tipa funkcije equation. To svojstvo omogućava definiranje jednadžbi modela kao lambda izraza koristeći pritom samo vitičaste zagrade. Na slici 19 je prikazan dijagram klasa.(Sovilj, Etinger, Sirotić, Pripužić, 2021.)

Slika 19: Dijagram klasa entiteta modela naslijeđenih iz apstraktne klase ModelEntity



Izvor: Sovilj, Etinger, Sirotić, Pripužić, 2021.

5.3. Funkcionalnosti i moguća poboljšanja ksdtoolkit-a

Ksdtoolkit primarno služi za kreiranje i simuliranje SD modela. To su mu ujedno i dvije najvažnije funkcionalnosti. On također omogućava eksportiranje dobivenih grafikona i slika u csv i png format. Neke od dodatnih mogućnosti koje bi trebalo importirati su naravno transpiler odnosno prevoditelj na čijoj implementaciji se bazira ovaj diplomski rad. Trenutno ne postoji grafičko sučelje za dizajniranje grafičkih modela. Isto tako ne postoji sustav koji bi upravljao scenarijima simulacije te nema kolekcije arhatipova i funkcija sistem-dinamike. Sve od navedenih funkcionalnosti bi se trebale implementirati u dogledno vrijeme. (Sovilj, Etinger, Sirotić, Pripužić, 2021.)

5.4. Implementacija transpilera u ksdtoolkit

Transpiler će se u *ksdtolkit* dodati naknadno unutar paketa pod nazivom transpiler. Načelno će se iz transpilera u ksdtoolkitu koristiti dvije funkcije. Jedna je *transpilerXmileToKsdtoolkit*, a druga *transpilerKsdtoolkitToXmile*. Transpiler će pomoću tih funkcija kreirati datoteke tipa .kt u slučaju prve navedene funkcije te .xmile ili .xml u slučaju druge navedene funkcije. Te kreirane datoteke će se spremati u predviđene pakete u *ksdtolkitu*.

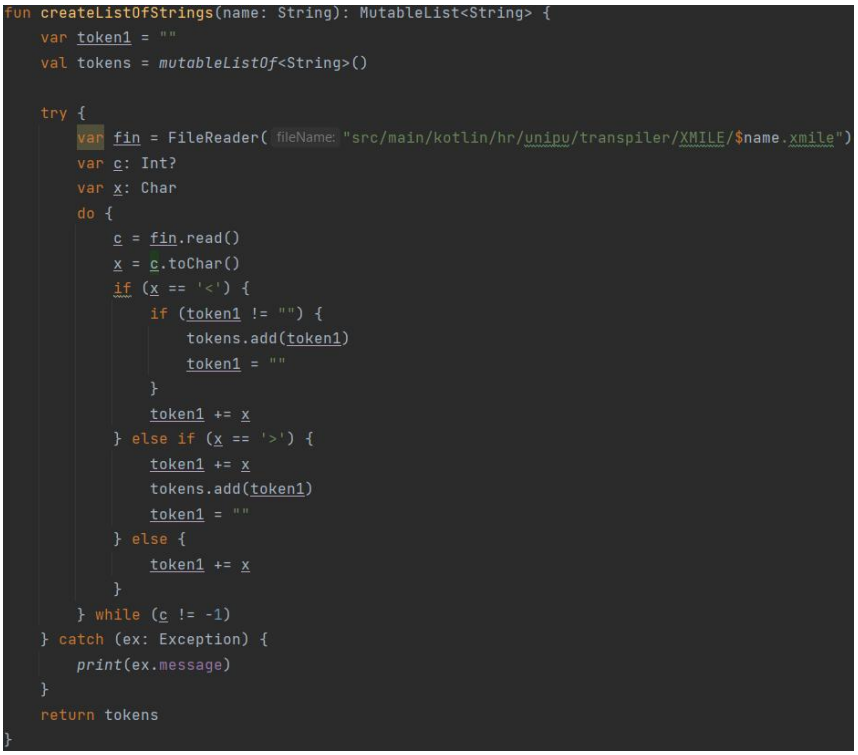
6. Izrada transpilera za ksdtoolkit

Ovo poglavlje bazirat će se na izradi programskog koda koji će se zatim integrirati u postojeći ksdtoolkit. U njemu će biti pojašnjeni svi problemi na koje se naišlo tijekom programiranja prevoditelja. Navest će se i poteškoće koje su se dogodile u početku izrade i definiranju strukture samog prevoditelja. Struktura prevoditelja će biti definirana u pet faza. Prva faza je dohvat podataka u kojoj se dohvaćaju podatci iz XMILE-a i spremaju u globalnu mapu *traspilerDataMap*. Druga faza je provjera jesu li svi obavezni podatci dohvaćeni. Treća faza je priprema tih podataka u oblik koji odgovara ksdtoolkitu. Naprimjer konstante moraju biti napisane velikim tiskanim slovima. Četvrta faza je postavljanje globalne mape podataka *traspilerDataMap* u novu globalnu mapu podataka *transpilerDataMapKsdToolkitSet* koja će biti u prikazu potrebnom za ksdtoolkit. Naprimjer *const val AREA_KEY = "AREA*" može biti jedan od kreiranih *stringova* pohranjenih kao varijabla nove globalne mape podataka. Peta faza će biti ispis podataka odnosno pohrana podataka u globalni *String* naziva transpilerString s kojim će se naknadno kreirati ksdtoolkit model. Također treba napomenuti da će za drugi smjer prevođenja; iz ksdtoolkita u XMILE; biti potrebna samo peta faza ispisa odnosno kreiranja globalnog stringa kako bi se ksdtoolkit preveo u XMILE format. Razlog tome je što su svi podatci koji se koriste prilikom kreiranja modela u ksdtoolkitu već dostupni i nema potrebe da ih se dohvaća ili radi neka veća prilagodba na njima. Na kraju ovog poglavlja će detaljnije biti pojašnjen *Base-Level Conformance* XMILE formata i razlozi zašto svi dijelovi istog nisu mogli biti do kraja implementirani, ali naravno ostavljen je prostor za moguću nadogradnju prevoditelja.

