Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

Marko Topolnik

Okružje za razvoj telekomunikacijske programske podrške zasnovano na jeziku XML

Doktorska disertacija

Zagreb, 2005.

Disertacija je izrađena na Zavodu za telekomunikacije Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu

Mentor: Prof. dr. sc. Marijan Kunštić

Disertacija ima: 249 stranica.

Disertacija br.:

Posvećeno G. Anđić,
mojem izvoru
duhovnog mira
i stalne inspiracije

Sadržaj

Popis slika

Uvod

Okosnica svakog okružja za razvoj programske podrške je *izvorni kod* koji predstavlja ključnu kariku u sučelju između čovjeka i računala. Gotovo od samih početaka programskih jezika udomaćio se pristup da čovjek zapisuje *tekst* koji se kodira prema jednom od svjetskih standarda te namjene, a računalo potpuno automatskim procesom taj tekst dekodira, utvrđuje njegovo značenje i proizvodi kod spreman za izravno izvršavanje na svojoj središnjoj obradbenoj jedinici. Tekst izvornog koda ugrubo podsjeća na pisani prirodni jezik i to njegovo svojstvo ključno je za uspješnost sučelja prema čovjeku.

U novije vrijeme, prije svega porastom kompleksnosti programskih sustava koji se razvijaju, pojavili su se novi zahtjevi na izvorni kod. On postaje ključan izvor različitih oblika informacija o razvojnom projektu i mjesto idealno za smještanje dodatnih informacija koje nisu potrebne za dobivanje samog izvršnog koda. Tradicionalni oblik izvornog koda pokazao se nedovoljno fleksibilnim da podrži ugradnju dodatnih informacija bez negativnih posljedica na njegovu primarnu namjenu.

Spomenute informacije potrebno je, zbog veličine i kompleksnosti projekata, izvlačiti automatskim putem. Također, kako projekt evoluira, potrebno je provoditi i neke sustavne zahvate nad velikim količinama izvornog koda kao cjelinom, što je ponovo isplativo jedino ako se može obaviti automatski. Zbog tih potreba automatizacije obrade izvornog koda pojavila se nova stavka u pitanju sučelja prema čovjeku. Postalo je bitno da zadatke obrade koda bude praktično formalno opisati i time omogućiti njihovo automatsko obavljanje. Rješenje je vrlo nepraktično formalizirati kao zahvate nad samim tekstom: mnogo ih je jednostavnije opisati koristeći se samim pojmovima kojima barata dotični programski jezik.

Za svaku automatizaciju obrade izvornog koda stoga je zajednički prvi korak njegovog automatskog razumijevanja do oblika kojim je eksplicitno opisano kombiniranje osnovnih jezičnih pojmova u opis programa. Shodno tome, znatno bi se olakšao i ubrzao razvoj automatskih zadataka obrade kad bi taj korak bio standardiziran i široko dostupan. To je motivacija za uvođenje novog oblika tekstualnog

zapisa izvornog koda koji će omogućiti takvu standardizaciju.

Ova disertacija bavi se problemom prelaska s tradicionalnog zapisa izvornog koda na zapis prema standardu XML (*Extensible Markup Language*) koji bi omogućio ostvarivanje navedenih ciljeva. U prvom poglavlju detaljnije je razjašnjena uloga izvornog koda u cjelokupnom razvojnom procesu. Dan je kratak pregled rezultata teorije formalnih jezika i načina njihove primjene na izvorni kod programskih jezika. U drugom poglavlju uvodi se jezik XML. Prvo se izlažu njegova općenita svojstva, a zatim koncepcija njegove primjene na zapis izvornog koda. U trećem poglavlju izneseni su rezultati usporedbe svojstava tradicionalnog izvornog koda s izvornim kodom u XML-u i istaknute ključne prednosti potonjeg. U četvrtom poglavlju dan je pregled najvažnijih zahtjeva koje treba ostvariti XML-ovski izvorni kod da bi omogućio maksimalno efikasno iskorištavanje mogućnosti XML-a. U petom poglavlju razmatraju se različiti aspekti oblikovanja sintakse XML-ovskog izvornog koda, njihov utjecaj na kvalitetu sintakse s obzirom na zahtjeve zacrtane u prethodnom poglavlju, te daju smjernice (načela) koja treba slijediti kako bi se ti zahtjevi ostvarili u što većoj mjeri. U šestom poglavlju predstavlja se XML-ovska sintaksa programskog jezika Java koja je razvijena radi potvrde opravdanosti načela iznesenih u prethodnom poglavlju. Posebno je obraćena pozornost na proces oblikovanja sintakse i na način na koji su načela korištena u razrješavanju dilema oko odabira najboljeg između niza mogućih rješenja za zapis pojedine jezične konstrukcije. U sedmom poglavlju dan je pregled drugih radova na području XML-ovskih sintaksi programskih jezika, a u osmom poglavlju posebno su izdvojene dvije postojeće sintakse za jezik Javu i provedena detaljna usporedba s predloženom sintaksom. Usporedba je provedena s obzirom na uspješnost u ispunjenju zahtjeva iznesenih u četvrtom poglavlju. Disertacija završava sa zaključkom.

Poglavlje 1

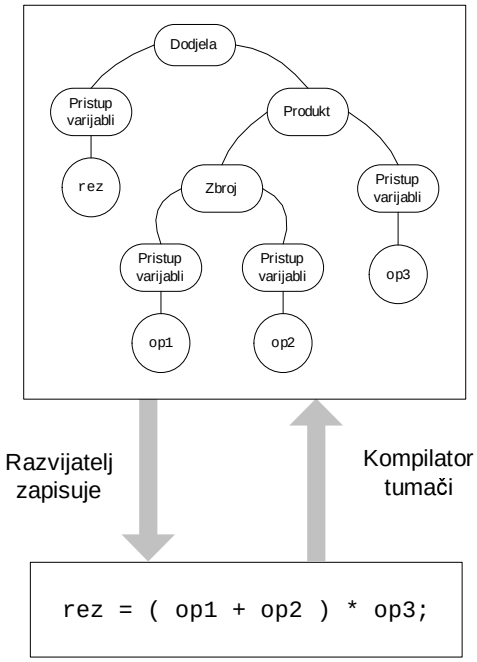
Uloga izvornog koda u razvojnom procesu

Budući da se u ovoj disertaciji razmatra uvođenje novog oblika izvornog koda, potrebno je prvo razlučiti neke ključne detalje oko njegove uloge u razvojnom procesu. Da bi se od elektroničkog računala dobilo željeno ponašanje, potrebno ga je precizno i nedvosmisleno opisati. Razvijatelj programa sastavlja opis ponašanja kombiniranjem osnovnih jedinica značenja (pojmova) koje poznaje dani programski jezik i zapisuje ga u obliku izvornog koda, a taj oblik se automatiziranim postupkom (kompiliranjem) prevodi u oblik spreman za izravno izvršavanje na računalu (strojni kod). Važno je razlučiti samo *značenje* programa od njegovog *zapisa* jer se u disertaciji razmatra zadržavanje identičnog značenja uz promjenu zapisa. Razlika će biti ilustrirana na primjeru tipične izjave u programskom jeziku.

Sveukupno, međuodnosi razvijatelja, kompilatora, izvornog koda i njegovog značenja mogu se prikazati slikom 1.1. Između ostalih, tipičan programski jezik će poznavati sljedeće osnovne pojmove: "izjava", "dodjela vrijednosti varijabli", "pristup varijabli", "produkt", "zbroj". Jedan smisleni način njihovog kombiniranja prikazan je na gornjem dijelu slike 1.1. Time je sastavljen precizan i nedvosmislen opis jedne izjave programskog jezika. Prema uvriježenom pristupu spomenuto značenje treba zapisati običnim tekstom, primjerice ovako:

```
rez = ( op1 + op2 ) * op3;
```

Formalno specificiranje pravila zapisa (sintakse) izvornog koda temeljito je razrađena disciplina pa će u nastavku će biti iznesen kratak pregled njenih teoretskih osnova.



Slika 1.1: Izvorni kod i njegovo značenje

1.1 Formalni jezici

Teorija formalnih jezika [Cho59, HMU01] grana je matematike proizašla iz istraživanja prirodnih (ljudskih) jezika. Ovdje će biti izloženi neki osnovni elementi teorije bitni za disertaciju.

Abeceda Σ je neprazan konačan skup *simbola*. U teoriji formalnih jezika važan pojam je *niz* dobiven nizanjem simbola iz neke abecede. Razmatraju se konačni nizovi. Definirajmo *potenciju* abecede, Σ^n za $n \geq 0$:

$$\Sigma^n = \begin{cases} \{\epsilon\} & , \text{n} = 0 \\ \{xy | x \in \Sigma, y \in \Sigma^{n-1}\} & , \text{n} > 0 \end{cases}$$

Ovdje ϵ označava prazan niz simbola, a xy označava rezultat *ulančavanja* nizova x i y. Riječima, n-ta potencija abecede je skup svih nizova duljine n.

Definirajmo još i sljedeće entitete:

$$\Sigma^? = \Sigma^0 \cup \Sigma^1$$

$$\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i$$

$$\Sigma^+ = \bigcup_{i=1}^{\infty} \Sigma^i$$

Dakle, $\Sigma^?$ je unija praznog niza i svih nizova od jednog simbola, Σ^* je skup svih nizova nad abecedom uključujući i prazan niz, a Σ^+ je skup svih nepraznih nizova. U ovim definicijama najvažnije je uočiti značenje *upitnika*, *zvjezdice* i *plusa* jer će se kasnije primjenjivati i na nešto drugačije matematičke entitete, s analognim značenjem.

Jezik L nad abecedom Σ je podskup skupa Σ^*:

$$L \subseteq \Sigma^*$$

Znakom $\emptyset$ označava se prazan jezik, takav koji ne sadrži nijedan niz.

1.1.1 Regularan jezik i regularni izraz

Regularni jezici su oni i samo oni jezici koje se može definirati *regularnim izrazom*. Jezik definiran nekim regularnim izrazom E zapisuje se kao $L(E)$. Algebra regularnih izraza raspolaže sljedećim operacijama:

- *Unija* dvaju jezika L i M, tj. $L \cup M$, je skup svih nizova koji se nalaze bilo u L, bilo u M.
- *Ulančenje* dvaju jezika L i M je skup svih nizova koje se može dobiti uzimanjem po jednog niza iz L i M i ulančavanjem prvog niza s drugim. Ovaj operator zapisuje se točkom ili bez ikakvog znaka, dakle $L.M$ ili jednostavno LM.
- *Kleeneovo zatvaranje* (ili, kraće, *zatvaranje*) jezika L označava se s L^*:

  $$L^* = \bigcup_{i=0}^{\infty} L^i$$

  gdje je

  $$L^n = \begin{cases} \{\epsilon\} & , \text{n} = 0 \\ \{L^{n-1}.L & , \text{n} > 0 \end{cases}$$

  Riječima, zatvaranje je skup svih nizova koje se može dobiti odabirom proizvoljnog broja proizvoljnih nizova iz L (uz moguć višestruki izbor istog niza) i ulančavanjem svih njih.

Osnova regularnog izraza su konstante:

- Posebne konstante $\emptyset$ i ϵ definiraju jezike $\{\epsilon\}$ i $\emptyset$. Dakle, $L(\emptyset) = \emptyset$ i $L(\epsilon) = \{\epsilon\}$.
- Za bilo koji simbol a regularni izraz koji definira jezik $\{a\}$ zapisuje se kao konstanta: $\mathbf{a}$. Dakle, $L(\mathbf{a}) = \{a\}$.

Operatori i grupiranje članova izraza zapisuju se prema pravilima koja slijede:

- Ako su E i F regularni izrazi, tada je $E + F$ regularni izraz koji definira uniju jezika definiranih izrazima E i F. Dakle, $L(E + F) = L(E) \cup L(F)$.
- Ako su E i F regularni izrazi, tada je EF regularni izraz koji definira ulančenje jezika definiranih izrazima E i F. Dakle, $L(EF) = L(E)L(F)$.
- Ako je E regularni izraz, tada je E^* regularni izraz koji definira zatvaranje jezika definiranog izrazom E. Dakle, $L(E^*) = (L(E))^*$.
- Ako je E regularni izraz, tada je (E) regularni izraz koji definira isti jezik kao i E. Dakle, $L((E)) = L(E)$.

1.1.2 Beskontekstan jezik i beskontekstna gramatika

Beskontekstni jezici pripadaju klasi široj od klase regularnih jezika. Svaki regularan jezik je beskontekstan, a obrat ne vrijedi. Klasu beskontekstnih jezika sačinjavaju oni i samo oni jezici koje se može definirati *beskontekstnom gramatikom*. Ovakvu gramatiku sačinjavaju sljedeći elementi:

- Abeceda Σ čije simbole u ovom kontekstu nazivamo *terminalnim simbolima* ili kraće *terminalima*.
- Konačan skup V *varijabli* ili *neterminala*. Svaki neterminal predstavlja neki beskontekstni jezik.
- Jedan, posebno izdvojeni neterminal S koji predstavlja jezik definiran gramatikom. Nazivamo ga *početnim simbolom*. Ostali neterminali predstavljaju pomoćne jezike koji pomažu u definiciji jezika početnog simbola.
- Konačan skup P *pravila* ili *produkcija* koje oblikuju rekurzivnu definiciju jezika. Svaka produkcija sastoji se od:

- Neterminala kojeg se (djelomično) definira produkcijom. Njega se naziva *glavom* produkcije. Jedan te isti neterminal može se pojaviti u glavi proizvoljnog broja različitih produkcija.
- Metaznaka $\rightarrow$ sa značenjem "može biti sastavljen od".
- Niza od nula ili više terminala i neterminala. Taj niz, zvan *tijelo* produkcije, predstavlja jedan način sastavljanja nizova u jeziku neterminala iz glave. Niz se sastavlja tako da se sve terminale ostavi nepromijenjenima, a umjesto svakog neterminala ubaci bilo koji niz iz jezika dotičnog neterminala. Nakon toga se sve nizove ulanča redom kojim su navedeni u produkciji.

Gramatiku možemo modelirati kao uređenu četvorku $G = (V, \Sigma, P, S)$. Produkciju p možemo modelirati kao uređeni par: $p = (A, \alpha)$ gdje je $A \in V$ glava, a $\alpha \in (\Sigma \cup V)^*$ tijelo. Zvjezdica, kao i ranije, označava uniju svih potencija skupa u zagradama. Produkciju se zapisuje ovako:

$$A \rightarrow \alpha$$

gdje se članovi niza α pišu jedan za drugim.

Za svaki neprazan beskontekstni jezik bez elementa ϵ postoji gramatika čija svaka produkcija ima jedan od sljedećih oblika:

$$A \rightarrow a \quad A \rightarrow BC$$

gdje je $A, B, C \in V, a \in \Sigma$. Takva gramatika naziva se *Chomskyjevim normalnim oblikom* (eng. *Chomsky Normal Form* [HMU01]). Ako je zadovoljeno i dodatno ograničenje da je svaki B u produkcijama drugog oblika neterminal čije sve produkcije imaju prvi oblik, gramatika opisuje *regularan* jezik i za svaki neprazan regularni jezik bez elementa ϵ postoji gramatika takvog oblika. Dakle, za svaki takav jezik postoji beskontekstna gramatika čija svaka produkcija ima jedan od sljedećih oblika:

$$A \rightarrow a \quad A \rightarrow aB$$

gdje je $A, B \in V, a \in \Sigma$. Ovime je najzornije ilustrirana razlika između regularnih i neregularnih beskontekstnih jezika.

Izvod niza pomoću gramatike

Primjenom produkcija može se doći do nizova koji su članovi jezika određenog neterminala. Odabere se jedna od produkcija dotičnog neterminala i zapiše niz iz tijela produkcije. Za svaki neterminal u tijelu postupak se ponovi, sve dok ne ostanu

sami neterminali. Njihovo ulančenje predstavlja član jezika početnog neterminala. Opisani postupak naziva se *izvodom* niza. Formalizacija izvoda zahtijeva uvođenje novog relacijskog simbola, $\Rightarrow$, koji predstavlja jedan korak izvoda. Pretpostavimo da je $G = (V, \Sigma, P, S)$ neka beskontekstna gramatika. Uzmimo u razmatranje niz $\alpha A \beta$ gdje je $A \in V$ neki neterminal, a $\alpha, \beta \in (\Sigma \cup V)^*$ neki nizovi terminala i neterminala. Nadalje, uzmimo da je $A \to \gamma$ jedna od produkcija u G. Tada je sljedećim izrazom definirano značenje relacije $\Rightarrow$:

$$\alpha A \beta \underset{G}{\Rightarrow} \alpha \gamma \beta$$

Dakle, jednim korakom izvoda zamjenjuje se jedan, bilo koji neterminal na lijevoj strani tijelom jedne od njegovih produkcija. Korisno je uvesti i proširenje relacije $\Rightarrow$ tako da se može izraziti izvod u proizvoljnom broju (nula ili više) koraka. Za to se koristi znak $\overset{*}{\Rightarrow}$, definiran sljedećom rekurzivnom definicijom:

Baza. Za svaki niz $\alpha \in (\Sigma \cup V)^*$ vrijedi $\alpha \overset{*}{\Rightarrow} \alpha$.

Indukcija. Ako $\alpha \underset{G}{\overset{*}{\Rightarrow}} \beta$ i $\beta \underset{G}{\Rightarrow} \gamma$, onda $\alpha \underset{G}{\overset{*}{\Rightarrow}} \gamma$.

Jezik gramatike G označava se s $L(G)$. To je skup nizova terminala za koje postoji izvod iz početnog simbola. Dakle,

$$L(G) = \{\alpha \in \Sigma^* | S \underset{G}{\overset{*}{\Rightarrow}} \alpha\}$$

1.1.3 Parsiranje i stablo parsiranja

Parsiranje je postupak provjere je li zadani niz član zadanog jezika. U slučaju jezika definiranog beskontekstnom gramatikom postupak parsiranja najčešće uključuje izgradnju *stabla parsiranja*. Uzmimo gramatiku $G = (V, \Sigma, P, S)$. Stablo parsiranja za G je uređeno stablo (djeca svakog čvora imaju definiran redoslijed) koje zadovoljava sljedeće uvjete:

- Svaki unutarnji čvor je označen neterminalom iz V.
- Svaki list je označen bilo terminalom, bilo ϵ-om. Ako je označen ϵ-om, taj list mora biti jedino dijete svog roditelja.
- Ako je unutarnji čvor označen s A, a djeca su mu označena s

  $$X_1, X_2, ..., X_k$$

  redom slijeva nadesno, tada je $A \to X_1 X_2 ... X_k$ produkcija u P.

Za parsiranje je bitan pojam *prinosa* stabla parsiranja. To je niz koji se dobije ulančavanjem svih terminala iz listova stabla, krećući se kroz stablo slijeva nadesno (prefiksni prolaz).

Stablo parsiranja ujedno predočuje postupak izvoda niza iz početnog simbola. Ono ima primjenu i izvan konteksta parsiranja pa se za njega koristi i termin *stablo sintakse*. Posebice, u primjeni na tekst izvornog koda, stablo parsiranja pokazuje na koji način su grupirani dijelovi teksta (kojim neterminalima pripadaju) i samim time je dobra osnova za utvrđivanje značenja teksta.

Ako za bilo koji niz–član jezika $L(G)$ možemo izgraditi više različitih stabala parsiranja čiji je prinos taj niz, tada je gramatika G *dvosmislena* jer ta stabla predstavljaju različite načine tumačenja niza. U slučaju da se definira gramatika koja neće služiti za parsiranje, njena dvosmislenost ne predstavlja problem. Na primjer, gramatika može služiti samo za izgradnju stabla izvodom iz početnog simbola, a jedno stablo nikad nije dvosmisleno.

1.1.4 Dvosmislenost gramatike i njeno otklanjanje

Mjesto na kojem zahtjev za nedvosmislenošću beskontekstne gramatike najviše dolazi do izražaja u njenoj primjeni na programske jezike jest zapis izraza s binarnim operacijama. Radi se o izrazima sljedećeg općenitog oblika:

operand1 operator1 operand2 operator2 operand3 ...

Uzmimo primjerice beskontekstnu gramatiku za izraze sa zbrajanjem i množenjem (slika 1.2[1]) i razmotrimo izraz

$$op1 + op2 * op3$$

Između ostalog, možemo provesti sljedeća dva valjana izvoda:

$$Izraz \Rightarrow Izraz * Izraz \Rightarrow Izraz + Izraz * Izraz$$

$$Izraz \Rightarrow Izraz + Izraz \Rightarrow Izraz + Izraz * Izraz$$

Dovršavanjem izvoda primjenom produkcije $Izraz \rightarrow Ime$ možemo dobiti dva stabla parsiranja prikazana na slici 1.3, identičnog prinosa jednakog gornjem izrazu.

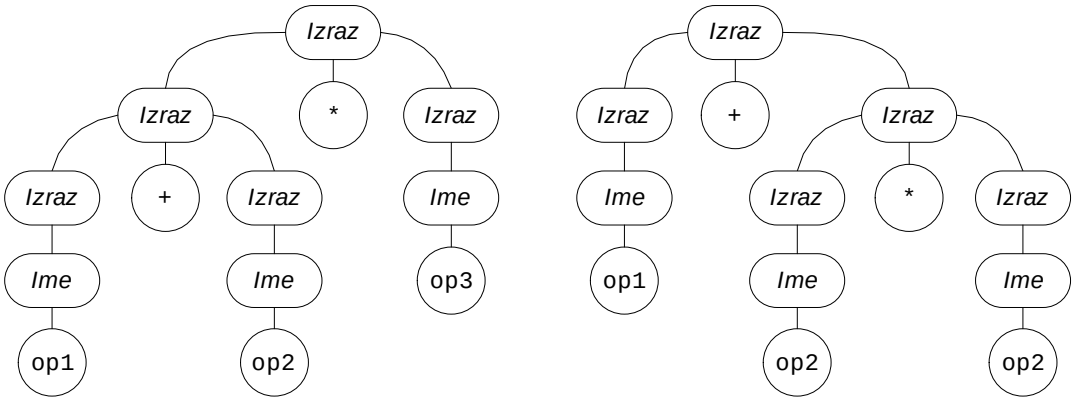
Gramatika je dvosmislena iz dva razloga:

1. Nije definiran prioritet operacija. Ovo je očito iz slike 1.3.

[1]Radi kratkoće zapisa produkcije s istom glavom navedene su u jednom retku: $A \rightarrow \alpha_1 | \alpha_2$ je ekvivalentno sa $A \rightarrow \alpha_1 \quad A \rightarrow \alpha_2$

$$
\begin{aligned}
Izraz &\rightarrow Ime \\
Izraz &\rightarrow Izraz + Izraz \\
Izraz &\rightarrow Izraz * Izraz \\
Izraz &\rightarrow (Izraz) \\
Ime &\rightarrow op1 \,|\, op2 \,|\, op3
\end{aligned}
$$

Slika 1.2: Dvosmislena beskontekstna gramatika



Slika 1.3: Dva stabla parsiranja izraza $op1 + op2 * op3$ gramatikom sa slike 1.2

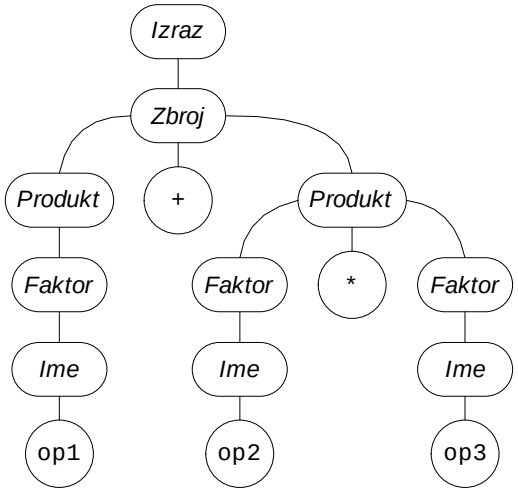
2. Nije definirano kojim poretkom treba parsirati niz od više istih operacija, npr. $op1 + op2 + op3$. Za ovaj izraz ponovo možemo izgraditi dva stabla parsiranja ista kao na slici 1.2, samo što se umjesto množenja pojavljuje opet zbrajanje.

Da bismo gramatiku učinili nedvosmislenom, potrebno je razriješiti i jedan i drugi problem. To se može učiniti uvođenjem posebnog neterminala za svaku operaciju, npr. u gramatici sa slike 1.4.

$$\begin{aligned} Izraz &\rightarrow Zbroj \\ Zbroj &\rightarrow Produkt \mid Zbroj + Produkt \\ Produkt &\rightarrow Faktor \mid Produkt * Faktor \\ Faktor &\rightarrow Ime \mid (\,Izraz\,) \end{aligned}$$

Slika 1.4: Nedvosmislena beskontekstna gramatika

Time je propisano da su operatori *lijevo asocijativni*, dakle izraz $op1+op2+op3$ ima smisao $(op1 + op2) + op3$ i da je množenje višeg prioriteta od zbrajanja. Također, izrazu u zagradama dodjeljuje se najviši prioritet. Stablo parsiranja ovom gramatikom prikazano je na slici 1.5.



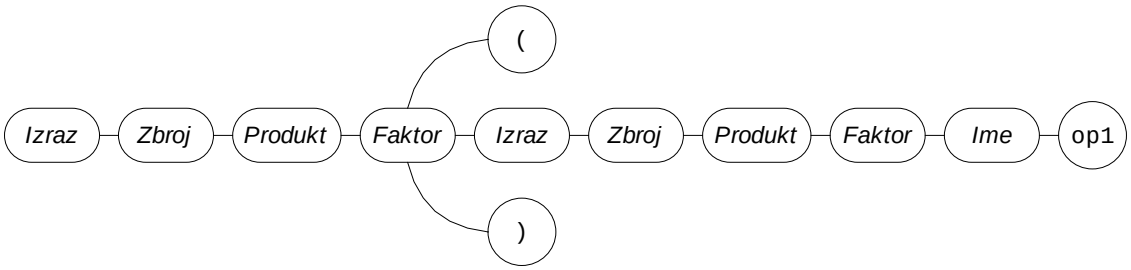
Slika 1.5: Stablo parsiranja izraza $op1 + op2 * op3$ gramatikom sa slike 1.4

U gramatici sa slike 1.4 može se uočiti i jedna njena osobina čija će važnost doći do izražaja kasnije: put do bilo kojeg neterminala *Ime* kroz stablo parsiranja

je

$$Izraz/Zbroj/Produkt/Faktor$$

bez obzira na to u kojem izrazu se nalazi. Za svake zagrade unutar kojih se nalazi identifikator, čitav taj put se ponavlja još jednom. Na primjer, ako se čitav izraz svodi samo na identifikator u zagradama, npr. ($op1$), stablo parsiranja će izgledati kako je prikazano na slici 1.6.



Slika 1.6: Stablo parsiranja izraza ($op1$) gramatikom sa slike 1.4

Jedna ideja za otklanjanje ovakve redundancije u stablu parsiranja mogla bi biti uvođenje oblika produkcija kao na slici 1.7, međutim time bi gramatika ponovo postala dvosmislena.

$$Izraz \rightarrow Zbroj \mid Produkt \mid Faktor \mid Ime$$

Slika 1.7: Primjer produkcije kojom se uklanja redundancija u stablu parsiranja

Još je važno istaknuti da je sama ideja binarne operacije usko povezana s uobičajenim načinom zapisivanja matematičkog izraza, tzv. *infiksnim* zapisom. Tim načinom zapisa nemoguće je zapisati operaciju nad više od dva operanda, iako bi takva operacija bila jednako dobro definirana. Primjerice, moglo bi se koristiti i *prefiksni* zapis uobičajen u matematici za zapis funkcija, npr:

$$Zbroj(Produkt(op1, op2, op3), op4)$$

Ovakav zapis nema problema s dvosmislenošću i za njega se može definirati puno jednostavnija beskontekstna gramatika. Općenito, takav zapis izravnije opisuje stablastu strukturu izraza, dok infiksni zapis više podsjeća na rečenicu prirodnog jezika i stoga je lakši za pročitati.

1.2 Primjena formalnih jezika na izvorni kod

Specifikacija programskog jezika sadrži dvije komponente:

1. Skup osnovnih pojmova i skup pravila (zvan *apstraktna sintaksa*) koja propisuju dozvoljene načine njihovog udruživanja u opis željenog ponašanja. Opis ima oblik stabla sintakse i stoga se naziva *stablom apstraktne sintakse* (eng. *Abstract Syntax Tree*, tj. AST [Luc82]).

2. Abecedu tipografskih znakova i skup pravila (zvan *konkretna sintaksa*) kako ih kombinirati radi zapisivanja opisa iz prethodne točke.

Dakle, programski tekst strukturiran je prema konkretnoj sintaksi, čija pravila se uobičajno formaliziraju u dvije razine: *leksičkom* i *sintaktičkom* gramatikom. Jezik leksičke gramatike je regularan i moguće ju je definirati regularnim izrazima. Sintaktička gramatika je beskontekstna i njena abeceda jednaka je jeziku leksičke gramatike.

Čitav proces kompiliranja može se konceptualno podijeliti u sljedeće korake, ilustrirane slikom 1.11:

1. Niz bitova dekodira se u niz znakova programskog teksta. Dekodiranje se redovito obavlja prema jednom od svjetskih standarda za kodiranje teksta (najvažniji primjeri su ASCII i u novije vrijeme Unicode).

2. *Leksičkom analizom* tekst izvornog koda rastavlja se na niz *tokena* (analognih riječima i interpunkcijama u prirodnom pisanom jeziku), ilustriran slikom 1.8. Ovaj korak obavlja se u skladu s pravilima leksičke gramatike. Tokeni su, dakle, članovi jezika leksičke i abecede sintaktičke gramatike.

3. Niz tokena se parsira prema pravilima sintaktičke gramatike. Rezultat ovog koraka je stablo parsiranja koje se u ovom slučaju još naziva *stablom konkretne sintakse* (eng. *Concrete Syntax Tree*, tj. CST, slika 1.9).

4. Obradom podataka iz stabla konkretne sintakse, kao i eventualnih dodatnih podataka iz šireg konteksta izgrađuje se stablo apstraktne sintakse prikazano na slici 1.10. Opis koji to stablo predstavlja može se smatrati *značenjem* dotičnog izvornog koda.

5. Na osnovu stabla apstraktne sintakse generira se strojni kod.

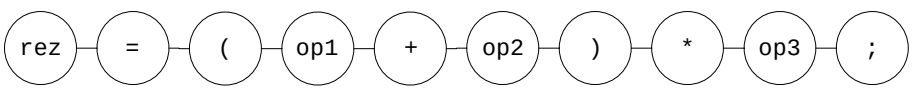
1.2.1 Stablo apstraktne sintakse

Proces obrade u koraku 4 konceptualno je najzahtjevniji. Svi raniji koraci odvijaju se prema jasno formaliziranim pravilima konkretne sintakse. Za razliku od njih, pravila apstraktne sintakse gotovo nikad se ne formaliziraju, već samo opisuju prirodnim jezikom. Tako i sam izgled (struktura i sadržaj) stabla apstraktne sintakse nije formalno propisan jezičnom specifikacijom. Apstraktna sintaksa uključuje i pravila za čije nametanje su potrebni podaci iz čitavog skupa programskih modula na koje se oslanja trenutno obrađivani modul. Na primjer, svaki identifikator treba protumačiti (ustanoviti na koji entitet se odnosi) i tek nakon toga može se provjeriti postoji li dotični entitet, koji može biti i u nekom vanjskom modulu. I unutar istog modula mnoga pravila provjeravaju se tek u fazi izgradnje stabla apstraktne sintakse, npr. nametanje jedinstvenosti identifikatora varijabli u nekom bloku koda.

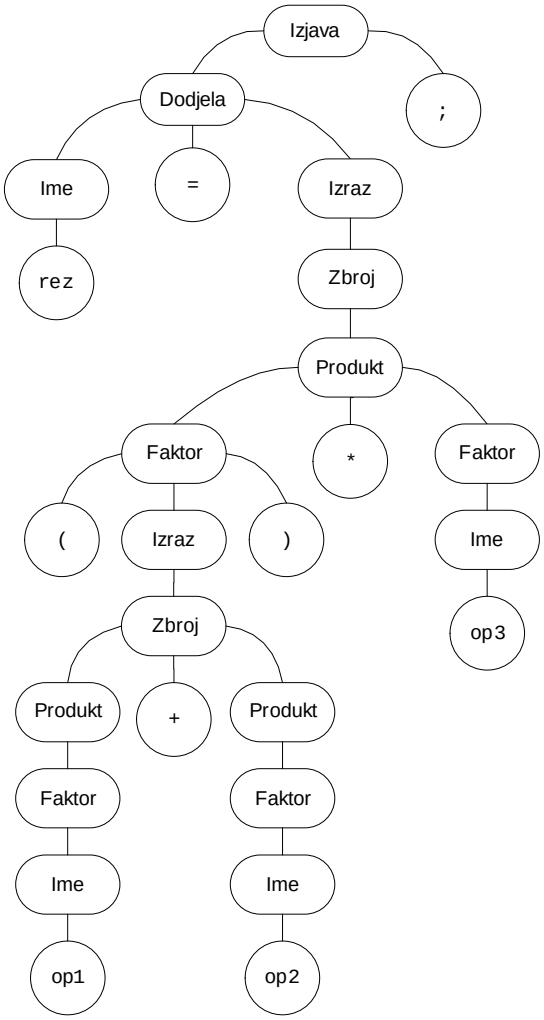
Apstraktnu sintaksu također se može (djelomično) formalizirati beskontekstnom gramatikom, s time da ona smije biti dvosmislena jer ne služi za parsiranje. Zahvaljujući, između ostalog, i tom ublažavanju zahtjeva stablo apstraktne sintakse može imati znatnu manju kompleksnost i veličinu. To se može jasno uočiti usporedbom slika 1.9 i 1.10 koje prikazuju ta dva stabla za jedan te isti odsječak izvornog koda.

1.3 Zapis gramatike u obliku EBNF

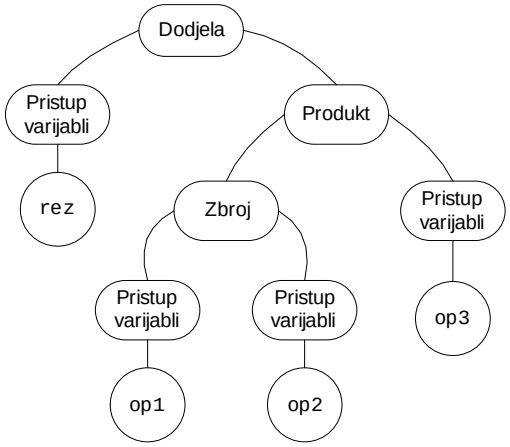
Formalizam izložen u odjeljku 1.1.2 ima svojstvo minimalnosti: to je najjednostavniji oblik dovoljno izražajan da se njime može definirati svaka beskontekstna gramatika. Minimalnost formalizma uvijek je značajna za teoretska razmatranja. Međutim, kada ga treba primijeniti u praksi, ključno svojstvo postaje mogućnost što konciznijeg i preglednijeg definiranja složenije gramatike. Stoga se za definiranje gramatike programskog jezika koristi nešto složeniji, *Prošireni Backus-Naurov oblik* (eng. *Extended Backus-Naur Form*, tj. EBNF). Zapis koristi isključivo ASCII-jeve znakove i uvodi dodatna pravila u zapis produkcije. Pravila su takva da se tijelo produkcije može definirati *regularnim izrazom* nad abecedom $(\Sigma \cup V)$. Ta-



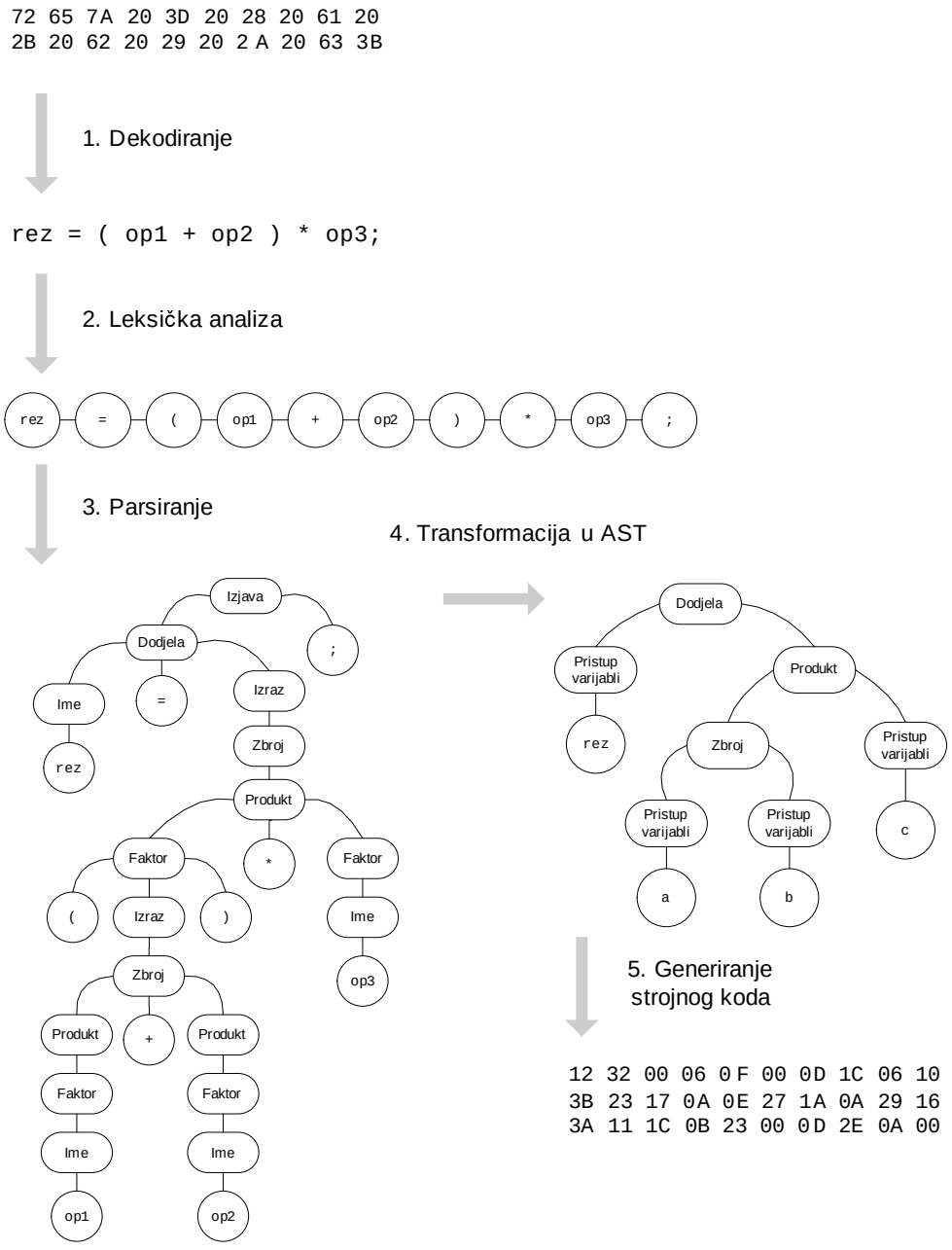
Slika 1.8: Niz tokena izjave "`rez = ( op1 + op2 ) * op3;`"



Slika 1.9: Stablo konkretne sintakse izjave "`rez = ( op1 + op2 ) * op3;`"



Slika 1.10: Stablo apstraktne sintakse izjave "`rez = ( op1 + op2 ) * op3;`"



Slika 1.11: Kompiliranje izjave "`rez = ( op1 + op2 ) * op3;`"

kođer se, osim operatora *zvjezdica* za zapis Kleeneovog zatvaranja, definiraju i preostala dva već spomenuta operatora, *plus* i *upitnik*. Konkretno, pravila zapisa su sljedeća:

- Umjesto znaka $\rightarrow$ koristi se niz ASCII-jevih znakova `::=`
- Upitnikom se označava niz koji se može i ne mora pojaviti. Zapis je ekvivalentan dvjema produkcijama od kojih jedna sadrži navedeni niz, a druga ne. Ako se niz na koji se odnosi upitnik sastoji od više simbola, potrebno ga je obuhvatiti oblim zagradama.