6.1. Kreiranje funkcija za manipulaciju stringovima i dohvat podataka

Ovo potpoglavlje opisuje početnu ideju, ali i napredak u razvoju samog koda iz perspektive spoznaja mogućnosti paketa koje pruža Kotlin. Prije početka kreiranja transpilera u svim prethodnim poglavljima, upoznali smo se sa svim potrebnim informacijama vezanim za strukture programskih jezika, gramatiku programskih jezika i pojmili smo što je to ustvari transpiler. Upoznali smo se s XMILE formatom, programskim jezikom Kotlin i ksdtoolkit-om vrstom DSL jezika koji služi za izradu SD modela. Sljedeći korak je bio krenuti kreirati prevoditelj. Logična opcija je bila prvo kreirati transpiler iz XMILE formata u ksdtoolkit jer je to zahtjevniji dio u prevođenju. Dohvat podataka znatno je teži iz razloga jer moramo dohvatiti podatke iz tekstualne datoteke tipa XMILE ili XML. Prvobitna ideja je bila da se prevoditelj u oba smjera kreira u zasebnom repozitoriju koji bi se naknadno lako prebacio u ksdtoolkit. Kreiranje prevoditelja započeto je kreiranjem funkcija koje bi manipulirale datotekom XMILE formata. On bi bio pretvoren u listu stringova funkcijom po imenu *createListOfStrings.* Ova funkcija kao ulazni parametar prima parametar *name* tipa *String* dohvaćen iz XMILE datoteke i vraća promjenjivu listu stringova. Koncept je bio zamišljen tako da se onda nad tom listom stringova radi dohvat podataka koji bi se pohranjivali u mapu. Prikaz i izgled funkcije *createListOfStrings* vidljiv je na slici 20.

Slika 20: Funkcija createListOfStrings

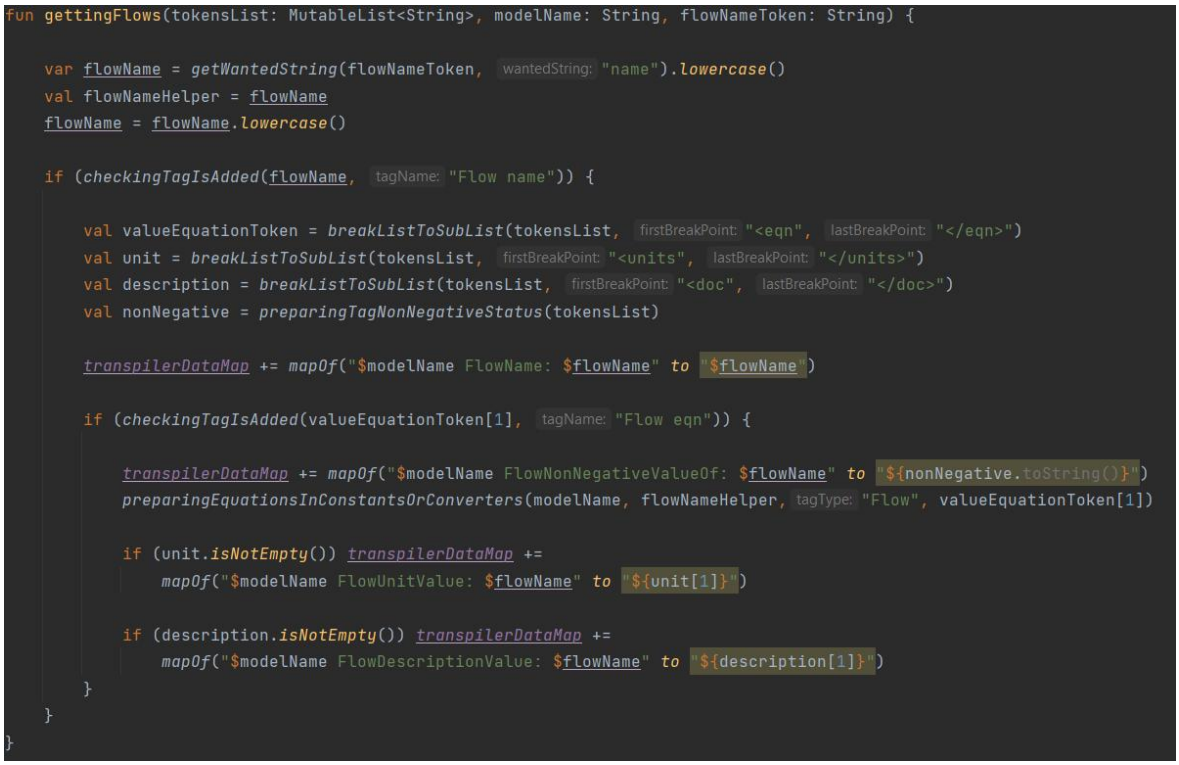


```
fun createListOfStrings(name: String): MutableList<String> {
    var token1 = ""
    val tokens = mutableListOf<String>()

    try {
        var fin = FileReader( fileName: "src/main/kotlin/hr/unipu/transpiler/XMILE/$name.xmile")
        var c: Int?
        var x: Char
        do {
            c = fin.read()
            x = c.toChar()
            if (x == '<') {
                if (token1 != "") {
                    tokens.add(token1)
                    token1 = ""
                }
                token1 += x
            } else if (x == '>') {
                token1 += x
                tokens.add(token1)
                token1 = ""
            } else {
                token1 += x
            }
        } while (c != -1)
    } catch (ex: Exception) {
        print(ex.message)
    }
    return tokens
}
```

Izvor: Izradio autor

Nakon što se pomoću funkcije *createListOfStrings* kreirala lista *stringova,* nad njom bi se radio niz operacija i koristio niz kreiranih funkcija. Prva funkcija je bila *removeWantedTagBlock.* Ova funkcija prima tri parametra list, firstBreakPoint i lastBrakPoint te vraća promjenjivu listu bez željene oznake i svih podoznaka te oznake. Ta funkcija je služila otklanjanju view oznake iz XMILE formata i olakšavala provjeru dohvaćenih podatka, ali i količinu podataka s kojom se treba raditi. Nakon te funkcije pozivala se skupina funkcija koja je za svoj cilj imala dohvatiti određene podatke. Te funkcije su: *gettingXMILETagData, gettingOptionsTagData, gettingHeaderTagData, gettingModelTagData, gettingModelsVariables, gettingModules, gettingAux, gettingFlows, gettingStocks,gettingSimSpecsTagData.* Te funkcije su kreirane kako bi ustvari dohvatile sve potrebne podatke koji trebaju za kreiranje modela i njegovih podmodula u ksdtoolkitu. U tekstu diplomskog rada nećemo prikazati sve funkcije već samo jednu od njih. To je funkcija *gettingFlows* kako bi demonstrirali izgled tog tipa funkcija.

Slika 21: Funkcija gettingFlows



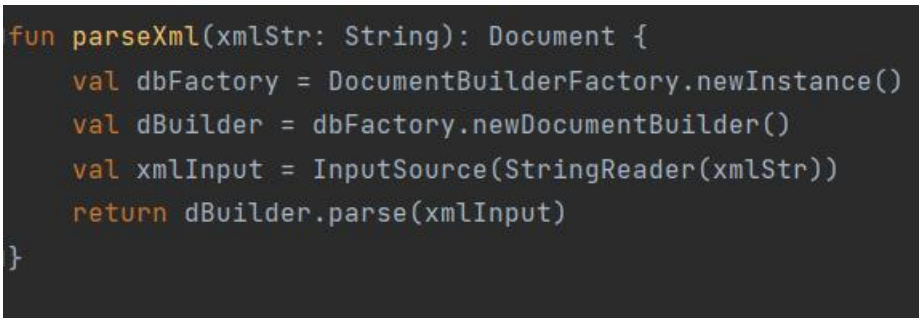
```
fun gettingFlows(tokensList: MutableList<String>, modelName: String, flowNameToken: String) {

    var flowName = getWantedString(flowNameToken,  wantedString: "name").lowercase()
    val flowNameHelper = flowName
    flowName = flowName.lowercase()

    if (checkingTagIsAdded(flowName,  tagName: "Flow name")) {

        val valueEquationToken = breakListToSubList(tokensList,  firstBreakPoint: "<eqn",  lastBreakPoint: "</eqn>")
        val unit = breakListToSubList(tokensList,  firstBreakPoint: "<units",  lastBreakPoint: "</units>")
        val description = breakListToSubList(tokensList,  firstBreakPoint: "<doc",  lastBreakPoint: "</doc>")
        val nonNegative = preparingTagNonNegativeStatus(tokensList)

        transpilerDataMap += mapOf("$modelName FlowName: $flowName" to "$flowName")

        if (checkingTagIsAdded(valueEquationToken[1],  tagName: "Flow eqn")) {

            transpilerDataMap += mapOf("$modelName FlowNonNegativeValueOf: $flowName" to "${nonNegative.toString()}")
            preparingEquationsInConstantsOrConverters(modelName, flowNameHelper,  tagType: "Flow", valueEquationToken[1])

            if (unit.isNotEmpty()) transpilerDataMap +=
                mapOf("$modelName FlowUnitValue: $flowName" to "${unit[1]}")

            if (description.isNotEmpty()) transpilerDataMap +=
                mapOf("$modelName FlowDescriptionValue: $flowName" to "${description[1]}")
        }
    }
}
```

Izvor: Izradio autor

Unutar bloka naredbi gore navedenih funkcija za dohvat, nalaze se i razne funkcije koje su kreirane kako bi dohvatile podatke SD modela. Neke od tih funkcija su: getWantedString, breakListToSublist, separateSameTags, preparingTagNonNegativeStatus, chackingTagIsAdded, preparingEquationsInConstantsOrConverters i brojne druge funkcije. Na slici 21 na kojoj je prikazana funkcija gettingFlows možemo vidjeti i neke od tih funkcija. Kao što se može zamijetiti ovaj postupak je dosta kompleksan i relativno nepregledan. Zbog toga kod testiranja unit testovima trebalo bi nam jako puno vremena da dobro istestiramo sve te funkcije za manipulaciju i dohvat podataka. Iz tog razloga bilo je potrebno tražiti alternativu ovome kodu. Pronađeni su paketi u Kotlinu kojima je olakšan cijeli proces dohvata podataka te testiranja funkcija koje to čine. Najvažniji paket podržan u Kotlinu koji omogućava te funkcionalnosti je *javax.xml.parsers.DocumentBuilderFactory.* Taj paket omogućava kreiranje DOM (eng. *Document Object Model*) stabla (eng. *tree*). To stablo se može izgenerirati iz dokumenata odnosno datoteke tipa XML. Također tri važna paketa koji pomažu u

kreiranju, ali i s kasnijom uporabom su i *org.w3c.dom.Element, org.w3c.dom.Document, import org.xml.sax.InputSource.* Svi ti paketi se koriste unutar funkcije koja se naziva *parseXml* ili kasnije prilikom rada na dohvatu podataka. Funkcija *parseXML* dohvaća varijablu naziva xmlStr tipa *String* i vraća *Dokument*. Prikaz te funkcije je vidljiv na slici 22.