  `N ::=` α `(`β`)?` γ

  ekvivalentno je sa

  $$N \rightarrow \alpha\gamma \mid \alpha\beta\gamma$$

- Zvjezdicom se označava niz koji se može pojaviti nula ili više puta, tj. zapisuje se njegovo Kleeneovo zatvaranje.

  `N ::=` α `(`β`)*` γ

  ekvivalentno je sa

  $$N \rightarrow \alpha\gamma \mid \alpha M\gamma$$
  $$M \rightarrow \beta \mid M\beta$$

  Zbog nužnosti uvođenja dodatnog neterminala, zapisi su ekvivalentni samo u smislu da definiraju isti jezik, međutim stabla parsiranja koja iz njih proizlaze neće biti identična. U praksi ta razlika je bitna.

- Plusom se označava niz koji se može pojaviti jedan ili više puta.

  `N ::=` α `(`β`)+` γ

  ekvivalentno je sa

  `N ::=` α β `(`β`)*` γ

- Doslovno navođenje terminala stavlja se unutar jednostrukih navodnika: `'terminal'`

- Umjesto doslovnog pobrojavanja svih potrebnih terminala može se regularnim izrazom definirati njihov jezik nad abecedom ASCII-jevih znakova. Posebice, unutar uglatih zagrada može se zapisati podskup abecede zadavanjem opsega. Na primjer, `N ::= [a-d]` ekvivalentno je sa

$$N \rightarrow a|b|c|d$$

  Ako treba sam znak *crtica* staviti u abecedu, ispred nje se stavlja *povratna kosa crta* \. Isto vrijedi i za neke druge znakove, npr. znak razmaka i samu povratnu kosu crtu.

Poglavlje 2

Izvorni kod u XML-u

XML (*Extensible Markup Language* ili *Proširivi jezik za označavanje* [xml04]) je beskontekstni jezik za opis stabla. Razvijen je pod okriljem međunarodnog tijela World Wide Web Consortium (skraćeno W3C). Abeceda XML-ove leksičke sintakse je skup znakova definiran standardom Unicode, suvremenim nasljednikom ASCII-ja. Konkretna sintaksa je prilično jednostavna i u potpunosti je definirana EBNF-om iznesenim u [xml04]. Međutim, od cjelokupne sintakse za većinu namjena dovoljan je samo njen manji dio. Svi XML-ovski dokumenti koji se pojavljuju u ovoj disertaciji ravnaju se po toj pojednostavljenoj sintaksi, čiji EBNF ima svega 32 produkcije i iznesen je u Dodatku A.

2.1 XML-ov *infoset*

Stablo kakvo se može opisati jezikom XML naziva se *infoset*[1] [inf04]. Jedan XML-ovski dokument opisuje jedan infoset. Prema [inf04] infoset je skup podataka u kojem se može pojavljivati ukupno jedanaest vrsta informacijskih stavki (eng. *information item*). Međutim, za svrhe ovog izlaganja infoset je najbolje promatrati kao stablo sastavljeno od tri vrste čvora, a to su *element*, *atribut* i *tekstualni čvor*. Ostale vrste imaju popratne namjene i mogu se zanemariti. Pojednostavljena gramatika iz Dodatka A omogućuje zapis te tri vrste čvora, a podržava i još jednu vrstu, čvor komentara. U opisu koji slijedi potrebno je obratiti pažnju na definiciju pojma "čvor-dijete" specifičnu za infoset.

Element je najvažniji čvor jer je jedini od navedene tri vrste koji može imati

[1] Budući da se riječ *infoset* vrlo često pojavljuje u tekstu disertacije, odlučeno je da se odavde nadalje navodi kao uobičajena stručna riječ, bez naglašavanja kurzivom.

čvorove-djecu. Djeca mogu biti tekstualni čvorovi i drugi elementi. Niz čvorova-djece naziva se *sadržajem* elementa. Element može imati još jednu vrstu podčvorova koji ne ulaze u njegov sadržaj – atribute (vidi niže). Proizlazi da je element vrsta čvora koja omogućuje izgradnju stabla, dok se ostali čvorovi pojavljuju kao listovi. Korijen čitavog infoseta je element[2]. Svaki element ima ime, a redoslijed između elemenata-braće definiran je redoslijedom njihovog navođenja u XML-ovskom dokumentu. Više elemenata-braće može imati isto ime jer se uvijek razlikuju prema položaju.

Atribut se pojavljuje kao podčvor nekog elementa. Svaki atribut ima ime i tekstualnu vrijednost. Redoslijed atributa nije definiran redoslijedom u XML-ovskom dokumentu i proizvoljan je. Među atributima-braćom ne smiju biti dva istog imena. Atributi se ne nalaze u sadržaju elementa, već u zasebnom skupu njegovih atributa, pa se tako ne smatraju ni njegovom djecom.

Tekstualni čvor pojavljuje se kao dijete nekog elementa. On nema ime, već samo tekstualnu vrijednost. U sadržaju jednog elementa tekstualni čvorovi mogu biti izmiješani s podelementima. Pojava više tekstualnih čvorova zaredom ekvivalentna je jednom tekstualnom čvoru čija vrijednost je dobivena ulančavanjem tekstualnih vrijednosti tih čvorova.

Beskontekstna gramatika apstraktne sintakse dovoljna za formalizaciju većine gornjih pravila može se prikazati EBNF-om iz ispisa 2.1[3]. Ostaje još nametanje jedinstvenosti imena atributa, što nije izrazivo beskontekstnom gramatikom.

Ispis 2.1: EBNF apstraktne sintakse XML-a

```
Element ::= Ime Atribut* Sadržaj?
Atribut ::= Ime Vrijednost
Sadržaj ::= (Element | Tekst)+
Ime ::= [A-Za-z] [A-Za-z0-9]*
Vrijednost ::= [A-Za-z0-9\ ]*
Tekst ::= [A-Za-z0-9\ ]+
```

Samo stablo infoseta ne može se formalizirati EBNF-om jer stablo beskontekstne gramatike sadrži drugačije vrste čvorova (neterminali i terminali). Međutim,

[2]Precizno govoreći, korijen svakog infoseta je čvor posebne vrste, tzv. *čvor dokumenta*, nadređen korijenskom elementu. On, međutim, nije bitan u ovom kontekstu.

[3]Gramatika apstraktne sintakse XML-a nije standardizirana i ovdje je definirana radi ilustracije. Radi jednostavnosti nisu navedene potpune abecede neterminala Ime, Vrijednost i Tekst. Potpune definicije mogu se deducirati iz Dodatka A.

postoji jednostavna dvosmjerna transformacija (*izomorfizam*) između ovakvog stabla apstraktne sintakse i infoseta.

Gornja EBNF-ovska definicija opisuje pravila *općenite* sintakse infoseta, tj. one koju mora zadovoljiti svaki infoset. Za XML-ovski dokument usklađen s tim pravilima kaže se da je *dobro oblikovan*. Kratkoća definicije općenite sintakse upućuje na to da je ona vrlo slobodna. Nju se za neku konkretnu primjenu može specijalizirati, tj. postrožiti. Za formalnu definiciju sintakse infoseta koristi se izraz *shema* dokumenta (iako je točnije reći shema infoseta) i postoji veći broj formalnih jezika za definiciju sheme, od kojih će kasnije u ovom poglavlju biti predstavljena dva najutjecajnija. Za dokument čiji infoset zadovoljava određenu shemu kaže se da je *valjan* s obzirom na nju.

Cjelokupan proces prevođenja XML-ovskog dokumenta u njegov infoset konceptualno se sastoji od sljedećih koraka, ilustriranih slikom 2.1:

1. Niz bitova dekodira se u niz znakova i dobiva se tekst XML-ovskog dokumenta.

2. Dokument se procesom leksičke analize rastavlja na niz tokena.

3. Niz tokena parsira se prema gramatici iz Dodatka A. Rezultat je stablo konkretne sintakse.

4. Stablo konkretne sintakse transformira se u stablo apstraktne sintakse.

5. Stablo apstraktne sintakse transformira se u infoset.

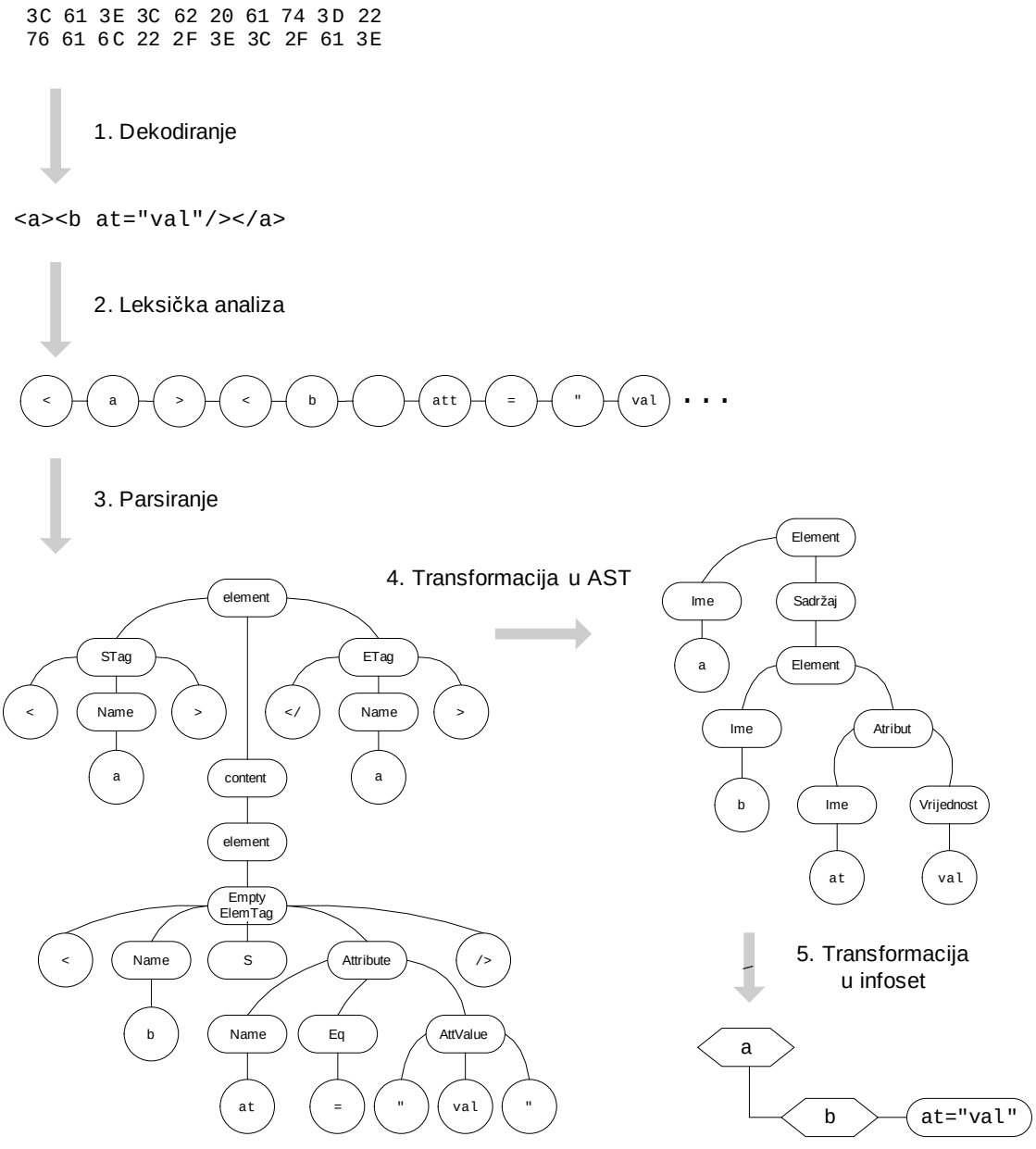
Potpuno strukturirani infoset

Korisno je uvesti sljedeću klasifikaciju *vrsta sadržaja* elementa:

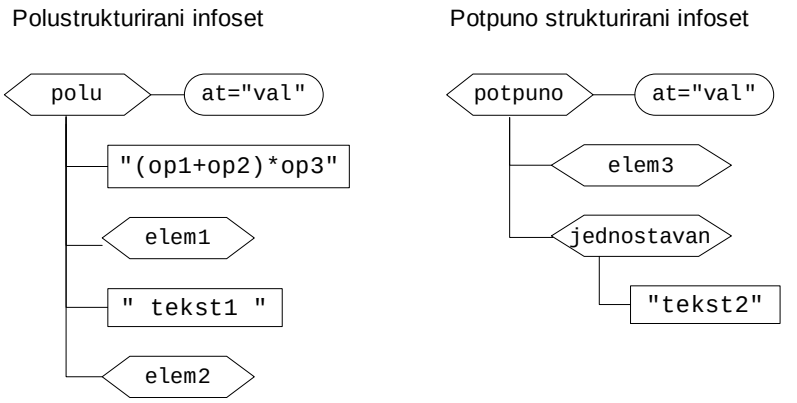
- *Jednostavan* sadržaj sastoji se od jednog tekstualnog čvora (npr. sadržaj elementa `elem4` na slici 2.2);

- *Složen* sadržaj sastoji se od niza elemenata (npr. sadržaj elementa `jednostavan` na slici 2.2).

- *Mješovit* sadržaj sastoji se od niza elemenata između kojih se pojavljuje barem jedan tekstualni čvor (npr. sadržaj elementa `polu` na slici 2.2).



Slika 2.1: Prevođenje XML-ovskog dokumenta `<a><b at="val"/></a>` u infoset




```
<polu at="val">(op1+op2)*op3<elem1/> tekst1 <elem2/></polu>

<potpuno at="val">
  <elem3/>
  <jednostavan>tekst2</jednostavan>
</potpuno>
```

Slika 2.2: Usporedba potpuno i polustruktuiranog infoseta

Atributi se ne smatraju dijelom sadržaja elementa pa njihova (ne)prisutnost ne utječe na vrstu sadržaja. Korisno je definirati i pojam *jednostavnog elementa* koji ima jednostavan sadržaj i nema atributa (npr. element `jednostavan` na slici 2.2).

Za ovu disertaciju ključan značaj ima infoset koji zadovoljava dva dodatna ograničenja, ilustrirana slikom 2.2:

1. Nijedan element nema mješovit sadržaj. Ovo se može formalizirati zamjenom produkcije sadržaja u ispisu 2.1 sljedećom:

   ```
   Sadržaj ::= Element+ | Tekst
   ```

2. Nijedan znakovni niz koji se pojavljuje u tekstualnom čvoru ili vrijednosti atributa nema strukturu, tj. ne zahtijeva parsiranje prema posebnoj sintaksi. Npr. niz "`(op1+op2)*op3`" nije prihvatljiv, a niz "`op2`" jest.

Takav infoset nazivat će se *potpuno strukturiranim*, za razliku od *polustrukturiranog* infoseta koji ne zadovoljava ta ograničenja. U zapisima na slici 2.2 može se uočiti kako zapis potpuno strukturiranog infoseta ima slobodniji oblik jer se može ignorirati prazan prostor u zapisu složenog sadržaja. U zapisu mješovitog sadržaja svi znakovi praznog prostora pripadaju tekstualnim čvorovima, kao što se vidi na primjeru `" tekst1 "`.

U potpuno strukturiranom infosetu postoji visok stupanj srodnosti između atributa i jednostavnih podelemenata – oba imaju samo ime i tekstualnu vrijednost, a temeljno razlikovno svojstvo atributa je njihova neporedanost.

2.2 XML-ovi prostori imena

XML-ov model stabla s imenovanim čvorovima pruža priliku da se primijeni koncepcija *prostora imena* – disjunktnih podskupova skupa svih dozvoljenih imena čvorova [nsp]. Svako ime pripada najviše jednom od prostora imena. Budući da je XML dizajniran za interoperabilnost na globalnoj razini, potrebno je osigurati da se za svaku primjenu može definirati jedinstveni prostor imena koji se neće moći pomiješati s drugima. Taj cilj ostvaruje se primjenom univerzalnog identifikatora resursa (*uniform resource identifier, URI*) kao identifikatora prostora imena. Najčešće se koristi HTTP-ovski URI u obliku `http://`*`ime-domene/segmenti`*. Za globalnu jedinstvenost najvažnije je ime (internetske) domene čiju jedinstvenost osigurava globalno internetsko tijelo za dodjelu prava na korištenje domenskih imena (*Internet Corporation for Assigned Names and Numbers*, ICANN). Daljnji segmenti omogućuju da se na osnovu jednog imena domene kreira više prostora imena. Ova konvencija korištena je i u primjeru koji slijedi niže.

Kojem prostoru pripada ime utvrđuje se prema početnom nizu znakova (*prefiksu*) koji je od ostatka (*lokalnog imena*) razdvojen dvotočkom. Dakle, općenit oblik imena čvora u infosetu je *`prefiks`*`:`*`lokalno-ime`*. URI se ne može izravno koristiti kao prefiks imena čvora – osim što je nepraktično dug, sadrži i znakove koji nisu dozvoljeni u imenu čvora. Stoga standard XML propisuje da se unutar dokumenta za svaki korišteni URI definira njegov kratki "nadimak" koji se koristi kao prefiks. U XML-ovskom dokumentu može se definirati *podrazumijevani* prostor imena i tada u njega spadaju sva imena koja nemaju prefiks. Primjer koji slijedi ilustrira ova dva slučaja:

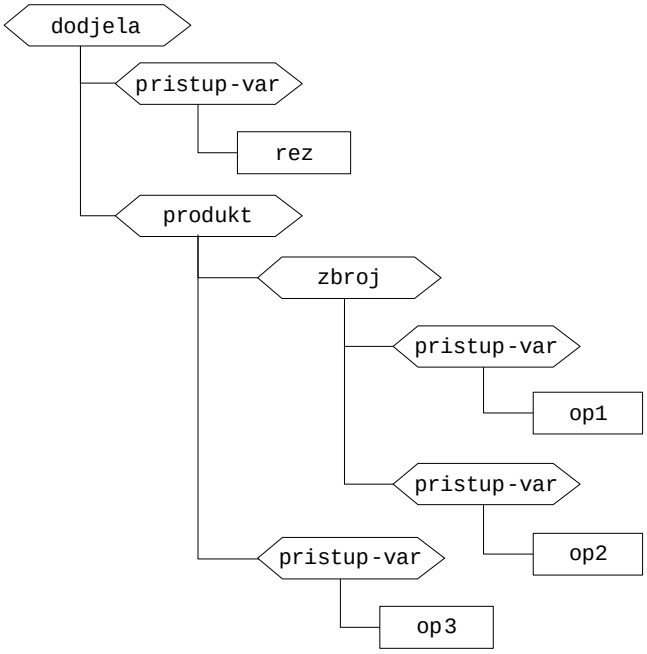
```
<root xmlns="http://tel.fer.hr/default" xmlns:j="http://tel.fer.hr/java">
  <a>Element iz prostora imena http://tel.fer.hr/default</a>
  <j:a>Element iz prostora imena http://tel.fer.hr/java</j:a>
</root>
```

U slučaju da ime nema prefiks, a nije definiran ni podrazumijevani prostor imena, prema specifikaciji [nsp] takvo ime nije ni u jednom prostoru imena, npr:

```
<root>Ovaj element nije ni u jednom prostoru imena</root>
```

2.3 Modeliranje stabla apstraktne sintakse infosetom

Potpuno strukturiranim infosetom može se modelirati stablo apstraktne sintakse programskog jezika[4], tj. za svako stablo apstraktne sintakse može se izgraditi ekvivalentan infoset. Njega će se nazivati *infosetom izvornog koda*. Na primjer, sve unutarnje čvorove može se modelirati elementima, a listove tekstualnim čvorovima. Slika 2.3 prikazuje infoset izvornog koda ekvivalentan stablu sa slike 1.10 (imena čvorova prilagođena su uobičajenim konvencijama za XML). Mogući su i drugačiji načini modeliranja, na primjer zapisom imena varijable u atributu.




```
<dodjela>
  <pristup-var>rez</pristup-var>
  <produkt>
    <zbroj>
      <pristup-var>op1</pristup-var>
      <pristup-var>op2</pristup-var>
    </zbroj>
    <pristup-var>op3</pristup-var>
  </produkt>
<dodjela>
```

Slika 2.3: Infoset izvornog koda ekvivalentan stablu sa slike 1.10 i njegov XML-ovski zapis

[4]Ovdje treba paziti da se ne pomiješa stablo apstraktne sintakse XML-a (slika 2.1 desno) sa stablom apstraktne sintakse programskog jezika (slika 1.10)

Rezimirajmo dosad utvrđene činjenice:

1. Programski jezik definira osnovne pojmove i pravila njihovog kombiniranja u formalnom opisu željenog ponašanja programa (stablu apstraktne sintakse).
2. Programski jezik također definira konkretnu sintaksu tekstualnog izvornog koda koji služi za zapis formalnog opisa ponašanja iz točke 1.
3. Za svako stablo apstraktne sintakse postoji ekvivalentan infoset.
4. Svaki infoset može se zapisati u tekstualnom obliku pomoću jezika XML, tj. u XML-ovskom dokumentu.

Zaključak: XML-ovskim dokumentom može se zapisati značenje proizvoljnog izvornog koda. Ovo je pojašnjeno slikom 2.4.

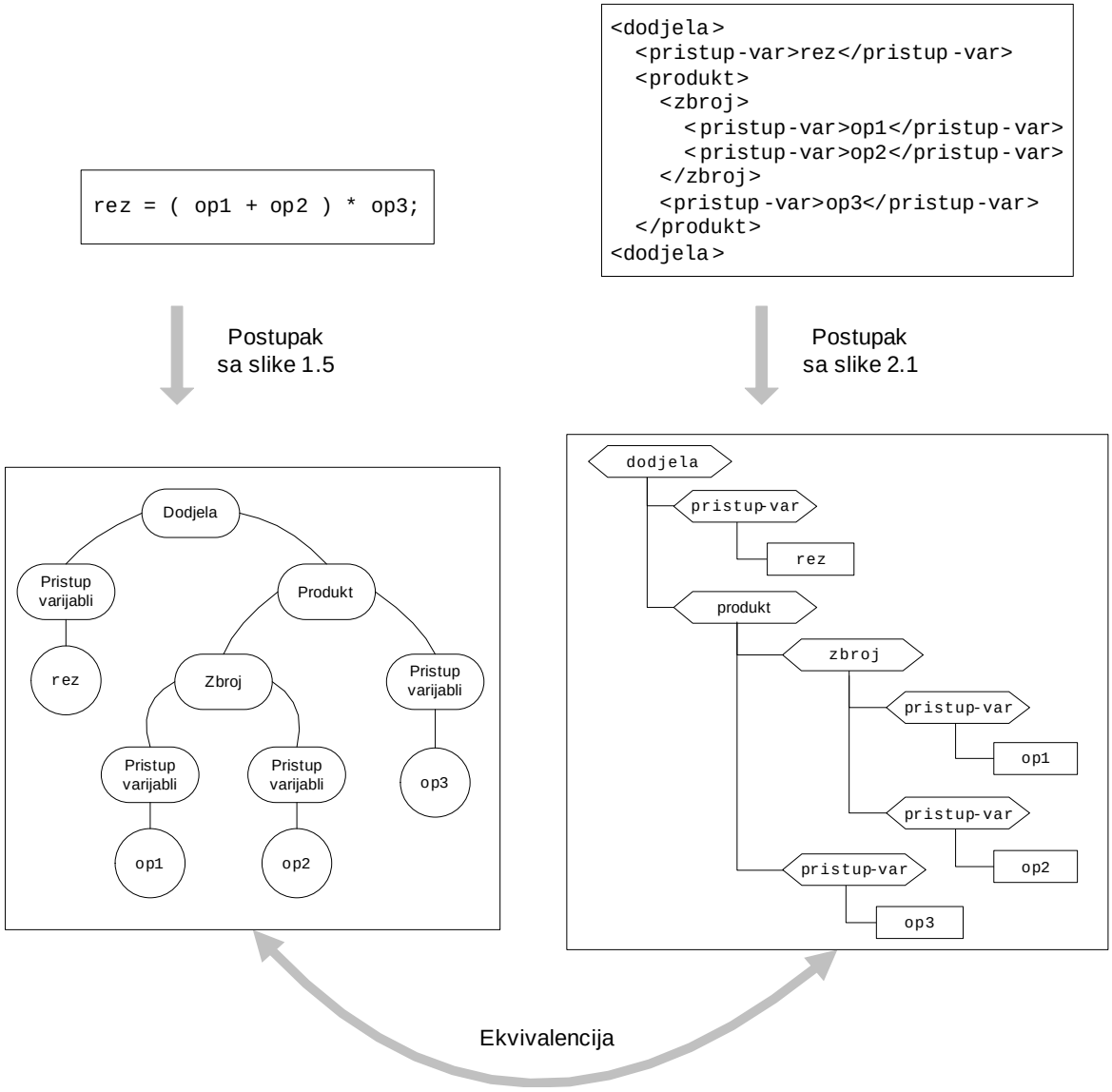
2.4 Standardni jezici za definiciju sheme dokumenta

2.4.1 Jezik DTD

Jezik *Document Type Definition* (Definicija vrste dokumenta), ili kraće DTD, standardiziran je 1986. godine kao dio standarda Međunarodne organizacije za standarde (*International Standards Organization*, ISO) zvanog *Standardni poopćeni jezik za označavanje* (*Standard Generalized Markup Language*, SGML). Jezik XML nastao je na osnovi SGML-a, prije svega pojednostavljenjem njegove konkretne sintakse, pa je razvijena i jednostavnija specifikacija DTD-a prilagođena XML-u. Ključna komponenta jezika je deklaracija elementa, kojom se definira njegovo ime i propisuje *model sadržaja*. Na slici 2.5 iznesena je (malo pojednostavljena) konkretna sintaksa deklaracije elementa prema specifikaciji [xml04]. Iz ispisa je izostavljeno pravilo deklaracije elementa s mješovitim sadržajem jer on nije bitan u ovom kontekstu. Izostavljena su također pravila za neterminale `Name` i `S`, koja su već navedena u Dodatku A.

Iz slike 2.5 mogu se razaznati sljedeći elementi definicije modela sadržaja:

1. Osnovna komponenta zove se *čestica*. To može biti jedan element ili *grupa* daljnjih čestica.
2. Unutar zagrada odvojeno zarezima navodi se grupa čestica koje se moraju pojaviti u sadržaju jedna za drugom.



Slika 2.4: Standardni i XML-ovski dokument izvornog koda

```
elementdecl ::= '<!ELEMENT' S Name S contentspec S? '>'
contentspec ::= 'EMPTY' | 'ANY' | Simple | children
Simple ::= '(' S? '#PCDATA' S? ')'
children ::= (choice | seq) ('?' | '*' | '+')?
choice ::= '(' S? cp ( S? '|' S? cp )* S? ')'
seq ::= '(' S? cp ( S? ',' S? cp )* S? ')'
cp ::= (Name | choice | seq) ('?' | '*' | '+')?
```

Slika 2.5: EBNF za DTD-ovu deklaraciju elementa

3. Unutar zagrada odvojeno okomitim crtama navodi se grupa čestica od kojih se treba jedna (bilo koja) pojaviti u sadržaju.

4. Zvjezdica desno od čestice propisuje Kleeneovo zatvaranje nad česticom, dakle nula ili više njenih pojava zaredom.

5. Upitnik desno od čestice propisuje da se česticu može izostaviti (nula ili jedna pojava).

6. Plus desno od čestice propisuje da se može pojaviti jednom ili više puta zaredom.

7. Pomoću `(#PCDATA)` propisuje se jednostavan sadržaj.

Ovaj popis pravila zapisa deklaracije elementa vrlo je srodan pravilima zapisa produkcije u EBNF-u. Na primjer, sljedeća deklaracija elementa:

```
<!ELEMENT Slozen ( A, B?, (C|D)* ) >
```

odgovarala bi sljedećoj produkciji neterminala:

```
Slozen ::= A B? (C|D)*
```

Stoga se može uspostaviti precizna analogija između DTD-ovske sheme infoseta i EBNF-ovske gramatike teksta: elementi odgovaraju neterminalima, a jednostavan sadržaj bilo kojem terminalu. U karakteristici DTD-a da, za razliku od EBNF-a, ne razlučuje terminale leži i ključna formalna razlika između sheme i gramatike: gramatika služi za parsiranje, a shema samo za provjeru valjanosti već izgrađenog stabla. Proces parsiranja oslanja se na razlikovanje terminala da bi utvrdio koji izvod primijeniti u izgradnji stabla parsiranja, a za provjeru valjanosti to je nepotrebno.

DTD uvodi i dva dodatna oblika sintaktičkih pravila koji nadilaze izražajnost beskontekstne gramatike i povezani su s relacijskim modelom podataka: ograničenje primarnog i stranog ključa. Za određeni atribut elementa može se propisati da je on primarni ključ i sve njegove vrijednosti u dokumentu moraju biti jedinstvene. Isto tako, za neki drugi atribut (nekog drugog elementa) može se propisati da je on strani ključ pa svaka njegova vrijednost mora biti jednaka nekoj postojećoj vrijednosti atributa-primarnog ključa.

2.4.2 Jezik WXS

W3C XML Schema (WXS) [xsc01a, xsc01b] je W3C-ov jezik za definiciju sheme razvijen specifično za XML i u zadnje vrijeme uspješno se nameće kao globalni standard. Sintaksa WXS-a je XML-ovska, tj. definicija sheme je XML-ovski dokument. WXS je po koncepciji vrlo srodan DTD-u, ali donosi i neke nove mogućnosti, od kojih su ovdje izdvojene najznačajnije:

1. Uvodi koncepciju *jednostavne vrste podatka* i primjenjuje ju na jednostavan sadržaj (ili vrijednost atributa) kojem se može detaljnije ograničiti struktura i, još važnije, omogućiti značenjem bogatije tumačenje vrijednosti. Postoji bogat skup ugrađenih vrsta na osnovu kojih se mogu definirati nove.

2. Uvodi koncepciju *složene vrste podatka* i primjenjuje ju na složen sadržaj elementa.

3. Uvodi *hijerarhiju* vrsta podataka. Novu vrstu se izvodi iz već postojeće proširivanjem ili sužavanjem. Sužavanje znači povećanje strogosti, a za proširivanje su pravila nešto složenija.

4. Poopćava mehanizam nametanja primarnog/stranog ključa tako da ključevi mogu biti i jednostavni elementi, a ne samo atributi. Može se također propisati više atributa/elemenata kao nositelje istog primarnog ključa.

5. Podržava XML-ove prostore imena. DTD, razvijen za SGML koji ne barata tim pojmom, ima vrlo ograničenu i nepraktičnu podršku za njih.

U WXS-ovskoj shemi se vrstama podataka, pa tako i čitavim modelima sadržaja u slučaju složene vrste, može dodijeliti identifikator, čime se omogućuje njihovo višestruko iskorištavanje. Identifikator se može dodijeliti i složenoj čestici, tj. grupi elemenata (izbor ili niz). Primjer zapisa ekvivalentnog gornjem primjeru za DTD-ovsku deklaraciju elementa:

```
<xs:group name="Cd">
  <xs:choice>
    <xs:element ref="C"/>
    <xs:element ref="D"/>
  </xs:choice>
</xs:group>
<xs:complexType name="Abcd">
  <xs:sequence>
    <xs:element ref="A"/>
    <xs:element ref="B" minOccurs="0"/>
```

```
    <xs:group ref="Cd" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="Slozen" type="Abcd"/>
```

Može se koristiti i jednostavniji način deklaracije, bez dodjele identifikatora česticama i vrstama:

```
<xs:element name="Slozen"/>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="A"/>
      <xs:element ref="B" minOccurs="0"/>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="C"/>
        <xs:element ref="D"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

2.5 Standardni jezici za baratanje infosetom

2.5.1 Opis puta kroz infoset jezikom *XPath*

XPath [xpt99] je W3C-ov standardni jezik čija prvenstvena namjena je adresiranje skupa čvorova u infosetu opisivanjem puta do njih. Izraz kojim se adresira skup čvorova naziva se *lokacijski put* (eng. *location path*). Označimo s I skup svih čvorova u nekom infosetu, a s 2^I skup svih podskupova od I. Lokacijski put je kompozicija funkcija zvanih *koraci* puta. To su funkcije sljedećeg oblika:

$$K : 2^I \to 2^I$$

Parametar koraka (skup čvorova iz infoseta) naziva se *kontekstom* i označava s C:

$$K = K(C)$$

Funkciju lokacijskog puta L možemo uvesti rekurzivno:

Baza. Za svaki korak K postoji put L takav da je $L(C) = K(C)$ za svaki kontekst C.

Indukcija. Ako su L i K neki lokacijski put i korak puta, tada je $(L \circ K)(C) = K(L(C))$ također lokacijski put.

Funkcija K uvijek je definirana preko funkcije k jednog čvora:

$$k : I \to 2^I$$

$$K(C) = \bigcup_{i=1}^{n} k(c_i, i, n)$$

gdje su c_i čvorovi iz C nad kojima je nametnut poredak u skladu s poretkom njihovog pojavljivanja u XML-ovskom dokumentu, a n je kardinalni broj skupa C. Najvažnije je uočiti da funkcija k ne ovisi o cjelokupnom sadržaju konteksta, već samo o položaju pojedinog čvora i ukupnom broju čvorova u njemu.

Općeniti lokacijski put je, dakle, kompozicija od n koraka puta:

$$L(C) = (K_1 \circ K_2 \circ ... \circ K_n)(C),$$

a *XPath*-ovim izrazom zapisuju se definicije odgovarajućih funkcija k_i, redom od k_1 do k_n, odvojene kosom crtom. U svom najjednostavnijem obliku, npr. `a/b/c`, lokacijski put izgleda isto kao zapis puta do neke datoteke kroz hijerarhiju datotečnog sustava. Suštinska je razlika u tome da navedeni izraz može adresirati i nekoliko čvorova u infosetu, dok u datotečnom sustavu uvijek adresira najviše jednu datoteku. Postoji sljedeća analogija između ta dva zapisa:

- Korak koji se sastoji samo od identifikatora predstavlja odabir čvora-djeteta kontekstualnog čvora. Ime djeteta mora biti jednako navedenom identifikatoru.
- Simbolom `.` odabire se kontekstualni čvor (funkcija identiteta).
- Simbolom `*` odabiru se svi čvorovi-djeca bez obzira na ime.
- Simbolom `..` odabire se roditeljski čvor (također bez obzira na ime).
- Može se definirati *apsolutni* put koji nije ovisan o kontekstu, npr. put `/a/b/c` bez obzira na kontekst kreće od korijena stabla. Formalno to znači da je $K_1(C) \equiv konst$.

U zapisu puta u datotečnom sustavu svaki sljedeći korak uglavnom implicira spuštanje niz hijerarhiju stabla. Simbol `..` implicira kretanje prema gore, ali u tom slučaju ne može se zadati ime čvora. U *XPath*-u se za svaki segment može zasebno definirati i smjer kretanja (prema gore, prema dolje ili pobočno) i ime čvora. Selektor smjera kretanja zove se *osovina* i zapisuje se ispred imena čvora uz poseban simbol (dvije dvotočke) kao u ovom primjeru: `b/parent::a`. Neke od osovina su: *roditelj*, *predak*, *dijete*, *potomak*, *brat*, *atribut* (do atributa se ne može doći osovinom *dijete* jer se njome dolazi samo do čvorova u sadržaju elementa).

Osim spomenutog izraza lokacijskog puta, jezik *XPath* podržava i numeričko-logičke izraze. Rezultat tih izraza nije skup čvorova, već brojevna ili logička vrijednost. Podržane su i operacije nad znakovnim nizovima. Logički izrazi mogu se primijeniti i kao dio lokacijskog puta, u *predikatu*. To je logički izraz vezan uz korak puta čija se vrijednost izračunava u odnosu na svaki čvor odabran kombinacijom osovine i imena. Ako čvor ne zadovolji izraz, on se izbacuje iz rezultata koraka. Predikat se zapisuje kao u ovom primjeru:

```
/a/b[@atr="val"]
```

Izraz će odabrati svaki `a`-ov podelement `b` kojem je vrijednost atributa[5] `atr` znakovni niz "val".

2.5.2 Upit nad infosetom u jeziku *XQuery*

Pred standardizacijom je jezik za upite nad infosetom, *XQuery* [xqr] (trenutno "public working draft" W3C-a). Sintaksa upita srodna je sintaksi jezika za upite nad relacijskim bazama podataka, SQL-a. Formalno, ovaj jezik možemo promatrati i kao proširenje *XPath*-a jer je svaki *XPath*-ov ujedno i *XQuery*-jev izraz. Glavni dodaci koje donosi *XQuery* su *konstruktori čvorova* i izraz "FLWOR" (skraćeno od "*for, let, where, order by, return*"; izgovara se kao engleska riječ *flower*) koji funkcionalno odgovara izrazu SELECT iz SQL-a. Jezik omogućuje sastavljanje XML-ovskog izvještaja na osnovu obrade podataka iz jednog ili više ulaznih dokumenata. Jedan primjerak izraza FLWOR izgledao bi ovako:

```
for $fourthMember in document( "OrExpressionTest.jmlm" )
  //ConditionalAndExpression/InclusiveOrExpression[4]
let $parExp := $fourthMember/PostfixExpression/*/PrimaryPrefix/Expression
where count( $fourthMember/* ) = 1 and exists( $parExp )
order by $parExp/@Identifier
return <fourth-member> { $parExp } </fourth-member>
```

Pomoću `for` deklarira se ime varijable (ovdje `$fourthMember`) koja će iterirati po svim elementima niza koji vrati izraz nakon ključne riječi `in`. Dakle, čitav ostatak prikazanog izraza izračunavat će se redom za svaki član tog niza, a varijabla `$fourthMember` će sadržavati dotični član. Pomoću `let` deklariraju se "obične" varijable (koje nisu iteratori, već sadrže čitave nizove); ovdje je deklarirana varijabla `$parExp` koja će sadržavati niz svih elemenata zadanih *XPath*-ovom stazom `$fourthMember/PostfixExpression/*/PrimaryPrefix/Expression`. Pomoću `where` obavlja se filtriranje niza elemenata odabranih pomoću `for`, a pomoću `order by` definira

[5] Znak `@` služi za kraći zapis osovine `attribute::`

se kriterij poretka po kojem treba prolaziti po nizu. Napokon, pomoću `return` definira se povratna vrijednost upita (izlaz).

U prvom retku treba uočiti važnu funkciju `document` pomoću koje se u samom upitu definira ulazni dokument. Općenito, specifikacija *XQuery*-ja dopušta i da se upit izvršava u kontekstu izvana određenog dokumenta, ali u ovom slučaju ne oslanja se na vanjski kontekst.

U izjavi `for` može se uočiti i primjena složenijeg *XPath*-ovog izraza u definiranju niza po kojem će iterirati varijabla.

U izjavi `return` može se uočiti konstruktor elementa, `<fourth-member>`. Njime se kreira element koji nije došao iz ulaznog dokumenta. Unutar zapisa elementa `fourth-member` naveden je ugniježđeni izraz u vitičastim zagradama `{$parExp}`. Njegova povratna vrijednost (dakle, sadržaj varijable `$parExp`) bit će ugrađena u element `fourth-member` kao njegov sadržaj.

2.5.3 Transformacija infoseta jezikom XSLT

Danas je već u širokoj uporabi W3C-ov jezik za opis transformacije iz infoseta u infoset – *Extensible Stylesheet Language Transformations* (XSLT) [xsl99]. Ime je dobio po tome što mu je prvotna ciljana primjena bila za pretvorbu čistog sadržaja u oblik spreman za tiskanje uporabom *stylesheet*-a (formalnog opisa stila prikaza). Jezik je dio tehnologije XSL, koja dodatno sadrži i vokabular za opis izgleda stranice. XSLT definira standardni tijek obrade dokumenta prolaskom po svim čvorovima i on je implicitan – ne opisuje se XSLT-ovim izvornim kodom. Ono što se opisuje su *predlošci* – kodni fragmenti s pridijeljenim uzorkom zapisanim u *XPath*-u. Proces obrade pri nailasku na svaki sljedeći čvor provjerava zadovoljava li on uzorak definiran u nekom od predložaka i ako se takav predložak nađe, izvrši se kod definiran u njemu. Ovaj pristup znatno olakšava razvoj automatiziranih zadataka u kojima treba većinu izvornog dokumenta zadržati nepromijenjenu te zadataka u kojima postoji veza bliska vezi jedan-na-jedan između konstrukcija u ulaznom i izlaznom dokumentu. Sam jezik XSLT ima XML-ovsku sintaksu čime je olakšano doslovno navođenje XML-ovskog sadržaja koji treba staviti u izlazni dokument. Slijedi primjer jednog XSLT-ovog predloška, izvađen iz *stylesheet*-a za transformaciju infoseta izvornog koda programskog jezika Java u infoset koji sadrži sve detalje zapisa istog izvornog koda u tradicionalnom obliku.

```
<xsl:template match="j:import">
  <s:import>
    <l:kw val="import"/>
```

```
    <l:ws/>
    <xsl:choose>
      <xsl:when test="j:reference-type">
        <xsl:apply-templates select="j:reference-type"/>
      </xsl:when>
      <xsl:otherwise>
        <s:all-from>
          <xsl:apply-templates select="j:all-from/@package"/>
          <l:sym val="."/>
          <l:sym val="*"/>
        </s:all-from>
      </xsl:otherwise>
    </xsl:choose>
    <l:sym val=";"/>
    <l:ws-opt nl="1"/>
  </s:import>
</xsl:template>
```

Ovaj predložak aktivirat će se pri nailasku na element `import` (iz prostora imena sintakse izvornog koda) u ulaznom dokumentu. Izvršenjem koda u izlazni dokument dodat će se element `import` (iz prostora imena posebne sintakse za opis teksta tradicionalnog izvornog koda) sa sadržajem koji opisuje prikazani kod. Svi elementi s prefiksom `xsl` su XSLT-ovi jezični elementi, a svi ostali elementi mogu se, u terminima *XQuery*-ja, smatrati konstruktorima. Od XSLT-ovih elemenata jedan od najvažnijih je `apply-templates` koji omogućuje izdavanje eksplicitne naredbe za obradu nekog skupa čvorova. Najvažnije je da to omogućuje da se rezultat njihove obrade ubaci na željeno mjesto unutar izlaznog dokumenta.

2.6 Sustavni pristup uvođenju XML-ovskog izvornog koda

Budući da se apstraktna sintaksa redovito ne formalizira jezičnom specifikacijom pa općenito nema ni mnogo iskustva u njenom oblikovanju, a naročito nedostaje iskustva u oblikovanju *sintakse infoseta izvornog koda*, potreban je sustavan i temeljit pristup toj problematici. Istraživanjem dosadašnjih rezultata uočeno je da u zajednici koja se bavi pitanjima primjene XML-a na izvorni kod izostaje dovoljno sustavan pristup i posebice da se već donekle i udomaćila **pogrešna interpretacija pojma "apstraktna sintaksa"**. Izraz "stablo apstraktne sintakse" koristi se za vrstu stabla kakvo se koristi u izvedbama kompilatora. Međutim, sintaksa tih stabala je redovito vrlo slična konkretnoj sintaksi jezika i zadržava mnoge detalje zapisa koji nisu relevantni za značenje, a često i ne razlučuje sva značenja (koristi

iste konstrukcije za različita značenja). Takva sintaksa nije, dakle, apstraktna. U mnogim raspravama, prije svega u [Bad00, ZK01], izraz se koristi u tom značenju, a često se navodi i kao uzor prema kojem treba oblikovati shemu dokumenta izvornog koda. Negativne posljedice takvog shvaćanja uočene su u analiziranim sintaksama.

Radi daljnjeg napretka na području primjene XML-a na izvorni kod neophodno je stoga sustavno provesti sljedeći postupak:

1. Utvrditi **prednosti** koje može donijeti primjena XML-a na izvorni kod.
2. Utvrditi **zahtjeve** na sintaksu infoseta izvornog koda koje ona treba zadovoljiti da bi omogućila maksimalno iskorištavanje tih prednosti.
3. Utvrditi **načela** kojima se treba voditi prilikom oblikovanja sintakse da bi zadovoljila identificirane zahtjeve.

U istraženim radovima korak 1 je proveden, ali ne sasvim zadovoljavajuće, a koraci 2 i 3 nisu gotovo uopće provedeni i bili su prvenstveni predmet vlastitog istraživanja. U nastavku će biti izloženi rezultati sustavnog provođenja sva tri koraka.

Važno je također imati na umu da se o jednoj jedinstvenoj apstraktnoj sintaksi programskog jezika može govoriti samo ako se pod time smatra skup neformalnih pravila opisanih u jezičnoj specifikaciji. Kad se govori o formaliziranoj sintaksi, ona uopće ne postoji dok ju se ne definira pa samim time može postojati mnoštvo različitih formalizacija od kojih se svaku može smatrati apstraktnom sintaksom dotičnog jezika, dok god ona zadovoljava neformalno opisana pravila. Nemoguće je i definirati formalni postupak kojim bi se utvrdilo zadovoljava li formalizirana sintaksa neformalna jezična pravila. To je uostalom nemoguće čak i za konkretnu sintaksu formaliziranu specifikacijom. Situacija je dodatno otežana time što različite sintakse, čak i uz pretpostavku ispravnosti u navedenom smislu, neće biti jednake po prikladnosti za predviđeni način njihovog korištenja u razvoju programske podrške. Iz ovih razloga i sama argumentacija rezultata provođenja gore skiciranog postupka ne služi se formalizmima i kao jedini zaista objektivni kriterij ocjene njihove valjanosti iznosi rezultate usporedbe prikladnosti predložene i nekoliko već postojećih sintaksi u rješavanju konkretnih problema.

Poglavlje 3

Prednosti primjene XML-a za zapis izvornog koda

3.1 XML kao globalni standard za zapis stabla

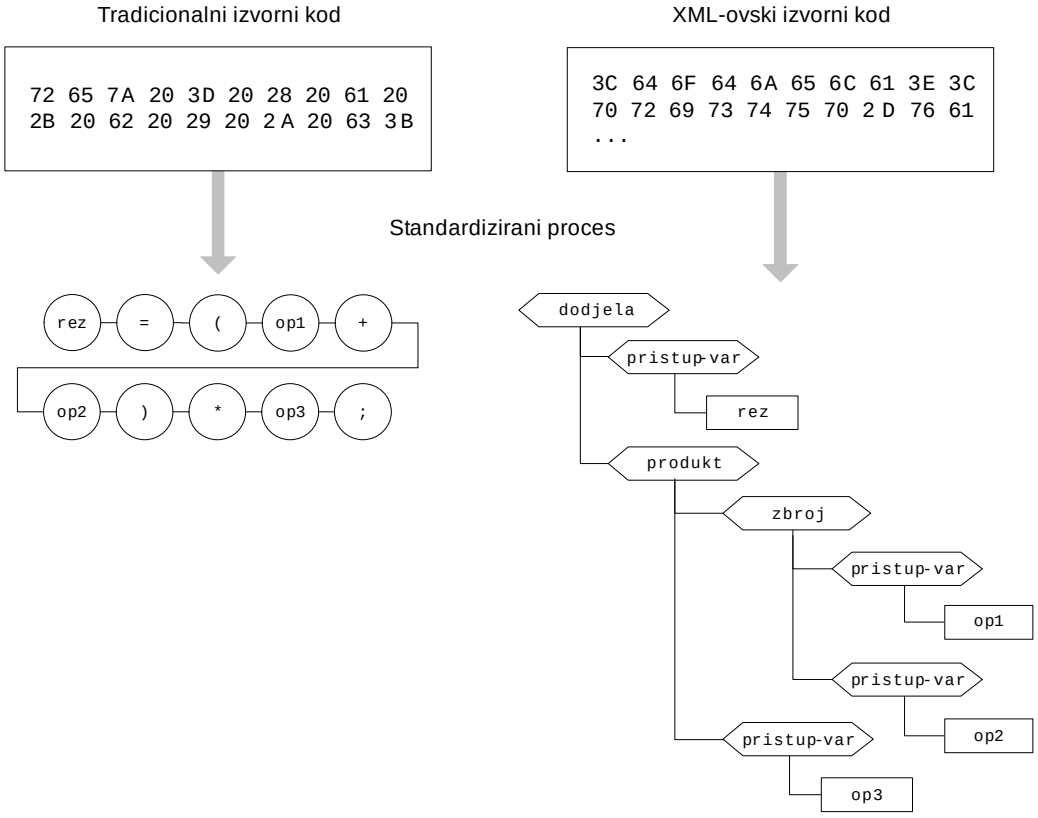
Globalni standardi zapisa podataka imaju velik utjecaj na računarski svijet. Njihovu važnost najbolje je ilustrirati na primjeru ASCII-ja koji već nekoliko desetljeća ima status globalnog standarda za zapis teksta. Danas je uobičajeno za tekst kodiran po ASCII-ju koristi izraz "običan tekst". Izraz dobro ilustrira posljedicu ASCII-jeve globalnosti – toliko je sveprisutan da se pitanje kodiranja većinom može ignorirati i smatrati da računalo izravno barata samim tekstom. Standard ASCII omogućio je razvoj velikog broja alata i programskih jezika posvećenih obradi teksta. U mnogo slučajeva središnju ulogu igra regularni izraz kojim se opisuju uzorci u tekstu.

U novije vrijeme stupanj prihvaćenosti standarda XML već je gotovo dosegao razinu ASCII-ja. XML se oslanja na postojeće standarde za kodiranje teksta i dodatno standardizira konkretnu sintaksu za opis infoseta. Analogno se dolazi do situacije gdje se u mnogo slučajeva može ignorirati problem tekstualnog zapisa XML-a i smatrati da računalo izravno barata samim infosetom.

Vratimo se još jednom na sliku 1.11. Samo korak 1 je standardiziran – binarni zapis se dekodira prema npr. ASCII-ju u tekst. Regularna gramatika potrebna za korak 2 (leksička analiza) slična je za većinu programskih jezika i, izuzevši nekoliko detalja, taj korak se može riješiti općenito. Za korak 3 (parsiranje) potrebna je kompletna beskontekstna gramatika konkretne sintakse, a ona se značajno razlikuje od jezika do jezika. Dakle, za izvođenje ovog koraka već je potrebno pozna-

vati detalje svakog pojedinog programskog jezika. Napokon, korak 4 je također specifičan za jezik, ali dodatan problem je da apstraktna sintaksa jezika u većini slučajeva nije formalizirana pa za njegovo provođenje nije dovoljno niti poznavanje specifikacije jezika.

Sve u svemu, bez formalizacije jezičnih pravila tradicionalni izvorni kod može se promatrati kao niz znakova, a uz blagu specijalizaciju općenitih pravila leksičke sintakse kao niz tokena. S druge strane, postupak sa slike 2.1 u potpunosti je standardiziran i isključivim oslanjanjem na globalne standarde moguće je doći od niza bitova do infoseta izvornog koda. Ova razlika ilustrirana je slikom 3.1



Slika 3.1: Usporedba dosega standardiziranog tumačenja izvornog koda

3.2 Ortogonalna proširenja

Primarni cilj dokumenta s izvornim kodom je da sadrži značenje izraženo preko osnovnih pojmova programskog jezika, drugim riječima da opisuje stablo apstraktne sintakse. Taj aspekt dokumenta definirat ćemo kao njegov *primarni sadržaj*.

Dokument igra i mnogo drugih uloga u širem kontekstu razvojnog procesa. Neke od tih uloga zahtijevaju uključivanje dodatnih sadržaja u dokument koji nisu potrebni kompilatoru. Svi takvi sadržaji mogu se smatrati *ortogonalnim proširenjima* dokumenta s izvornim kodom [ST03]: "proširenjima" zato što nisu dio primarnog sadržaja dokumenta, a "ortogonalnim" zato što su tako uvedena da ne utječu na automatski proces obrade primarnog sadržaja (npr. prilikom kompilacije). Slijedi popis nekoliko ideja za korisna ortogonalna proširenja. Podrška za njih u današnjim okružjima varira od nepostojeće do prihvatljive.

Dokumentacija programskog sučelja može se ubaciti uz sam izvorni kod, npr. uz svaku javnu metodu objekta. To je praktično ne samo kao podsjetnik razvijatelju, već i zato što je razvijatelj taj koji je najčešće zadužen i za pisanje dokumentacije. Ovaj pristup osigurava i usklađenost koda i dokumentacije – nemoguće je da postoji dokumentacija za nepostojeći kod, a i lako je ustanoviti za koji dio koda nedostaje dokumentacija. Ovo proširenje trenutno je najkvalitetnije podržano pa je dobar primjer za ilustraciju fundamentalnih nedostataka tradicionalnog izvornog koda (vidjeti niže).

Osnovni podaci o dokumentu: datum stvaranja i zadnje izmjene, izvorni autor, autori izmjena, podaci o autorskim pravima itd. Za ovu vrstu proširenja XML-ovskog dokumenta već je standardizirana sintaksa zvana Dublin Core [dc02].

Bilješke, pojašnjenja i ostali komentari koda su primjeri vrsta podataka koje razvijatelji obično smještaju unutar kodnih komentara. Sadržaj takvih komentara je potpuno slobodan i samim time ne nosi nikakvo formalno značenje. Zbog toga se ne može automatski ustanoviti njihova priroda i namjena.

Zadaci su kratki komentari vezani uz određeni dio koda koje postavlja razvijatelj kad uoči neki problem, npr. nepravilnu ili nepostojeću funkcionalnost, koji nije mogao odmah riješiti. To također mogu biti i upute voditelja tima pojedinim razvijateljima što i gdje trebaju napraviti. Dobra podrška za ovu mogućnost uključivala bi atribute kao što su važnost, autor zadatka, razvijatelj kojem je dodijeljeno riješavanje, rok dovršenja i sl. Na osnovu toga bilo bi moguće analizirati čitav projekt i sastaviti npr. popis svih zadataka dodijeljenih pojedinom razvijatelju. U današnjim alatima postoji dosta skromna podrška za zadatke: tekst zadatka ubacuje se u komentare koji počinju nekom ključnom riječi, npr. `TODO`. Alat prikuplja sve pojave zadataka i može sastaviti izvještaj o njima. Nema podrške za bilo što osim teksta zadatka.

Praćenje verzija i izmjena neizostavan je dio timskog razvoja koda. O tome često vodi računa poseban repozitorij dokumenata izvornog koda i čuva podatke o verzijama i promjenama izvan samih dokumenata. Budući da su podaci pospremljeni lokalno u repozitoriju, dolazi do problema pri seljenju dokumenata u drugi repozitorij i čak pri seljenju dokumenta na drugu lokaciju unutar istog repozitorija.

Kontrola pristupa kodu. Ako na istom dokumentu radi više osoba, one često imaju različite uloge. Na primjer, jedan razvijatelj može biti zadužen za pisanje koda, a drugi za pisanje dokumentacije. Razvojni proces bi profitirao kad bi se takve uloge mogle eksplicitno specificirati unutar dokumenta tako da alat za obradu dokumenta može pojedinom razvijatelju nametati ograničenja u skladu s njegovom ulogom. Posebnu ulogu u razvoju ima recenzent koda. On ne smije ništa mijenjati, ali smije ubacivati komentare i predlagati izmjene koda. Razvijatelj tada može proučiti komentare i prijedloge i donijeti konačnu odluku. Podrška za recenzenta uključivala bi kontrolu pristupa, recenzentske bilješke i praćenje izmjena.

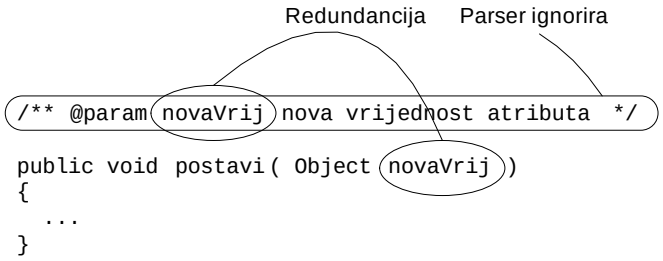
Privremeno neaktivan kod. Tijekom životnog vijeka nekog kodnog modula neki njegovi dijelovi zastarijevaju. Također, može se ukazati potreba za privremenim izbjegavanjem korištenja nekog modula čija funkcionalnost je trenutno narušena. U ovim i sličnim slučajevima ukazuje se potreba za privremenom deaktivacijom dijelova koda bez trajnog uklanjanja. Tradicionalno, ovo se postiže označavanjem deaktiviranog koda kao komentara (*iskomentirani kod*). Taj pristup izvor je mnogih komplikacija, npr. razlikovanje iskomentiranog koda od ostalih komentara i tretiranje komentara koji su već prisutni i iskomentiranom kodu i koji sada postaju ugniježđeni (npr. Java ne podržava ugniježđene komentare). Iskomentirani kod izuzet je od bilo kakve provjere valjanosti, međutim često je poželjno zadržati tu provjeru bez obzira na trenutačnu deaktiviranost.

Uvjetna kompilacija. Izvorni kod redovito sadrži neke dijelove koji ne ostvaruju ciljnu funkcionalnost, već su prisutni samo za potrebe provjeravanja i održavanja koda. Sve takve dijelove treba isključiti iz krajnjeg softverskog proizvoda, ali su neizostavni tijekom razvoja. Budući da je razvoj ispresijecan s objavljivanjem uporabnih verzija koda, ne postoji trenutak u kojem bi se sav takav kod mogao jednostavno ukloniti. Stoga je potreban mehanizam centraliziranog uključivanja/isključivanja raspršenih odsječaka koda. Ta teh-

nika se naziva uvjetnom kompilacijom. Srodna je privremenoj deaktivaciji koda, ali je potreban bogatiji model kontrole statusa akvitnosti odsječaka koda.

Kodni predlošci. U mnogim slučajevima, a posebno u primjeni raznih *framework*-a, u slučaju jezika Jave npr. *Remote Method Invocation* ili *Java Data Objects*, kod se mora držati nekih dodatnih propisa. U mnogim projektima propisuje se i držanje određenih konvencija u rješavanju standardnih problema, npr. korištenje uzorka *typesafe enum* (točka 21 u [Blo02]). Takvi propisi uvode stereotipnu redundanciju u kod. Moguće je uvesti ortogonalno proširenje unutar kojeg se zapisuju samo neredundantni podaci, a poseban alat ih uklapa u kodni predložak i generira kod koji se drži svih propisa.

U tradicionalni izvorni kod ortogonalna proširenja redovito se ubacuju kao *komentari*. Pravila zapisa svakog programskog jezika propisuju kako se označava početak i kraj odsječka teksta koji parser treba zanemariti. Izvanjezičnim pravilima propisuje se struktura teksta unutar komentara u skladu s potrebama proširenja. Ovaj pristup stvara krupne probleme pri razvijanju alata koji barataju proširenjem. Njima je redovito potreban dostup do sadržaja ne samo proširenja, već i izvornog koda. Sadržaj koda dostupan je jezičnom parseru i kompilatoru – zasebnim alatima koji prema specifikaciji ignoriraju komentare, pa time i ortogonalna proširenja. Alatu za proširenje preostaje jedino da duplicira posao koji oni obavljaju i implementira vlastito parsiranje koda.



Slika 3.2: Problemi s *Javadoc*-om

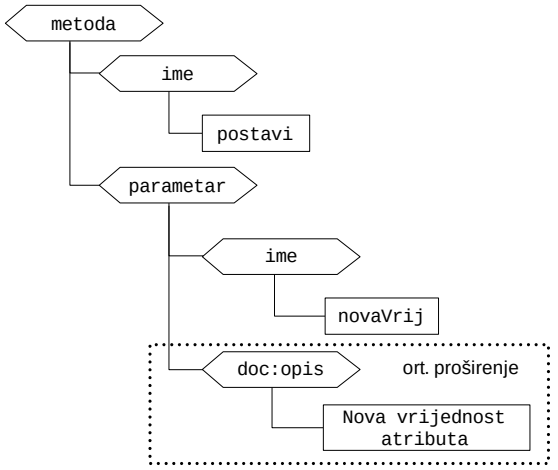
Manifestiranje ovog problema u praksi može se vidjeti na najpoznatijem primjeru ortogonalnih proširenja u Javi, dokumentaciji programskog sučelja *Javadoc*. Dokumentacija za npr. neku metodu ubačena je u dokument izvornog koda odmah iznad definicije metode, kao komentar. Poseban alat na osnovu takvih komentara generira HTML-ovsku dokumentaciju programskog sučelja. Razmotrimo primjer sa slike 3.2. Unutar *Javadoc*-ovog proširenja spominje se identifikator parametra

`novaVrij`. Postoji dakle redundancija između proširenja i osnovnog koda, a stoga i potreba održavanja konzistentnosti. Ako razvijatelj preimenuje parametar, konzistentnost će biti narušena ako zaboravi unijeti promjenu i u *Javadoc*. Ručno održavanje konzistentnosti ne samo da je nepouzdano već i zamorno. Stoga je najbolje rješenje uvesti alat koji automatizira proces održavanja konzistencije. Takav alat očito mora razumjeti i *Javadoc* i osnovni kod da bi znao gdje i kada intervenirati. Osim ako ne reimplementira funkcionalnost Javinog parsera, alatu neće biti dostupna jedinstvena struktura u kojoj se nalazi i parsirani kod i parsirani *Javadoc*, a upravo nad takvom strukturom bi bilo najprirodnije formulirati njegov posao. Izvedba alata bit će dakle nepotrebno kompleksna. Svaki daljnji sličan alat bit će prisiljen također reimplementirati parserovu funkcionalnost.

Ortogonalna proširenja ubačena u komentare ne samo da su nevidljiva tamo gdje su potrebna, već su i vidljiva tamo gdje samo smetaju – u prozoru editora izvornog koda. *Javadoc* je možda jedina iznimka gdje je vidljivost proširenja ponekad korisna. Kad bi u dokumentu bila prisutna i raznorazna druga proširenja, razvijatelj bi među njima jedva uspio pronaći razbacane dijelove izvornog koda.

Ortogonalna proširenja XML-ovskog dokumenta s izvornim kodom moguće je elegantno uvesti služeći se XML-ovim prostorima imena. Dokument treba definirati svoj prostor imena i čvorovi iz tog prostora izgrađivat će osnovno stablo dokumenta koje odgovara njegovoj gramatici. Za svako ortogonalno proširenje definira se poseban prostor imena i čvor iz tog prostora može se pojaviti bilo gdje u osnovnom stablu. Svi njegovi podčvorovi također se smatraju dijelom proširenja pa proširenja poprimaju oblik zasebnih grana u osnovnom stablu. Pripadnost čvora proširenju lako se utvrđuje na osnovu pripadnosti drugom prostoru imena. Sve grane koje pripadaju proširenjima lako se može isfiltrirati iz glavnog dokumenta, čime proširenja dobivaju svoje ključno svojstvo ortogonalnosti. Budući da je u ovom slučaju vrlo jednostavno za svaku situaciju definirati politiku filtriranja pojedinih proširenja, nema više prepreke uključivanju proizvoljnog broja proširenja fino raspršenih po dokumentu jer to neće izazvati poteškoće u baratanju njime. Na slici 3.3 shematski je prikazano funkcioniranje ortogonalnih proširenja. Osnovni elementi dokumenta su u podrazumijevanom prostoru imena, a za proširenja deklariran je prefiks `ext`. Lokalno ime elementa proširenja je `a`, isto kao i ime korijenskog elementa glavnog dijela, međutim budući da je u drugom prostoru imena, ne dolazi do njihovog miješanja. Nadalje, unutar proširenja pojavljuje se element `b` iz glavnog prostora imena. Međutim, budući da se pojavljuje kao podelement elementa proširenja, on nedvosimsleno pripada proširenju. Primjeri primjene

ortogonalnih proširenja izneseni su u odjeljku 6.5.