Slika 22: Funkcija parseXML



Izvor: Izradio autor

Kao što je vidljivo iz priloženog na gornjoj slici jako mali broj linija je bio potreban kako bi se kreiralo stablo koje će se kasnije koristiti za dohvat podataka iz XMILE-a. Samim time to je logičniji odabir za završnu verziju koda prevoditelja. Primjer funkcije koja se koristi za dohvat podataka iz kreiranog stabla te ekvivalentna funkcija funkciji *gettingFlows;* koja se nalazi na 21. slici; je funkcija *getFlows.* Razlog iz kojeg su navedene samo ove dvije ekvivalentne funkcije je taj što funkcije za dohvat ostalih podataka funkcioniraju na isti način kao i te dvije funkcije. Funkcija *getFlows* kao ulazni parametar dohvaća element tipa *Element* i modelName tipa *String* te kao izlazni parametar vraća mapu podatka. Funkcija *getFlows* ne mora nužno vraćati ništa jer se vrijednosti koje dohvaća pohranjuju u globalnu mapu *transpilerDataMap* u kojoj se nalaze svi podatci iz svih ostalih funkcija koje dohvaćaju podatke. Razlog iz kojeg je odlučeno da svaka od funkcija za dohvat ima povratnu vrijednost je taj što to dosta olakšava testiranje samih funkcija unit i integracijskim testovima. Na slici 23 može se vidjeti koliko je ustvari olakšan dohvat podataka i koliko je funkcija *getFlows* ustvari preglednija u odnosu na funkciju *gettingFlows.* Funkcija *getFlows* koristi samo dvije pod funkcije koje koriste DOM stablo, elemente i čvorove za dohvat podataka. To su *getName i getWantedTagValue.* Te dvije funkcije će biti potrebno testirati unit testovima, ali to

je značajno manji broj unit testova u odnosu na broj unit testova koji bi morao biti kreiran prilikom testiranja funkcije *gettingFlows*.

Slika 23: Funkcija getFlow

```
fun getFlows(element: Element, modelName: String): Map<String, String?> {
    val flow = element
    val name = getName(flow, tagName: "Flow")
    if (name.isEmpty()) error("Flow 'name' attribute is empty")

    val eqn = getWantedTagValue(flow, tagIdentifier: "eqn")
    val unit = getWantedTagValue(flow, tagIdentifier: "units")
    val description = getWantedTagValue(flow, tagIdentifier: "doc")

    transpilerDataMap += mapOf("$modelName FlowName: $name" to "$name")
    if (eqn != null) {
        prepareEquationsInConstantsOrConverters(modelName, name, tagType: "Flow", eqn)
    }
    if (unit != null) transpilerDataMap += mapOf("$modelName FlowUnitValue: $name" to "$unit")
    if (description != null) transpilerDataMap += mapOf("$modelName FlowDescriptionValue: $name" to "$description")

    return mapOf("name" to name, "eqn" to eqn)
}
```

Izvor: Izradio autor

Kako bi prikazali i dohvat podataka pomoću DOM stabla. Prikazana će biti i funkcija *getName.* Ona koristi funkcije iz gore navedenih paketa *org.w3c.dom.Element, org.w3c.dom.Document*. Funkcija getName je ujedno i najniža razina apstrakcije do koje će se ići kako bi kreirali parser. Za razliku od prvotne ideje po kojoj bi se morali spustiti još jednu razinu apstrakcije niže do načina na koji se dohvaćaju podatci iz same liste stringova. Ovako se možemo usredotočiti samo na to da li funkcija *getName* uistinu dohvaća imena podataka.

Slika 24: Funkcija getName

```
fun getName(element: Element, tagName: String): String {
    if (!element.hasAttribute( name: "name")) error("$tagName missing 'name' attribute")
    return transformName(element.getAttribute( name: "name"))
}
```

Izvor: Izradio autor

6.2. Provjera podataka

U ovoj fazi se provjerava jesu li svi podatci koji moraju biti uneseni po propisima *Base-Level Conformancea* doista uneseni i poštuju li se pravila XMILE dokumentacije. Provjera podataka se radi unutar funkcija za dohvat objašnjenih u prethodnom poglavlju. Razlog tome je taj što u slučaju da neki podatak ili tag nedostaje, želimo korisniku odmah ukazati da to treba popraviti prije nego što krenemo u daljnje faze prevođenja. Korisniku se u slučaju krivo konfiguriranog XMILE dokumenta vraća *error* poruku koja ukazuje u kojoj je fazi i zbog čega došlo do greške. Ova faza sastoji se od provjera koje su zapisane unutar datoteke checkTranspilerData.kt. U nastavku ćemo navesti par funkcija koje provjeravaju podatke, ali važno je napomenuti da nisu sve funkcije koje se bave ovom fazom prevođenja izvučene u zasebnu funkciju nego se neke od jednostavnijih provjera ove faze mogu odraditi samo s if, else if ili else izjavama (eng. *statement*). Primjer jedne od check funkcija je funkcija *checkModelName* koja provjerava jesu li imena modela unesena po svim propisima *Base-Level Conformancea*. Ona dohvaća *Element* i *Int* vrijednost i vraća *String* ako su svi uvjeti zadovoljeni. U protivnom korisniku vraća *error* grešku koja mu ukazuje da nešto kod unosa imena modela nije bilo dobro napravljeno u XMILE dokumentu.

Slika 25: Funkcija checkModelName

```
fun checkModelName(element: Element, i: Int): String {
    var modelName = element.getAttribute( name: "name")
    if (!element.hasAttribute( name: "name") && i > 0) error("Model missing 'name' attribute")
    else if (element.hasAttribute( name: "name") && i == 0) error("Root model MUST NOT have attribute 'name'.")
    else if (!element.hasAttribute( name: "name") && i == 0) modelName = modelRootNameConst
    if (modelName == null) error("Model 'name' attribute is empty!")

    return modelName
}
```

Izvor: Izradio autor

Funkcija getXmileTagData prikazuje na koji način funkcioniraju jednostavni ukomponirani *if statementi* u fazi provjere željenih podataka. Na slici 26 je prikazana ta funkcija. Funkcija dohvaća vrijednost tipa *Element* i vraća *error* u slučaju da jedan od željenih podataka nije unesen. Također, možemo primijetiti da

u slučaju da su svi željeni podatci uneseni funkcija ih sprema u globalnu mapu *transpilerDataMap*.