```
<metoda>
  <ime>postavi</ime>
  <parametar>
    <ime>novaVrij</ime>
    <doc:opis>Nova vrijednost atributa </doc:opis>
  </parametar>
<metoda>
```

Slika 3.3: Ortogonalno proširenje u infosetu

3.3 Jezična proširenja

Nakon što novi programski jezik zaživi u masovnoj uporabi, redovito se javljaju ideje i potrebe za dodatnim osnovnim pojmovima. Radi uvođenja jezičnog proširenja kod jezika s tradicionalnim izvornim kodom uvijek je potrebno intervenirati u konkretnu sintaksu izvornog koda. Često bi se proširenje najelegantnije izvelo dodavanjem nove ključne riječi, međutim to rješenje je većinom nedostupno jer bi tako proširena sintaksa izgubila kompatibilnost s postojećim izvornim kodom u kojem se ta ključna riječ slobodno koristi kao identifikator. Stoga se u praksi sintaksa proširuje uvođenjem novih konstrukcija izgrađenih pomoću postojećih ključnih riječi i simbola. To ponekad rezultira nezgrapnim konstrukcijama čija opravdanost proizlazi jedino iz povijesnih okolnosti razvoja jezika, tj. potrebe zadržavanja kompatibilnosti s ranijim verzijama.

Jezična proširenja znatno je lakše uvesti u XML-ovsku sintaksu jer se konkretna sintaksa nikad ne mijenja. Treba samo proširiti apstraktnu sintaksu, tj. sintaksu infoseta, a to se uvijek može izvesti na način koji zadržava kompatibilnost sa starom verzijom sintakse.

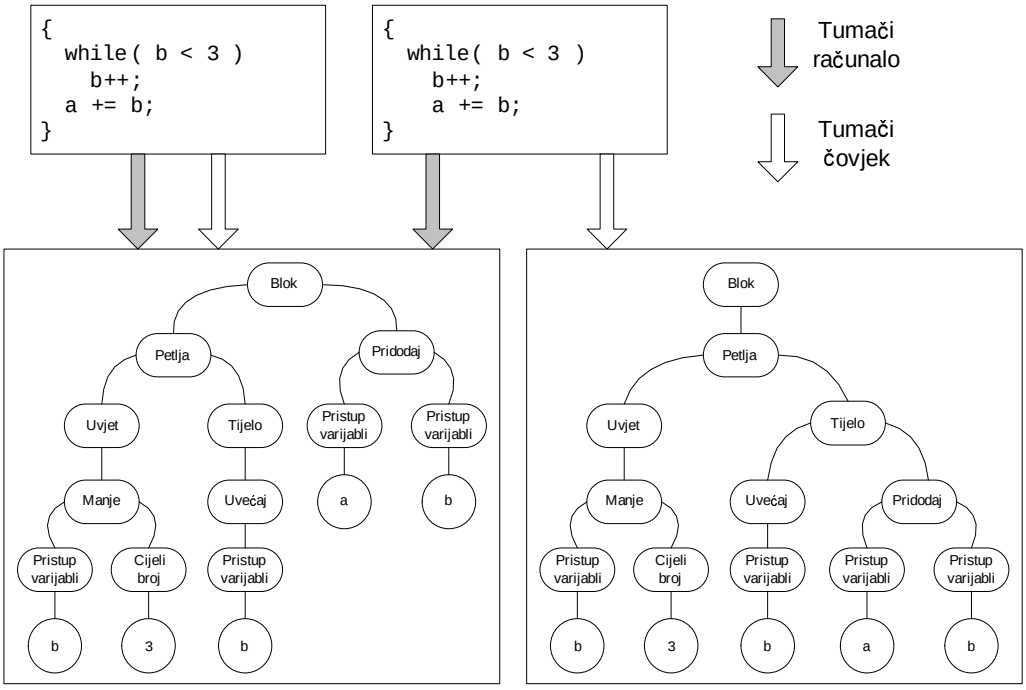
3.4 Razlika između sadržaja i njegovog prikaza

Tekstualni dokument uvijek ima i dodatne aspekte koji ne utječu na značenje zapisanog sadržaja, ali utječu na čitljivost, tj. razumljivost čovjeku. Takvi aspekti spadaju u *prikaz* sadržaja. U slučaju tradicionalnog izvornog koda najčešće se koristi tzv. *slobodno oblikovan* dokument koji dozvoljava proizvoljan razmještaj praznog prostora između pojedinih tokena. Već je objašnjeno kako je u zapisu složenog sadržaja XML-ovskog elementa također dopušteno slobodno ubacivanje praznog prostora. Čovjeku se tekstualni dokument ne prikazuje kao jednodimenzionalan niz znakova, već kao dvodimenzionalna stranica na kojoj značenje prenose i horizontalni i vertikalni međuodnosi znakova. To oblikovanje dokumenta zadovoljava definiciju *ortogonalnog proširenja* dokumenta – nije dio sadržaja, ali ima važnost u širem kontekstu razvojnog procesa. U slučaju XML-a oblikovanje je od sporednog značaja jer je predviđeno da čovjek izravno editira infoset, a ne njegov tekstualni zapis. Međutim, u slučaju tradicionalnog izvornog koda ono je važan element korisničkog sučelja.

Velik dio značenja prenesenog oblikovanjem ponavlja značenje već prisutno u sadržaju – ono je, dakle, redundantno [ST03]. Čovjek se najviše oslanja na oblikovanje u razumijevanju koda, dok se računalo u potpunosti oslanja na redoslijed tokena. Svaka redundancija koja se ne održava nekim pouzdanim automatskim procesom omogućuje pojavu nekonzistentnosti, a u ovom slučaju održavanje redundancije je u nepouzdanim rukama razvijatelja. To rezultira situacijom u kojoj postoji mogućnost nesporazuma između čovjeka i računala, dakle do zakazivanja korisničkog sučelja. Ovo je ilustrirano slikom 3.4. Oblikovanje izvornog koda na lijevoj strani prati organizaciju stabla apstraktne sintakse i stoga navodi na ispravno razumijevanje, a onog na desnoj strani navodi na pogrešno razumijevanje. Međutim, u automatskom postupku kompiliranja već prilikom leksičke analize (slika 1.11, korak 2) razlika između dva zapisa se u potpunosti gubi i jednako nepogrešivo se izgrađuje stablo apstraktne sintakse.

Tradicionalni izvorni kod, dakle, sadrži neke aspekte prikaza. S druge strane, običan tekst je vizualno siromašan, tj. ne iskorištava mnogo mogućnosti čovjekove vizualne percepcije. Prikaz koda u suvremenim editorima sadrži i dodatne elemente, npr. prikaz različitih sintaktičkih elemenata u različitim bojama, podcrtavanje koje ukazuje na moguće probleme itd. Dakle, običan tekst je nezadovoljavajuć kao nositelj prikaza, ali svejedno sadrži neke njegove aspekte, među kojima i one posebno nepoželjne – redundantne.

Infoset je, s druge strane, čisto logička struktura i ne podrazumijeva nijedan



Slika 3.4: Usporedba utjecaja oblikovanja na čovjeka i na računalo

konkretan način prikaza. Oblikovanje prisutno u zapisu infoseta je irelevantno – na njega može gledati kao na implementacijski detalj nevidljiv korisniku.

3.5 Utjecaj primjene XML-a na korisničko sučelje

Već je spomenuto da je teorija formalnih jezika proizašla iz pokušaja formalizacije prirodnih jezika. To je i razlog zašto se upravo nju primjenjuje na izvorni kod. Taj pristup omogućava izvornom kodu da naslijedi strukturalne osobine prirodnog jezika i stoga bude lak za razumijevanje i zapisivanje. Na primjer, jedna izjava u proceduralnom kodu može podsjećati na naredbenu rečenicu prirodnog jezika.

Isti formalizam (beskontekstna gramatika) primjenuje se i za XML, međutim postoji fundamentalna razlika, također već spominjana, a to je da tradicionalni izvorni kod koristi vlastitu sintaksu posebno prilagođenu svojoj apstraktnoj sintaksi. Korištenjem XML-a obvezno se koristi njegova konkretna sintaksa, koja je univerzalna i samim time nije bliska ciljnoj apstraktnoj sintaksi. Upravo zbog toga XML-ovski tekst je nepogodan za čitanje i pisanje, a sadrži i više redundancije pa je od ekvivalentnog teksta tradicionalnog izvornog koda dulji za faktor između tri i pet.

Kombinacija općenitog editora teksta i tradicionalnog zapisa koda sačinjava sasvim upotrebljivo korisničko sučelje. Rad na samom zapisu XML-a u tom istom editoru naporan je i neproduktivan. Međutim, u kontekstu gdje je XML jednako udomaćen kao što je danas običan tekst treba pretpostaviti i postojanje odgovarajućih editora XML-ovog infoseta. U takvoj situaciji XML-ovski editor će imati svoje vlastite prednosti. Infoset može biti prikazan npr. kao stablo imenovanih čvorova. Korisniku će biti nemoguće kreirati dokument koji krši osnovna XML-ova pravila dobre oblikovanosti. Ako pretpostavimo da će osim samog XML-a biti udomaćen i standard za formalnu definiciju sintakse infoseta (npr. WXS), editor će moći nadzirati korisnikov rad na dokumentu i onemogućiti stvaranje nevaljanog dokumenta. Također će moći i ponuditi popis čvorova dozvoljenih na trenutnom mjestu unosa. Editori običnog teksta mogu ostvariti ovakvu funkcionalnost jedino ako su specifično dizajnirani za rad s određenim programskim jezikom, dok bi u slučaju XML-a to mogao najopćenitiji editor koji podržava samo globalne standarde.

Editor infoseta zasnovan na stablastom prikazu s kontrolom valjanosti na osnovu sheme dokumenta mogao bi biti prilično dobro korisničko sučelje za početnika. Imena čvorova mogu učiniti očitim značenje opisano infosetom izvornog koda. Međutim, iskusnom razvijatelju to više ne mora predstavljati prednost i rad na stablu dokumenta može postati zamoran i neefikasan. Razvijatelj će htjeti raspolagati bržim i konciznijim načinom izražavanja – možda nečim sličnim tradicionalnom izvornom kodu. Sintaksa takvog zapisa u ovom slučaju ne mora biti normirana jezičnom specifikacijom i to otvara veću slobodu u izboru najboljeg načina izražavanja u pojedinim situacijama. Svaki takav zapis bio bi samo još jedan način prikaza sadržaja opisanog infosetom.

3.6 Automatizirano baratanje kodom

Izvorni programski kod nije dovoljno promatrati samo kao posrednika između čovjeka i kompilatora. Nad kodom se, osim kompiliranja, obavlja i mnogo drugih operacija za koje je jednako poželjno da budu u što većoj mjeri automatizirane. U nastavku su navedeni neki primjeri.

- U većim projektima za neke namjene kod nije uputno promatrati u njegovoj potpunosti, već je potrebno ekstrahirati samo neke njegove aspekte. Na primjer, korisno je imati statističke podatke o broju veza prema pojedinim dijelovima koda, o brojnosti i veličinama kritičnih odsječaka u konkurentnom

kodu i sl. Prikupljanje mnogih vrsta takvih podataka moguće je automatizirati.

- Uporaba gotovo svih ortogonalnih proširenja zahtijeva automatsku obradu. Primjerice, spomenuti kodni predlošci (3.2) zahtijevaju automatizirano generiranje koda koji opisuju.

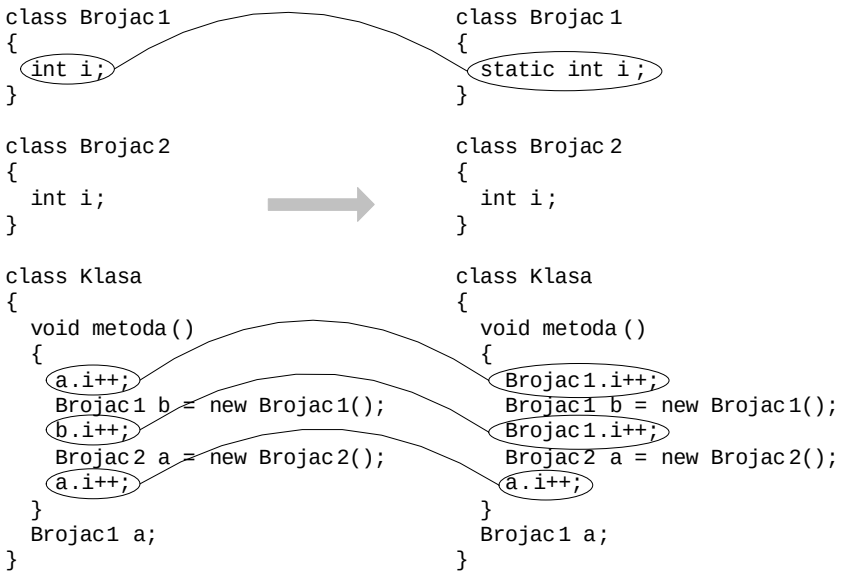
- Tijekom razvoja koda često se primijenjuju tipizirani zahvati (transformacije) nad kodom radi bolje organizacije (povećanje čitljivosti, proširivosti). Takvi zahvati poznati su pod imenom *refaktorizacija* [Fow99]. Mnoge se može formalizirati i zatim automatizirati.

Ključan preduvjet za automatizaciju obrade je formalizacija njenog opisa. U praksi to znači da ju treba biti moguće opisati nekim programskim jezikom. Od posebnog interesa su oni zadaci obrade koje je praktično opisati služeći se osnovnim pojmovima programskog jezika, a takvi su svi gore navedeni primjeri. Njih je moguće formalizirati kao transformaciju stabla apstraktne sintakse. Općenit problem na čije rješenje se oslanja svaki takav zadatak je automatizacija postupka izgradnje stabla apstraktne sintakse na osnovu izvornog koda. U praksi će biti presudno da je rješenje tog problema široko dostupno, tako da ga razvijatelj programa za automatizaciju obrade ne mora svaki put iznova rješavati. Ranije u ovom poglavlju pokazano je zašto je široko dostupno rješenje znatno lakše ostvariti uz primjenu XML-a.

U svim navedenim primjerima treba baratati isključivo sadržajem izvornog koda, a aspekti prikaza u tom kontekstu su irelevantni. U razvojnom okružju organiziranom oko tradicionalnog izvornog koda pojavljuju se višestruki problemi. Izvorni kod ne pretvara se u stablo apstraktne sintakse, već se obrada tipično formalizira kao transformacija programskog teksta. Takva formalizacija vrlo je kompleksna, teška za osmišljanje i, nakon što je izvedena, za razumijevanje pa je stoga teško locirati pogreške ili dodavati poboljšanja. Za kod na izlazu iz procesa obrade problem je što je nemoguće izbjeći aspekte prikaza jer su oni nerazdvojni od samog načina zapisa sadržaja. Alat za obradu je prisiljen brinuti se o urednom oblikovanju koda, što je samo po sebi zahtjevan postupak (vidjeti [jcc99] kao primjer standarda oblikovanja koda). Dakle, ostvarivanje ukupne funkcionalnosti u značajnom je dijelu opterećeno sporednim aspektom prikaza irelevantnim za samu funkcionalnost.

Za kod na ulazu u proces obrade problem je s dostupnosti njegovog značenja. Na primjer, uzmimo da neki zahvat refaktorizacije koda u Javi uključuje pretvara-

nje instancinog atributa u klasin atribut. Tijekom zahvata bit će potrebno locirati sve pristupe tom atributu. Cilj nije niti da se nađu sve pojave imena atributa u tekstu, niti da se razlikuju različite sintaktičke varijante pristupa atributu. Treba jednostavno naći sve primjerke značenja "pristup instancinom atributu X" i izmijeniti ih u "pristup klasinom atributu X". Značenje postaje dostupno tek nakon čitavog procesa ilustriranog slikom 1.11. Jedan primjer ovakve refaktorizacije ilustriran je slikom 3.5. Posebice treba uočiti kako se identičan tekst, `a.i++`, u jednom slučaju treba transformirati, a samo nekoliko redaka niže već ima drugačije značenje i treba ostati neizmijenjen. Značenje ponekih sintaktičkih konstrukcija ne možemo ustanoviti čak niti parsiranjem čitavog dokumenta nad kojim izvodimo zahvat, već je potrebna inspekcija svih programskih modula koji će biti vidljivi tijekom izvršavanja programa. Drugim riječima, značenje konstrukcije je ovisno o (ponekad vrlo širokom) kontekstu u kojem se pojavljuje.



Slika 3.5: Refaktorizacija

U postojećim razvojnim okružjima donekle su se udomaćili alati koji omogućuju rad sa stablom "apstraktne sintakse". Međutim, taj pristup pati od već spomenutog problema da apstraktna sintaksa većine programskih jezika nije formalizirana pa ju je potrebno formalizirati eksplicitno za taj alat, obično u nekom nestandardnom obliku. Stoga svaki takav alat barata svojom sintaksom, koja osim toga često pati od raznih nedostataka lošeg oblikovanja.

Već je objašnjeno kako se zapisom izvornog koda u XML-u može potpuno eliminirati opis prikaza. Alat koji generira izvorni kod treba se dakle brinuti samo

oko opisivanja sadržaja. Koristeći se samo dvjema već opisanim jezicima za baratanje infosetom (*XQuery* i XSLT, vidjeti 2.5.2, 2.5.3) razvijatelju je omogućeno da bez mnogo truda opiše funkcionalnost koja bi predstavljala značajan izazov da se opiše u jeziku za baratanje tekstom. Time se također pomiče i gornja granica kompleksnosti problema u čije rješavanje se isplati upustiti.

3.7 Zaključci poglavlja

U ovom poglavlju identificirana su sljedeća svojstva XML-ovskog izvornog koda koja ga razlikuju od tradicionalnog:

- Može se standardnim i široko dostupnim postupkom pretvoriti u infoset koji modelira stablo apstraktne sintakse.
- Koristi univerzalnu konkretnu sintaksu, što omogućuje ubacivanje novih čvorova u infoset bez potrebe za promjenom sintakse.
- Tekst služi samo za opis infoseta u obliku prikladnom za spremanje i prijenos, a ne i prikaz čovjeku. Sam infoset ne treba nametati nijedan aspekt prikaza.

Time XML-ovski izvorni kod donosi ove ključne prednosti programskim jezicima:

- Omogućuje jednostavnije uvođenje ortogonalnih proširenja.
- Omogućuje praktičniju formalizaciju zadataka obrade izvornog koda.
- Olakšava evoluciju programskog jezika.
- Ostavlja potpunu slobodu načina prikaza izvornog koda.

Poglavlje 4

Zahtjevi na sintaksu infoseta izvornog koda

Nakon što su identificirane ključne prednosti primjene XML-a na izvorni kod, treba pristupiti definiranju zahtjeva na sintaksu infoseta izvornog koda. Zahtjevi moraju biti takvi da njihovo ispunjavanje garantira mogućnost maksimalnog iskorištavanja prednosti XML-a. Stoga oni prirodno proizlaze iz činjenica izloženih u prethodnom poglavlju. Ukratko, zahtjevi su sljedeći:

1. Semantičko podudaranje s jezičnom specifikacijom
2. Prilagođenost upitima
3. Stroga definicija u WXS-u
4. Svi meta-podaci u ortogonalnim proširenjima

4.1 Semantičko podudaranje s jezičnom specifikacijom

Ovaj zahtjev slijedi izravno iz namjene XML-ovske sintakse da preuzme ulogu izvornog koda programskog jezika. Ako se uvodi nova sintaksa, ona mora omogućiti izražavanje točno istih značenja koja su izraziva postojećom sintaksom tj. propisana jezičnom specifikacijom. Specifikaciju treba obraditi i deducirati pravila apstraktne sintakse. Nakon toga kreće proces oblikovanja sintakse infoseta s ciljem da budu izraziva sva predviđena značenja. Za ovu disertaciju kao ogledni primjer

odabran je programski jezik Java pa je popis značenja koja treba izraziti opisan u dokumentu *Java Language Specification, 2nd Edition* [GJSB00].

Dva su moguća načina odstupanja od doslovne semantičke podudarnosti: izostavljanje nekih propisanih značenja i uvođenje novih značenja. Prvi način sigurno ne dolazi u obzir jer bi tada bilo nemoguće zapisati proizvoljan izvorni kod. Drugi način je zanimljiviji jer ne bi rezultirao nepridržavanjem specifikacije – još uvijek bi se XML-ovskim izvornim kodom moglo izraziti sve ono što specifikacija propisuje, ali bi se mogla izraziti i dodatna značenja. Ona ne bi bila izraziva u tradicionalnom izvornom kodu, ali u slučaju postojanja kompilatora koji kao ulaz prima XML-ovski izvorni kod to ne bi predstavljalo problem. Takvi dodaci bili bi ustvari jezična proširenja kakva se pojavljuju i tijekom normalne evolucije jezika, samo što ne bi bila standardizirana. Ovaj pristup ima svoje prednosti i budući da se već nalazimo u procesu razvoja potpuno nove sintakse, može iskrsnuti ideja za korisna jezična proširenja. Međutim, u kontekstu prvotnog oblikovanja sintakse vrlo je uputno držati se isključivo značenja propisanih specifikacijom. Svako jezično proširenje stupa u suptilne interakcije sa svim postojećim svojstvima jezika i teško je unaprijed predvidjeti sve finese tih interakcija. Definiranje proširenja ozbiljan je problem sam za sebe i nije razumno upuštati se u njega dok još sintaksa nije ni razvijena.

Iako se svojstva koja bi predstavljala jezično proširenje mogu izbjeći, praktički je nemoguće izbjeći svaku vrstu razlike između stare i nove sintakse. Značenje se izražava putem bitno različitih konstrukcija i to će izazvati razlike u detaljima njihovog ponašanja i interakcije. Postoje slučajevi gdje su takve razlike ne samo poželjne, nego upravo ključne za ostvarivanje prednosti novog zapisa. Jedan od primjera za ovo je odnos značenja konstrukcije i konteksta u kojem se pojavljuje (vidjeti 5.1).

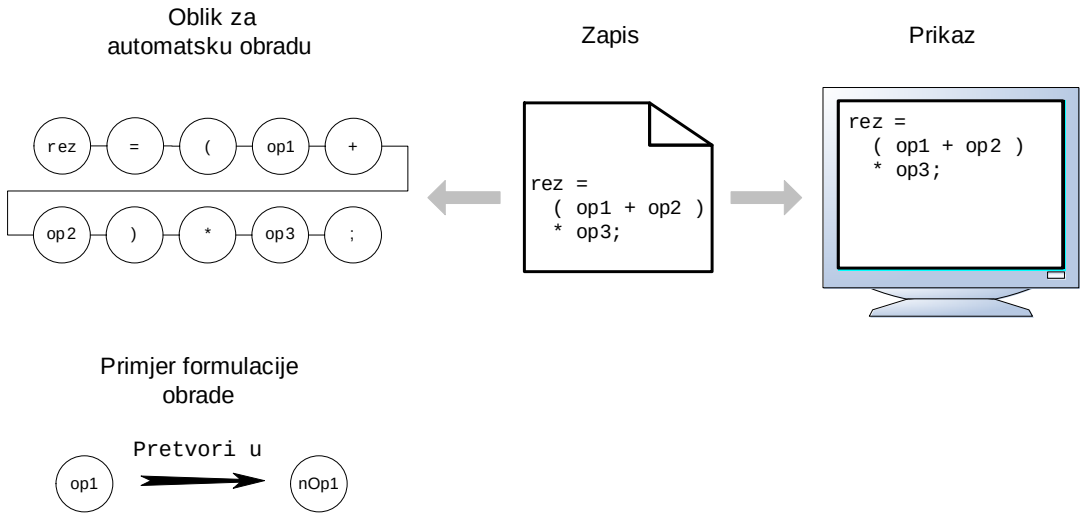
4.2 Prilagođenost upitima

Upit je općenito zahtjev za dohvaćanje zadane informacije iz nekog izvora podataka, u ovom slučaju iz infoseta izvornog koda. Jedna od ključnih prednosti XML-a istaknutih u odjeljku 3.7 je praktičnija formalizacija zadataka obrade izvornog koda, a gotovo je nemoguće naći zadatak obrade koji ne bi uključivao dohvaćanje neke informacije. Ustvari, imajući na umu kompleksnost strukture koda, formuliranje upita često će predstavljati najkompleksniji dio cjelokupne funkcionalnosti. U slučaju da se sintaksa infoseta izvornog koda pažljivo ne oblikuje s ovim zahtje-

vom u vidu, neki podatak koji je u načelu trivijalno dohvatiti mogao bi iziskivati vrlo složen upit. Upiti trebaju biti što koncizniji i jednostavniji za pisanje i čitanje, tj. njihova formulacija treba prirodno proizlaziti iz značenja zahtjeva koji predstavljaju.

Ovdje je važno naglasiti svojevrstan preokret u zahtjevima na XML-ovsku u odnosu na tradicionalnu sintaksu u pogledu sučelja između računala i čovjeka:

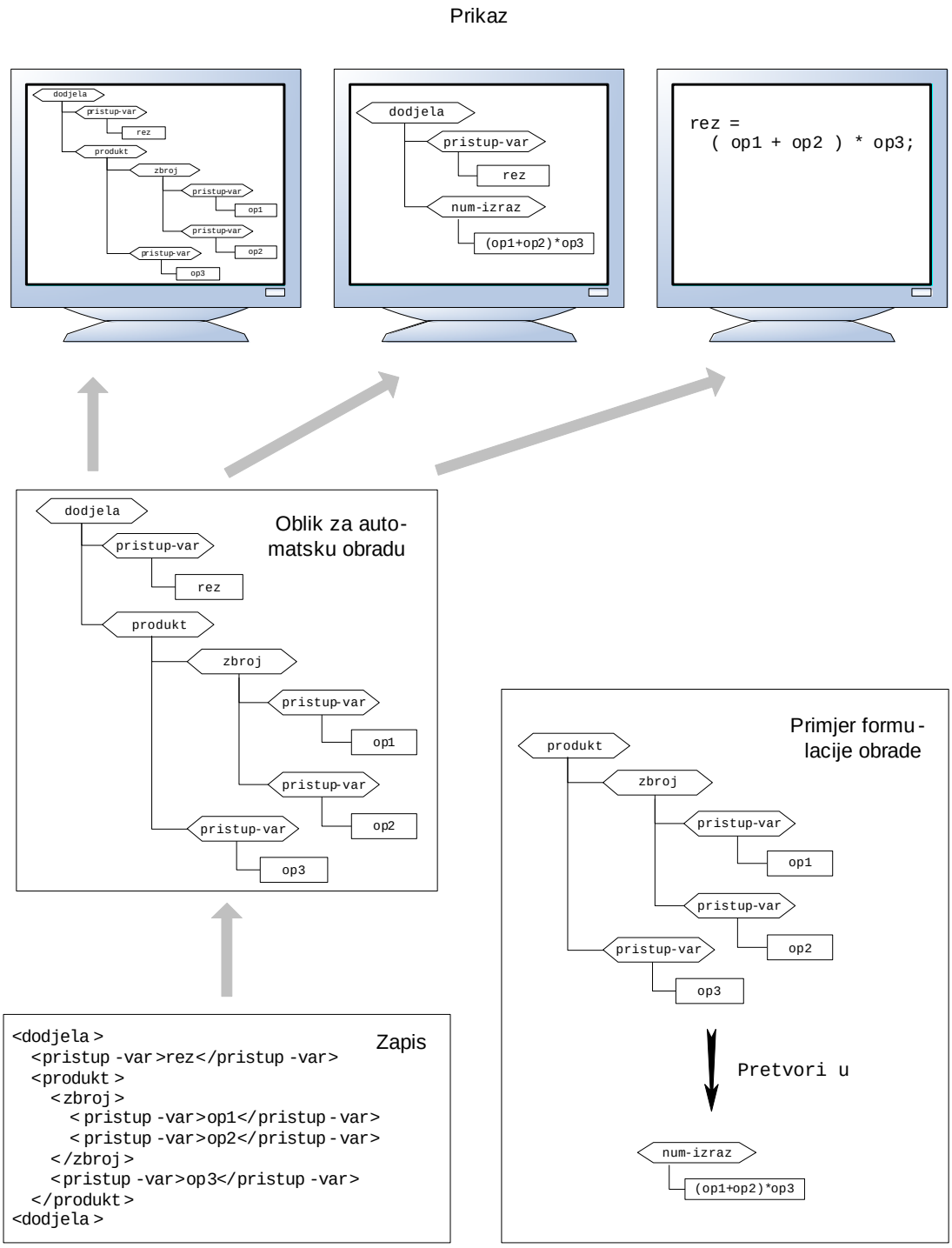
- Kod tradicionalnog izvornog koda sam njegov tekst predstavlja ključnu kariku u sučelju prema čovjeku i stoga je u prvom planu njegova čitljivost i lakoća zapisivanja (slika 4.1).
- Kod XML-ovskog izvornog koda tekst je u predviđenom razvojnom okružju nevidljiv čovjeku, a ne treba se držati ni doslovnog prikaza infoseta. S druge strane, važan element sučelja prema čovjeku postaje formalni opis strukturalnih uzoraka u infosetu radi automatizacije obrade, koja je uostalom potrebna i za generiranje prikaza (slika 4.2).



Slika 4.1: Tradicionalno razvojno okružje

4.3 Stroga definicija u WXS-u

Teoretski, čitava sintaksa infoseta izvornog koda mogla bi biti opisana neformalno, kao što je to slučaj s apstraktnom sintaksom. Dio razvoja svakog alata koji barata izvornim kodom bila bi i provjera njegove valjanosti, barem u opsegu koji je nužan



Slika 4.2: Razvojno okružje zasnovano na XML-u

za ispravno funkcioniranje alata. Da bi se olakšao razvoj takvih alata, sintaksu infoseta treba formalizirati definicijom sheme u strojno obradivom obliku. U ovom trenutku postoji niz jezika za definiciju sheme koji se razlikuju po mogućnostima izražavanja i po stupnju prihvaćenosti. Obje ove karakteristike vrlo su značajne pri izboru jezika, a WXS je pri vrhu po oba kriterija. Spomenuto je da se već nameće kao globalni standard. Zasad DTD još uvijek ima širu prihvaćenost, ali će ga WXS uskoro sustići i postati podrazumijevani dio XML-ovske infrastrukture na svakoj računalnoj platformi. Po širini spektra mogućnosti WXS je najjači, iako pojedini drugi jezici imaju neke mogućnosti koje WXS nema.

Postoji još jedan snažan argument u korist WXS-a, a to je njegovo spomenuto svojstvo da omogućava definiciju vrste podataka. Uz prisutnost definicije u WXS-u iz dokumenta se može izvući bogatije značenje na automatiziran način. Na primjer, nova generacija osnovnih XML-ovskih tehnologija (XPath 2.0 i XQuery 1.0, [xqr]) izravno podržavaju informaciju o vrstama iz WXS-ovske definicije sheme i omogućuju fomulaciju moćnijih upita koji se njome služe. Iz toga proizlazi da će postojati XML-ovski alati čija će puna funkcionalnost moći biti iskorištena samo uz prisutnost kvalitetne definicije u WXS-u. Iz svih ovih razloga postajat će sve kritičnije za svaku XML-ovsku sintaksu, a ne samo za onu o kojoj se ovdje raspravlja, da ima definiciju u WXS-u.

Jednom kad se definicija u WXS-u prihvati kao definitivan zahtjev, od značajnog interesa postaje detaljno istraživanje njegovih svojstava i mogućnosti. Definicija sheme nikad neće pokrivati sva pravila koja nameće jezična specifikacija i upravo zato je važno uložiti napor da ih se pokrije što je više moguće. Svako pravilo koje je izostavljeno iz strojno obradive definicije morat će svaki alat nametati naknadno. Sposobnost nametanja nekih od pravila ovisit će o odlukama donesenim u oblikovanju same sintakse infoseta. Uputno je dozvoliti ograničenjima u izražajnosti WXS-a da utječu na odabir rješenja u oblikovanju sintakse, ali samo do određene granice, preko koje bi došlo do narušavanja drugih zahtjeva na dizajn ili do prekomjernog porasta kompleksnosti WXS-ovske definicije.

4.4 Svi meta-podaci u ortogonalnim proširenjima

Superiornost XML-a u lakoći uvođenja ortogonalnih proširenja dokumenta zahtijeva drugačiji pristup kad se razmišlja o načinu kodiranja meta-podataka. Svaka definicija sintakse tradicionalnog izvornog koda bavi se ovim problemom barem na najpovršniji način – propisivanjem leksičke strukture komentara. Novije speci-

fikacije u osnovnu sintaksu uvrštavaju i konkretniju podršku za meta-podatke. Na primjer, sintaksa jezika C# [HWG03] propisuje način zapisivanja dokumentacije programskog sučelja, a sintaksa verzije 5 jezika Java uvodi općeniti *framework* za uvođenje ortogonalnih proširenja bilo koje namjene.

Standard XML već definira posebnu vrstu čvora za komentare koji se mogu pojaviti bilo gdje u dokumentu pa ih shema dokumenta ne treba eksplicitno uvoditi. Nadalje, standard *XML Namespaces* omogućuje ubacivanje stranih elemenata i atributa (koje specifikacija ne propisuje) na takav način da ih je lako ignorirati prilikom provjere valjanosti dokumenta. S tim svojstvima XML-a na umu u shemu dokumenta ne treba uvoditi eksplicitnu podršku za bilo koji sadržaj izvan samog izvornog koda. Čak i ako se neka mogućnost – npr. dodatne upute za transformaciju u tradicionalni zapis – planira otpočetka, to nije razlog da ju shema propisuje. Ovaj pristup ima nekoliko prednosti. Alat koji barata samo izvornim kodom neće biti opterećen proširenjima; slično, svaki pojedini alat tipično će baratati samo jednim proširenjem i neće se morati baviti ostalima. Ako se specifikacija nekog proširenja promijeni, bilo bi vrlo nepraktično kad bi to povlačilo izmjene na svim alatima samo da uspješno ignoriraju novu verziju proširenja. Ako se svakom proširenju dodijeli poseban prostor imena, alat se može osigurati od utjecaja nepodržanih proširenja. Karakter nekih proširenja je takav da se može kreirati jedna specifikacija primjenjiva na više programskih jezika. Alat koji barata takvim proširenjem automatski bi radio sa svim tim jezicima.

Poglavlje 5

Načela u oblikovanju sintakse

Tijekom rada na oblikovanju sintakse infoseta izvornog koda programskog jezika Java, a i analizom svojstava postojećih sintaksi, uočeno je da su dva najkritičnija zahtjeva (najsloženija za ispunjavanje) prilagođenost sintakse upitima i strogost sheme. Od njih, prilagođenost upitima je konceptualno zahtjevnija i suptilnija pa je većina načela izloženih u ovom poglavlju posvećena tom zahtjevu. Načela imaju dvojaku namjenu: kao upute o tome kako ispuniti zahtjev i kao njegovo pobliže pojašnjenje ukazivanjem na njegovu važnost i na posljedice njegovog neispunjavanja. Pritom se mnogi argumenti pozivaju na najvažniji kriterij kvalitete sintakse – lakoću razvijanja alata koji barataju izvornim kodom. Neki od takvih alata razvijeni su tijekom istraživanja radi provjere sintakse, međutim argumenti se odnose općenito na svaki potencijalni alat te vrste. Kratak pregled načela:

1. Neovisnost značenja konstrukcije o kontekstu
2. Potpuna strukturiranost
3. Razdvajanje identifikatora od znakovnih nizova s posebnim značenjem
4. Višestruko iskorištavanje konstrukcija
5. Minimiziranje broja razina ugniježđenja
6. Maksimiziranje strogosti WXS-ovske definicije sheme
7. Odabir između atributa i jednostavnog elementa
8. Izbor imena čvorova

5.1 Neovisnost značenja konstrukcije o kontekstu

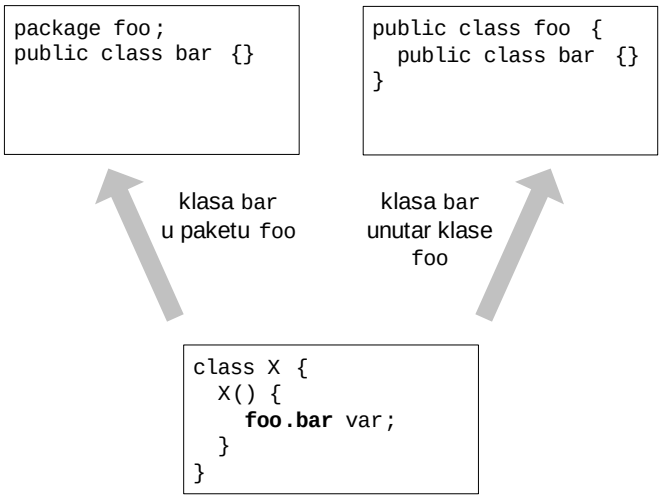
U tradicionalnom izvornom kodu često je velik dio značenja neke sintaktičke konstrukcije pohranjen izvan nje same: u okolini u kojoj se pojavljuje, tj. u njenom *kontekstu*. U kontekst prvenstveno spada ostatak dokumenta u kojem je konstrukcija, ali redovito i mnogi drugi dokumenti izvornog koda na koje se on oslanja.

U Javi je najbolji primjer "operator *točka*". Točka ustvari i nije operator, već se koristi kao element konkretne sintakse čitavog niza primarnih izraza te kao znak za razdvajanje segmenata punog imena Javine referentne vrste. Može imati najmanje šest vrlo različitih značenja ovisno o kontekstu pojavljivanja (*qualified* `new` i *class literal* su neki od manje poznatih primjera). Neka od mogućih značenja lako je identificirati prema prisustvu određene ključne riječi s lijeve ili desne strane operatora. Na primjer, ako je desna riječ `class`, radi se o spomenutom *class literal*-u. Međutim, ako se ustanovi da su i lijeva i desna riječ identifikatori (a ne ključne riječi), ostaje čitav niz mogućih značenja, ovisno o tome što oni identificiraju. Javina konkretna sintaksa propisuje identičnu leksičku strukturu za identifikatore vrlo raznorodnih entiteta kao što su paketi, vrste, atributi, metode, parametri i lokalne varijable.

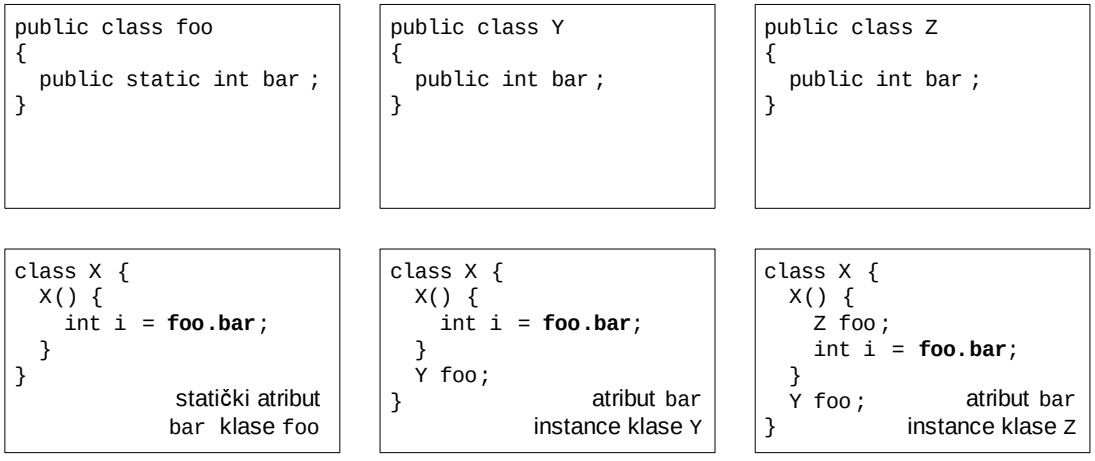
Uzmimo na primjer konkretan niz tokena `foo . bar`. Riječ `bar` može biti ime vrste te instancinog ili vrstinog atributa. Riječ `foo` može također biti bilo koje od toga, a dodatno može biti i ime lokalne varijable ili paketa. Gotovo svaki par značenja je dozvoljen, a ukupno značenje konstrukcije je za svaki izbor drugačije. Na primjer, konstrukcija bi mogla biti ime vrste `bar` u paketu `foo`, ime unutarnje vrste `bar` sadržane u vrsti `foo` (ilustrirano slikom 5.1), ime atributa `bar` instance na koju pokazuje atribut `foo` trenutne instance, ime atributa `bar` instance na koju lokalna varijabla `foo` (ilustrirano slikom 5.2) itd.

Za ustanovljenje značenja pojedinog identifikatora u općenitom slučaju potrebno je poznavanje ne samo sadržaja trenutnog dokumenta, već i imena svih Javinih paketa, vrsta i njihovih članova koji će biti dostupni tijekom izvršavanja koda. Skup vidljivih entiteta definira se izvanjezično, na način koji nije propisan specifikacijom i ovisi o konkretnom razvojnom/izvršnom okružju. To znači da samo promjenom okružja, bez ikakve promjene u izvornom kodu, možemo promijeniti značenje navedene sintaktičke konstrukcije do te mjere da će biti prevedena u sasvim drugačiji izvršni kod.

Jedan od argumenata za uvođenje ovisnosti o kontekstu u tradicionalnom zapisu je to što konstrukcije tako mogu biti konciznije i time čitljivije – ne ponavljaju se komponente značenja koje se mogu deducirati iz konteksta. Također je omo-



Slika 5.1: Ovisnost značenja o kontekstu, primjer 1



Slika 5.2: Ovisnost značenja o kontekstu, primjer 2

gućena višestruka iskoristivost iste konkretne sintakse za zapis različitih značenja, čime se štedi na broju zauzetih ključnih riječi i simbola. Ni prvi ni drugi argument više ne vrijede u slučaju XML-a jer se on u svakom slučaju treba oslanjati na bogatiji prikazni sloj koji riješava pitanja čitljivosti, a brojnost "ključnih riječi", u ovom slučaju imena elemenata i atributa, nema nikakvog utjecaja na slobodu odabira identifikatora. U spomenutom primjeru skup svih mogućih značenja "operatora *točka*" nema ništa više smisla podvesti pod jednu te istu konstrukciju u infosetu nego to isto učiniti s bilo kojim drugim skupom nasumično odabranih značenja.

Osim ovih argumenata koji ukazuju na to da je ovisnost značenja o kontekstu nepotrebna u sintaksi infoseta, postoji i jak argument koji ukazuje na to da je ona i štetna. Uzmimo na primjer da se u infosetu gore spominjana konstrukcija modelira doslovno, na primjer na sljedeći način:

```
<dot-operation>
  <identifier>foo</identifier>
  <identifier>bar</identifier>
</dot-operation>
```

Razmotrimo sada pokušaj sastavljanja upita koji treba locirati sve pristupe instancinom atributu (bez obzira na vrste instance i atributa). To bi moglo predstavljati jedan od podataka potrebnih za neku analizu, ili bi moglo biti dio automatiziranog nametanja konvencije da se atributima instance nikad ne pristupa izravno, već preko pristupnih metoda. Upit će morati prvo locirati sve pojave konstrukcije `dot-operation`. Da se za konkretnu konstrukciju navedenu gore ustanovi njeno značenje, bit će potrebno puno dodatnih informacija. Trebat će pobrojati sve lokalne varijable vidljive u bloku gdje se konstrukcija nalazi i vidjeti zove li se neka od njih `foo`. Vidljive lokalne varijable mogle su biti deklarirane u bilo kojem od okružujućih blokova koda. Proces će trebati ponoviti i za parametre trenutne metode. Ako se konstrukcija nalazi unutar bloka `catch`, trebat će provjeriti i njegovu deklaraciju imena iznimke. U slučaju da nema lokalne varijable `foo`, treba pregledati imena atributa trenutne instance. Napokon, ako ni tamo nije nađen identifikator `foo`, treba provjeriti postoji li klasa `foo` koja ima statički atribut `bar`. Rezultirajući upit bit će vrlo velik i kompleksan i samim time težak i za formulaciju i za razumijevanje.

Uzmimo naprotiv slučaj da je značenje "pristup atributu instance" izraženo posebnom konstrukcijom, npr.

```
<field-access name="bar">
```

```
  <containing-instance>
    <variable-access name="foo"/>
  </containing-instance>
</field-access>
```

Sve što je u ovom slučaju potrebno je jednostavan *XPath*-ov izraz

```
//field-access[containing-instance]
```

Kad zapisuje neku jezičnu konstrukciju, njen autor redovito očekuje da će ona imati točno jedno, željeno značenje. Nikad se neće željeti okoristiti mogućnošću da konstrukcija može promijeniti značenje ovisno o kontekstu – dapače, poduzet će korake da se osigura od takve mogućnosti. Na primjer, Javina jezična specifikacija preporučuje, iako ne nameće, konvencije za takvu leksičku strukturu identifikatora koja omogućuje leksičko razlikovanje imena vrste od imena ostalih entiteta. U mnogim projektima također se koristi konvencija *kvalificiranja* svakog pristupa atributu trenutne instance ključnom riječju `this`, čime se onemogućuje promjena značenja u "pristup lokalnoj varijabli".

Uz promišljeno oblikovanje sintakse cjelokupna informacija potrebna za utvrđivanje značenja neke konstrukcije može biti sadržana u putu kroz XML-ovsko stablo do nje, uključujući usput i neke čvorove-braću – posebice atribute.

Situaciju kompliciraju identifikatori jer se njima može pozivati na entitete opisane udaljenim konstrukcijama, pa puno značenje promatrane ovisi i o tim udaljenim konstrukcijama. Međutim, i u ovom slučaju može se napraviti iskorak u odnosu na tradicionalni zapis jer se barem može znati na kakav entitet (lokalna varijabla, klasa, atribut) se identifikatorom poziva.

5.2 Potpuna strukturiranost

Već je objašnjeno da jedna od najvažnijih prednosti XML-a leži u eliminaciji parsiranja kao problema kojim se treba baviti prilikom baratanja izvornim kodom. Da bi se ta prednost u potpunosti iskoristila, nikakva struktura ne smije ostati ugrađena u znakovne nizove koji se pojavljuju kao vrijednosti atributa ili jednostavnih elemenata. Kao tipičan primjer može poslužiti puno kvalificirano ime Javine vrste. U najopćenitijem slučaju ono se sastoji iz imena paketa, imena sadržavajućih vrsta i napokon nekvalificiranog imena same vrste. Svi dijelovi imena razdvojeni su točkom. U XML-u se ne smije propisati zapis čitavog imena kao jednog znakovnog niza jer ne samo da bi bio potreban proces parsiranja da se ono raščlani, već bi

taj proces zahtijevao (kao što je razjašnjeno u prethodnom odjeljku) i poznavanje širokog konteksta pojave promatranog imena.

Neke jezične konstrukcije mogu zauzimati "sivu zonu" između strukturiranosti i atomarnosti pa može doći do dileme oko izbora njihovog zapisa. U slučaju Jave postoji dilema oko zapisa imena paketa. Striktno s gledišta jezične semantike to ime je nedjeljiva cjelina i bilo koja dva različita imena jednostavno predstavljaju dva različita paketa. Na primjer, paketi `a.b` i `a.b.c` nisu u nikakvom bližem odnosu nego paketi `a.b` i `x.y.z`. Međutim, već specifikacija Javine konkretne sintakse mora se baviti strukturom imena budući da konstrukcija s dva identifikatora odvojena točkom može imati i razna druga značenja. Također postoji i konvencija o hijerarhijskom imenovanju paketa radi osiguravanja globalne jedinstvenosti njihovih imena i radi povezivanja po namjeni povezanih paketa. U ovom slučaju, budući da se XML-ovska sintaksa treba strogo ravnati prema jezičnoj a ne prema nekoj dodatnoj, izvanjezičnoj razini značenja, treba odabrati zapis imena paketa kao jednog niza.

5.3 Razdvajanje identifikatora od znakovnih nizova s posebnim značenjem

U sintaksi se osim imena elemenata i atributa pojavljuje još jedna vrsta nizova znakova s posebnim značenjem: vrijednosti iz neke enumeracije. Na primjer, u Javi će se tipično razina dostupnosti nekog entiteta propisati atributom čija vrijednost mora biti jedna od četiri dozvoljene. Svaka od tih vrijednosti ima posebno značenje propisano jezičnom specifikacijom.

S druge strane, na vrijednosti atributa koje predstavljaju identifikatore ne postoje takva ograničenja i osim toga sve moguće vrijednosti su ekvivalentne po značenju. Na mjestima u sintaksi predviđenim za zapis identifikatora ne smiju biti propisani posebni nizovi koji nemaju smisao identifikatora, nego neko posebno značenje. Kao primjer u Javi možemo razmotriti ključnu riječ `this` koja se u tradicionalnoj konkretnoj sintaksi pojavljuje na istim mjestima gdje se može pojaviti i identifikator. Značenje koje se izražava ključnom riječju `this` moglo bi biti zapisano u XML-u na analogan način – sintaksa bi mogla propisivati da na nekom mjestu predviđenom za identifikator svi nizovi imaju ekvivalentno značenje, osim posebnog niza `this`, na primjer:

```
<!-- običan identifikator --> <variable-access name="var1"/>
```

```
<!-- posebno značenje --> <variable-access name="this"/>
```

Takav pristup izazvao bi niz problema. Zapis bi bio kontradiktoran jer ime atributa koji sadrži tu vrijednost upućuje na to da se radi o identifikatoru a ne o ključnoj riječi. Sintaksa bi morala nametati nepotrebna ograničenja na izbor identifikatora, a već je rečeno (3.3) da je jedna od ključnih prednosti koje donosi XML gubitak potrebe za takvim ograničenjima. Najteži problem je što bi pravila za utvrđivanje značenja konstrukcije postala nepotrebno složena jer bi ono za većinu vrijednosti atributa bilo srodno, osim za jedan poseban slučaj. Stoga se čitava konstrukcija u kojoj se pojavljuje ključna riječ ne bi mogla tretirati općenito, oslanjajući se na to da ima uvijek isto (točnije, vrlo srodno) značenje. Time bi i upiti koji uključuju tu konstrukciju postali ne samo manje razumljivi nego i nepotrebno kompleksni. Na primjer, XPath-ov izraz koji prikuplja sve pristupe lokalnim varijablama izgledao bi ovako:

```
//variable-access[ not( @name = "this" ) ]
```

umjesto jednostavno

```
//variable-access
```

U slučaju da ima više ključnih riječi, izraz bi morao eksplicitno isključivati jednu po jednu.

U kontekstu faze prijelaza s tradicionalne na XML-ovsku sintaksu ipak postoji opravdanje za uvođenje ograničenja na izbor identifikatora. Naime, bez tog ograničenja neće biti moguća transformacija u tradicionalni zapis bez intervencije nad identifikatorima koji se poklapaju s ključnim riječima.

5.4 Višestruko iskorištavanje konstrukcija

Mnoge sintaktičke konstrukcije pojavljuju se uvijek u istom lokalnom kontekstu, na primjer izjave za kontrolu tijeka uvijek se pojavljuju kao elementi-djeca elementa koji predstavlja blok izvršnog koda, deklaracije metoda kao djeca elementa za deklaraciju klase itd. Međutim, ima takvih konstrukcija koje se pojavljuju na mnogo mjesta u sintaksi. Najbolji primjer (u Javi) je konstrukcija pozivanja na vrstu podatka jer je istovremeno vrlo proširena po sintaksi i prilično kompleksna. Razmotrimo na primjer deklaraciju lokalne varijable i najavu iznimki koje baca metoda. Ovo bi moglo izgledati kao razuman pristup:

```
<variable name="var1">
  <type package="a.b" name="Exception1"/>
</variable>

<method>
  <throws package="a.b" name="Exception1"/>
  <throws package="a.b" name="Exception2"/>
</method>
```

Međutim, razmotrimo (vrlo čest i samim time značajan) upit koji pronalazi sve pozive na neku vrstu. Bilo bi vrlo poželjno da ga je moguće ostvariti jednostavnim *XPath*-ovim izrazom kao što je `//type[@package="a.b" @name="Exception1"]`, međutim sa sintaksom oblikovanom kao u primjeru to će biti nemoguće. Ekstrapolirajući isti pristup kroz cijelu sintaksu, za očekivati je da će postojati još čitav niz daljnjih različitih konstrukcija identičnog značenja – pozivanje na vrstu. Upit će morati eksplicitno pobrojati svaku konstrukciju, na primjer:

```
//variable[type/@name="T"] | //throws[exception/@name="T"]
| //cast[cast-to/@name="T"] | ...
```

Da bi se to izbjeglo, bitno je da sve konstrukcije istog značenja imaju i istu strukturu. U navedenom primjeru najava iznimki bi trebala izgledati ovako:

```
<method>
  <throws>
    <type package="a.b" name="Exception1"/>
    <type package="a.b" name="Exception2"/>
  </throws>
</method>
```

5.5 Minimiziranje broja razina ugniježđenja

Nekoliko je različitih vrsta "sila" koje vuku sintaksu u smjeru većeg broja razina ugniježđenja elemenata. Tri vrste će ovdje biti navedene kao primjeri. U većini slučajeva postoji znatna suprotna sila koja proizlazi iz primarnog zahtjeva na sintaksu, prilagođenosti upitima. Svaku priliku za smanjivanje broja razina ugniježđenja treba ozbiljno razmotriti. Čak i kad smanjivanje broja razina dolazi u sukob s drugim zahtjevima, konačnu odluku ne smije se donijeti bez analiziranja utjecaja na upite.

5.5.1 Izbjegavanje elementa-spremnika kolekcije

Možemo razmotriti konstrukciju za zapis popisa parametara metode. Informacija koju treba zapisati je položaj parametra u nizu, njegova vrsta, ime i moguće modifikatore. Logično je da položaj u nizu bude definiran položajem elementa koji opisuje parametar u nizu drugih takvih elemenata. Oznaka vrste je višestruko korištena jezična komponenta pa je smještena u svoj podelement. Ime može biti atribut glavnog elementa, kao i modifikatori. Imamo dva prijedloga za konstrukciju:

```
<method>
  <parameters>
    <parameter name="param1">
      <primitive-type name="int"/>
    </parameter>
    <parameter name="param2">...</parameter>
  </parameters>
  <body>...</body>
</method>

<method>
  <parameter name="param1">
    <primitive-type name="int"/>
  </parameter>
  <parameter name="param2">...</parameter>
  <body>...</body>
</method>
```

Prva konstrukcija izgleda logičnije i čišće jer jedan element, `parameters`, sadrži kompletnu informaciju o parametrima i ne dolazi do miješanja niza parametara s drugim elementima na istoj razini. To svojstvo korisno je i u stablastom prikazu izvornog koda – detalje o parametrima može se lako sakriti neovisno o ostalim detaljima, sakrivanjem sadržaja samo jednog elementa. Međutim, svaki hijerarhijski put do nekog parametra morat će proći kroz taj spremnički element pa će njegov zapis biti ponavaljajuć i nepotrebno dugačak. Konkretno, u slučaju prve konstrukcije, *XPath*-ov izraz koji vraća drugi parametar glasio bi `//method/parameters/parameter[2]`, a u slučaju druge `//method/parameter[2]` – kraći i pregledniji zapis.

Srodan, ali suptilno drugačiji primjer je već prikazana konstrukcija za popis najavljenih iznimki metode. To je niz poziva na vrstu podatka bez dodatnih detalja. Stoga je nepotrebno uvoditi poseban element npr. `exception`. Elementi niza su generički elementi `type`. U ovom slučaju potreban je nadelement-spremnik koji definira smisao generičkih elemenata. Upiti nad takvom konstrukcijom neće biti repetitivni, npr. `//method/throws/type[2]`. Slična konstrukcija pogodna je i za zapis

argumenata metode, gdje član u nizu argumenata može biti bilo koji Javin *izraz*. Izraz je još jedan značajan primjer višestruko iskoristive jezične komponente.

5.5.2 Iterativne umjesto rekurzivnih konstrukcija

Zbog svojstava specifičnih za beskontekstnu gramatiku konstrukcije koje sadrže niz istih ili srodnih dijelova lakše je modelirati u konkretnoj sintaksi kao binarno stablo.

Možemo kao primjer uzeti Javin izraz za pristup članu dvodimenzionalnog polja. Dvodimenzionalno polje može se tretirati kao jednodimenzionalno polje kojem su članovi jednodimenzionalna polja. Pristup nekom njegovom članu modelira se kao pristup članu unutarnjeg polja koje je rezultat izraza za pristup članu vanjskog polja. Dakle, u striktnom smislu i ne postoji poseban izraz za pristup članu dvodimenzionalnog polja, već se on izgrađuje ugnježđavanjem unutar samog sebe izraza za pristup članu jednodimenzionalnog polja. Ako XML-ovska sintaksa doslovno odražava ovo modeliranje tog izraza, on će biti zapisan u takvom, ugniježđenom obliku. Ovako bi mogao izgledati zapis Javinog izraza `array[1][2]`:

```
<array-access>
  <array-access>
    <variable-access name="array"/>
    <integer value="1"/>
  </array-access>
  <integer value="2"/>
</array-access>
```

Takav izraz bit će vrlo nespretan za formulaciju upita. Na primjer, upit koji vraća sve fiksne pristupe gornjem lijevom članu dvodimenzionalnog polja (oblika *array*[0][0]) izgledao bi ovako:

```
//array-access[array-access[integer/@value="0"]][integer/@value="0"]
```

U slučaju još više dimenzija izraz bi postao iznimno nepregledan. Umjesto toga, izraz se može zapisati ovako:

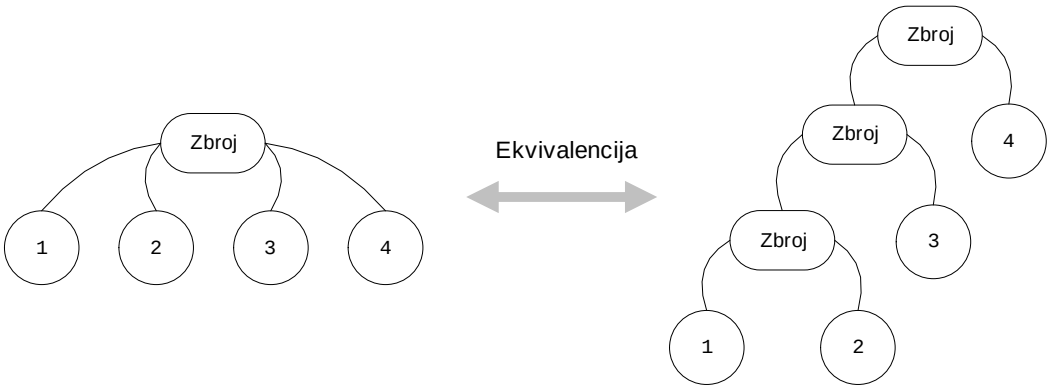
```
<array-access>
  <variable-access name="array"/>
  <integer value="1"/>
  <integer value="2"/>
</array-access>
```

Prvi element identificira polje, a daljnji zapisuju izraze za indekse po svakoj dimenziji. XPath-ov upit nad ovakvom konstrukcijom, ekvivalentan gornjem, glasio bi

```
//array-access [integer[1]/@value="0"] [integer[2]/@value="0"]
```

Povećanjem broja dimenzija upit bi se samo produljavao, bez dodatnih razina ugnježđavanja. Takav izraz može se lako rastaviti na niz dijelova koje se može tumačiti jedan po jedan.

Drugi primjer konstrukcije, koja se puno češće u praksi koristi mnogostruko ulančena, je binarna numerička operacija, npr. zbrajanje. Već je u odjeljku 1.1.4 objašnjeno kako je sama koncepcija binarne operacije uvedena radi modeliranja infiksnog zapisa koji je najlakši za čitanje pa se koristi u konkretnoj sintaksi većine programskih jezika. Primjerice, izraz `1 + 2 + 3 + 4` modelira se kao tri binarne operacije ugniježđene jedna u drugu, dakle `((1 + 2) + 3) + 4`. U sintaksi infoseta nema potrebe zadržavati taj pristup jer je model operacije s nizom operanada potpuno ekvivalentan, a rezultirajuća struktura infoseta znatno prilagođenija automatskoj obradi. Ovo je ilustrirano slikom 5.3.



Slika 5.3: Niz i ekvivalentno binarno stablo

Moguće je uvesti i zajedničku konstrukciju za nekoliko srodnih operacija, npr. niz operacija zbrajanja i oduzimanja zapisati kao niz podelemenata od npr. elementa `additive-expression`.

5.5.3 Izbacivanje prioriteta operacija i zagrada

Osim upravo opisane mane modeliranja operacija po uzoru na konkretnu sintaksu, taj pristup pati od još jedne, čak i znatno veće mane. U odjeljku 1.1.4 objašnjeno je kako je definiranje prioriteta operacija nužno radi uklanjaja dvosmislenosti infiksnog zapisa operacija. Također je potrebno uvesti i zagrade radi grupiranja operanada mimo prioriteta. Budući da sintaksa infoseta nema problema s dvosmislenošću, prioriteti postaju izlišni, a samim time i eksplicitno modeliranje zagrada.

Stoga je neprimjereno zadržati sintaktičko pravilo ilustrirano slikom 1.6 prema kojem se svaki izraz mora sastojati iz najmanje onoliko elemenata ugniježđenih jedan u drugi koliko je razina prioriteta definirano (u slučaju Jave radi se o njih petnaestak). Drugim riječima, otpada argument protiv uvođenja produkcija kao što je ona sa slike 1.7. Ne samo da nema argumenata protiv takvih sintaktičkih pravila, već postoje i vrlo snažni argumenti u njihovu korist, najbolje ilustrirani ispisom 8.4 na kojem se vidi izgled upita koji mora uzeti u obzir sve nepotrebne razine ugniježđavanja elemenata.

5.5.4 Izbjegavanje općenitih nadelemenata

U radu s izvornim kodom redovito se pojavljuju skupovi jezičnih konstrukcija koje su po nekom kriteriju srodne i trebaju se tretirati zajednički, npr. upit treba locirati sve konstrukcije u nekom kontekstu koje zadovoljavaju dotični kriterij. Mogućih kriterija je mnogo, ali neki su vrlo prirodni i često praktični. Većinom su kriteriji takvi da ne postoji način izravne formulacije značenja kriterija u upitu pa preostaje domisliti se upita drugačijeg značenja, ali s istim krajnjim efektom. Na primjer, upit može skup željenih konstrukcija pobrojati član po član, što znači da će vjerojatno biti dugačak i nerazumljiv. Problem je posebno izražen u slučajevima u praksi učestalih kriterija i velikih skupova konstrukcija koje ih zadovoljavaju. Upit se može znatno pojednostaviti ako se u sintaksu uvede općeniti nadelement svakog člana skupa pa će se moći pretraživati po tom nadelementu. Ako postoji nadelement, on u većini upita smeta kao redundantan, ali tu jednu konkretnu vrstu upita čini drastično jednostavnijom.

Dobar primjer za ovo je konstrukcija za zapis jedne cijele izjave. Postoji mnogo različitih izjava, a posebno su brojni *izrazi* koji mogu igrati i ulogu samostalne izjave. Dakle, postoji mnogo različitih elemenata koji se mogu naći u ulozi nositelja čitave izjave. Nadalje, rastavljanje bloka koda na niz izjava vrlo je tipičan i učestao zadatak. Moguće je uvesti nadelement, npr. `statement`, koji sadrži jednu izjavu. Tada će uočavanje niza izjava postati trivijalno jednostavan posao, ali će mnogi drugi upiti biti opterećeni redundantnom razinom ugniježđenja koja ne donosi nikakvu informaciju. Ovo je primjer pristupa sa i bez nadelementa:

```
<block>
  <variable-declaration>...</variable-declaration>
  <assignment>...</assignment>
  <method-invocation>...</method-invocation>
  <postfix-increment>...</postfix-increment>
</block>
```

```
<block>
  <variable-declaration>...</variable-declaration>
  <statement>
    <assignment>...</assignment>
  </statement>
  <statement>
    <method-invocation>...</method-invocation>
  </statement>
  <statement>
    <postfix-increment>...</postfix-increment>
  </statement>
</block>
```

Načelo je i ovdje izbjeći dodatnu razinu ugniježđenja. Ona ionako pomaže samo jednoj konkretnoj vrsti upita. U suprotnom takve bi se razine mogle uvoditi posvuda, prema raznoraznim kriterijima koji se učine smislenim. U slučaju razmatranog upita, on može ipak biti prilično jednostavan i bez dodatne razine ako se može osloniti na jedinstveni element koji sadrži niz izjava, npr. element `block`. U konkretnom slučaju Jave trebat će iz niza isfiltrirati deklaracije lokalnih klasa i varijabli jer to nisu izjave. Tako formuliran upit manje je čitak jer željeni skup opisuje negativno, isključujući suvišne članove. Svejedno, alternativa s uvođenjem nadelementa svake izjave je ukupno lošija.

5.6 Maksimiziranje strogosti WXS-ovske definicije sheme

Svako sintaktičko pravilo koje nije nametnuto shemom dokumenta morat će biti nametano nakon standardne provjere valjanosti nekim drugim putem. To će vjerojatno značiti da će svaki automatski proces morati izvesti vlastitu provjeru u opsegu koji je za njega relevantan. Shema dokumenta stoga treba biti što je moguće stroža, tj. pokrivati što veći dio pravila sintakse. Razvijatelj sheme mora biti temeljito upoznat s WXS-om da bi mogao osmisliti najbolji način za nametanje pravila.

Na primjer, WXS ne dozvoljava propisivanje ovisnosti modela sadržaja elementa o vrijednostima njegovih atributa ili međuovisnosti između vrijednosti samih atributa. Takva vrsta ograničenja često je vrlo poželjna. Dobar primjer je deklaracija metode u Javi. Postoji složena interakcija između različitih ograničenja na osobine metode. Na primjer, apstraktna metoda ne smije biti privatna ni statička i ne smije imati tijelo, statička metoda mora imati tijelo, itd. Postoji uobi-

čajen način, korišten u oblikovanju samog definicijskog jezika WXS, za rješenje ove situacije, ali on ima svoju cijenu. Umjesto atributa o čijoj vrijednosti ovisi model sadržaja može se uvesti izbor između više podelemenata, po jednog za svaku dopuštenu vrijednost atributa. Dakle, umjesto `<method abstract="true">...</method>` imali bismo `<method> <abstract>...</abstract> </method>`. Svaki podelement može imati različit model sadržaja. Atributi nad kojima postoje ograničenja ovisna o vrijednosti tog atributa mogu biti preseljeni u podelemente. Cijena primjene ovog rješenja je dodatna razina ugniježđenja i smanjena preglednost čitave konstrukcije.

Uvijek će biti propisa za čiju ugradnju je cijena plaćena na strani elegancije sintakse veća nego dobitak na strani strogoće sheme. Prilikom odlučivanja da li ugraditi takav propis u shemu dobro je razmotriti koliko se često koristi jezično svojstvo u pitanju i koliko će biti složeno provesti naknadnu provjeru drugim mehanizmima – drugim riječima, koliki učinak u praksi će imati izostavljanje propisa.

5.7 Odabir između atributa i jednostavnog elementa

U odjeljku 2.1 razjašnjeno je kako su u slučaju potpuno strukturiranog infoseta atribut i jednostavan element srodnog karaktera. Kad se prilikom oblikovanja sintakse susretne s podatkom koji se modelira kao par (*ključ*, *vrijednost*), on će se najčešće moći modelirati bilo atributom, bilo jednostavnim elementom. Na primjer, za zapis pristupa varijabli postoji alternativa:

```
<variable name="var1">
  ...
</variable>
```

ili

```
<variable>
  <name>var1</name>
  ...
</variable>
```

Stoga se otvara prilika za definiranje jasnijih kriterija po kojima treba napraviti odabir. Ključna razlika između atributa i jednostavnog elementa je u definiciji njihovog identiteta: atribut je identificiran svojim imenom, dok je element identificiran i imenom i položajem u listi čvorova-braće. Stoga je poredak između atributa-braće nevažan i ne smiju postojati dva atributa-braće istog imena, dok je poredak između elemenata-braće važan i nema ograničenja na brojnost istoimenih

elemenata-braće. Atributov model identiteta je jednostavniji i stoga se preporuča njegova uporaba kad god priroda modeliranog podatka to dozvoljava.

Postoji i srodna dilema u slučaju kad treba koristiti element. Taj element može biti kao gore, jednostavan s tekstualnim sadržajem, ali može biti i prazan element s jednim atributom. Na primjer, doslovan broj može se zapisati na ova dva načina:

```
<int>1</int>
```

ili

```
<int value="1"/>
```

U potpuno strukturiranom dokumentu konzistentnije je koristiti drugi oblik jer se time u potpunosti izbjegava korištenje tekstualnih čvorova pa je ukupan broj različitih vrsta čvorova manji. Međutim, prvi oblik ima nešto kraći zapis i ne treba mu dodatni identifikator za atribut. Ako se u dokumentu vrlo često pojavljuju ovakvi elementi, preferira se prvi oblik, u protivnom je bolje koristiti drugi.

5.8 Izbor imena čvorova

Izbor imena za elemente i atribute u XML-ovskoj sintaksi vrlo je značajan jer snažno utječe na razumljivost i preglednost kako dokumenata, tako i upita. U XML-u se udomaćio stil imena s isključivo malim slovima i crticom "-" za odvajanje riječi. Bitno je također paziti i na konciznost imena. Nepotrebno dugačka imena ne donose nikakvu informaciju, a znatno produljavaju i zamućuju izgled dokumenta i upita. S druge strane, za imena u XML-u je uobičajeno koristiti pune riječi umjesto skraćenica.

Poglavlje 6

Predložena sintaksa

U ovom poglavlju bit će izneseni odabrani detalji svojstava predložene sintakse programskog jezika Java i postupka njenog razvoja. Formalizacija sintakse WXS-ovskom definicijom sheme iznesena je u Dodatku D. Ukratko, bit će izloženi sljedeći aspekti oblikovanja sintakse:

1. Primarni izraz
2. Binarni izrazi
3. Deklaracija metode u klasi
4. Napredna ograničenja nametnuta shemom

6.1 Primarni izraz

Najopsežniji dio sintakse je onaj koji se bavi *izrazima*, glavnim nositeljima informacije u naredbenom dijelu koda (za razliku od deklarativnog). Na primjer, tijelo metode većinom se sastoji od izraza – što samostalnih, što uklopljenih u upravljačke izjave (za upravljanje tijekom izvršavanja i sl.). Izrazi se redovito pojavljuju ugniježđeni jedni u druge. Središnji dio svakog izraza u Javi je *primarni izraz*; svaki izraz može se raščlanjivati na podizraze sve dok se ne dođe do primarnih. U gramatici iz odjeljka 1.1.4 neterminal *Faktor* odgovarao bi primarnom izrazu.

Primarni izrazi najraznolikiji su dio Javine sintakse i jedini (uz *type cast*) čiji rezultat može biti referentne vrste. Vrlo srodna konstrukcija koja je u specifikaciji jezika Java [GJSB00] izdvojena zbog problema specifičnih za konkretnu sintaksu jest *izrazno ime* (*expression name*), a ovdje će se i ona tretirati kao primarni izraz.

U većini primarnih izraza kao element konkretne sintakse koristi se točka. Na primjer, najčešća varijanta sintakse izraza za pristup atributu instance *field access* je *Primary*.*Identifier* ([GJSB00], 15.11), sintaksa izraza *class literal* ([GJSB00], 15.8.2) je *Type*.class itd. Međutim, u skladu s načelima iz odjeljaka 5.1 i 5.3 u sintaksi infoseta nikako se ne smije ove raznorodne izraze povezivati po toj zajedničkoj karakteristici jer je ona isključivo vezana uz konkretnu sintaksu. Primjerice, od spomenutih izraza u predloženoj sintaksi prvi se zapisuje ovako:

```
<field-access name="instanceInt">
  ... primarni izraz ...
</field-access>
```

Njemu je kao srodna pridružena formalno sasvim različita konstrukcija opisana na udaljenom dijelu specifikacije ([GJSB00], 6.5.6), izrazno ime koje opisuje pristup statičkom (klasinom) atributu i zapisuje se isto kao gore, samo se unutar elementa `field-access` umjesto primarnog izraza nalazi element `reference-type`.

Drugi navedeni izraz (*class literal*) zapisuje se u sasvim drugačijem elementu:

```
<class-literal>
  <reference-type> ... </reference-type>
</class-literal>
```

U XML-ovskom zapisu, kao što se može vidjeti, nigdje se i ne spominje točka. U ranijoj verziji sintakse pristup atributu zapisivao se posebnim podelementom koji je služio za razlikovanje pristupa statičkom od pristupa instancinom atributu:

```
<field-access name="instanceInt">
  <containing-instance>
    ... primarni izraz ...
  </containing-instance>
</field-access>
```

odnosno

```
<field-access name="instanceInt">
  <containing-type>
    ... primarni izraz ...
  </containing-type>
</field-access>
```

Međutim, vodeći se načelom iz odjeljka 5.5, ti nadelementi su ukinuti jer je i dalje lako ustanoviti razliku: ili je podelement `reference-type` pa se radi o pristupu statičkom atributu, ili nije pa se radi o pristupu instancinom atributu.

6.1.1 Poziv na vrstu

Povezano s raspravom o sintaktičkoj ulozi točke je i pitanje zapisa poziva na vrstu podatka. Pod pozivom na vrstu smatra se navođenje njenog imena, npr. u deklaraciji varijable ili u pristupu klasinom atributu. U Javi se vrste dijele na primitivne i referentne i shodno tome svaka ima svoj element u predloženoj sintaksi. Sintaksa dodatno razlikuje vrste polja kao poseban slučaj. Element za primitivnu vrstu je jednostavne strukture: `<reference-type name="..."/>`, gdje je ime jedno od njih sedam iz skupa primitivnih vrsta propisanih specifikacijom. S druge strane, ime referentne vrste može biti složeno iz više različitih dijelova koji se u tradicionalnom zapisu svi odvajaju točkom. Primjer imena sa svim dijelovima je `jezix.srctest.DotTest.Inner1.Inner2`. Ovdje je `Inner2` lokalno (nekvalificirano) ime vrste na koju se poziva; `Inner1` je ime njene sadržavajuće vrste (unutar koje je deklarirana); `DotTest` je ime sadržavajuće vrste od `Inner1`; i napokon, `jezix.srctest` je ime paketa koje se, bez obzira na pojavu točke, ne raščlanjuje. Dakle, u primjeru se točka pojavljuje u više uloga: kao dio imena paketa, za odvajanje imena paketa od imena vrste, za odvajanje imena sadržavajućih vrsta itd. U predloženoj sintaksi isti primjer poziva na vrstu zapisuje se ovako:

```
<reference-type name="Inner2" package="jezix.srctest">
  <enclosing-type name="DotTest"/>
  <enclosing-type name="Inner1"/>
</reference-type>
```

Za zapis poziva na vrstu polja postoji više razumnih rješenja. Jedan od pristupa je da se u elemente `primitive-type` i `reference-type` ugradi i atribut `dimensions` koji bi govorio koliko dimenzija ima polje. Međutim, tada bi bilo otežano razlikovanje između "običnih" vrsta i vrsta polja. Nadalje, svaka vrsta polja je referentna vrsta pa ne bi bilo konzistentno pozivati se na nju elementom `primitive-type`. Stoga je odlučeno da vrsta polja ima zaseban element, `array-type`, npr. za jednodimenzionalno polje `int`-ova:

```
<array-type dimensions="1">
  <primitive-type name="int"/>
</array-type>
```

Ovaj pristup, osim što je logički konzistentan i lako razumljiv, omogućava i strožu definiciju sheme – npr. u izrazu kreiranja instance polja dozvoljena je isključivo vrsta polja, a u izrazu za pristup atributu vrsta polja nije dozvoljena (vrste polja nemaju statičke atribute).

Ovdje navedena tri elementa jedini su način u čitavoj sintaksi za poziv na vrstu. Time je u skladu s načelom iz odjeljka 5.4 osiguran jednostavan način pronalaženja svih poziva na neku vrstu, a nije izgubljeno ništa na ostalim važnim svojstvima sintakse. Dobiveno je i na lakoći zapisivanja jer nije potrebno razmišljati kako se na vrstu poziva u svakom konkretnom slučaju.

6.1.2 Poziv metode

Struktura poziva metode ima sličnosti sa strukturom za pristup atributu. Ključna je razlika što se ovdje treba omogućiti i navođenje argumenata poziva. Puni oblik je sljedeći:

```
<method-invocation name="methodName">
  ... primarni izraz, poziv na vrstu ili from-superclass ...
  <arguments>
    ... izraz 1 ...
    ... izraz 2 ...
    itd.
  </arguments>
</method-invocation>
```

Ovaj oblik nastao je odabirom između više mogućnosti. Kao i kod pristupa atributu, u ranijoj verziji prvi dio (lokacija metode) navodio se unutar elementa `containing-instance` ili `containing-type`. Postojala je i dilema je li potreban nadelement `arguments` ili se argumente može navoditi izravno, kao djecu od `method-invocation`. Također, u slučaju da se on koristi, postojala je dilema treba li se on navoditi uvijek ili samo kad ima argumenata. Nakon što je izbačen nadelement `containing-instance`, element `arguments` postao je nužan da razlikuje izraz čiji rezultat je lokacija metode od izraza za argument metode. Iz razloga konzistentnosti i kratkoće propisano je da se element `arguments` nikad ne navodi prazan – dakle, ako argumenata nema, on se izostavlja.

6.2 Binarni izrazi

Najbrojnija vrsta izraza više razine, onih koji se ne mogu nalaziti kao listovi u stablu raščlanjenog Javinog izraza, je binarni izraz. Takav izraz definiran je operacijom te lijevim i desnim operandom. Rezultat operacije može biti numerički ili logički, a operandi mogu biti numerički, logički ili referentni. Operacija čiji rezultat je iste vrste kao operandi može se ugnježđivati unutar same sebe, točnije lijevi operand može biti ta ista operacija (zbog lijeve asocijativnosti binarnih operatora u

Javi). Takve operacije preglednije je promatrati kao *ulančane* u niz operacija koje se obavljaju slijeva nadesno, gdje se kao lijevi operand uzima akumulirani rezultat prethodnih operacija. Upravo za takve operacije postojala je najveća nedoumica kako ih najbolje zapisati. Dvije su osnovne varijante, rekurzivna i iterativna konstrukcija. Nadalje, konkretna sintaksa Jave propisuje složenu ljestvicu prioriteta operacija potrebnu za parsiranje izraza u tradicionalnom zapisu koji sadrži niz operacija bez zagrada za eksplicitno grupiranje. Budući da prioriteti postoje isključivo zbog prirode zapisa po tradicionalnoj sintaksi, u XML-ovskom zapisu, bez obzira na odabir varijante zapisa, ne treba nametati prioritete.

6.2.1 Rekurzivna konstrukcija

Ova konstrukcija drži se strukture infiksnog zapisa u konkretnoj sintaksi. Element operacije ima točno dva podelementa od kojih svaki može biti taj isti element. U slučaju zapisa normalno ulančanih operacija, samo prvi podelement smije biti isti kao nadelement. Primjer zapisa:

```
<operation-x>
  <operand>
    <operation-x>
      <operand><variable name="a"/></operand>
      <operand><variable name="b"></operand>
    </operation-x>
  </operand>
  <operand><variable name="c"></operand>
</operation-x>
```

Razlozi protiv ovakve sintakse već su izneseni u odjeljku 5.5.2.

6.2.2 Iterativna konstrukcija

Ova konstrukcija niz ulančanih operacija srodnih vrsta zapisuje kao niz elemenata. Element operacije ima dva ili više podelemenata s operandima. Konstrukcija je znatno preglednija, a i upiti nad njom će biti jednostavniji za sastavljanje i razumljiviji. U odabiru detalja sintakse na raspolaganju je više varijanti.

Varijanta 1

Ovdje je sadržaj glavnog elementa niz naizmjeničnih elemenata `operand` i `operator`. Unutar elemenata `operand` nalazi se element izraza koji predstavlja dotični operand. Shematski prikaz konstrukcije:

```
<expression-x>
  <operand>
    <expression-y>...</expression-y>
  </operand>
  <operator @type/>
  <operand>
    <expression-z>...</expression-z>
  </operand>
  <operator @type/>
  <operand>
    <expression-w>...</expression-w>
  </operand>
</operation-x>
```

Varijanta 2

U ovom slučaju sadržaj glavnog elementa je niz isključivo elemenata `operand` unutar kojih se nalazi zapis izraza koji predstavlja dotični operand. Svi podelementi osim prvog imaju i atribut `operation` koji govori kojom operacijom je dotični operand povezan s prethodnim.

```
<operation-x>
  <operand>...</operand>
  <operand @operation>...</operand>
  <operand @operation>...</operand>
</operation-x>
```

Varijanta 3

Ovdje je izbačen nadelement `operand` i izravno naveden element izraza koji predstavlja operand.

```
<expression-x>
  <expression-y>...</expression-y>
  <operator @type/>
  <expression-z>...</expression-z>
  <operator @type/>
  <expression-w>...</expression-w>
</operation-x>
```

Ova varijanta je odabrana jer izbjegava dodatnu razinu ugniježđenja unutar elementa `operand`, što je imalo negativan učinak na upite. Time je postalo nužno zadržati element `operator` koji sadrži vrstu operacije. Međutim, ovaj element je potreban samo u izrazima u kojima se mogu ulančavati različite operacije, npr. zbrajanje i oduzimanje. Kod mnogih izraza ulančava se samo jedna operacija, npr. logičko "i", logičko "ili" itd. U tim slučajevima sadržaj glavnog elementa izraza je čisti niz operanada.

6.3 Deklaracija metode u klasi

Ukupno najzahtjevnijom i najkompleksnijom pokazala se konstrukcija za deklaraciju metode u klasi (u sučelju je situacija vrlo jednostavna). Specifikacija Jave propisuje složena ograničenja na pojavu raznih modifikatora odnosno tijela metode. Konkretno, ograničenja su sljedeća:

1. Smije biti naveden najviše jedan od modifikatora `abstract`, `static` i `final`
2. Smije biti naveden najviše jedan od modifikatora `abstract`, `native` i `strictfp`
3. Tijelo metode ne postoji ako i samo ako je naveden modifikator `abstract` ili `native`
4. Ako je naveden `abstract`, ne smiju biti navedeni `private` niti `synchronized`
5. Modifikator `abstract` smije se pojaviti samo u apstraktnoj klasi.

U traženju najboljeg kompromisa između složenosti zapisa i strogosti definicije sheme u WXS-u razmatrano je ukupno šest varijanti (neke su grupirane u podvarijante).

6.3.1 Varijanta 1a: općeniti element s više logičkih atributa

Ovo je najjednostavniji, "naivan" pristup. Postoji jedan općeniti element `method` koji podržava niz logičkih atributa, po jedan za svaki modifikator. Shematski prikaz konstrukcije:

```
<method @abstract @static @final @native @strictfp @synchronized>
  <returns>...</returns>
  <throws>...</throws>
  <parameters>...</parameters>
  <body> (može se izostaviti)... </body>
</method>
```

Ovaj zapis nema nikakvih komplikacija i najpregledniji je. Međutim, ne može se nametnuti nijedno od pobrojanih ograničenja i stoga je odbačen.

6.3.2 Varijanta 1b: općeniti element s jednim trovrijednosnim atributom

Pristup je srodan gornjem, samo što se nameće ograničenje 1. Umjesto logičkih atributa `abstract`, `static` i `final` uveden je jedan trovrijednosni atribut `modifier` s

dozvoljenim vrijednostima "abstract", "static" i "final". Shematski prikaz konstrukcije:

```
<method modifier="abstract" | "static" | "final" | null @access @native @strictfp>
  <returns>...</returns>
  <throws>...</throws>
  <parameters>...</parameters>
  <body> (može se izostaviti)... </body>
</method>
```

Ova varijanta malo je manje pregledna od gornje jer se neki modifikatori zadaju logičkim atributima, a neki pomoću posebnog trovrijednosnog atributa. Uspijeva pokriti tek jedno od nabrojanih ograničenja.

6.3.3 Varijanta 2: općeniti element s jednim dvovrijednosnim atributom i izborom tri podelementa

Ovdje se primjenjuje uzorak spominjan u odjeljku 5.6: umjesto samo body uveden je izbor između tri podelementa: body, abstract ili native. Druga dva su prazni elementi i samo donose informaciju o modifikatoru metode. Atribut modifier ovdje poprima dvije moguće vrijednosti: "static" ili "final", ili uopće ne postoji. Shematski prikaz:

```
<method @name modifier="static" | "final" | null @access @native @strictfp>
  <returns>...</returns>
  <throws>...</throws>
  <parameters>...</parameters>
  <body>...</body> | <abstract/> | <native/>
</method>
```

Ovim pristupom u potpunosti je nametnuto ograničenje 3, a ograničenja 1 i 2 su nametnuta djelomično. Zapis je dodatno zakompliciran zbog neobičnog navođenja modifikatora abstract i native.

6.3.4 Varijanta 3: tri različita konkretna elementa

U ovom slučaju više ne postoji jedinstven općenit element method, već ih je tri: instance-method, static-method i abstract-method. Za svakog se sad mogu nametnuti ograničenja koja se tiču njegovog slučaja deklaracije. Shematski prikaz:

```
<instance-method @name @access @final @synchronized @strictfp>
  <returns>...</returns>
  <throws>...</throws>
```

```
  <parameters>...</parameters>
  <body>...</body> | <native/>
</instance-method>

<static-method @name @access @synchronized @strictfp>
  <returns>...</returns>
  <throws>...</throws>
  <parameters>...</parameters>
  <body>...</body> | <native/>
</static-method>

<abstract-method @name @access>
  <returns>...</returns>
  <throws>...</throws>
  <parameters>...</parameters>
</abstract-method>
```

Ključna prednost ovog pristupa je izvrsna mogućnost nametanja ograničenja. Nametnuta su sva ograničenja osim 5 i jednog dijela 2: međusobna isključivost modifikatora `native` i `strictfp`, koju se ne može nametnuti ni u varijantama koje slijede. Međutim, zapis ima čitav niz mana. Upit za nalaženje svih metoda deklariranih u klasi bit će složen i neintuitivan; zapis je nekonzistentan s deklaracijom metode u sučelju, kao i s deklaracijom atributa, koje koriste jedinstveni element.

6.3.5 Varijanta 4a: općeniti element s tri konkretna podelementa

Ovim pristupom pokušava se ujediniti ključne prednosti Varijante 3 s ključnim prednostima ranijih varijanti. Zadržan je općeniti element `method`, ali ispod njega dolazi izbor između tri konkretna podelementa `abstract`, `instance` i `static`. Općeniti element zadržava atribut `name`, a ostali su raspoređeni po podelementima u skladu s ograničenjima:

```
<method @name>
  <abstract @access |
   instance @access @final @synchronized @strictfp |
   static @access @synchronized @strictfp>
     <returns>...</returns>
     <throws>...</throws>
     <parameters>...</parameters>
     <body>...</body> | <native/>
  </abstract | instance | static>
</method>
```

U ovom slučaju zadržana su sva ograničenja iz Varijante 3, ali je poboljšana kvaliteta upita i vraćena je konzistentnost s drugim deklaracijama. Ukupna ele-

gancija konstrukcije narušena je u odnosu na Varijantu 1, ali odlučeno je da se ta žrtva prinese u korist drastično veće strogosti.

6.3.6 Varijanta 4b: kao 4a, ali zajednički podelementi kao djeca općenitog elementa

Ova, posljednja varijanta je odabrana kao ukupno najbolja i uvrštena u predloženu sintaksu. Elementi `returns`, `parameters` i `throws` izvađeni su iz konkretnih podelemenata i stavljeni izravno u sadržaj općenitog elementa kao njegova djeca:

```
<method @name @access>
  <returns>...</returns>
  <throws>...</throws>
  <parameters>...</parameters>

  <abstract/> |
  <instance @final @synchronized @strictfp |
   static @synchronized @strictfp>
    <body> | <native/>
  </instance |static>
</method>
```

U odnosu na 4a povećana je kvaliteta upita koji pristupaju informaciji u navedena tri podelementa jer *XPath*-ovi izrazi ne moraju prolaziti kroz dodatnu razinu ugniježđenja. Vrlo je značajno da je sad konstrukcija u potpunosti usklađena s konstrukcijom za deklaraciju metode sučelja. Smještaj atributa `access` bio je najkontroverzniji jer su ograničenja na njega različita ovisno o prirodi deklaracije metode. Konkretno, ograničenje 4 nalaže da apstraktna metoda ne smije biti privatna. Moralo se odabrati između mogućnosti nametanja tog ograničenja i potpune konzistentnosti s deklaracijom metode sučelja. Izabrana je slabija strogost u korist elegantnijih upita koji će proizlaziti iz konzistentnog modela. Ostali atributi iz podelemenata uopće se ne pojavljuju u deklaraciji metode sučelja pa nema ni nekonzistentnosti.

U čitavoj predloženoj sintaksi ovo je jedino mjesto na kojem je korišten uzorak spominjan u odjeljku 5.6, naime korištenje podelemenata za informaciju koju je elegantnije zapisati u atributima. Jedino ovdje je cijena u pogledu elegantnosti smatrana opravdanom s obzirom na strogoću WXS-ovske sheme koju omogućava.

U daljnjem postupku profinjavanja sintakse razmatrale su se dvije varijante za zapis parametara: unutar nadelementa-sadržitelja kolekcije ili bez njega. Prva varijanta bila je bez nadelementa, kao niz elemenata `parameter`. Kasnije je uočeno

da bi trebalo uskladiti srodne konstrukcije za deklaraciju lokalnih varijabli, iteratora u petlji `for`, parametara i uhvaćenih iznimki. Što se tiče njihovog korištenja, parametri i uhvaćene iznimke identični su lokalnim varijablama. Razlika je samo u načinu njihove inicijalizacije: parametri i uhvaćene iznimke automatski su inicijalizirani kad počne izvršavanje bloka u kojem su deklarirani, dok lokalne varijable treba eksplicitno inicijalizirati. Mnogo zadataka automatske obrade tretirat će svo troje na identičan način. Stoga je odlučeno da se u deklaraciju parametara metode uvede nadelement `parameters`, a njegova djeca da budu elementi `variable` koji su već korišteni za deklaraciju lokalnih varijabli. Sukladno tome izmijenjena je i konstrukcija za deklaraciju uhvaćene iznimke.

6.4 Napredna ograničenja nametnuta shemom

6.4.1 Ograničenja na vrstu operanada

Prvenstveno zahvaljujući tome što je sintaksa oslobođena potrebe nametanja prioriteta operacija, bilo je moguće uvesti smislenije ograničenje – na vrstu operanada. Budući da je za mnoge operacije jasno definirana njihova vrsta rezultata kao logički, numerički ili referentni, a isto tako i vrsta operanada, bilo je moguće nametnuti ograničenja koja ne dozvoljavaju pojavu operanada pogrešne vrste. Na primjer, operandi u zbrajanju moraju biti numerički, a operandi u logičkom "i" moraju biti logički. Time je postignuta strogoća koja nije obuhvaćena beskontekstnom gramatikom tradicionalne sintakse. Ovo ograničenje puno je smislenije i korisnije od nametanja prioriteta koji su "nužno zlo" u tradicionalnom zapisu.

6.4.2 Usklađenost vrste deklarirane varijable i vrste inicijalizacijskog izraza

WXS-ovskom shemom propisano je da, ovisno o tome je li element za deklaraciju vrste varijable `primitive-type`, `reference-type` ili `array-type`, inicijalizacijski izraz mora biti primitivne ili referentne vrste, s dodatnom mogućnošću elementa `array-initializer` za vrstu polja.

6.4.3 Jedinstvenost identifikatora

Još jedan primjer ograničenja koje se može nametnuti WXS-ovskom shemom, a nije ostvarivo beskontekstnom gramatikom, jest jedinstvenost identifikatora.

WXS-ov mehanizam za nametanje jedinstvenosti (`xs:unique`) iskorišten je za nametanje jedinstvenosti imena atributa, metoda, klasa, sučelja, itd. Lokalne varijable (i već spomenuti njima srodni entiteti) moraju imati jedinstvena imena ne samo u bloku u kojem su deklarirani, već i u svim podblokovima. Ovo ograničenje nije se moglo u potpunosti nametnuti WXS-om pa je nametnuta samo jedinstvenost unutar istog bloka. Slijedi primjer definicije elementa `class` unutar kojeg se propisuje jedinstvenost imena atributa i metoda. Skupina lokacija na kojima se pojavljuju atributi koji moraju biti jedinstveni zadaje se *XPath*-ovim izrazom u elementu `selector`, a samo ime atributa u elementu `field`.