Slika 26: Funkcija getXmileTagData

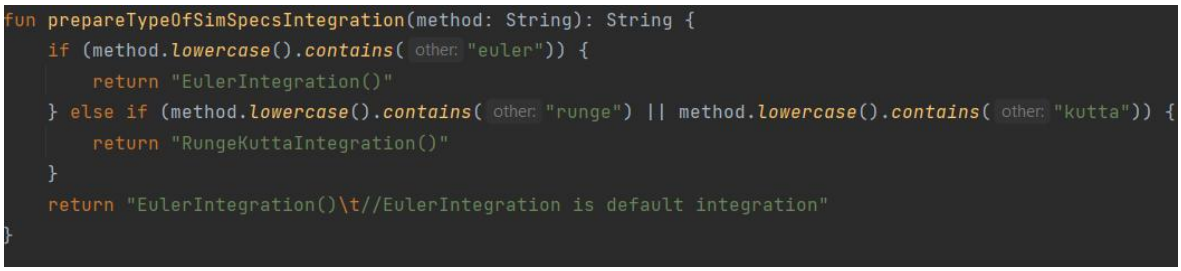
```
fun getXmileTagData(element: Element) {
    if(!element.hasAttribute( name: "version")) error("XMILE needs to have attribute version!")
    if(!element.hasAttribute( name: "xmlns")) error("XMILE needs to have attribute xmlns!")

    val version = element.getAttribute( name: "version").toString()
    val xmlns = element.getAttribute( name: "xmlns").toString()

    if(version == "") error("XMILE version must be added in XMILE tag.")
    if(xmlns == "" ) error("XMILE namespace must be added in XMILE tag.")

    transpilerDataMap += mapOf("Version" to version)
    transpilerDataMap += mapOf("xmlns" to xmlns)
}
```

Izvor: Izradio autor

6.3. Priprema podataka

Faza pripreme podataka je također usko povezana s fazom dohvata podataka. Podatci prije pohrane u globalnu mapu *transpilerDataMap* moraju biti pripremljeni u željeni oblik *ksdtoolkita*. Već smo prethodno spomenuli da se u *ksdtoolkitu* nazivi *stocka* i *modula* označavaju s prvim velikim početnim slovom. Nazivi konstanta su označeni svim velikim slovima, a nazivi konvertera i *flowova* se označavaju svim malim slovima. Međutim to nije jedina priprema podataka koja je potrebna kako bi se svi željeni uvjeti *ksdtoolkita* zadovoljili. Naprimjer jedna od funkcija koja priprema podatke je *prepareTypeOfSimSpecsIntegration.* Ta funkcija kao ulazne parametre sadrži varijablu method tipa *String* te vraća vrijednost *String* koja je izmijenjena u skladu sa željenim zapisom methode u *ksdtoolkitu*. Primjer te funkcije je vidljiv na slici 27.

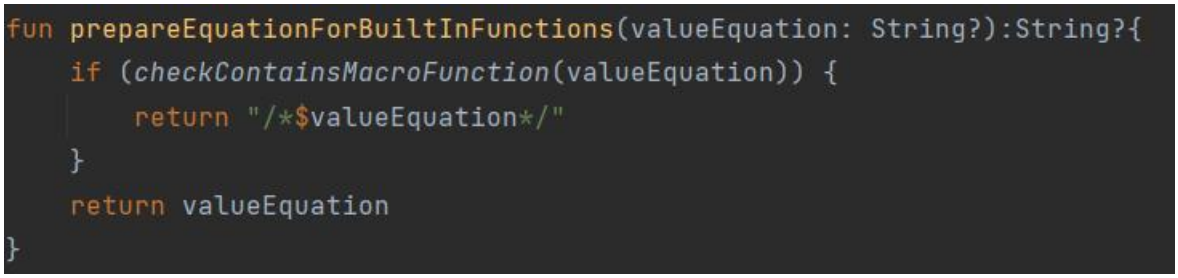
Slika 27: Funkcija prepareTypeOfSimSpecsIntegration



Izvor: Izradio autor

Faza pripreme podataka je usko povezana s fazom dohvata podataka jer podatci prije pohrane u globalnu mapu *transpilerDataMap* moraju biti pripremljeni u željenom formatu za ksdtoolkit. Također, treba napomenuti da se unutar funkcija koje pripremaju podatke, nalaze i funkcije iz faze dva provjere podataka. Jedna od tih funkcija koja sadrži funkcije provjere je i funkcija *prepareEquationForBuiltInFunctions.* Ova funkcija služi tome da se provjere svi podatci izvučeni iz eqn oznake i u slučaju da sadrže neku od built-in funkcija koja trenutno nije podržana u ksdtoolkitu, a zastupljena je u XMILE formatu. Ta funkcija će zakomentirati vrijednost dohvaćenu iz eqn oznake jer ksdtoolkit neće imati mogućnost da tu vrijednost direktno prevede. Ova funkcija je prikazana na slici 28.

Slika 28: Funkcija prepareEquationForBuiltInFunctions



Izvor: Izradio autor

Funkcija *prepareEquationForBuiltInFunctions* prima parametar naziva *valueEquation* tipa *String?.* Upitnik označava da *string* može biti i vrijednosti *null*. Funkcija pripreme vraća tip podatka *String?,* a vidljivo je i kako u slučaju da funkcija *checkContainesMacroFunction* vrati vrijednost *true* te se zadovolji *if* uvjet funkcija pripreme će vratiti zakomentiranu vrijednost. S ova dva primjera funkcija

pripreme podataka smo zaokružili cijelu treću fazu koja se odnosi na pripremu podataka.

6.4. Postavljanje podataka

Četvrta faza pripreme podataka odnosi se na postavljanje podataka u novu globalnu mapu *transpilerDataMapKsdToolkitSet* iz koje će se poslije prilikom faze printanja te iste podatke pohraniti željenim redoslijedom u globalnu varijablu *transpilerString*. U ovom potpoglavlju ćemo prikazati dvije funkcije koje se koriste prilikom postavljanja podataka u mapu. Prvo ćemo prikazati dodavanje vrijednosti koje se odnose na konstantne vrijednosti unutar *companion objecta,* a u drugom primjeru postavljanje vrijednosti sistemskih elemenata unutar modela.

Slika 29: Funkcija setCompanionObjects

```
fun setCompanionObjects(name: String, modelName: String, equationValueConstant: String?) {
    if (equationValueConstant != null) {
        transpilerDataMapKsdToolkitSet +=
            mapOf(
                "$modelName $name companion object keys" to
                        "const val ${name.uppercase()}" + "_KEY = \"${name.uppercase()}\""
            )
        transpilerDataMapKsdToolkitSet +=
            mapOf(
                "$modelName $name companion object values" to
                        "const val ${name.uppercase()}" + "_VALUE = $equationValueConstant"
            )
    }
}
```

Izvor: Izradio autor

Na slici 29 je prikazana *funkcija setCompanionObject*. Ona ima parametre *name* tipa *String*, *modelName* tipa *String* te *equationValueConstant* tipa *String?.* Njena primarna svrha je odrediti da li postoji *equationValueConstant* i ako postoji dodati dva podatka u mapu *transpilerDataMapKsdToolkitSet*. Može se zamijetiti da funkcije za postavljanje podataka nisu prekompleksne. Važno je još napomenuti da je prva String vrijednost u mapi ustvari ključ pomoću kojeg ćemo prilikom printanja ustvari dohvaćati drugi *String,* a on je podatak koji će se dodavati u globalni *String transpilerString* u zadnjoj fazi printanja. Na slici 30 je prikazana

funkcija *setSystemElements* koja je malo kompleksnija od funkcije *setCompanionObject,* ali se također zasniva na istom principu postavljanja podataka.

Slika 30: Funkcija setCompanionObjects

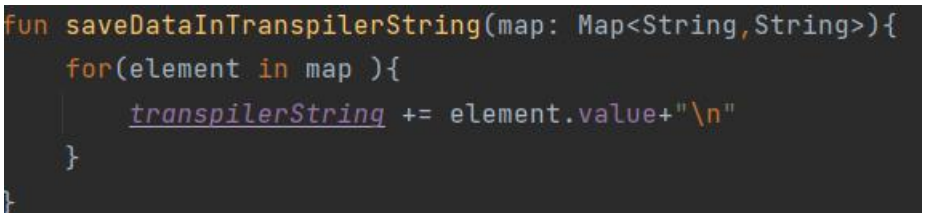
```
fun setSystemElements(
    name: String, modelName: String, tagType: String, equationValueConstant: String?,
    equationValueConverter: String?
) {
    if (equationValueConstant != null && tagType == "Aux") transpilerDataMapKsdToolkitSet +=
        mapOf(
            "$modelName $name 2. Constants" to
                    "val ${name.uppercase()} = model.constant(${name.uppercase()}" + "_KEY)"
        )
    if (tagType == "Stock")
        transpilerDataMapKsdToolkitSet += mapOf("$modelName $name 2. Stocks"
                to "val ${name.lowercase().replaceFirstChar { it.uppercase()}} = model.stock(\"$name\")")
    if (tagType == "Flow") transpilerDataMapKsdToolkitSet +=
        mapOf("$modelName $name 2. Flows" to "val ${name.lowercase()} = model.flow(\"$name\")")
    if (equationValueConverter != null && tagType == "Aux") transpilerDataMapKsdToolkitSet +=
        mapOf("$modelName $name 2. Converters" to "val ${name.lowercase()} = model.converter(\"$name\")")

}
```

Izvor: Izradio autor

6.5. Ispis podataka

Faza ispis podataka radi samo spremanje podataka u globalnu varijablu *transpilerString* tipa *String*. Podatci se pohranjuju željenim redoslijedom kako bi se na propisan način kasnije pomoću transpilerStringa kreirala kotlin datoteka po propisima *ksdtoolkita*. U ovom potpoglavlju su prikazane dvije funkcije: *saveDataInTranspilerString* te *transpilerStringCreateAllSystemElements.* Prva funkcija je ustvari funkcija koja izvršava ulogu svojevrsne for petlje za sve podatke koji su filtrirani iz globalne mape *transpilerDataMapKsdToolkitSet*. Ti podatci su prepoznati kao željeni podatci za taj dio ispisa i pohranjuju se tim željenim redom u varijablu *transpilerString.* Ova funkcija je prikazana na slici 31.

Slika 31:Funkcija saveDataInTranspilerString



Izvor: Izradio autor

Funkcija *transpilerStringCreateAllSystemElements* je ustvari svojevrsna šablona (eng. *template*) koja se koristi kako bi se sistemski elementi unutar *stringa* poredali željenim redom. Na slici 32 možemo vidjeti od čega se sve sastoji ta funkcija i možemo zamijetiti da poziva funkciju *saveDataTranspilerString*. Prikazane su samo ove dvije funkcije jer su kao i u prethodnim fazama ostale funkcije jako slične prikazanim funkcijama i nema svrhe sve ih navoditi i ponavljati načelno iste stvari. Drugi smjer prevođenja radi se na sličan način kao i ovaj ispis samo podatci neće biti dohvaćeni iz mape nego iz klasa. Funkcije za ispis i kreiranje stringa bit će načelno iste samo s promijenjenim string vrijednostima koje će u ovom slučaju biti prilagođene XMILE formatu.