```
<xs:element name="class" type="ClassDeclaration">
  <xs:unique name="Field">
    <xs:selector xpath="field"/>
    <xs:field xpath="@name"/>
  </xs:unique>
  <xs:unique name="ClassMethod">
    <xs:selector xpath="method"/>
    <xs:field xpath="@name"/>
  </xs:unique>
</xs:element>
```

6.5 Ortogonalna proširenja nad predloženom sintaksom

S ciljem pokazivanja koristi od primjene XML-ovske sintakse s ortogonalnim proširenjima razvijeno je nekoliko konkretnih primjera – dokumentacija programskog sučelja (funkcionalni ekvivalent *Javadoc*-a) te dva primjera kodnih predložaka za često korištene konstrukcije. Proširenja se obrađuju XSLT-ovim transformacijama, iznesenim u Dodatku E.

6.5.1 Dokumentacija programskog sučelja

Ovo proširenje definira nekoliko elemenata za meta-podatke – npr. `description` za opis nekog dijela sučelja (klase, metode, parametra). Ti elementi ubacuju se unutar zapisa Javinog koda i XSLT-om se od njih izgrađuje HTML-ovska dokumentacija. Primjer izvornog koda s dokumentacijskim elementima naveden je u nastavku.

Ispis 6.1: Izvorni kod s ortogonalnim proširenjem za dokumentaciju

```
<?xml version="1.0" encoding="UTF-8"?>
<compilation-unit package="hr.fer.tel.jezix"
```

```
    xmlns="http://tel.fer.hr/jezix/java"
    xmlns:jd="http://tel.fer.hr/jezix/java/doc"
    xmlns:h="http://www.w3.org/1999/xhtml">
  <class name="ClassTest" access="public">
    <jd:description>Sample <h:em>class</h:em> in Jezix-Java.</jd:description>
    <jd:author value="Marko Topolnik"/>
    <jd:author value="Joe Developer"/>
    <jd:version value="2.0"/>
    <extends package="java.lang" name="Object"/>
    <implements package="java.lang" name="Cloneable"/>
    <field access="public" name="i">
      <jd:description>A variable used in test code.</jd:description>
      <primitive-type name="int"/>
    </field>
    <field access="package-private" name="b">
      <primitive-type name="boolean"/>
    </field>
    <field access="private" final="true" name="j">
      <primitive-type name="int"/>
    </field>
    <field access="protected" name="k">
      <primitive-type name="int"/>
    </field>
    <field access="package-private" static="true" name="l">
      <primitive-type name="int"/>
    </field>
    <constructor access="public">
      <jd:description>Primary constructor.</jd:description>
      <parameter name="i">
        <jd:description>An integer</jd:description>
        <primitive-type name="int"/>
      </parameter>
      <parameter name="b">
        <jd:description>A boolean</jd:description>
        <primitive-type name="boolean"/>
      </parameter>
      <throws name="Exception">
        <jd:description>Thrown a lot these days.</jd:description>
      </throws>
      <body>
        <super-constructor-invocation/>
      </body>
    </constructor>
    <method name="instanceMethod1">
      <jd:description>An instance method.</jd:description>
      <instance>
        <returns>
          <jd:description>A boolean. What you give is what you get.</jd:description>
          <primitive-type name="boolean"/>
        </returns>
        <parameter name="a">
          <jd:description>A boolean to be returned.</jd:description>
          <primitive-type name="boolean"/>
        </parameter>
        <body>
          <return>
            <variable name="a"/>
          </return>
        </body>
      </instance>
    </method>
  </class>
</compilation-unit>
```

U primjeru se mogu uočiti prednosti pristupa s ortogonalnim proširenjima XML-a u odnosu na komentare u tradicionalnom zapisu. Dokumentacijski elementi smješteni su u kontekst onoga što se dokumentira pa je nepotrebno ponavljati identifikatore unutar njih, kao što je to slučaj s *Javadoc*-om. Stoga ne može doći ni do gubitka konzistentnosti. Također, tako je za većinu dokumentacije dovoljan samo element `description`, dok u *Javadoc*-u treba koristiti različite oznake za dokumentiranje različitih entiteta, npr. `return`, `param`, `exception` itd. HTML-ovski elementi koriste se za oblikovanje teksta i provjerom dobre oblikovanosti čitavog dokumenta automatski će biti provjereni i ti elementi, dok u slučaju tradicionalnog zapisa takvu provjeru može obavljati samo specijalizirani alat.

XSLT-ova transformacija iznesena u dodatku ilustrira kako se informacija iz ortogonalnih proširenja kombinira s osnovnim sadržajem dokumenta i generira HTML-ovski dokument. Pri izradi XSLT-a nije posvećeno mnogo pažnje atraktivnom oblikovanju izlaznog dokumenta jer taj dio posla nije relevantan u ovom kontekstu.

6.5.2 Predložak za metodu `equals`

Pomoću ovog proširenja moguće je zadati kriterije jednakosti dva objekta, a transformacijom na osnovu toga generirati metodu `equals` koja radi prema tim kriterijima. Kostur metode je sljedeći (parom znakova $ označeni su promjenjivi dijelovi):

```
public boolean equals( java.lang.Object o )
{
  if ( this == o )
    return true;
  if ( !( o instanceof $trenutna-klasa$ ) )
    return false;
  $trenutna-klasa$ that = ($trenutna-klasa$) o;
  return $uvjet-jednakosti$;
}
```

Slijedi primjer izvornog koda s navedenim parametrima predloška. Proširenje je smješteno unutar elementa `method`. Njegova struktura u ovom obliku je nevaljana (ne zadovoljava shemu), međutim nakon transformacije postat će valjana. Samo proširenje i nakon transformacije će ostati u dokumentu i to na istom mjestu, unutar generirane metode `equals`. U današnjim razvojnim okružjima također postoje kodni predlošci, no njihov je problem što služe samo za jednokratno generiranje koda, a ne postoji trajna veza između predloška i generiranog koda. Stoga se bilo kakva izmjena nad kodom mora obavljati ručno. Prednost pristupa iznesenog ovdje je u trajnoj vezi između predloška i koda.

Ispis 6.2: Izvorni kod s proširenjem za generiranje metode `equals`

```
<?xml version="1.0" encoding="UTF-8"?>
<compilation-unit package="hr.fer.tel.jezix"
    xmlns="http://tel.fer.hr/jezix/java"
    xmlns:t="http://tel.fer.hr/jezix/java/template-equals">
  <class name="EqualsTemplateTest" access="public">
    <extends package="java.lang" name="Object"/>
    <implements package="java.lang" name="Cloneable"/>
    <field access="private" name="i">
      <primitive-type name="int"/>
    </field>
    <field access="private" name="b">
      <primitive-type name="boolean"/>
    </field>
    <field access="private" name="s">
      <reference-type name="String"/>
    </field>
    <method name="equals">
      <t:template-equals>
        <t:conditions>
          <t:same>
            <t:field name="i"/>
            <t:field name="b"/>
          </t:same>
          <t:equal>
            <t:field name="s"/>
          </t:equal>
        </t:conditions>
      </t:template-equals>
    </method>
  </class>
</compilation-unit>
```

XSLT-ova transformacija, iznesena u dodatku, u stanju je i pobrojati atribute u klasi i na osnovu njih automatski generirati uvjete jednakosti dva objekta. Atribute primitivne vrste uspoređuje se operatorom "==", a atribute referentne vrste pozivom njihove metode `equals`. Automatsko generiranje uvjeta zahtijeva se sljedećim zapisom proširenja:

```
<t:template-equals>
  <t:conditions>
    <t:automatic/>
  </t:conditions>
</t:template-equals>
```

Najvažnije za istaknuti je da je za razvoj XSLT-ove transformacije bilo potrebno svega 3-4 sata, što se može smatrati vrlo brzim razvojem za ovakav zadatak.

Jezik XSLT vrlo je dobro prilagođen ovakvim namjenama, gdje se većina izvornog koda preuzima nepromijenjena, a na manjem dijelu obavlja se transformacija. Ekvivalentno rješenje koje radi sa tradicionalnim zapisom, čak i uz oslanjanje na parserske alate, zahtijevalo bi mnogo više vremena i u mnogim situacijama bi se pokazalo kao neisplativo.

Metoda `equals` generirana prema oba proširenja (bilo navedeni eksplicitni, bilo automatski kriteriji) daje identičan rezultat:

Ispis 6.3: Generiran kod metode `equals`

```
<method access="public" name="equals">
  <t:template-equals>
    ... sadržaj proširenja prekopiran iz ulaznog dokumenta ...
  </t:template-equals>
  <returns>
    <primitive-type name="boolean"/>
  </returns>
  <parameters>
    <variable name="o">
      <reference-type package="java.lang" name="Object"/>
    </variable>
  </parameters>
  <instance>
    <body>
      <if>
        <condition>
          <equal>
            <this/>
            <variable-access name="o"/>
          </equal>
        </condition>
        <then>
          <return>
            <boolean value="true"/>
          </return>
        </then>
      </if>
      <if>
        <condition>
          <not>
            <instance-of>
              <variable-access name="o"/>
              <reference-type name="EqualsTemplateTest"/>
            </instance-of>
          </not>
        </condition>
        <then>
          <return>
            <boolean value="false"/>
          </return>
        </then>
      </if>
      <variable name="that">
        <reference-type name="EqualsTemplateTest"/>
        <reference-cast>
          <reference-type name="EqualsTemplateTest"/>
          <variable-access name="o"/>
        </reference-cast>
      </variable>
      <return>
```

```
        <conditional-and>
          <equal>
            <field-access name="i">
              <this/>
            </field-access>
            <field-access name="i">
              <variable-access name="that"/>
            </field-access>
          </equal>
          <equal>
            <field-access name="b">
              <this/>
            </field-access>
            <field-access name="b">
              <variable-access name="that"/>
            </field-access>
          </equal>
          <method-invocation name="equals">
            <field-access name="s">
              <this/>
            </field-access>
            <arguments>
              <field-access name="s">
                <variable-access name="that"/>
              </field-access>
            </arguments>
          </method-invocation>
        </conditional-and>
      </return>
    </body>
  </instance>
</method>
```

odnosno, prevedeno na tradicionalni zapis:

```
public boolean equals( java.lang.Object o )
{
  if ( this == o )
    return true;
  if ( !( o instanceof EqualsTemplateTest ) )
    return false;
  EqualsTemplateTest that = (EqualsTemplateTest) o;
  return ( this.i == that.i ) && ( this.b == that.b ) && this.s.equals( that.s );
}
```

6.5.3 Predložak za prolazak po kolekciji

Kodni uzorak za prelazak iteratorskim objektom po kolekciji objekata vrlo se često koristi, a i prilično je složen pa ga je pogodno generirati na osnovu predloška. Kostur uzorka je sljedeći:

```
for ( java.util.Iterator iter = $kolekcija$.iterator(); iter.hasNext(); )
{
  $vrsta-elementa$ $ime-varijable-s-elementom$ = ($vrsta-elementa$) iter.next();
  $posao-koji-treba-obaviti$
}
```

Slijedi primjer izvornog koda s ortogonalnim proširenjem za ovaj predložak.

Ispis 6.4: Izvorni kod s proširenjem za generiranje prelaska po kolekciji

```
<?xml version="1.0" encoding="UTF-8"?>
<compilation-unit package="hr.fer.tel.jezix"
    xmlns="http://tel.fer.hr/jezix/java"
    xmlns:t="http://tel.fer.hr/jezix/java/template-iterator">
  <import>
    <all-from package="java.util"/>
  </import>
  <class name="IteratorTemplateTest" access="public">
    <field name="clues">
      <reference-type name="Collection"/>
      <new-instance-of-class>
        <reference-type name="ArrayList"/>
      </new-instance-of-class>
    </field>
    <method name="iterateOverCollection">
      <instance>
        <body>
          <for>
            <t:iterate-collection>
              <t:collection-access>
                <field-access name="clues">
                  <this/>
                </field-access>
              </t:collection-access>
              <t:item-type>
                <reference-type name="Clue" package="hr.fer.tel.clue"/>
              </t:item-type>
              <t:item-var name="clue"/>
              <t:do>
                <method-invocation name="getKey">
                  <t:item-access/>
                </method-invocation>
              </t:do>
            </t:iterate-collection>
          </for>
        </body>
      </instance>
    </method>
  </class>
</compilation-unit>
```

Kod ovog proširenja najzanimljivije je uočiti kako su isprepleteni elementi izvornog koda i proširenja. Unutar proširenja pojavljuju se isječci izvornog koda koji predstavljaju predložak, a unutar elementa `method-invocation` pojavljuje se element proširenja `t:item-access` na koji se može gledati i kao na predložak unutar predloška – na tom mjestu ubacit će se standardni predložak za pristup elementu

kolekcije.

XSLT-ova transformacija za ovo proširenje jednostavnija je od one za `equals` jer ne treba prikupljati podatke iz udaljenih dijelova dokumenta. Za njen razvoj trebalo je svega dva sata. Transformacijom gore navedenog primjera dobiva se sljedeća konstrukcija `for`:

Ispis 6.5: Generiran kod za prelazak po kolekciji

```
<for>
  <t:iterate-collection>
    ... sadržaj proširenja prekopiran iz ulaznog dokumenta ...
  </t:iterate-collection>
  <initialize>
    <variables>
      <reference-type package="java.util" name="Iterator"/>
      <variable name="iter">
        <method-invocation name="iterator">
          <field-access name="clues">
            <this/>
          </field-access>
        </method-invocation>
      </variable>
    </variables>
  </initialize>
  <condition>
    <method-invocation name="hasNext">
      <variable-access name="iter"/>
    </method-invocation>
  </condition>
  <loop>
    <block>
      <variable name="clue">
        <reference-type name="Clue" package="hr.fer.tel.clue"/>
        <reference-cast>
          <reference-type name="Clue" package="hr.fer.tel.clue"/>
          <method-invocation name="next">
            <variable-access name="iter"/>
          </method-invocation>
        </reference-cast>
      </variable>
      <method-invocation name="getKey">
        <variable-access name="clue"/>
      </method-invocation>
    </block>
  </loop>
</for>
```

odnosno, prevedeno na tradicionalni zapis:

```
for ( java.util.Iterator iter = this.clues.iterator(); iter.hasNext(); )
```

```
{
  hr.fer.tel.clue.Clue clue = (hr.fer.tel.clue.Clue) iter.next();
  clue.getKey();
}
```

Poglavlje 7

Srodni radovi

U okviru softverskog razvoja postoji velik broj situacija u kojima je uputno primijeniti XML. Danas se XML primjenjuje u velikom broju slučajeva gdje je potrebno pohranjivati ili komunicirati strukturiranim podacima. Pohranjivanje koda u XML-u je rjeđe. Ovdje će se razmatrati radovi u kojima se pohranjuje kod ili barem neki njegovi aspekti.

7.1 Analiza i projektiranje primjenom XML-a

U ovim situacijama XML se ne koristi za opis samog izvornog koda koji i dalje ostaje u tradicionalnom zapisu, već za opis nekih aspekata značenja koda. Na primjer, u fazi dizajna aplikacije ne specificiraju se implementacijski detalji nego samo klasne hijerarhije i međuodnosi objekata. Tijekom razvoja većeg softverskog projekta potrebno je provoditi i raznovrsne analize koda na statističkoj razini, zanemarujući detalje. Podaci koje treba zapisati su potpuno strukturirani i stoga pogodni za zapis u XML-u. To olakšava izvedbu alata koji barataju tim podacima jer se mogu osloniti na XML-ovske tehnologije. Budući da sintaksa za ove namjene treba obuhvatiti samo odabrane aspekte koda, moguće ju je dizajnirati tako da bude primjenjiva na kod više različitih programskih jezika. Takva integracija vrlo je korisna u projektima koji primjenjuju više jezika.

Ovaj pristup ima svoju primjenu, ali ne rješava problem zapisa izvornog koda u XML-u. Ovdje je ipak spomenut jer bi postao korisniji kad bi i sam kod bio u XML-u. Tada bi prijenos podataka između zapisa koda i njegovih ekstrahiranih aspekata bio znatno olakšan.

Software Development Foundation (SDS) je otvorena arhitektura dizajnirana

da olakša razvoj alata za razvijanje programske podrške. Jezgra SDS-a je XML-ovski *Code Structure Format* (CSF) [San00]. Svrha tog formata je da se izgradi baza analitičkih podataka visoke razine o sveukupnom kodu softverskog projekta. Informacija se na taj način stavlja na raspolaganje raznim analitičkim alatima. CSF ne sadrži pune detalje programskog koda – na primjer, ne opisuje implementacije metoda.

GXL [WKR01] je XML-ovski format za opis grafova s vrstama i atributima. Podržava hijerarhijske grafove u kojima se čvor može rastaviti u podgraf. Također podržava i hipergrafove u kojima grane povezuju više od dva čvora. Namjena mu je da služi kao standardni format za razmjenu podataka između alata za rad nad grafovima i donekle se udomaćio u zajednici softverskog inženjerstva. Ovaj format nije namijenjen da zapisuje doslovan programski kod, nego njegove analitičke i dizajnerske aspekte više razine. Općeniti jezici za baratanje infosetom poput XSLT-a i *XQuery*-ja mogli bi se iskoristiti za reverzni inženjering programskog koda (zapisanog po XML-ovskoj sintaksi) u GXL, a također i za generiranje punog koda na osnovu GXL-ovskih grafova.

7.2 Izvorni kod označen XML-ovim oznakama

U ovom slučaju također se zadržava tradicionalni zapis koda, ali se u sam dokument izvornog koda dodaju XML-ove oznake. Oznake obuhvaćaju semantički povezane nizove tokena, na primjer čitava definicija klase je obuhvaćena jednom oznakom, a unutar nje daljnje oznake obuhvaćaju pojedine metode. Oznake su dakle ugniježđene i izgrađuju stablastu strukturu koju se može zamisliti kao da se nalazi u sloju iznad teksta izvornog koda. Slijedi kratak primjer:

```
<class><access>public</access> class <name>Class1</name> {
 <field><type primitive="true">int</type> <name>i</name>;</field>
}</class>
```

XML se ovdje primjenjuje na model polustrukturiranih podataka i stoga je relevenatno razmatrati sam zapis i u njemu uočiti podjelu na sadržaj i strukturu. Ako se iz zapisa uklone sve oznake, tj. sve šiljate zagrade zajedno sa svim tekstom unutar njih, ostat će čisti izvorni kod u tradicionalnom zapisu. Iz toga slijedi da atribut, nalazeći se unutar otvarajuće oznake, nije dio sadržaja pa ima poseban status jer omogućava ubacivanje dodatnih podatka u sloj iznad teksta izvornog koda.

Jedan od problema s ovim pristupom proizlazi iz već rečenog – sam zapis dokumenta ima primarni značaj, što narušava jednu od spomenutih prednosti XML-a – da se taj zapis može zanemariti. Konkretno, standard XML dozvoljava da se prilikom parsiranja dokumenta *normalizira* prazan prostor između elemenata – npr. da se nekoliko razmaka zaredom pretvori u samo jedan razmak. Time bi se izgubilo oblikovanje običnog teksta koje je u ovom pristupu značajno. Još jedan problem je što se većina pravila ovakvog zapisa ne može formalizirati shemom dokumenta. Pravila zapisa uključuju i sadržaj tekstualnih čvorova između elemenata, a taj sadržaj je potpuno izvan kontrole jezika za definiciju sheme.

Source Markup Language, srcML [CMM02a, CMM02b, CMK04], predstavnik je ovog pristupa. To je XML-ovski format za označavanje izvornog koda jezika C++.

7.3 Modeliranje stabla apstraktne sintakse

U ovom odjeljku spominju se radovi s pristupom najsrodnijim onom koji se zastupa u ovoj disertaciji. Pri kraju odjeljka posebno su izdvojene dvije najrelevantnije sintakse koje slijede ovaj pristup i koje su iskorištene radi usporedbe s predloženom sintaksom. Relevantne su iz dva razloga: najbliže su predloženoj sintaksi po koncepciji (programski jezik Java, modeliranje stabla apstraktne sintakse) i najcjenjenije su u istraživačkoj zajednici koja se bavi ovim područjem.

Povremeno se u raspravama o izvornom kodu u XML-u pojavljuje i ideja razvoja jedne univerzalne sintakse koja bi obuhvatila semantiku "svih objektnih jezika". Pritom se zaboravlja da "objektni" ne može biti definicija jezika, već samo jedna od njegovih karakteristika. Dapače, prvenstveni razlog postojanja različitih programskih jezika nije njihova razlika u sintaksi, već upravo u skupu podržanih semantičkih konstrukcija. Zamislivo je da se razvije sintaksa koja bi podržavala semantiku više programskih jezika podržavajući uniju njihovih pojedinačnih skupova semantičkih konstrukcija. Osim što takva sintaksa očito ne bi bila univerzalna, bila bi i prilično besmislena jer, kao što je već objašnjeno, svaki programski jezik podržava pažljivo odabran skup semantičkih konstrukcija i kvaliteta jezika mjeri se kroz uspješnost njihove kompleksne interakcije. Nije jasno što bi se postiglo miješanjem više takvih skupova unutar jedne sintakse. Ono što bi ipak moglo biti korisno je definiranje širih konvencija kako zapisati neku tipičnu jezičnu konstrukciju koja se koristi u mnogo jezika. To bi povećalo stupanj interoperabilnosti jezika i olakšalo razvoj alata koji rade s više jezika.

XQueryX [xqx03] je W3C-ova XML-ovska sintaksa, još uvijek u razvoju, sa semantikom ekvivalentnom jeziku *XQuery*. Ova sintaksa predviđa opis *XPath*-ovih izraza u potpuno strukturiranom obliku. S druge strane, neki od izbora u dizajnu u suprotnosti su sa zahtjevima definiranim u ovoj disertaciji:

- Sintaksa je snažno utjecana od EBNF-ovske formalizacije konkretne sintakse *XQuery*-ja (dokument [xqx03] ju pogrešno naziva apstraktnom sintaksom). To rezultira već opisanim redundantnim razinama ugniježđenja. Značenje takvih konstrukcija ne proizlazi prirodno iz njihove strukture i to će narušiti kvalitetu upita nad njima.

- Atributi su u potpunosti zaobiđeni i koriste se isključivo jednostavni podelementi. To nameće nepotrebnu disciplinu u zapisu dokumenta jer podelementi moraju biti navedeni u strogom poretku čak i kad je taj poredak semantički irelevantan.

- Hijerarhija vrsta WXS-a i s njom povezan poseban atribut `xsi:type` koriste se tamo gdje su potrebni različiti modeli sadržaja ovisno o vrijednosti atributa. Posljedica toga je da će upiti biti prisiljeni koristiti taj poseban atribut. Takvi upiti bit će slabije razumljivi i samo radi tog atributa trebat će uvijek deklarirati prefiks za prostor imena za instance WXS-a.

XSLT [xsl99] je već opisani jezik za transformaciju XML-ovskog dokumenta. Ovdje se spominje zato što ima XML-ovsku sintaksu. Ona kombinira zapis po XML-ovoj sa zapisom po vlastitoj konkretnoj sintaksi. Svi *XPath*-ovi izrazi ostavljeni su u tekstualnom obliku. Kontekst u kojem je ovaj jezik nastao, 1999. godine, opravdava ovakav pristup. Zapis je prilagođen tome da razvijatelj ručno zapisuje XML-ovski dokument u editoru teksta, a u takvom okružju bilo bi vrlo nepraktično zapisivati potpuno parsirani XPath.

o:XML [Kla03] je objektni jezik zasnovan otpočetka na XML-ovskoj sintaksi. Iskorištava XML na dva načina: uvodi XML-ovski fragment kao osnovnu vrstu podatka i koristi XML-ovo stablo dokumenta da iskaže ugniježđenje jezičnih konstrukcija. Korištenje XML-ovskog fragmenta kao osnovnog podatka znatno olakšava manipulaciju podacima složene strukture u odnosu na "klasične" programske jezike. o:XML omogućava baratanje XML-om u stilu XSLT-a (prepoznavanjem uzoraka) i DOM-a (objektni model podataka). Projekt koristi i koncepciju ortogonalnih proširenja za dokumentaciju programskog sučelja i za generiranje jediničnih ispita

za provjeru ispravnosti koda. Iz istih razloga kao i XSLT, i ovaj jezik propisuje vlastitu konkretnu sintaksu za zapis izraza (*XPath*-ovih, regularnih i aritmetičko-logičkih). Time je smanjena mogućnost automatiziranog baratanja kodom pomoću jezika za obradu infoseta i onemogućena stroga definicija sheme.

Superx++ [spx] je također objektni jezik zasnovan na XML-ovskoj sintaksi. Zanimljiv je po tome što radnu memoriju također modelira kao XML-ov infoset, što bi trebalo omogućiti laku inspekciju memorije, serijalizaciju objekata itd. I ovaj jezik djelomično koristi vlastitu konkretnu sintaksu. Također, umjesto *XPath*-a koristi se vlastiti jezik iste namjene, ali različite sintakse i slabije izražajnosti. Ovo se može smatrati nedostatkom jer se razvijatelj koji već poznaje *XPath* mora priučavati drugačijoj sintaksi za izražavanje istih značenja. Trenutna definicija sheme u WXS-u je vrlo siromašna, tj. labava i mnogim elementima dozvoljava proizvoljan sadržaj.

XML-Encoded Source ili XES [xes] je XML-ovski format za zapis izvornog koda više programskih jezika. Format ne zapisuje značenje, već samo sintaksu. Točnije, konstrukcije srodnog značenja iz raznih jezika zapisuju se u istom obliku. Ovisno o programskom jeziku čiji kod se zapisuje, ista konstrukcija u XES-u imat će različito značenje. Definicija sheme je u WXS-u i, iako je format navodno općenit, strogo je vođena Javinom sintaksom. Sintaksa pati od raznih nedostataka već viđenih i drugdje.

eXtenDE je projekt koji je započeo kao rad na proširivom razvojnom okruženju, ali prvi važan korak je bilo odlučiti kako predstavljati izvorni kod XML-om. U tom kontkstu napisan je članak [Arm00] koji analizira prednosti korištenja XML-a za izvorni kod. Rezultati analize izravno su relevantni za ovu disertaciju, ali predloženi pristup se drastično razlikuje. XML se koristi samo da iskaže hijerarhiju, korištenjem bezimenih čvorova.

7.3.1 JavaML Grega Badrosa

Ova sintaksa [Bad00] najsličnija je predloženoj po istaknutim načelima dizajna, a i po samoj izvedi. Međutim, izvedba sintakse odudara od istaknutih načela na više načina, a i ne obuhvaća čitav jezik Javu. Slijedi podrobniji popis nedostataka.

Nepodržane konstrukcije. Badrosova sintaksa nema eksplicitnu podršku za izraze "*class variable name*", "*static method invocation*", "*qualified this*", "*qualified super*" i "*class literal*". Ove izraze moguće je zapisati alternativnom

konstrukcijom pogrešnog značenja, ali koja nakon transformacije daje ispravan kod u običnom tekstu. Stoga je moguće dodefinirati posebna pravila za značenje takvih konstrukcija, nakon čega bi se moglo reći da sintaksa navedene izraze podržava, ali uz neprikladan način zapisa koji otežava razumijevanje ili ga onemogućava bez poznavanja detalja o kontekstu u kojem se nalazi dokument s izvornim kodom. Primjer za *class literal*, `Object.class`:

```
<field-access name="class">
  <var-ref name="Object"/>
</field-access>
```

Ovdje se koristi konstrukcija za pristup atributu objekta na koji pokazuje lokalna varijabla. Umjesto imena varijable navodi se ime vrste, a umjesto imena atributa ključna riječ `class`. Takav zapis sam za sebe nema smisla, a smisao postaje jasan tek ako ako se uoči sličnost zapisa pristupa instanci i *class literal*-a u tradicionalnom izvornom kodu.

Nadalje, sintaksa uopće ne podržava "*qualified class instance creation*" i "*qualified superclass constructor invocation*". Značenje tih izraza nemoguće je zapisati, a nije moguće zapisati ni neku alternativnu konstrukciju koja bi proizvela ispravan kod u običnom tekstu.

Neraščlanjeno ime vrste, miješanje s imenom varijable. Sintaksa ne raščlanjuje puno ime vrste na paket, sadržavajuće vrste i lokalno ime vrste. Osim što to narušava zahtjev na potpunu strukturiranost infoseta, bez poznavanja konteksta je i nemoguće parsirati ime jer se ne može ustanoviti što je dio imena paketa, a što je ime sadržavajuće vrste. Na primjer, ako imamo navedeno ime vrste "`java.util.List`", to može, između ostalog, biti klasa `List` unutar klase `util` u paketu `java` ili klasa `List` u paketu `java.util`.

Situaciju pogoršava činjenica da se na isto mjesto u strukturi zapisuje i ime varijable i ime vrste pa je potrebna zahtjevna analiza čitavog dokumenta samo da se ustanovi priroda navedenog identifikatora. Na primjer, u sljedećem zapisu,

```
<field-access name="out">
  <var-ref name="System"/>
</field-access>
```

`System` je možda ime lokalne varijable, parametra ili atributa, a možda i vrste. Ovisno o prirodi tog imena mijenja se vrsta izraza koji konstrukcija

zapisuje. Potrebno je, dakle, analizirati čitav dokument da bi se locirale sve deklaracije vidljive u kontekstu pojave konstrukcije i ustanoviti zauzima li neka od tih deklaracija identifikator `System`. Time je i smanjena mogućnost otkrivanja pogreški – dovoljno je pogrešno napisati ime varijable da bi se to ime protumačilo kao ime vrste. Da bi se ustanovilo postoji li navedena vrsta, potrebno je analizirati širi kontekst razvojnog okružja i sve vrste dostupne u trenutku kompilacije.

Zapis dimenzija polja u znakovnom nizu imena vrste. Broj dimenzija polja definiran je brojem parova pravokutnih zagrada na kraju imena vrste, npr. `<type name="int[][]"/>` označava dvodimenzionalno polje podataka vrste `int`.

Rekurzivne konstrukcije umjesto iterativnih. Za zapis izraza s binarnim operatorima koristi se zapis u obliku binarnog stabla, kao što proizlazi iz Javine konkretne sintakse. Iako formalno ispravne, takve konstrukcije drastično otežavaju sastavljanje upita nad njima.

Labava definicija sheme u DTD-u. DTD-ovska shema dokumenta Badrosove sintakse dopušta mnogobrojna odstupanja od ispravnosti. Neki od primjera su: dozvoljen bilo kakav tekst kao znak binarne operacije, dozvoljeno bilo kakvo ugniježđavanje binarnih operacije (narušavanje lijeve asocijativnosti i prioriteta operacija), dozvoljena bilo koja kombinacija modifikatora metode (private, static, final, abstract, native itd.) bez obzira na kontekst, nema ograničenja na izbor znakova u identifikatorima.

Uključuje podatke o zapisu u običnom tekstu. Sintaksa ima posebnu podršku za dodatne podatke o tradicionalnom zapisu. Može se zapisati redak i stupac na kojem počinje izraz. Ovakvi podaci, koji ne spadaju u primarni sadržaj, spadaju u ortogonalna proširenja.

Redundantno navođenje imena konstruktora. Sintaksa zahtijeva da se unutar deklaracije svakog konstruktora ponovi ime klase u kojoj se nalazi. To je redundancija koja omogućava uvođenje pogreški u dokument, a ne donosi nikakvu prednost.

Nema definiran prostor imena. Ovaj nedostatak povezan je i s uporabom DTD-a koji ima slabu (zaobilaznu) podršku za prostore imena. Bez prostora imena dodatno je otežano provođenje ideje ortogonalnih proširenja.

7.3.2 JavaML Evana Mamasa

Mamasova sintaksa [MK00, ZK01] udaljenija je od predložene, iako također dijeli s njom mnoga načela dizajna. Najvažnija sličnost je u tome što ova sintaksa isto propisuje potpuno strukturirani model dokumenta. Međutim, u znatnoj mjeri je utjecana od Javine konkretne sintakse. Slijedi pregled najvažnijih nedostataka.

Visoko redundantan zapis izraza. Svaki izraz, bez obzira na vrstu, potrebno je zapisati u istoj strukturi koja ugniježđene sadrži sve vrste izraza definirane u konkretnoj sintaksi. Takva struktura ne samo da ne donosi nikakvu informaciju, već i drastično smanjuje razumljivost i preglednost kako samog dokumenta, tako i upita nad njim. Na primjer, najjednostavniji mogući izraz, doslovan broj, npr. `0` zapisuje se ovako:

```
<Expression>
  <ConditionalExpression>
    <ConditionalOrExpression>
      <ConditionalAndExpression>
        <InclusiveOrExpression>
          <ExclusiveOrExpression>
            <AndExpression>
              <EqualityExpression>
                <InstanceOfExpression>
                  <RelationalExpression>
                    <ShiftExpression>
                      <AdditiveExpression>
                        <MultiplicativeExpression>
                          <UnaryExpression>
                            <UnaryExpressionNotPlusMinus>
                              <PostfixExpression>
                                <PrimaryExpression>
                                  <PrimaryPrefix>
                                    <Literal Value="0"/>
                                  </PrimaryPrefix>
                                </PrimaryExpression>
                              </PostfixExpression>
                            </UnaryExpressionNotPlusMinus>
                          </UnaryExpression>
                        </MultiplicativeExpression>
                      </AdditiveExpression>
                    </ShiftExpression>
                  </RelationalExpression>
                </InstanceOfExpression>
              </EqualityExpression>
            </AndExpression>
          </ExclusiveOrExpression>
        </InclusiveOrExpression>
      </ConditionalAndExpression>
```

```
    </ConditionalOrExpression>
  </ConditionalExpression>
</Expression>
```

Nepotrebno složene i neintuitivne konstrukcije. Neke konstrukcije, prije svega za deklaracije (klasa, metoda, varijabla itd.) i primarne izraze (pristup atributu, poziv metode itd.), imaju nepotrebno složenu i nepreglednu strukturu. Primjer deklaracije klase:

```
<TypeDeclaration>
  <ClassDeclaration isPublic="True">
    <UnmodifiedClassDeclaration Identifier="DotTest" Extends="True">
      <Name Identifier="BaseClass"/>
      <Name Identifier="Interface1"/>
      <Name Identifier="Interface2"/>
      <ClassBody> ... </ClassBody>
    </UnmodifiedClassDeclaration>
  </ClassDeclaration>
</TypeDeclaration>
```

Tri elementa, `TypeDeclaration`, `ClassDeclaration` i `UnmodifiedClassDeclaration` ponavljaju isto značenje. Nadalje, atribut `Extends` neočekivano je logičke vrste i govori o tome označava li prvi od elemenata `Name` koji slijede ime natklase ili ime implementiranog sučelja.

Nepodržane konstrukcije. Sintaksa ne podržava konstrukciju "*qualified super*". Moguće je zapisati konstrukciju koja daje ispravan rezultat u običnom tekstu, ali je nekonzistentna s predviđenim načinom zapisivanja konstrukcija sa `super`. Primjer predviđenog korištenja (izraz `super.i`):

```
<PrimaryExpression>
  <PrimaryPrefix isSuper="True" Identifier="i"/>
</PrimaryExpression>
```

Treba uočiti kako sintaksa predviđa korištenje posebnog atributa, `isSuper`. Primjer zapisa "*qualified super*" (izraz `DotTest.Inner.super.i`):

```
<PrimaryExpression>
  <PrimaryPrefix>
    <Name Identifier="DotTest.Inner1"/>
  </PrimaryPrefix>
  <PrimarySuffix Identifier="super"/>
  <PrimarySuffix Identifier="i"/>
</PrimaryExpression>
```

Ovdje se ključna riječ `super` pojavljuje kao identifikator, a korištenjem atributa `isSuper` ne može se postići potreban zapis jer je on definiran samo za element `PrimaryPrefix`.

Labava definicija sheme u DTD-u. Najvažniji primjer je način zapisa modifikatora dostupnosti. Za svaku razinu dostupnosti definiran je po jedan logički atribut, npr. `isPrivate` i `isPublic`. DTD-om, kao ni WXS-om, nemoguće je nametnuti ograničenje da samo jedan od tih atributa smije imati vrijednost `True`. Dakle, to je primjer gdje se samim načinom oblikovanja sintakse onemogućava nametanje ograničenja.

Neraščlanjeno ime vrste, miješanje s imenom varijable. Isto kao kod Badrosa (vidjeti gore).

Redundantno navođenje imena konstruktora. Isto kao kod Badrosa (vidjeti gore).

Nema definiran prostor imena. Isto kao kod Badrosa (vidjeti gore).

Poglavlje 8

Usporedba predložene sintakse s postojećima

U ovom poglavlju bit će na nekoliko primjera pokazane karakteristike i napravljena usporedba s predloženom sintaksom.

8.1 Zadaci

Slijede zadaci u kojima će biti opisan upit koji treba formalizirati, naveden primjerak izvornog koda (u tradicionalnoj sintaksi) nad kojim se obavlja upit i konačno tekst izvedbe upita u jeziku *XQuery* za svaku od tri sintakse. XML-ovski dokumenti izvornog koda nad kojima su izvršavani upiti nalaze se u Dodatku B, a rezultati upita u Dodatku C. Zadaci su osmišljeni tako da istaknu koristi od pridržavanja načela dizajna navedenih u 5 i da prikažu na koji način odstupanje od njih utječe na upite. Također se pazilo da zadaci budu realistični, tj. da budu odabrani između zadataka koji se zaista pojavljuju u praksi.

8.1.1 Zadatak 1: prepoznavanje uzorka u izrazu

Opis: pronaći sve pojave izraza koji zadovoljava uzorak

```
_ && _ && _ && ( _ || _ ) && ...
```

dakle, izraz logičke konjunkcije (točnije, "*conditional and*") koji ima najmanje četiri člana i četvrti član je disjunkcija ("*conditional or*") između dva člana. Vratiti XML-ovski element `and-operations` koji za svaki nađeni izraz vraća po jedan element `fourth-member` sa sadržajem četvrtog člana izraza.

Zadatak je prvenstveno osmišljen s ciljem da istakne opravdanost načela iz 5.5 koje kaže da treba minimizirati broj razina ugniježđenja. Konkretno, u Badrosovoj sintaksi ne koristi se iterativna umjesto rekurzivne konstrukcije za zapis binarnih izraza, a u Mamasovoj sintaksi zapis izraza dizajniran je po uzoru na EBNF-ovsku specifikaciju konkretne sintakse pa ima velik broj razina ugniježđenja.

Ispis 8.1: Ispitni kod, OrExpressionTest.java

```
package jezix.srctest;

public class OrExpressionTest
{
  public OrExpressionTest()
  {
    super();
    boolean b;
    b = true && true && true;
    b = true && true && true && true && true;
    b = true && true && true && 1 > 1;
    b = true && true && true && 1 != 1;
    b = true && true && true && true & true;
    b = true && true && true && true | true;

    // samo ova dva izraza zadovoljavaju uzorak:
    b = true && true && true && ( true || true );
    b = true && true && true && ( true || true ) && true;

    b = true && true && true && ( false || false || false ) && true;
    b = true && true && true && true && ( false || false );
    b = true && true && true && true && ( false || false ) && true;
  }
}
```

Ispitni kôd osmišljen je da sadrži više varijanti traženog izraza i razne varijante izraza koji nisu traženi, ali od traženog odstupaju u manjoj mjeri.[1]

U XML-ovskim zapisima ispitnog koda može se uočiti da zapis po predloženoj sintaksi ima oko tri puta više znakova od običnog teksta. Zapis po Badrosu je u ovom slučaju oko dva puta veći od predloženog, ali općenito su im veličine sličnije, uz manju prednost na strani predloženog zapisa. S druge strane, zapis po Mamasu je redovito veći od predloženog za gotovo čitav red veličine, oko 8 do 10 puta – dakle 30 puta u odnosu na tradicionalni zapis. Zbog enormne veličine zapisa po Mamasu bilo je potrebno uvesti neka skraćenja u ispise u dodacima.

[1] Korišteno ime paketa, `jezix.srctest`, odudara od preporuke za imenovanje paketa prema internetskoj domeni koja osigurava jedinstvenost. Ispravno puno ime paketa bilo bi `hr.fer.tel.jezix.srctest`, ali takvo bi svojom duljinom opteretilo zapis pa je za potrebe primjera skraćeno)

Mamasova sintaksa ne drži se udomaćenog stila identifikatora (5.8), već koristi takozvani *CamelCase*, a ne pazi ni na kratkoću identifikatora pa se npr. u svim elementima vezanim za izraz ponavlja riječ Expression.

Badrosova sintaksa koristi rekurzivnu konstrukciju za zapis višečlanog izraza (5.5.2), dok ostale dvije koriste iterativnu konstrukciju. I Badrosova i Mamasova sintaksa zahtijevaju eksplicitno navođenje zagrada u izrazima. Sama ideja zagrada je usko vezana uz tradicionalni zapis, a u XML-u one mogu samo biti izvor nepotrebne složenosti i redundancije i samim time otvarati dodatne mogućnosti za pogreške u zapisu.

Upit nad dokumentom po predloženoj sintaksi

Ispis 8.2: Upit nad predloženom sintaksom

```
declare default element namespace "http://tel.fer.hr/jezix/java";

<and-operations>
{
  for $fourthMember in document( "OrExpressionTest.jxj" )
    //conditional-and/*[4]/self::conditional-or[ count( * ) = 2 ]
  return <fourth-member> { $fourthMember } </fourth-member>
}
</and-operations>
```

Za obavljanje najvažnijeg dijela posla, nalaženje izraza koji zadovoljavaju uzorak, dovoljan je samo jedan XPath-ov izraz. Kompleksnost izraza gotovo je identična kompleksnosti samog teksta zadatka. Izraz u prvom koraku odabire sve elemente `conditional-and`; u drugom koraku odabire četvrti podelement; u trećem koraku provjerava je li taj element `conditional-or` i ima li točno dva podelementa. Može se reći da zapis koda upita proizlazi izravno iz zapisa zadatka, a i da je smisao upita lako dokučiti čitajući njegov kod.

Za razvoj koda ovog upita bilo je potrebno oko deset minuta. Većina vremena nije potrošena na glavni XPath-ov izraz, koji je prilično jednostavan i intuitivan, već na osmišljavanje sveukupnog izgleda upita i domišljanje kako najbolje iskoristiti jezična svojstva XQuery-ja radi postizanja što konciznijeg zapisa.

Upit nad dokumentom po Badrosovoj sintaksi

Ispis 8.3: Upit nad Badrosovom sintaksom

```
<and-operations>
{
```

```
  for $fourthMember in document( "OrExpressionTest.jmlb" )
    //binary-expr[ @op="&amp;&amp;" ]
      [ not( exists( binary-expr[ @op="&amp;&amp;" ] ) ) ]
      /parent::binary-expr[ @op="&amp;&amp;" ]
      /parent::binary-expr[ @op="&amp;&amp;" ]
      /*[2]/self::paren/binary-expr[ @op="||" ]
      [ not( exists( binary-expr[ @op="||" ] ) ) ]

  return <fourth-member> { $fourthMember } </fourth-member>
}
</and-operations>
```

Ovdje također ključni posao obavlja samo jedan XPath-ov izraz. Međutim, taj izraz je znatno složeniji i vrlo teško razumljiv. Budući da Badrosova sintaksa sve izraze zapisuje kao ugniježđene binarne izraze, zahtjevno je uopće identificirati što predstavlja četvrti član izraza logičke konjunkcije. Konstrukcija koju tražimo ima oblik

```
<binary-expr op="&amp;&amp;">
  <binary-expr op="&amp;&amp;">
    <binary-expr op="&amp;&amp;">
      <!-- ... prvi član ... -->
      <!-- ... drugi član ... -->
    </binary-expr>
    <!-- ... treći član ... -->
  </binary-expr>
  <!-- četvrti član: -->
  <paren>
    <binary-expr op="||">
      <!-- ... prvi član disjunkcije ... -->
      <!-- ... drugi član disjunkcije ... -->
    </binary-expr>
  </paren>
</binary-expr>
```

Takva konstrukcija konceptualno odgovara ideji ulančanog izraza jer, slijedeći pravilo da se prvo izračunava lijevi, a zatim desni operand, dobiva redoslijed izračunavanja kako je označeno u komentarima – dakle, redoslijed pojave u dokumentu odgovara redoslijedu izračunavanja. Izraz u prvom koraku pronalazi sve elemente `binary-expr`; u prvom predikatu koraka odabire elemente s potrebnim znakom operacije. U drugom predikatu odbacuje elemente koji imaju identičan podelement. Time se osigurava da je odabrani element najdublje u hijerarhiji ugniježđavanja istovrsnih izraza. Taj unutarnji izraz trebao bi opisivati prva dva člana čitavog izraza konjunkcije. Zatim se u sljedeća dva koraka kreće dva stupnja ugniježđavanja prema gore, provjeravajući svaki put vrstu operacije. Drugi podelement tako

odabranog elementa je traženi četvrti član konjunkcije. Na kraju se za njega provjerava je li to izraz disjunkcije unutar zagrada (po Badrosu treba eksplicitno zapisati zagrade) i da nije sastavljen od podizraza iste vrste, što bi značilo da ima više od dva člana.

Za razvoj ovog upita bilo je potrebno oko dva sata. Velik dio vremena potrošen na identifikaciju uzorka u XML-u koji treba tražiti, tj. na ustanovljavanje kako izgleda konstrukcija koja zapisuje izraz od četiri člana. Konačni upit razvijen je tek nakon dva drugačija pokušaja. Trebalo je domisliti pristup po kojem se u analizu izraza ne kreće izvana, već iznutra, od najdublje ugniježđenog podizraza. Na kraju, potrošeno je i dosta vremena na razmišljanje o svim mogućima situacijama i možebitnoj potrebi izbjegavanja nekih posebnih slučajeva.

Upit nad dokumentom po Mamasovoj sintaksi

Ispis 8.4: Upit nad Mamasovom sintaksom