Slika 32:Funkcija transpilerStringCreateAllSystemElements

Izvor: Izradio autor

6.6. Base-Level Conformance

Base-Level Conformance je osnovna razina prilagođenosti koja mora postojati kako bi se na neki prevoditelj, kompajler ili interpreter gledalo kao da zadovoljava i odgovara normama XMILE formata. Postoji 13 točaka *Base-Level Conformancea*. Nažalost neke od tih točaka neće moći biti zadovoljene jer *ksdtoolkit* još uvijek nema sve funkcionalnosti koje ima XMILE format. Naprimjer *ksdtoolkit* još uvijek nema sve *built-in* funkcije koje podržava XMILE format. Treba uzeti u obzir da XMILE postoji već dugi niz godina i postepeno je dolazio do ove faze u kojoj se trenutno nalazi. Poanta ovog prevoditelja pogotovo iz smjera XMILE prema *ksdtoolkitu* bila je napraviti istu takvu mogućnost i za ksdtoolkit. To je mogućnost da se postepeno sve više prilagođava XMILE formatu. Naprimjer možebitni logični nastavak *ksdtoolkita* u smjeru prilagođavanja XMILE formatu bi možda mogao ići prema oznaci *behavior*. *Behavior* je vrsta oznake XMILE formata koja služi tome da se određena varijabla definira kao naprimjer ne negativna (eng. *non negative*) varijabla. Naprimjer neki *stock* ne bi mogao poprimiti negativnu vrijednost. Također postoji i oznaka options koji naglašava koje se opcionalne funkcionalnosti koriste unutar tog XMILE dokumenta. Trenutno jedina opcionalna funkcionalnost koji ksdtoolkit podržava je korištenje podmodela (eng. *uses submodels*). Dobra stvar je što se pomoću gore navedenih pet faza znatno olakšalo proširenje mogućnosti prevoditelja odnosno olakšano je dodavanje novih značajki unutar prevoditelja. Prva ne refaktorirana verzija koja nije koristila DOM stablo i *xmlparser* bila je znatno ne prilagođenija promjenama. Također, ne refaktorirana verzija nije sadržala globalne varijable i imala je puno ne prilagođeniju funkciju za ispis. S globalnim varijablama i korištenjem funkcije *filterMap* prilikom ispisa dobio se lakše nadogradiv, pregledniji i jednostavniji kod.

7. Testiranje

Ovo poglavlje pojasnit će što su to jedinični (eng. *unit*) testovi te što su integracijski testovi i kako se testirao kreirani kod. Bavit će se pojašnjavanjem razloga iz kojih su unit i integracijski testovi nužni prilikom testiranja, ali i same izrade koda.

7.1. Jedinični (eng. *Unit*) testovi

Jedinični testovi služe tome kako bi se svaka funkcija najniže razine unutar koda provjerila jednostavnim testovima. Tako se možemo uvjeriti da svaka od tih funkcija koju smo kreirali doista radi ono za što je kreirana. Jedinični testovi su pouzdani iz razloga jer se mogu raditi paralelno s kreiranim funkcijama odnosno prilikom programiranja koda. Taj postupak se naziva TDD (eng. *test driven development*). Također, jedinični testovi se mogu raditi i nakon kreiranja programskog koda kako bi se provjerilo radi li on doista ono za što je kreiran. Na slici 33 je prikazana jednostavna funkcija *checkAlphaOrUnderscore* i njen test *checkAlphaOrUnderscoreTest.*

Slika 33:Test checkAlphaOrUnderscore

```
fun checkAlphaOrUnderscore(c: Char): Boolean {
    return c.isLetter() || c == '_'
}

@Test
fun checkAlphaOrUnderscoreTest() {
    //Test if parameter char=a -> return true
    assertTrue(checkAlphaOrUnderscore( c: 'a'))
    //Test if parameter char=_  -> return true
    assertTrue(checkAlphaOrUnderscore( c: '_'))
    //Test if parameter char=% -> return false
    assertFalse(checkAlphaOrUnderscore( c: '%'))
}
```

Izvor: Izradio autor

Slika 33 prikazuje funkciju i njene testove koji su uspješno izvršeni i samim time ukazuju da ta funkcija nema grešaka i izvršava zadatak za koji je kreirana. Njen zadatak je u slučaju da je unesena vrijednost podcrta ili slovo vratit vrijednost istine (eng. *true),* a u slučaju da je bilo koji drugi znak u pitanju vratiti vrijednost ne istine (eng*. false*).

7.2. Integracijski testovi

Integracijski testovi su testovi koji agregiraju te provjeravaju funkcije koje pozivaju više funkcija unutar svog bloka naredbi. Najčešće te funkcije koje se provjeravaju integracijskim testovima također sadrže i funkcije koje su provjerene jediničnim testovima. Tako se obuhvaćaju sve razine unutar koda. Veći broj jediničnih testova može se nazvati sistemski test. Granica kada se integracijski test može nazvati sistemski nije baš najjasnije naznačena zato što neki integracijski testovi mogu biti i sistemski testovi nekog manjeg sistema. Cilj integracijskih i jediničnih testova prilikom kreacije prevoditelja je smanjiti mogućnost grešaka u kodu. Zato su testirane funkcije iz svih faza prevođenja. Svi testovi neće biti navedeni u diplomskom radu, ali će biti vidljivi unutar *ksdtoolkita* koji je objavljen na github repozitoriju.

8. Zaključak

U izradu diplomskog rada krenuo sam korektno upoznat s tematikom istog. Vođen razmišljanjem da bi taj diplomski rad trebao biti sličan izradi programskog jezika; moje prvotne ideje za temu diplomskog rada. Pokazalo se da je tematika ovog rada vrlo zanimljiva i da teme SD modela sežu daleko u prošlost čak do pedesetih godina prošlog stoljeća. Došao sam do saznanja da SD modeli obuhvaćaju širok spektar tema u ljudskom životu. Mogu donijeti uvid iz totalno drugačije perspektive nego što je to uobičajeno. Prevoditelj je doista sličan izradi programskog jezika, ali možda i u nekim aspektima i lakši, iz razloga jer se u slučaju ksdtoolkita i XMILE formata nisam morao zamarati kreiranjem svoje gramatike i razno raznih problema na koje bih možda naišao u tom slučaju. Prvi cilj mi je bio čim bolje se upoznati sa svim aspektima odnosno barem onim glavnim aspektima koji su vezani uz moj diplomski rad. Tu prvenstveno mislim na programski jezik Kotlin koji sam do tad poznavao kao svojevrsnu novu verziju Jave. SD modele koji su mi bili važni kako bih uopće spoznao koja je njihova svrha i zašto ustvari kreiram prevoditelj. Mislim na XMILE format koji je poprilično kompleksan i ima brojne aspekte koji su važni kako bi prevoditelj dobro funkcionirao. Također bilo je važno dobro razumjeti značenje samog prevoditelja i njegovu razliku u odnosu na interpreter i kompajler. Nakon svega toga krenuo sam u kreiranje koda gdje sam se par puta našao u situaciji da sam morao refaktorirati kod jer sama struktura koda nije bila optimalna, onakva kakvu sam zamislio u tom trenutku. Kod je značajno pojednostavljen u odnosu na prve verzije i uvjeren sam da je puno veća mogućnost nadogradnje sada nego što je to bilo u početku. Kao dobru možebitnu nadogradnju ksdtoolkita, ali i samog prevoditelja nekako mi se najviše ističe izgradnja grafičkog sučelja u ksdtoolkitu te dodatak u prevoditelj koji bi potom dohvatio podatke iz oznake *view* XMILE dokumenta. To bi definitivno olakšalo korisnicima korištenje ksdtoolkita.

Literatura

1. Božikov J., (2006.), Modeliranje i simulacija, dostupno 12.08.2021. na https://bib.irb.hr/datoteka/347082.modeliranje_i_simulacija_-_v2a2.pdf

2. Chichakly K., (2013.), XMILE – An open standard for system dynamics models,dostupno 28.08.2021 na https://blog.iseesystems.com/newsannouncements/xmile-faq/

3. ClubOfRome, (2021.), About us & history, dostupno 11.08.2021. na https://www.clubofrome.org/about-us/history/

4. ETHW,(2021.), dostupno 12.08.2021.na https://ethw.org/Jay_W._Forrester

5. Farrell, J. A. (1995.), Compiler Basics, dostupno 16.08.2021. na http://www.cs.man.ac.uk/~pjj/farrell/compmain.html

6. Forrester J. W., (1990.), The Begining of System Dynamics, dostupno 9.08.2021. na http://web.mit.edu/sysdyn/sd-intro/D-4165-1.pdf

7. Jakupović A., Šuman S., (2014.), Osnove programiranja, Veleučilište u Rijeci, Rijeka

8. OASIS, (2021.), dostupno 28.08.2021. na http://docs.oasisopen.org/xmile/xmile/v1.0/errata01/csprd01/xmile-v1.0-errata01csprd01-complete.html

9. Pacandi H., (2018.), IMPLEMENTACIJA SISTEM-DINAMIČKIH MODELA U EDUKATIVNIM RAČUNALNIM IGRAMA KORIŠTENJEM LIBGDX OKVIRA, dostupno, 9.08.2021.na https://urn.nsk.hr/urn:nbn:hr:137:760161

10. Sovilj S., Etinger D., Sirotić Z., Pripužić K., (2021.), Kotlin System Dynamics Toolkit, dostupno, 10.11.2021. na https://systemdynamics.org/kotlin-system-dynamics-toolkit/

Popis slika

Slika 1: Jay W. Forrester 6
Slika 2: Primjeri grafičkih simbola korištenih na dijagramima uzročnih petlji .. 10
Slika 3: Primjeri grafičkih simbola korištenih na dijagramima toka 11
Slika 4: Primjer CNF gramatike 15
Slika 5: Primjer EBNF gramatike 16
Slika 6: Primjer model_units oznake XMILE formata 23
Slika 7: Primjer oznake dimensions XMILE formata 24
Slika 8: Primjer oznake behavior XMILE formata 25
Slika 9: Primjer oznake model XMILE formata 26
Slika 10: Primjer oznake stock XMILE formata 29
Slika 11: Primjer oznake stock s podoznakom conveyor XMILE formata 30
Slika 12: Primjer oznake flow XMILE formata 31
Slika 13: Primjer oznake aux XMILE formata 32
Slika 14: Primjer oznake group XMILE formata 32
Slika 15: Primjer oznake module koji se referencira na submodel unutar iste datoteke 34
Slika 16: Primjer oznake module koja se referencira na podmodel unutar druge datoteke 34
Slika 17: Primjer oznake macro XMILE formata 35
Slika 18: Primjer oznake options XMILE formata 36
Slika 19: Dijagram klasa entiteta modela naslijeđenih iz apstraktne klase ModelEntity 39
Slika 20: Funkcija createListOfStrings 43
Slika 21: Funkcija gettingFlows 44
Slika 22: Funkcija parseXML 45
Slika 23: Funkcija getFlow 46
Slika 24: Funkcija getName 46

Slika 25: Funkcija checkModelName 47
Slika 26: Funkcija getXmileTagData 48
Slika 27: Funkcija prepareTypeOfSimSpecsIntegration 49
Slika 28: Funkcija prepareEquationForBuiltInFunctions 49
Slika 29: Funkcija setCompanionObjects 50
Slika 30: Funkcija setCompanionObjects 51
Slika 31: Funkcija saveDataInTranspilerString 52
Slika 32: Funkcija transpilerStringCreateAllSystemElements 52
Slika 33: Test checkAlphaOrUnderscore 54