```
<and-operations>
{
  for $fourthMember in document( "OrExpressionTest.jmlm" )
    //ConditionalAndExpression/InclusiveOrExpression[4]
  let $parExp := $fourthMember/*/*/*/*/*/*/*/*/*/*
    /UnaryExpressionNotPlusMinus/PostfixExpression/*/PrimaryPrefix/Expression
  where
    count( $fourthMember/* ) = 1 and
    count( $fourthMember/*/* ) = 1 and
    count( $fourthMember/*/*/* ) = 1 and
    count( $fourthMember/*/*/*/* ) = 1 and
    count( $fourthMember/*/*/*/*/* ) = 1 and
    count( $fourthMember/*/*/*/*/*/* ) = 1 and
    count( $fourthMember/*/*/*/*/*/*/* ) = 1 and
    count( $fourthMember/*/*/*/*/*/*/*/* ) = 1 and
    count( $fourthMember/*/*/*/*/*/*/*/*/* ) = 1 and
    exists( $parExp ) and
    count( $parExp/* ) = 1 and
    count( $parExp/*/* ) = 1 and
    count( $parExp/*/*/* ) = 2

  return <fourth-member> { $parExp/*/* } </fourth-member>

}
</and-operations>
```

U kodu ovog upita odmah se primjećuje posljedica velikog broja razina ugniježđavanja. Izraz koji tražimo ima oblik

```
<ConditionalAndExpression>
```

```
  <!-- prvi član -->
  <InclusiveOrExpression>
    ...
  </InclusiveOrExpression>

  <!-- drugi član -->
  <InclusiveOrExpression>
    ...
  </InclusiveOrExpression>

  <!-- treći član -->
  <InclusiveOrExpression>
    ...
  </InclusiveOrExpression>

  <!-- četvrti član -->
  <InclusiveOrExpression>
    <ExclusiveOrExpression>
      <AndExpression>
        <EqualityExpression>
          <InstanceOfExpression>
            <RelationalExpression>
              <ShiftExpression>
                <AdditiveExpression>
                  <MultiplicativeExpression>
                    <UnaryExpression>
                      <UnaryExpressionNotPlusMinus>
                        <PostfixExpression>
                          <PrimaryExpression>
                            <PrimaryPrefix>
                              <Expression>
                                <ConditionalExpression>
                                  <ConditionalOrExpression>
                                     <!-- prvi član disjunkcije -->
                                     <ConditionalAndExpression>
                                       ...
                                     </ConditionalAndExpression>

                                     <!-- drugi član disjunkcije -->
                                     <ConditionalAndExpression>
                                       ...
                                     </ConditionalAndExpression>
                                  </ConditionalOrExpression>
                                </ConditionalExpression>
                              </Expression>
                            </PrimaryPrefix>
                          </PrimaryExpression>
                  :::
          :::               (izostavljen je niz zatvarajućih oznaka)
  </InclusiveOrExpression>
  ... eventualni daljnji članovi konjunkcije
</ConditionalAndExpression>
```

Rješenje nije više praktično izvesti jednim XPath-ovim izrazom. Prvi XPath-ov izraz (čiji rezultat se pridjeljuje iteratoru `$fourthMember`) u prvom koraku odabire sve izraze logičke konjunkcije, a u drugom sve četvrte članove tih izraza. Tu će automatski otpasti svi izrazi koji imaju manje od četiri člana. Zatim se deklarira varijabla `$parExp` koja sadrži duboko ugniježđeni podizraz četvrtog člana. Podizraz mora biti izraz u zagradama (Mamasova sintaksa također nameće eksplicitno navođenje zagrada). Izraz u zagradama je prema Javinoj definiciji sintakse jedna od podvrsta primarnog izraza i stoga se unutar elementa `PrimaryExpression` nalazi `Expression`. Slijedi niz provjera potrebnih da se ustanovi da se zaista radi o samostalnom izrazu u zagradama a ne o dijelu nekog drugog izraza. Za to je nužno da svi provjeravani elementi imaju točno jedan podelement. Posljednja provjera funkcijom `count` ustanovljava da izraz disjunkcije u zagradama ima točno dva člana.

Na razvoj ovog izraza potrošeno je oko tri sata. Bilo je zahtjevno utvrditi koji uzorak u XML-u treba tražiti i točno koje provjere je potrebno ugraditi da bi se osiguralo od neželjenih slučajeva. Mnogo vremena utrošeno je i na pronalaženje i otklanjanje pogreški zbog nepreglednosti upita.

8.1.2 Zadatak 2: pronalaženje pristupa statičkim atributima

Zadatak: pronaći sve pristupe statičkom (klasinom) atributu, točnije sve primjerke konstrukcije "*class variable name*" ([GJSB00], odjeljak 6.5.6). Vratiti XML-ovski element `static-accesses` koji za svaki pronađeni izraz sadrži po jedan element `access` s atributima `type` za ime vrste čijem atributu se pristupa, `package` za ime njenog paketa (ako je naveden u kodu), `enclosing` za eventualna imena sadržavajućih vrsta odvojena točkom i `field` za ime atributa kojem se pristupa.

Ovaj zadatak zamišljen je da istakne prednosti od pridržavanja načela iz 5.1 (neovisnost značenja o kontekstu), 5.2 (eliminiranje složenih znakovnih nizova) i 5.3 (problem identifikatora s posebnim značenjem). Ni Badrosova ni Mamasova sintaksa ne pridržavaju se ovih načela. Obje zapisuju ime vrste kao jedan znakovni niz pa dolazi do miješanja imena paketa i vrsta. Nadalje, Badrosova sintaksa propisuje korištenje elementa `var-ref` i za ime varijable u zapisu pristupa varijabli (odnosno atributu, parametru ili uhvaćenoj iznimci) i za ime vrste u zapisu pristupa statičkom članu vrste. Mamasova sintaksa je konzistentna u tome da propisuje zapis izraza po uzoru na konkretnu sintaksu, tj. na zapis u običnom tekstu pa dolazi do miješanja mnogih različitih značenja "operatora *točka*" i samim time do miješanja imena varijabli i vrsta.

Ispis 8.5: Ispitni kod, DotTest.java

```
package jezix.srctest;

import java.util.Set;                                    // import type
import java.io.*;                                        // import package

public class DotTest
extends java.lang.RuntimeException                       // extends type
implements java.lang.Cloneable                           // implements type
{
  private int instanceInt;
  private static int staticInt;

  private jezix.srctest.DotTest.Inner1 instanceInner =       // field type
    new jezix.srctest.DotTest.Inner1();                      // new instance of type

  private static Inner1 staticInner;

  private void instanceMethod()
  {}

  private static void staticMethod()
  {}

  private class Inner1                                   // inner subclass
  extends DotTest
  {
    private int innerInstanceInt;

    private DotTest.Inner1 getMe()                       // method return type
    {
      return this;
    }

    private void inner1Method( jezix.srctest.DotTest Inner1 ) // method parameter type
    throws jezix.srctest.DotTest                         // method thrown type
    {
      int j;
      for ( Inner1 Inner2 = DotTest.staticInner; ; )  // local variable declared in for-init
      {
        try
        {
          j = super.instanceInt;                     // field-access from-superclass
          super.instanceMethod();                    // method-invocation from-superclass
          j = Inner2.innerInstanceInt;               // method param access
        }
        catch ( jezix.srctest.DotTest DotTest )
        {
          j = DotTest.instanceInt;                   // caught param access
          j = Inner1.instanceInt;                    // method param access
          j = StaticNested.nestedInt;                // StaticNested declared below as param
        }
        catch ( StaticNested StaticNested )
        {
          Inner1 = StaticNested.util;                // caught param access
          j = DotTest.staticInt;                     // DotTest declared above as param
        }
      }
    }

    private class Inner2                           // inner class of an inner class
    {
      private int inner2InstanceInt;

      private Inner2()
```

```
      {
        super();
        java.lang.Class c;                          // variable type
        c = DotTest.class;                          // class-literal
        c = DotTest.Inner1.class;                   // class-literal
        c = jezix.srctest.DotTest.Inner1.class;     // class-literal
        c = void.class;                             // void-class-literal
        DotTest dt;
        dt = DotTest.this;                          // qualified-this
        Object o;
        o = DotTest.Inner1.this;                    // qualified-this
        o = jezix.srctest.DotTest.Inner1.this;      // qualified-this
        int j;
        int k = 1;
        j = DotTest.staticInt;                      // static field access
        j = jezix.srctest.DotTest.staticInt;        // static field access
        j = dt.instanceInt;                         // instance field access
        j = dt.instanceInner.innerInstanceInt;      // instance field access, two layers
        j = java.util.Arrays.instanceInt;           // instance field access, multilayer
        j = DotTest.StaticNested.nestedInt;         // inner type's static field access
        j = this.inner2InstanceInt;                 // instance field access through this
        j = DotTest.Inner1.this.innerInstanceInt;   // qualified this
        j = DotTest.Inner1.super.instanceInt;       // variable from-superclass, qualified
        Inner1 in1;
        in1 = (jezix.srctest.DotTest.Inner1) o;     // cast type
        in1 = new DotTest.Inner1();                 // new instance of inner type
        o = in1.new Inner2();                       // qualified new
        jezix.srctest.DotTest.staticMethod();       // static method invocation
        in1.inner1Method( null );                   // instance method-invocation
        DotTest.Inner1.super.instanceMethod();      // method-invocation from-superclass qualified
        j = jezix.srctest.DotTest.Inner1.this.getMe().innerInstanceInt; // multiuse dot
        j = jezix.srctest.DotTest.StaticNested.IntArray.length; // static instance's field access
        DotTest DotTest = dt;
        j = DotTest.instanceInt;                    // instance field access!
        DotTest.instanceMethod();                   // instance method invocation!
      }
    }
  }

  private class Inner3                       // inner subclass of an inner class
  extends DotTest.Inner1
  {
    private Inner3(
        jezix.srctest.DotTest String        // constructor parameter type
    )
    throws java.lang.Exception              // constructor thrown type
    {
      DotTest.staticInner.super(); // qualified superclass-constructor invocation
      String.instanceInt = String.instanceInt; // instance field access
    }
  }

  private static class StaticNested
  extends RuntimeException
  {
    DotTest util;
    static int nestedInt = 1;
    static int[] IntArray;
  }

  private StaticNested java;
  private DotTest Arrays;
}
```

Kôd ovog primjera koncipiran je tako da sadrži većinu mogućih kombinacija

uporabe "operatora *točka*" u Javi. U Badrosovoj i Mamasovoj sintaksi najveći je problem ustanoviti je li prvi u nizu identifikatora odvojenih točkom zauzet u nekoj deklaraciji vidljivoj u kontekstu pojave izraza. U primjer su stoga uvršteni primarni izrazi čiji je prvi član identifikator zauzet na sve moguće načine – atribut, lokalna varijabla, `for`-varijabla, parametar, uhvaćena iznimka. Na svaki od ovih slučaja mora se posebno obratiti pozornost prilikom sastavljanja upita.

Upit nad dokumentom po predloženoj sintaksi

Ispis 8.6: Upit nad predloženom sintaksom

```
declare default element namespace "http://tel.fer.hr/jezix/java";

<static-accesses>
{
  for $fieldAccess in document( "DotTest.jxj" )//field-access[reference-type]
  let
    $type := $fieldAccess/reference-type,
    $enclosing := string-join( $type/enclosing-type/@name, "." ),
    $localName := $type/@name,
    $fieldName := $fieldAccess/@name
  order by $type/@package, $enclosing, $localName, $fieldName
  return
    <access
      package="{ $type/@package }" enclosing="{ $enclosing }"
      type="{ $localName }" field="{ $fieldName }"/>
}
</static-accesses>
```

Ovdje ponovo glavni zadatak, nalaženje svih pristupa statičkom atributu, obavlja jedan, i to vrlo jednostavan XPath-ov izraz "`//j:field-access[j:reference-type]`". Izraz nalazi sve elemente `field-access` koji sadrže element `reference-type`. To je dovoljno za jedinstveno identificiranje svih pristupa statičkom atributu i nijedne druge vrste izraza. Ostatak koda upita bavi se oblikovanjem ispisa. Budući da sintaksa propisuje zapis imena svake sadržavajuće vrste zasebno, potrebno je sva imena povezati u jedan znakovni niz, odvojena točkom. To obavlja funkcija `string-join`.

Sveukupno je za razvoj ovog upita bilo potrebno oko 30 minuta, potrošenih gotovo isključivo na osmišljavanje i izvedbu koda za oblikovanje ispisa – kod za pretraživanje dokumenta nije tražio više od 2 minute.

Upit nad dokumentom po Badrosovoj sintaksi

Ispis 8.7: Upit nad Badrosovom sintaksom

```
declare function local:isKeyword( $name as xs:string ) as xs:boolean
{
  $name = "this" or $name = "super" or $name = "class"
}

declare function local:isInScope( $varRef as element() ) as xs:boolean
{
  let $declsInScope :=
    $varRef/ancestor::*/field |
    $varRef/ancestor::catch/formal-argument |
    $varRef/ancestor::*/formal-arguments/formal-argument |
    $varRef/ancestor::*/loop/init/local-variable |
    ( $varRef/preceding::* intersect
      ( $varRef/ancestor::*/local-variable ) )
  return exists( $declsInScope[ @name = $varRef/@name ] )
}

declare function local:parseName( $typeName as xs:string ) as xs:string*
{
  let $names := tokenize( $typeName, "\." )
  return (
    string-join( $names[ not( position() = last() ) ], "." ),
    $names[ last() ]
  )
}

<static-accesses>
{
  for $fieldAccess in document( "DotTest.jmlb" )//*
    [ exists( self::field-access | self::field-set ) ]
    [ exists( var-ref ) ]
    [ not( local:isKeyword( @field ) ) ]
    [ not( local:isInScope( var-ref ) ) ]
  let
    $parsedName := local:parseName( $fieldAccess/var-ref/@name ),
    $fieldName := $fieldAccess/@field
  order by $parsedName[1], $parsedName[2], $fieldName
  return <access
    package-and-enclosing="{ $parsedName[1] }"
    type="{ $parsedName[2] }"
    field="{ $fieldName }"
  />
}
</static-accesses>
```

Ovaj upit ne rješava u potpunosti zadatak jer je u Badrosovom zapisu nemoguće razlikovati između imena paketa i sadržavajuće vrste (posljedica nepridržavanja načela iz 5.2). Stoga se ime paketa navodi zajedno s imenima sadržavajućih

vrsta u atributu `package-and-enclosing`.

Glavni posao, pronalaženje traženih izraza, obavlja složen XPath-ov izraz koji se oslanja i na dvije funkcije deklarirane u prologu upita. Pristup statičkom atributu identične je strukture kao i pristup instancinom atributu. Stoga je potrebna složena analiza konteksta u kojem se nalazi izraz da bi se utvrdilo njegovu prirodu (odstupanje od načela iz 5.1). Potrebno je pronaći sve identifikatore zauzete u kontekstu pojave izraza i usporediti ih s identifikatorom navedenim u elementu `var-ref`.

Glavni XPath-ov izraz počinje odabirom svih elemenata u prvom koraku. Elementi se filtriraju nizom od četiri predikata. Prvi provjerava ime elementa (može biti `field-access` ili `field-set`); drugi osigurava da postoji podelement `var-ref` u kojem se možda nalazi ime vrste. Treći predikat provjerava nije li umjesto imena atributa navedena ključna riječ `class`, `this` ili `super`, što bi značilo da se ne radi o pristupu atributu, već o izrazu *class literal*, *qualified this* odnosno *qualified super*. Ovdje se najbolje očituje posljedica nepridržavanja načela iz 5.3.

Napokon, posljednji predikat pozivom funkcije `isInScope` provjerava je li identifikator u elementu `var-ref` zauzet, što bi značilo da se zaista radi o pristupu varijabli. Ako se među zauzetim identifikatorima ne pronađe onaj iz `var-ref`, zaključuje se da se radi o imenu neke vrste. Dakle, do konačnog zaključka dolazi se negativnom selekcijom – eliminacijom svih drugih mogućnosti.

Funkcija `isInScope` kao parametar `$varRef` prima element `var-ref` analiziranog izraza. Varijabli `$declsInScope` pridjeljuje se rezultat složenog XPath-ovog izraza koji vraća niz svih elemenata koji sadrže zauzete identifikatore vidljive u kontekstu elementa `$varRef`. Čitav XPath-ov izraz je unija pet podizraza kojima je zajedničko da od `$varRef` kreću prema gore po stablu XML-ovskog dokumenta (osovina `ancestor::`). To je zato što se svaka vidljiva deklaracija mora nalaziti unutar nekog bloka koji obuhvaća `$varRef`. Prvi podizraz nalazi svaki element `field` koji je dijete nekog od predaka elementa `$varRef`. Takvi elementi sadrže vidljive deklaracije atributa. Drugi podizraz nalazi pretka po imenu `catch` i odabire njegov podelement `formal-argument` koji sadrži ime dodijeljeno uhvaćenoj iznimci. Treći podizraz nalazi pretke `method-declaration` ili `constructor-declaration`, što nije navedeno radi kratkoće jer su to jedini elementi koji mogu imati podelement `formal-arguments`. Podizraz odabire sve njegove podelemente `formal-argument`. Četvrti podizraz nalazi svakog pretka `loop` i odabire njegov eventualni element s deklaracijom lokalne varijable u inicijalizacijskom dijelu (može ga imati petlja `for`). Posljednji podizraz je najsloženiji jer radi presjek između svih čvorova-prethodnika `$varRef` (svi koji

se u XML-ovskom dokumentu pojavljuju prije njega, bez obzira na hijerarhiju) i elemenata `local-variable` koji su dijete nekog od predaka od `$varRef`. Ovo je potrebno jer se deklaracije lokalnih varijabli mogu pojavljivati i prije i nakon `$varRef`, a vidljive su samo one prije.

Za razvoj ovog upita trebalo je oko 7 sati. Vrlo je zahtjevan posao bilo samo ustanoviti koje sve konstrukcije treba ispitati kao moguće sadržitelje deklaracije koja zauzima identifikator, a koja je vidljiva iz konteksta izraza čiju prirodu treba ustanoviti. Zatim je uslijedila formulacija koda koji će obavljati tu složenu provjeru. Kao dio razvoja upita treba uračunati i osmišljavanje sadržaja ispitnog dokumenta koji će omogućiti temeljito ispitivanje koda upita. To osmišljavanje je bilo iznimno zahtjevno i dugotrajno i više puta se uočilo da još neki poseban slučaj nedostaje. Izvedba parsiranja imena vrste odnijela je oko pola sata. Napokon, znatno vrijeme utrošeno je na mukotrpno pronalaženje i ispravljanje pogreški, a ni na kraju nije jasno je li upit sasvim korektan i nije li možda izostavljen još neki slučaj u ispitnom dokumentu pa ispravnost za taj slučaj nije provjerena.

Upit nad dokumentom po Mamasovoj sintaksi

Ispis 8.8: Upit nad Mamasovom sintaksom

```
declare function local:isInScope( $name as element() ) as xs:boolean
{
  let $declsInScope :=
    $name/ancestor::Block[ exists( parent::TryStatement ) ]
      /preceding-sibling::FormalParameter[1]/VariableDeclaratorId |
    (
      $name/ancestor::*/FieldDeclaration |
      $name/ancestor::MethodDeclaration/MethodDeclarator |
      $name/ancestor::ConstructorDeclaration |
      $name/ancestor::ForStatement/ForInit/LocalVariableDeclaration |
      $name/preceding::* intersect $name/ancestor::*/LocalVariableDeclaration
    )/*/VariableDeclaratorId

  return exists( $declsInScope[ @Identifier = $name/@Identifier ] )
}

declare function local:parseName( $typeName as xs:string ) as xs:string*
{
  let $names := tokenize( $typeName, "\." )
  return (
    string-join( $names[ not( position() = last() ) ], "." ),
    $names[ last() ]
  )
}
```

```
<static-accesses>
{
  for $primaryExpr in document( "DotTest.jmlm" )//PrimaryExpression
    [ exists( PrimaryPrefix/Name ) ]
    [ exists( PrimarySuffix[1]/@Identifier ) ]
    [ not( PrimarySuffix[1]/@Identifier = "super" ) ]
    [ if ( exists( PrimarySuffix[2] ) )
      then exists( PrimarySuffix[2]/@Identifier ) else true() ]
    [ not( local:isInScope( PrimaryPrefix/Name ) ) ]
  let
    $parsedName := local:parseName( $primaryExpr/PrimaryPrefix/Name/@Identifier ),
    $fieldName := $primaryExpr/PrimarySuffix[1]/@Identifier
  order by $parsedName[1], $parsedName[2], $fieldName
  return
    <access
      package-and-enclosing="{ $parsedName[1] }"
      type="{ $parsedName[2] }"
      field="{ $fieldName }"
    />
}
</static-accesses>
```

Rješenje upita nad Mamasovom sintaksom dijeli mnogo zajedničkih svojstava s upitom nad Badrosovom sintaksom. I ovdje je nemoguće razdvojiti ime paketa od imena sadržavajućih vrsta. Funkcija `parseName` koja obavlja djelomično parsiranje identična je onoj u Badrosovom slučaju. Koncepcija rješenja također je ista: provodi se eliminacija svih nepovoljnih slučaja sve dok na kraju ne ostanu samo povoljni. U prvom koraku XPath-a odabiru se svi primarni izrazi koji se zatim filtriraju kroz pet predikata. U prvom se provjerava je li prvi dio primarnog izraza (prefiks) identifikator. U slučaju pristupa statičkom atributu, to će biti puno ime vrste. U drugom predikatu isto se provjerava za drugi dio izraza (prvi u nizu sufiksa). Tu bi trebalo biti zapisano ime statičkog atributa. U trećem predikatu provjerava se da umjesto drugog identifikatora nije navedena ključna riječ `super`, što bi značilo da se radi o izrazu *qualified super* koji treba odbaciti. U četvrtom predikatu provjerava se da je, u slučaju da postoji, i treći dio primarnog izraza identifikator – ako nije, onda čitav izraz ne predstavlja pristup statičkom atributu jer drugi dio izraza tada ne označava ime atributa. Napokon, u posljednjem, petom predikatu provjerava se je li identifikator iz prvog dijela izraza zauzet, u kojem slučaju se ne radi o imenu vrste.

Funkcija `isInScope` radi po istom načelu kao i u slučaju Badrosove sintakse, samo što traži drugačije konstrukcije. Element čiji kontekst treba ispitati spremljen je u varijablu `$name`. U cijelosti, izraz je unija između dva podizraza od kojih je prvi jednostavniji. On pronalazi elementa-pretka `Block` čiji roditelj je `TryStatement`

– dakle, radi se o jednom od blokova u složenoj izjavi `try-catch-finally`. Ispituje se neposredni prethodnik elementa `Block` i ako je to `FormalParameter`, tada navedeni `Block` predstavlja jedan od blokova `catch`, a u `FormalParameter` je zapisana deklaracija varijable koja sadrži uhvaćenu iznimku, točnije u njegovom podelementu `VariableDeclaratorId`, koji je rezultat čitavog podizraza.

Svim elementima koje vraća složeni drugi podizraz zajedničko je da od njih treba krenuti dvije razine prema dolje, do elementa `VariableDeclaratorId`. Prvi podelement razlikuje se od slučaja do slučaja pa je umjesto njega radi konciznosti stavljena zvjezdica. S druge strane, stavljanjem zvjezdice zapis je nejasniji i podložniji pogrešci jer, iako se točno zna koji podelementi se očekuju na tom mjestu, to nije navedeno eksplicitno. Problem je povezan s načelom iz 5.5. Glavni dio podizraza je unija između četiri člana. Prvi član unije odabire sve elemente `FieldDeclaration` koji su dijete nekog od predaka proslijeđenog elementa `$name`. Ti elementi sadrže vidljive deklaracije atributa. Drugi član unije nalazi sve pretke zvane `MethodDeclaration` i odabire njihov podelement `MethodDeclarator`. Tu su deklarirani parametri metoda unutar kojih se nalazi `$name`. Moguća je pojava metoda ugniježđenih jedna u drugu putem lokalne klase. Treći član unije sličan posao obavlja za sadržavajuće konstruktore. Četvrti član odabire pretke `ForStatement` i unutar njih eventualne deklaracije lokalnih varijabli u inicijalizacijskom dijelu. Peti član je najsloženiji jer se radi o presjeku između svih prethodnika od `$name` i svih elemenata `LocalVariableDeclaration` koji su dijete nekog od predaka elementa `$name` - dakle, član odabire sve vidljive deklaracije lokalnih varijabli.

Za razvoj ovog upita trebalo je oko 6 sati, prvenstveno zato što je razvijan nakon upita za Badrosovu sintaksu pa ga je bilo lakše izgraditi na tim temeljima. U protivnom bi vjerojatno trebalo i više od 7 sati (koliko je trebalo za Badrosovu sintaksu) jer je sintaksa vrlo neintuitivna pa ju je teško pratiti i shvaćati. Posebno je trebalo osmišljavati i problematične slučajeve u izvornom kodu. I ovdje ostaje nejasno je li konačan upit sasvim korektan.

8.1.3 Zadatak 3: pronalaženje svih korištenih referentnih vrsta

Zadatak: pronaći sva spominjanja referentnih vrsta (izuzevši vrste polja) u jednom dokumentu izvornog koda. Vratiti element `reference-types` s po jednim elementom `reference-type` za svako pronađeno spominjanje vrste. Element mora imati atribute `type` za ime vrste, `package` za ime njenog paketa (ako je naveden u kodu) i `enclosing`

za eventualna imena sadržavajućih vrsta odvojena točkom.

Zadatak je zamišljen da prvenstveno istakne prednosti od pridržavanja načela iz 5.4. Međutim, budući da se zadatak, kao i prethodni, bavi imenima vrsta, na upite će ponovo imati utjecaj i načela iz 5.1, 5.2 i 5.3.

Upiti se izvršavaju nad istim dokumentom izvornog koda kao u Primjeru 2. Kôd tog primjera, osim već navedenog u Primjeru 2, koncipiran je i tako da sadrži sve moguće kontekste u kojima se spominje referentna vrsta: natklasa, implementirano sučelje, najavljena iznimka itd. Također, primjer sadrži zapise koji bi mogli biti statički pristup, ali nisu jer je identifikator zauzet kao ime atributa, parametra, varijable ili uhvaćene iznimke. To je potrebno za provjeru ispravnosti upita nad Mamasovom i Badrosovom sinstaksom kod kojih se mora provjeravati zauzetost identifikatora.

Upit nad dokumentom po predloženoj sintaksi

Ispis 8.9: Upit nad predloženom sintaksom

```
declare default element namespace "http://tel.fer.hr/jezix/java";

<reference-types>
{
  for $type in document( "DotTest.jxj" )//reference-type
  let
    $enclosing := string-join( $type/enclosing-type/@name, "." ),
    $localName := $type/@name
  order by $type/@package, $enclosing, $localName
  return
    <reference-type
      package="{ $type/@package }" enclosing="{ $enclosing }"
      type="{ $localName }" />
}
</reference-types>
```

Ovdje izraz za pronalaženje zadanih podataka dosiže vrhunac jednostavnosti i bez pretjerivanja se može nazvati trivijalnim. Sve što je potrebno je naći sve elemente `reference-type` u dokumentu. Ostatak koda bavi se oblikovanjem izlaza i identičan je kao u Primjeru 2.

Oslanjajući se na već riješen Primjer 2, ovaj upit bio je gotov u manje od 5 minuta.

Upit nad dokumentom po Badrosovoj sintaksi

Ispis 8.10: Upit nad Badrosovom sintaksom

```
declare function local:isKeyword( $name as xs:string ) as xs:boolean
{
  $name = "void" or $name = "byte" or $name = "char" or $name = "short" or
  $name = "int" or $name = "long" or $name = "float" or $name = "double"
}

declare function local:isInScope( $varRef as element() ) as xs:boolean
{
  let $declsInScope :=
    $varRef/ancestor::*/field |
    $varRef/ancestor::catch/formal-argument |
    $varRef/ancestor::*/formal-arguments/formal-argument |
    $varRef/ancestor::*/loop/init/local-variable |
    ( $varRef/preceding::* intersect
      ( $varRef/ancestor::*/local-variable ) )
  return exists( $declsInScope[ @name = $varRef/@name ] )
}

declare function local:parseName( $typeName as xs:string ) as xs:string*
{
  let $names := tokenize( $typeName, "\." )
  return (
    string-join( $names[ not( position() = last() ) ], "." ),
    $names[ last() ]
  )
}

<reference-types>
{
  let $root := document( "DotTest.jmlb" )/*
  for $typeName in
    $root//superclass/@name |
    $root//implement/@interface |
    $root//throws/@exception |
    $root//import[ not( ends-with( @module, "*" ) ) ]/@module |
    $root//type
      [ not( ends-with( @name, "[]" ) ) ]
      [ not( @primitive = "true" ) ]/@name |
    $root//var-ref
      [ not( ends-with( @name, "[]" ) ) ]
      [ not( local:isKeyword( @name ) ) ]
      [ not( local:isInScope( . ) ) ]/@name
  let $parsedName := local:parseName( $typeName )
  order by $parsedName[1], $parsedName[2]
  return <reference-type
    package-and-enclosing="{ $parsedName[1] }" type="{ $parsedName[2] }" />
}
</reference-types>
```

Kao i u Primjeru 2, ovaj upit ne ispunjava u potpunosti zadatak jer je u Badrosovom zapisu nemoguće razlikovati između imena paketa i sadržavajuće vrste. Funkcije `isInScope` i `parseName` preuzete su iz Primjera 2, ali glavni XPath-ov izraz

trebalo je razvijati ispočetka i upravo u njemu se odražava posljedica odstupanja on načela iz 5.4. Trebalo je istražiti sintaksu da bi se identificirale sve moguće konstrukcije u kojima se spominje ime referentne vrste. To su redom zapis natklase, implementiranog sučelja, najavljene iznimke, uvezene vrste (treba izbaciti slučajeve koji predstavljaju uvoz čitavog paketa), zatim element `type` koji se koristi u više konteksta (treba izbaciti zapise vrste polja i primitivne vrste) te vrsta navedena kao ime varijable (također treba izbaciti zapise primitivnih vrsta i polja, a tu je i već objašnjena provjera zauzetosti identifikatora).

Za razvoj upita bilo je potrebno oko 5 sati, u što nije uračunat posao preuzet iz Primjera 2. Glavnina vremena potrošena je na spomenuto istraživanje sintakse, zatim na smišljanje svih mogućih posebnih slučajeva koje treba uvrstiti u ispitni dokument, zatim na razvoj koda upita uz postupno uvođenje elegantnijih i konciznijih zapisa i napokon na pronalaženje i ispravak pogrešaka.

Upit nad dokumentom po Mamasovoj sintaksi

Ispis 8.11: Upit nad Mamasovom sintaksom

```
declare function local:isInScope( $name as element() ) as xs:boolean
{
  let $declsInScope :=
    $name/ancestor::Block[ exists( parent::TryStatement ) ]
      /preceding-sibling::FormalParameter[1]/VariableDeclaratorId |
    (
      $name/ancestor::*/FieldDeclaration |
      $name/ancestor::MethodDeclaration/MethodDeclarator |
      $name/ancestor::ConstructorDeclaration |
      $name/ancestor::ForStatement/ForInit/LocalVariableDeclaration |
      $name/preceding::* intersect $name/ancestor::*/LocalVariableDeclaration
    )/*/VariableDeclaratorId

  return exists( $declsInScope[ @Identifier = $name/@Identifier ] )
}

declare function local:parseName( $typeName as xs:string ) as xs:string*
{
  let $names := tokenize( $typeName, "\." )
  return (
    string-join( $names[ not( position() = last() ) ], "." ),
    $names[ last() ]
  )
}

let $root := document( "DotTest.jmlm" )/*
return
<reference-types>
```

```
{
  for $nameEl in $root//Name[
    exists( parent::ConstructorDeclaration | parent::MethodDeclaration |
            parent::UnmodifiedClassDeclaration | parent::AllocationExpression ) or
    exists( parent::ImportDeclaration ) and not( ends-with( @Identifier, "*" ) ) or
    exists( parent::Type ) and not( @Identifier = "void" ) or
    exists( parent::PrimaryPrefix ) and not( local:isInScope( . ) )
  ]
  let $parsedName := local:parseName( $nameEl/@Identifier )
  order by $parsedName[1], $parsedName[2]
  return <reference-type
    package-and-enclosing="{ $parsedName[1] }" type="{ $parsedName[2] }" />
}
</reference-types>
```

Ponovo, rješenje za Mamasovu sintaksu nije potpuno zbog neraščlanjenosti imena paketa i sadržavajućih vrsta. Postupak razvoja bio je srodan onom opisanom za Badrosovu sintaksu. Iz Primjera 2 preuzete su funkcije `isInScope` i `parseName`, a glavni XPath-ov izraz, koji odabire sve pojave imena vrste, razvijan je ispočetka.

U prvom koraku izraza odabiru se svi elementi `Name` koji se filtriraju u jednom složenom predikatu. Predikat za svaki element provjerava kontekst u kojem se pojavljuje. Provjera je izvedena kao logička disjunkcija u kojoj se pobrojava sve moguće povoljne slučajeve, a iz nekih rezultata potrebno je dodatno izbaciti nepovoljne podslučajeve (u drugom članu konjunkcije). Prvi član disjunkcije odabire najjednostavnije slučajeve, one koji ne zahtijevaju dodatnu provjeru. To su deklaracija klase (element `Name` sadrži ime natklase ili implementiranog sučelja), deklaracije konstruktora i metode (`Name` sadrži ime najavljene iznimke), te izraz "`new`" kojim se instancira neka klasa. U ovom dijelu upita dolazi do izražaja i neintuitivnost sintakse – npr. podelementi `Name` elementa `UnmodifiedClassDeclaration` ne sadrže ime klase koju se deklarira već ime natklase ili implementiranog sučelja. I samo ime elementa, `UnmodifiedClassDeclaration`, neobično je i nejasno, pogotovo ako se uzme u obzir da je ono obvezan podelement elementa očekivanog imena, `ClassDeclaration`.

Drugi član disjunkcije odabire zapis uvoza vrste (treba odbaciti uvoze čitavog paketa); treći odabire element `Type` koji se koristi u više konteksta (treba odbaciti slučajeve navođenja ključne riječi `void` umjesto vrste); posljednji član odabire pojave imena vrste u primarnom izrazu (treba odbaciti slučajeve u kojima je navedeni identifikator zauzet). Ostatak koda upita bavi se oblikovanjem zapisa rezultata i identičan je kao u Primjeru 2.

Na razvoj ovog upita potrošeno oko 4 sata, ne uračunavajući posao preuzet iz Primjera 2. Posao je bio srodan onom za Badrosovu sintaksu. Trebalo je istražiti

sintaksu, osmisliti sve moguće posebne slučajeve za ispitni dokument, razviti kod upita te proći kroz proces pronalaženja i ispravka pogreški te usavršavanja zapisa.

8.1.4 Osvrt na rezultate usporedbe

Iz navedenog jasno proizlazi da je navedene zadatke znatno lakše riješiti nad predloženom sintaksom. Prirodno se nameće pitanje postoje li i takvi zadaci koje je lakše riješiti nad Badrosovom ili Mamasovom sintaksom. Jedini uočeni oblik takvog zadatka bio bi onaj koji je formuliran u terminima tradicionalnog zapisa, npr. 'Nađi sve konstrukcije čiji tradicionalni zapis glasi "`id1.id2`".' U kontekstu u kojem se taj zapis više ne koristi osim eventualno za prikaz takvi zadaci nemaju gotovo nikakvu praktičnu vrijednost. Tradicionalni zapis, ako bi se i koristio, bio bi samo jedan od mogućih načina prikaza.

Također treba istaknuti da će mnogi zadaci biti takvi da ih je podjednako lako riješiti nad bilo kojom od tri sintakse. Takvi zadaci nisu navođeni jer je ovdje cilj istaknuti razlike, a ne sličnosti između sintaksi.

Zaključak

Ključne prednosti koje primjena XML-a za zapis izvornog koda programskih jezika može donijeti u odnosu na tradicionalni zapis su bolja podrška za automatizaciju zadataka obrade i za uvođenje ortogonalnih proširenja. Da bi se ove prednosti mogle u potpunosti iskoristiti, infoset XML-ovskog dokumenta treba modelirati stablo apstraktne sintakse jer ono najizravnije opisuje značenje izvornog koda pa stoga odgovara čovjekovom misaonom modelu programa.

Već postoji stanovit broj XML-ovskih sintaksi izvornog koda raznih programskih jezika, pa tako i jezika Java, ali u svima su uočeni razni nedostaci. Glavni uzrok tih nedostataka leži u tome što se sintakse previše ravnaju prema konkretnoj sintaksi tradicionalnog zapisa izvornog koda i stoga su pravila utvrđivanja značenja određenih kodnih konstrukcija nepotrebno složena. To njihovo svojstvo ima vrlo negativan utjecaj na lakoću formulacije i konačnu preglednost upita koji trebaju pristupiti tim značenjima. Sintakse također u velikom broju slučajeva ignoriraju standard *XML Namespaces* koji je ključan za provedbu ideje ortogonalnih proširenja.

Na temelju ovih utvrđenih činjenica uočeno je da je potrebno sustavno provesti postupak koji uključuje preciznu identifikaciju **prednosti** primjene XML-a, formuliranje jasnih **zahtjeva** na sintaksu infoseta izvornog koda te sintezu **načela** kojima se treba rukovoditi u oblikovanju sintakse da bi ona ostvarila zacrtane zahtjeve i time u omogućila maksimalno iskorištavanje identificiranih prednosti XML-a.

Primarni predmet istraživanja i provođenja opisanog postupka bio je problem sintakse infoseta za programski jezik Java. Sintaksa je oblikovana tako da uspješno ostvaruje ciljeve i omogućuje efikasnu primjenu jezika za obradu infoseta i uvođenje ortogonalnih proširenja. Radi provjere uspješnosti sintakse definirano je nekoliko konkretnih ortogonalnih proširenja i jezikom XSLT je automatizirana obrada njihovog sadržaja. Pokazalo se da je uz relativno malo napora moguće riješiti prilično kompleksne zadatke. Obavljena je također i usporedba predložene

s dvije najrelevantnije postojeće sintakse iste namjene. Definirano je nekoliko zadataka automatske obrade i razvijeni su upiti u jeziku *XQuery* koji obavljaju isti zadatak za svaku od tri sintakse. Upiti nad dokumentom prema predloženoj sintaksi bili su znatno lakši za formulaciju, kraći i razumljiviji. Potraženi su točni uzroci problema u formulaciji upita nad alternativnim sintaksama i pokazalo se da oni leže u nepridržavanju istaknutih načela oblikovanja sintakse. To je najizravniji pokaz njihove valjanosti i opravdanosti.

Literatura

[Arm00] Eric Armstrong. Encoding source code in XML, a strategic analysis. http://www.treelight.com/software/encodingSource.html, 2000.

[Bad00] Greg J. Badros. JavaML: a markup language for Java source code. In *Proceedings of the 9th international World Wide Web conference on Computer networks*, pages 159–177. North-Holland Publishing Co., 2000.

[Blo02] Joshua Bloch. *Effective Java Programming Language Guide*. Addison-Wesley, 2002.

[Cho59] Noam Chomsky. On certain formal properties of grammars. *Information and Control*, 2:137–167, 1959.

[CMK04] Michael L. Collard, Jonathan I. Maletic, and Huzefa Kagdi. Leveraging XML technologies in developing program analysis tools. In *Proceedings of the 4th International Workshop on Adoption-Centric Software Engineering*, 2004.

[CMM02a] Michael L. Collard, Jonathan I. Maletic, and Andrian Marcus. Source code files as structured documents. In *Proceedings of the 10th International Workshop on Program Comprehension (IWPC'02)*, 2002.

[CMM02b] Michael L. Collard, Jonathan I. Maletic, and Andrian Marcus. Supporting document and data views of source code. In *Proceedings of the 2002 ACM symposium on Document engineering*, 2002.

[dc02] Dublin Core Metadata Element Set, Version 1.1, 2002.

[Fow99] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

[GJSB00] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java Language Specification*. Sun Microsystems, Inc., second edition, 2000.

[HMU01] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*, chapter Context-Free Grammars and Languages, pages 169–217. Addison-Wesley, 2001.

[HWG03] Anders Hejlsberg, Scott Wiltamuth, and Peter Golde. *The C# Programming Language*. Addison Wesley Professional, 2003.

[inf04] XML Information Set. http://www.w3.org/TR/2004/REC-xml-infoset-20040204, 2004.

[jcc99] Code Conventions for the Java Programming Language. http://java.sun.com/docs/codeconv, 1999.

[Kla03] Martin Klang. XML and the art of code maintenance. In *Proceedings of Extreme Markup Languages 2003*, 2003.

[Luc82] Peter Lucas. *Formal Specification and Software Development*, chapter Main Approaches to Formal Specifications, pages 11–12. Prentice-Hall, 1982.

[MK00] Evan Mamas and Kostas Kontogiannis. Towards Portable Source Code Representations Using XML. In *Proceedings of WCRE'00*, pages 172–182, 2000.

[nsp] Namespaces in XML. http://www.w3.org/TR/REC-xml-names.

[San00] Stig E. Sandø. CSF specification. http://sds.sourceforge.net/doc/csf.html, 2000.

[spx] Superx++, XML-based OO Programming. http://xplusplus.sourceforge.net.

[ST03] Hrvoje Simic and Marko Topolnik. Prospects of encoding Java in XML. In *Proceedings of ConTEL 2003*, pages 635–641, June 2003.

[WKR01] A. Winter, B. Kullbach, and V. Riediger. *An Overview of the GXL Graph Exchange Language*. Springer Verlag, 2001.

[xes] XML-encoded Source.
http://www.euclideanspace.com/software/language/xes/index.htm.

[xml04] Extensible Markup Language (XML) Version 1.1.
http://www.w3.org/TR/2004/REC-xml11-20040204/, 2004.

[xpt99] XML Path Language (XPath) Version 1.0.
http://www.w3.org/TR/xpath, 1999.

[xqr] XQuery 1.0 and XPath 2.0 Data Model.
http://www.w3.org/TR/xpath-datamodel/.

[xqx03] XML Syntax for XQuery 1.0 (XQueryX).
http://www.w3.org/TR/2003/WD-xqueryx-20031219, 2003.

[xsc01a] XML Schema Part 1: Structures.
http://www.w3.org/TR/2001/REC-xmlschema-1, 2001.

[xsc01b] XML Schema Part 2: Datatypes.
http://www.w3.org/TR/2001/REC-xmlschema-2, 2001.

[xsl99] XSL Transformations (XSLT) Version 1.0.
http://www.w3.org/TR/xslt, 1999.

[ZK01] Ying Zou and Kostas Kontogiannis. Towards a portable XML-based source code representation. In *Proceedings of XML Technologies and Software Engineering (XSE2001)*, 2001.

Dodatak A

Gramatika XML-a u EBNF-u

Napomena: leksička struktura XML-a definirana je nad abecedom definiranom standardom Unicode. Zapisi `#x...` odnose se na heksadecimalne kodove pojedinih Unicode-ovih znakova.

```
document ::= prolog element Misc*

prolog ::= XMLDecl? Misc*

XMLDecl ::= '<?xml' VersionInfo EncodingDecl? SDDecl? S? '?>'

VersionInfo ::= S 'version' Eq (' VersionNum ' | " VersionNum ")

VersionNum ::= ([a-zA-Z0-9_.:] | '-')+

EncodingDecl ::= S 'encoding' Eq ('"' EncName '"' | "'" EncName "'" )

EncName ::= [A-Za-z] ([A-Za-z0-9._] | '-')*

SDDecl ::= S 'standalone' Eq (("'" ('yes' | 'no') "'") | ('"' ('yes' |
'no') '"'))

element ::= EmptyElemTag | STag content ETag

content ::= (element | CharData | Reference | Comment)*

EmptyElemTag ::= '<' Name (S Attribute)* S? '/>'

STag ::= '<' Name (S Attribute)* S? '>'

ETag ::= '</' Name S? '>'

Attribute ::= Name Eq AttValue

AttValue ::= '"' ([^<&"] | Reference)* '"' | "'" ([^<&'] | Reference)* "'"

S ::= (#x20 | #x9 | #xD | #xA)+

Eq ::= S? '=' S?

Misc ::= Comment | S

Comment ::= '<!--' ((Char - '-') | ('-' (Char - '-')))* '-->'
```

```
Name ::= (Letter | '_' | ':') (NameChar)*

Reference ::= EntityRef | CharRef

EntityRef ::= '&' Name ';'

CharRef ::= '&#' [0-9]+ ';' | '&#x' [0-9a-fA-F]+ ';'

CharData ::= [^<&]* - ([^<&]* ']]>' [^<&]*)

Char ::= #x9 | #xA | #xD | [#x20-#xD7FF] | [#xE000-#xFFFD] | [#x10000-#x10FFFF]

Letter ::= BaseChar | Ideographic

NameChar ::= Letter | Digit | '.' | '-' | '_' | ':' | CombiningChar | Extender

BaseChar ::= [#x0041-#x005A] | [#x0061-#x007A] | [#x00C0-#x00D6] | [#x00D8-#x00F6] |
   [#x00F8-#x00FF] | [#x0100-#x0131] | [#x0134-#x013E] | [#x0141-#x0148] | [#x014A-#x017E] |
   [#x0180-#x01C3] | [#x01CD-#x01F0] | [#x01F4-#x01F5] | [#x01FA-#x0217] | [#x0250-#x02A8] |
   [#x02BB-#x02C1] | #x0386 | [#x0388-#x038A] | #x038C | [#x038E-#x03A1] | [#x03A3-#x03CE] |
   [#x03D0-#x03D6] | #x03DA | #x03DC | #x03DE | #x03E0 | [#x03E2-#x03F3] | [#x0401-#x040C] |
   [#x040E-#x044F] | [#x0451-#x045C] | [#x045E-#x0481] | [#x0490-#x04C4] | [#x04C7-#x04C8] |
   [#x04CB-#x04CC] | [#x04D0-#x04EB] | [#x04EE-#x04F5] | [#x04F8-#x04F9] | [#x0531-#x0556] | #x0559
   | [#x0561-#x0586] | [#x05D0-#x05EA] | [#x05F0-#x05F2] | [#x0621-#x063A] | [#x0641-#x064A] |
   [#x0671-#x06B7] | [#x06BA-#x06BE] | [#x06C0-#x06CE] | [#x06D0-#x06D3] | #x06D5 | [#x06E5-#x06E6]
   | [#x0905-#x0939] | #x093D | [#x0958-#x0961] | [#x0985-#x098C] | [#x098F-#x0990] |
   [#x0993-#x09A8] | [#x09AA-#x09B0] | #x09B2 | [#x09B6-#x09B9] | [#x09DC-#x09DD] | [#x09DF-#x09E1]
   | [#x09F0-#x09F1] | [#x0A05-#x0A0A] | [#x0A0F-#x0A10] | [#x0A13-#x0A28] | [#x0A2A-#x0A30] |
   [#x0A32-#x0A33] | [#x0A35-#x0A36] | [#x0A38-#x0A39] | [#x0A59-#x0A5C] | #x0A5E | [#x0A72-#x0A74]
   | [#x0A85-#x0A8B] | #x0A8D | [#x0A8F-#x0A91] | [#x0A93-#x0AA8] | [#x0AAA-#x0AB0] |
   [#x0AB2-#x0AB3] | [#x0AB5-#x0AB9] | #x0ABD | #x0AE0 | [#x0B05-#x0B0C] | [#x0B0F-#x0B10] |
   [#x0B13-#x0B28] | [#x0B2A-#x0B30] | [#x0B32-#x0B33] | [#x0B36-#x0B39] | #x0B3D | [#x0B5C-#x0B5D]
   | [#x0B5F-#x0B61] | [#x0B85-#x0B8A] | [#x0B8E-#x0B90] | [#x0B92-#x0B95] | [#x0B99-#x0B9A] |
   #x0B9C | [#x0B9E-#x0B9F] | [#x0BA3-#x0BA4] | [#x0BA8-#x0BAA] | [#x0BAE-#x0BB5] | [#x0BB7-#x0BB9]
   | [#x0C05-#x0C0C] | [#x0C0E-#x0C10] | [#x0C12-#x0C28] | [#x0C2A-#x0C33] | [#x0C35-#x0C39] |
   [#x0C60-#x0C61] | [#x0C85-#x0C8C] | [#x0C8E-#x0C90] | [#x0C92-#x0CA8] | [#x0CAA-#x0CB3] |
   [#x0CB5-#x0CB9] | #x0CDE | [#x0CE0-#x0CE1] | [#x0D05-#x0D0C] | [#x0D0E-#x0D10] | [#x0D12-#x0D28]
   | [#x0D2A-#x0D39] | [#x0D60-#x0D61] | [#x0E01-#x0E2E] | #x0E30 | [#x0E32-#x0E33] |
   [#x0E40-#x0E45] | [#x0E81-#x0E82] | #x0E84 | [#x0E87-#x0E88] | #x0E8A | #x0E8D | [#x0E94-#x0E97]
   | [#x0E99-#x0E9F] | [#x0EA1-#x0EA3] | #x0EA5 | #x0EA7 | [#x0EAA-#x0EAB] | [#x0EAD-#x0EAE] |
   #x0EB0 | [#x0EB2-#x0EB3] | #x0EBD | [#x0EC0-#x0EC4] | [#x0F40-#x0F47] | [#x0F49-#x0F69] |
   [#x10A0-#x10C5] | [#x10D0-#x10F6] | #x1100 | [#x1102-#x1103] | [#x1105-#x1107] | #x1109 |
   [#x110B-#x110C] | [#x110E-#x1112] | #x113C | #x113E | #x1140 | #x114C | #x114E | #x1150 |
   [#x1154-#x1155] | #x1159 | [#x115F-#x1161] | #x1163 | #x1165 | #x1167 | #x1169 | [#x116D-#x116E]
   | [#x1172-#x1173] | #x1175 | #x119E | #x11A8 | #x11AB | [#x11AE-#x11AF] | [#x11B7-#x11B8] |
   #x11BA | [#x11BC-#x11C2] | #x11EB | #x11F0 | #x11F9 | [#x1E00-#x1E9B] | [#x1EA0-#x1EF9] |
   [#x1F00-#x1F15] | [#x1F18-#x1F1D] | [#x1F20-#x1F45] | [#x1F48-#x1F4D] | [#x1F50-#x1F57] | #x1F59
   | #x1F5B | #x1F5D | [#x1F5F-#x1F7D] | [#x1F80-#x1FB4] | [#x1FB6-#x1FBC] | #x1FBE |
   [#x1FC2-#x1FC4] | [#x1FC6-#x1FCC] | [#x1FD0-#x1FD3] | [#x1FD6-#x1FDB] | [#x1FE0-#x1FEC] |
   [#x1FF2-#x1FF4] | [#x1FF6-#x1FFC] | #x2126 | [#x212A-#x212B] | #x212E | [#x2180-#x2182] |
   [#x3041-#x3094] | [#x30A1-#x30FA] | [#x3105-#x312C] | [#xAC00-#xD7A3]

Ideographic ::= [#x4E00-#x9FA5] | #x3007 | [#x3021-#x3029]

CombiningChar ::= [#x0300-#x0345] | [#x0360-#x0361] | [#x0483-#x0486] | [#x0591-#x05A1] |
   [#x05A3-#x05B9] | [#x05BB-#x05BD] | #x05BF | [#x05C1-#x05C2] | #x05C4 | [#x064B-#x0652] | #x0670
   | [#x06D6-#x06DC] | [#x06DD-#x06DF] | [#x06E0-#x06E4] | [#x06E7-#x06E8] | [#x06EA-#x06ED] |
   [#x0901-#x0903] | #x093C | [#x093E-#x094C] | #x094D | [#x0951-#x0954] | [#x0962-#x0963] |
   [#x0981-#x0983] | #x09BC | #x09BE | #x09BF | [#x09C0-#x09C4] | [#x09C7-#x09C8] | [#x09CB-#x09CD]
   | #x09D7 | [#x09E2-#x09E3] | #x0A02 | #x0A3C | #x0A3E | #x0A3F | [#x0A40-#x0A42] |
   [#x0A47-#x0A48] | [#x0A4B-#x0A4D] | [#x0A70-#x0A71] | [#x0A81-#x0A83] | #x0ABC | [#x0ABE-#x0AC5]
   | [#x0AC7-#x0AC9] | [#x0ACB-#x0ACD] | [#x0B01-#x0B03] | #x0B3C | [#x0B3E-#x0B43] |
   [#x0B47-#x0B48] | [#x0B4B-#x0B4D] | [#x0B56-#x0B57] | [#x0B82-#x0B83] | [#x0BBE-#x0BC2] |
   [#x0BC6-#x0BC8] | [#x0BCA-#x0BCD] | #x0BD7 | [#x0C01-#x0C03] | [#x0C3E-#x0C44] | [#x0C46-#x0C48]
   | [#x0C4A-#x0C4D] | [#x0C55-#x0C56] | [#x0C82-#x0C83] | [#x0CBE-#x0CC4] | [#x0CC6-#x0CC8] |
   [#x0CCA-#x0CCD] | [#x0CD5-#x0CD6] | [#x0D02-#x0D03] | [#x0D3E-#x0D43] | [#x0D46-#x0D48] |
```

```
   [#x0D4A-#x0D4D] | #x0D57 | #x0E31 | [#x0E34-#x0E3A] | [#x0E47-#x0E4E] | #x0EB1 | [#x0EB4-#x0EB9]
   | [#x0EBB-#x0EBC] | [#x0EC8-#x0ECD] | [#x0F18-#x0F19] | #x0F35 | #x0F37 | #x0F39 | #x0F3E |
   #x0F3F | [#x0F71-#x0F84] | [#x0F86-#x0F8B] | [#x0F90-#x0F95] | #x0F97 | [#x0F99-#x0FAD] |
   [#x0FB1-#x0FB7] | #x0FB9 | [#x20D0-#x20DC] | #x20E1 | [#x302A-#x302F] | #x3099 | #x309A

Digit ::= [#x0030-#x0039] | [#x0660-#x0669] | [#x06F0-#x06F9] |
   [#x0966-#x096F] | [#x09E6-#x09EF] | [#x0A66-#x0A6F] | [#x0AE6-#x0AEF] | [#x0B66-#x0B6F] |
   [#x0BE7-#x0BEF] | [#x0C66-#x0C6F] | [#x0CE6-#x0CEF] | [#x0D66-#x0D6F] | [#x0E50-#x0E59] |
   [#x0ED0-#x0ED9] | [#x0F20-#x0F29]

Extender ::= #x00B7 | #x02D0 | #x02D1 | #x0387 | #x0640 | #x0E46
   | #x0EC6 | #x3005 | [#x3031-#x3035] | [#x309D-#x309E] | [#x30FC-#x30FE]
```

Dodatak B

Zadaci: dokumenti izvornog koda

NAPOMENA: u ispisu dokumenata i kasnije rezultata upita po Mamasovoj sintaksi zbog velikog broja razina ugniježđenja korištene su neke znakovne kratice. Prvi primjerak zapisa izraza (`Expression`) ostavljen je u punom obliku kao podsjetnik na pravi izgled konstrukcije, a svi sljedeći izostavljaju ponavljajući niz otvarajućih i zatvarajućih oznaka, što je istaknuto nizovima dvotočki (po tri dvotočke u dva retka). Nadalje, zbog predubokog uvlačenja dijelove ispisa trebalo je prikazati sa smanjenim uvlačenjem. Kose strelice udesno označavaju prelazak na dio s još dubljom razinom ugnježđavanja koja se prikazuje s manjim uvlačenjem, a kose strelice ulijevo označavaju povratak na normalno uvlačenje.

B.1 Zadatak 1

Ispis B.1: Izvorni kod po predloženoj sitaksi, OrExpressionTest.jxj

```
<?xml version="1.0" encoding="UTF-8"?>
<compilation-unit package="jezix.srctest" xmlns="http://tel.fer.hr/jezix/java">
  <class name="OrExpressionTest" access="public">
    <constructor access="public">
      <body>
        <super-constructor-invocation/>
        <variable name="b">
          <primitive-type name="boolean"/>
        </variable>
        <assignment>
          <variable-access name="b"/>
          <conditional-and>
            <boolean value="true"/>
            <boolean value="true"/>
            <boolean value="true"/>
          </conditional-and>
        </assignment>
        <assignment>
          <variable-access name="b"/>
          <conditional-and>
            <boolean value="true"/>
```

```
    <boolean value="true"/>
    <boolean value="true"/>
    <boolean value="true"/>
    <boolean value="true"/>
  </conditional-and>
</assignment>
<assignment>
  <variable-access name="b"/>
  <conditional-and>
    <boolean value="true"/>
    <boolean value="true"/>
    <boolean value="true"/>
    <greater-than>
      <int value="1"/>
      <int value="1"/>
    </greater-than>
  </conditional-and>
</assignment>
<assignment>
  <variable-access name="b"/>
  <conditional-and>
    <boolean value="true"/>
    <boolean value="true"/>
    <boolean value="true"/>
    <not-equal>
      <int value="1"/>
      <int value="1"/>
    </not-equal>
  </conditional-and>
</assignment>
<assignment>
  <variable-access name="b"/>
  <conditional-and>
    <boolean value="true"/>
    <boolean value="true"/>
    <boolean value="true"/>
    <and>
      <boolean value="true"/>
      <boolean value="true"/>
    </and>
  </conditional-and>
</assignment>
<assignment>
  <variable-access name="b"/>
  <conditional-and>
    <boolean value="true"/>
    <boolean value="true"/>
    <boolean value="true"/>
    <or>
      <boolean value="true"/>
      <boolean value="true"/>
    </or>
  </conditional-and>
</assignment>
<assignment>
  <variable-access name="b"/>
  <conditional-and>
    <boolean value="true"/>
    <boolean value="true"/>
    <boolean value="true"/>
    <conditional-or>
      <boolean value="true"/>
      <boolean value="true"/>
    </conditional-or>
  </conditional-and>
</assignment>
```

```
        <assignment>
          <variable-access name="b"/>
          <conditional-and>
            <boolean value="true"/>
            <boolean value="true"/>
            <boolean value="true"/>
            <conditional-or>
              <boolean value="true"/>
              <boolean value="true"/>
            </conditional-or>
            <boolean value="true"/>
          </conditional-and>
        </assignment>
        <assignment>
          <variable-access name="b"/>
          <conditional-and>
            <boolean value="true"/>
            <boolean value="true"/>
            <boolean value="true"/>
            <conditional-or>
              <boolean value="false"/>
              <boolean value="false"/>
              <boolean value="false"/>
            </conditional-or>
            <boolean value="true"/>
          </conditional-and>
        </assignment>
        <assignment>
          <variable-access name="b"/>
          <conditional-and>
            <boolean value="true"/>
            <boolean value="true"/>
            <boolean value="true"/>
            <boolean value="true"/>
            <conditional-or>
              <boolean value="false"/>
              <boolean value="false"/>
            </conditional-or>
          </conditional-and>
        </assignment>
        <assignment>
          <variable-access name="b"/>
          <conditional-and>
            <boolean value="true"/>
            <boolean value="true"/>
            <boolean value="true"/>
            <boolean value="true"/>
            <conditional-or>
              <boolean value="false"/>
              <boolean value="false"/>
            </conditional-or>
            <boolean value="true"/>
          </conditional-and>
        </assignment>
      </body>
    </constructor>
  </class>
</compilation-unit>
```

Ispis B.2: Izvorni kod po Badrosovoj sintaksi, OrExpressionTest.jmlb

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE java-source-program SYSTEM "java-ml.dtd">
<java-source-program>
```

```
<java-class-file name="AdditiveExpressionTest.java">
  <package-decl name="jezix.srctest"/>
  <class name="AdditiveExpressionTest" visibility="public">
    <constructor visibility="public" name="AdditiveExpressionTest" id="p">
      <formal-arguments/>
      <super-call>
        <arguments/>
      </super-call>
      <block>
        <local-variable name="b" id="a">
          <type name="boolean" primitive="true"/>
        </local-variable>
        <assignment-expr op="=">
          <lvalue>
            <var-set name="b"/>
          </lvalue>
          <binary-expr op="&amp;&amp;">
            <binary-expr op="&amp;&amp;">
              <literal-boolean value="true"/>
              <literal-boolean value="true"/>
            </binary-expr>
            <literal-boolean value="true"/>
          </binary-expr>
        </assignment-expr>
        <assignment-expr op="=">
          <lvalue>
            <var-set name="b"/>
          </lvalue>
          <binary-expr op="&amp;&amp;">
            <binary-expr op="&amp;&amp;">
              <binary-expr op="&amp;&amp;">
                <binary-expr op="&amp;&amp;">
                  <literal-boolean value="true"/>
                  <literal-boolean value="true"/>
                </binary-expr>
                <literal-boolean value="true"/>
              </binary-expr>
              <literal-boolean value="true"/>
            </binary-expr>
            <literal-boolean value="true"/>
          </binary-expr>
        </assignment-expr>
        <assignment-expr op="=">
          <lvalue>
            <var-set name="b"/>
          </lvalue>
          <binary-expr op="&amp;&amp;">
            <binary-expr op="&amp;&amp;">
              <binary-expr op="&amp;&amp;">
                <literal-boolean value="true"/>
                <literal-boolean value="true"/>
              </binary-expr>
              <literal-boolean value="true"/>
            </binary-expr>
            <binary-expr op=">">
              <literal-number value="1" kind="integer"/>
              <literal-number value="1" kind="integer"/>
            </binary-expr>
          </binary-expr>
        </assignment-expr>
        <assignment-expr op="=">
          <lvalue>
            <var-set name="b"/>
          </lvalue>
          <binary-expr op="&amp;&amp;">
            <binary-expr op="&amp;&amp;">
```

```
      <binary-expr op="&amp;&amp;">
        <literal-boolean value="true"/>
        <literal-boolean value="true"/>
      </binary-expr>
      <literal-boolean value="true"/>
    </binary-expr>
    <binary-expr op="!=">
      <literal-number value="1" kind="integer"/>
      <literal-number value="1" kind="integer"/>
    </binary-expr>
  </binary-expr>
</assignment-expr>
<assignment-expr op="=">
  <lvalue>
    <var-set name="b"/>
  </lvalue>
  <binary-expr op="&amp;&amp;">
    <binary-expr op="&amp;&amp;">
      <binary-expr op="&amp;&amp;">
        <literal-boolean value="true"/>
        <literal-boolean value="true"/>
      </binary-expr>
      <literal-boolean value="true"/>
    </binary-expr>
    <binary-expr op="&amp;">
      <literal-boolean value="true"/>
      <literal-boolean value="true"/>
    </binary-expr>
  </binary-expr>
</assignment-expr>
<assignment-expr op="=">
  <lvalue>
    <var-set name="b"/>
  </lvalue>
  <binary-expr op="&amp;&amp;">
    <binary-expr op="&amp;&amp;">
      <binary-expr op="&amp;&amp;">
        <literal-boolean value="true"/>
        <literal-boolean value="true"/>
      </binary-expr>
      <literal-boolean value="true"/>
    </binary-expr>
    <binary-expr op="|">
      <literal-boolean value="true"/>
      <literal-boolean value="true"/>
    </binary-expr>
  </binary-expr>
</assignment-expr>
<assignment-expr op="=">
  <lvalue>
    <var-set name="b"/>
  </lvalue>
  <binary-expr op="&amp;&amp;">
    <binary-expr op="&amp;&amp;">
      <binary-expr op="&amp;&amp;">
        <literal-boolean value="true"/>
        <literal-boolean value="true"/>
      </binary-expr>
      <literal-boolean value="true"/>
    </binary-expr>
    <paren>
      <binary-expr op="||">
        <literal-boolean value="true"/>
        <literal-boolean value="true"/>
      </binary-expr>
    </paren>
```

```
  </binary-expr>
</assignment-expr>
<assignment-expr op="=">
  <lvalue>
    <var-set name="b"/>
  </lvalue>
  <binary-expr op="&amp;&amp;">
    <binary-expr op="&amp;&amp;">
      <binary-expr op="&amp;&amp;">
        <binary-expr op="&amp;&amp;">
          <literal-boolean value="true"/>
          <literal-boolean value="true"/>
        </binary-expr>
        <literal-boolean value="true"/>
      </binary-expr>
      <paren>
        <binary-expr op="||">
          <literal-boolean value="true"/>
          <literal-boolean value="true"/>
        </binary-expr>
      </paren>
    </binary-expr>
    <literal-boolean value="true"/>
  </binary-expr>
</assignment-expr>
<assignment-expr op="=">
  <lvalue>
    <var-set name="b"/>
  </lvalue>
  <binary-expr op="&amp;&amp;">
    <binary-expr op="&amp;&amp;">
      <binary-expr op="&amp;&amp;">
        <binary-expr op="&amp;&amp;">
          <literal-boolean value="true"/>
          <literal-boolean value="true"/>
        </binary-expr>
        <literal-boolean value="true"/>
      </binary-expr>
      <paren>
        <binary-expr op="||">
          <binary-expr op="||">
            <literal-boolean value="false"/>
            <literal-boolean value="false"/>
          </binary-expr>
          <literal-boolean value="true"/>
        </binary-expr>
      </paren>
    </binary-expr>
    <literal-boolean value="true"/>
  </binary-expr>
</assignment-expr>
<assignment-expr op="=">
  <lvalue>
    <var-set name="b"/>
  </lvalue>
  <binary-expr op="&amp;&amp;">
    <binary-expr op="&amp;&amp;">
      <binary-expr op="&amp;&amp;">
        <binary-expr op="&amp;&amp;">
          <literal-boolean value="true"/>
          <literal-boolean value="true"/>
        </binary-expr>
        <literal-boolean value="true"/>
      </binary-expr>
      <literal-boolean value="true"/>
    </binary-expr>
```

```
            <paren>
              <binary-expr op="||">
                <literal-boolean value="false"/>
                <literal-boolean value="false"/>
              </binary-expr>
            </paren>
          </binary-expr>
        </assignment-expr>
        <assignment-expr op="=">
          <lvalue>
            <var-set name="b"/>
          </lvalue>
          <binary-expr op="&amp;&amp;">
            <binary-expr op="&amp;&amp;">
              <binary-expr op="&amp;&amp;">
                <binary-expr op="&amp;&amp;">
                  <binary-expr op="&amp;&amp;">
                    <literal-boolean value="true"/>
                    <literal-boolean value="true"/>
                  </binary-expr>
                  <literal-boolean value="true"/>
                </binary-expr>
                <literal-boolean value="true"/>
              </binary-expr>
              <paren>
                <binary-expr op="||">
                  <literal-boolean value="false"/>
                  <literal-boolean value="false"/>
                </binary-expr>
              </paren>
            </binary-expr>
            <literal-boolean value="true"/>
          </binary-expr>
        </assignment-expr>
      </block>
    </constructor>
  </class>
 </java-class-file>
</java-source-program>
```

Ispis B.3: Izvorni kod po Mamasovoj sintaksi, OrExpressionTest.jmlm

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE CompilationUnit SYSTEM "javaml.dtd">
<CompilationUnit>
  <TypeDeclaration>
    <ClassDeclaration isPublic="True">
      <UnmodifiedClassDeclaration Identifier="OrExpressionTest">
        <ClassBody>
          <ConstructorDeclaration Identifier="OrExpressionTest" isPublic="True">
            <ExplicitConstructorInvocation Type="super">
              <Arguments/>
            </ExplicitConstructorInvocation>
            <Block>
              <LocalVariableDeclaration>
                <Type ArraySize="0">
                  <PrimitiveType Type="boolean"/>
                </Type>
                <VariableDeclarator>
                  <VariableDeclaratorId ArraySize="0" Identifier="b"/>
                </VariableDeclarator>
              </LocalVariableDeclaration>
              <StatementExpression>
                <PrimaryExpression>
```

```
  <PrimaryPrefix>
    <Name Identifier="b"/>
  </PrimaryPrefix>
</PrimaryExpression>
<AssignmentOperator Type="Simple"/>
<Expression>
  <ConditionalExpression>
    <ConditionalOrExpression>
      <ConditionalAndExpression>
        <InclusiveOrExpression>
          <ExclusiveOrExpression>
            <AndExpression>
              <EqualityExpression>
                <InstanceOfExpression>
                  <RelationalExpression>
                    <ShiftExpression>
                      <AdditiveExpression>
                        <MultiplicativeExpression>
                          <UnaryExpression>
                            <UnaryExpressionNotPlusMinus>
                              <PostfixExpression>
                                <PrimaryExpression>
                                  <PrimaryPrefix>
                                    <Literal Value="true" Type="Boolean"/>
                                  </PrimaryPrefix>
                                </PrimaryExpression>
                              </PostfixExpression>
                            </UnaryExpressionNotPlusMinus>
                          </UnaryExpression>
                        </MultiplicativeExpression>
                      </AdditiveExpression>
                    </ShiftExpression>
                  </RelationalExpression>
                </InstanceOfExpression>
              </EqualityExpression>
            </AndExpression>
          </ExclusiveOrExpression>
        </InclusiveOrExpression>
        <InclusiveOrExpression>
          <ExclusiveOrExpression>
            <AndExpression>
              <EqualityExpression>
                <InstanceOfExpression>
                  <RelationalExpression>
                    <ShiftExpression>
                      <AdditiveExpression>
                        <MultiplicativeExpression>
                          <UnaryExpression>
                            <UnaryExpressionNotPlusMinus>
                              <PostfixExpression>
                                <PrimaryExpression>
                                  <PrimaryPrefix>
                                    <Literal Value="true" Type="Boolean"/>
                                  </PrimaryPrefix>
                                </PrimaryExpression>
                              </PostfixExpression>
                            </UnaryExpressionNotPlusMinus>
                          </UnaryExpression>
                        </MultiplicativeExpression>
                      </AdditiveExpression>
                    </ShiftExpression>
                  </RelationalExpression>
                </InstanceOfExpression>
              </EqualityExpression>
            </AndExpression>
          </ExclusiveOrExpression>
```

```
            </InclusiveOrExpression>
            <InclusiveOrExpression>
              <ExclusiveOrExpression>
                <AndExpression>
                  <EqualityExpression>
                    <InstanceOfExpression>
                      <RelationalExpression>
                        <ShiftExpression>
                          <AdditiveExpression>
                            <MultiplicativeExpression>
                              <UnaryExpression>
                                <UnaryExpressionNotPlusMinus>
                                  <PostfixExpression>
                                    <PrimaryExpression>
                                      <PrimaryPrefix>
                                        <Literal Value="true" Type="Boolean"/>
                                      </PrimaryPrefix>
                                    </PrimaryExpression>
                                  </PostfixExpression>
                                </UnaryExpressionNotPlusMinus>
                              </UnaryExpression>
                            </MultiplicativeExpression>
                          </AdditiveExpression>
                        </ShiftExpression>
                      </RelationalExpression>
                    </InstanceOfExpression>
                  </EqualityExpression>
                </AndExpression>
              </ExclusiveOrExpression>
            </InclusiveOrExpression>
          </ConditionalAndExpression>
        </ConditionalOrExpression>
      </ConditionalExpression>
    </Expression>
  </StatementExpression>
  <StatementExpression>
    <PrimaryExpression>
      <PrimaryPrefix>
        <Name Identifier="b"/>
      </PrimaryPrefix>
    </PrimaryExpression>
    <AssignmentOperator Type="Simple"/>
    <Expression>
      <ConditionalExpression>
        <ConditionalOrExpression>
          <ConditionalAndExpression>
            <InclusiveOrExpression>
                    :::
                              :::
                                   <PrimaryExpression>
                                     <PrimaryPrefix>
                                       <Literal Value="true" Type="Boolean"/>
                                     </PrimaryPrefix>
                                   </PrimaryExpression>
                              :::
                    :::
            </InclusiveOrExpression>
            <InclusiveOrExpression>
                    :::
                              :::
                                 <PostfixExpression>
                                   <PrimaryExpression>
                                     <PrimaryPrefix>
                                       <Literal Value="true" Type="Boolean"/>
                                     </PrimaryPrefix>
                                   </PrimaryExpression>
```

```
                             :::
                   :::
          </InclusiveOrExpression>
          <InclusiveOrExpression>
                   :::
                             :::
                                  <PrimaryExpression>
                                    <PrimaryPrefix>
                                      <Literal Value="true" Type="Boolean"/>
                                    </PrimaryPrefix>
                                  </PrimaryExpression>
                             :::
                   :::
          </InclusiveOrExpression>
          <InclusiveOrExpression>
                   :::
                             :::
                                  <PrimaryExpression>
                                    <PrimaryPrefix>
                                      <Literal Value="true" Type="Boolean"/>
                                    </PrimaryPrefix>
                                  </PrimaryExpression>
                             :::
                   :::
          </InclusiveOrExpression>
          <InclusiveOrExpression>
                   :::
                             :::
                                  <PrimaryExpression>
                                    <PrimaryPrefix>
                                      <Literal Value="true" Type="Boolean"/>
                                    </PrimaryPrefix>
                                  </PrimaryExpression>
                             :::
                   :::
          </InclusiveOrExpression>
        </ConditionalAndExpression>
      </ConditionalOrExpression>
    </ConditionalExpression>
  </Expression>
</StatementExpression>
<StatementExpression>
  <PrimaryExpression>
    <PrimaryPrefix>
      <Name Identifier="b"/>
    </PrimaryPrefix>
  </PrimaryExpression>
  <AssignmentOperator Type="Simple"/>
  <Expression>
    <ConditionalExpression>
      <ConditionalOrExpression>
        <ConditionalAndExpression>
          <InclusiveOrExpression>
                   :::
                             :::
                                  <PrimaryExpression>
                                    <PrimaryPrefix>
                                      <Literal Value="true" Type="Boolean"/>
                                    </PrimaryPrefix>
                                  </PrimaryExpression>
                             :::
                   :::
          </InclusiveOrExpression>
          <InclusiveOrExpression>
                   :::
                             :::
```

```
                                    <PrimaryExpression>
                                      <PrimaryPrefix>
                                        <Literal Value="true" Type="Boolean"/>
                                      </PrimaryPrefix>
                                    </PrimaryExpression>
                               :::
                         :::
                  </InclusiveOrExpression>
                  <InclusiveOrExpression>
                         :::
                               :::
                                    <PrimaryExpression>
                                      <PrimaryPrefix>
                                        <Literal Value="true" Type="Boolean"/>
                                      </PrimaryPrefix>
                                    </PrimaryExpression>
                               :::
                         :::
                  </InclusiveOrExpression>
                  <InclusiveOrExpression>
                    <ExclusiveOrExpression>
                      <AndExpression>
                        <EqualityExpression>
                          <InstanceOfExpression>
                            <RelationalExpression>
                              <ShiftExpression>
                                   :::
                                         :::
                                    <PrimaryExpression>
                                      <PrimaryPrefix>
                                        <Literal Value="1" Type="Integer"/>
                                      </PrimaryPrefix>
                                    </PrimaryExpression>
                                     :::
                                   :::
                              </ShiftExpression>
                              <RelationalOperator Type="Greater"/>
                              <ShiftExpression>
                                   :::
                                         :::
                                    <PrimaryExpression>
                                      <PrimaryPrefix>
                                        <Literal Value="1" Type="Integer"/>
                                      </PrimaryPrefix>
                                    </PrimaryExpression>
                                       :::
                                    :::
                              </ShiftExpression>
                            </RelationalExpression>
                          </InstanceOfExpression>
                        </EqualityExpression>
                      </AndExpression>
                    </ExclusiveOrExpression>
                  </InclusiveOrExpression>
                </ConditionalAndExpression>
              </ConditionalOrExpression>
            </ConditionalExpression>
          </Expression>
        </StatementExpression>
        <StatementExpression>
          <PrimaryExpression>
            <PrimaryPrefix>
              <Name Identifier="b"/>
            </PrimaryPrefix>
          </PrimaryExpression>
          <AssignmentOperator Type="Simple"/>
```

```
<Expression>
  <ConditionalExpression>
    <ConditionalOrExpression>
      <ConditionalAndExpression>
        <InclusiveOrExpression>
             :::
                        :::
                               <PrimaryExpression>
                                 <PrimaryPrefix>
                                   <Literal Value="true" Type="Boolean"/>
                                 </PrimaryPrefix>
                               </PrimaryExpression>
                        :::
             :::
        </InclusiveOrExpression>
        <InclusiveOrExpression>
             :::
                        :::
                               <PrimaryExpression>
                                 <PrimaryPrefix>
                                   <Literal Value="true" Type="Boolean"/>
                                 </PrimaryPrefix>
                               </PrimaryExpression>
                          :::
                  :::
        </InclusiveOrExpression>
        <InclusiveOrExpression>
             :::
                        :::
                               <PrimaryExpression>
                                 <PrimaryPrefix>
                                   <Literal Value="true" Type="Boolean"/>
                                 </PrimaryPrefix>
                               </PrimaryExpression>
                          :::
                   :::
        </InclusiveOrExpression>
        <InclusiveOrExpression>
          <ExclusiveOrExpression>
            <AndExpression>
              <EqualityExpression>
                <InstanceOfExpression>
                     :::
                                 :::
                               <PrimaryExpression>
                                 <PrimaryPrefix>
                                   <Literal Value="1" Type="Integer"/>
                                 </PrimaryPrefix>
                               </PrimaryExpression>
                                :::
                     :::
                <InstanceOfExpression>
                <EqualityOperator Type="NotEqual"/>
                <InstanceOfExpression>
                   :::
                           :::
                               <PrimaryExpression>
                                 <PrimaryPrefix>
                                   <Literal Value="1" Type="Integer"/>
                                 </PrimaryPrefix>
                               </PrimaryExpression>
                            :::
                       :::
                <InstanceOfExpression>
              </EqualityExpression>
            </AndExpression>
```

```
          </ExclusiveOrExpression>
        </InclusiveOrExpression>
      </ConditionalAndExpression>
    </ConditionalOrExpression>
  </ConditionalExpression>
 </Expression>
</StatementExpression>
<StatementExpression>
 <PrimaryExpression>
  <PrimaryPrefix>
   <Name Identifier="b"/>
  </PrimaryPrefix>
 </PrimaryExpression>
 <AssignmentOperator Type="Simple"/>
 <Expression>
            :::
                       :::
                              <PrimaryExpression>
                                <PrimaryPrefix>
                                  <Literal Value="true" Type="Boolean"/>
                                </PrimaryPrefix>
                              </PrimaryExpression>
                         :::
                  :::
        </InclusiveOrExpression>
        <InclusiveOrExpression>
                  :::
                            :::
                                 <PrimaryExpression>
                                   <PrimaryPrefix>
                                     <Literal Value="true" Type="Boolean"/>
                                   </PrimaryPrefix>
                                 </PrimaryExpression>
                            :::
                  :::
        </InclusiveOrExpression>
        <InclusiveOrExpression>
             :::
                       :::
                              <PrimaryExpression>
                                <PrimaryPrefix>
                                  <Literal Value="true" Type="Boolean"/>
                                </PrimaryPrefix>
                              </PrimaryExpression>
                         :::
                  :::
        </InclusiveOrExpression>
        <InclusiveOrExpression>
          <ExclusiveOrExpression>
            <AndExpression>
              <EqualityExpression>
                   :::
                                 :::
                                 <PrimaryExpression>
                                   <PrimaryPrefix>
                                     <Literal Value="true" Type="Boolean"/>
                                   </PrimaryPrefix>
                                 </PrimaryExpression>
                            :::
                     :::
              </EqualityExpression>
              <EqualityExpression>
                   :::
                                 :::
                                 <PrimaryExpression>
                                   <PrimaryPrefix>
```

```
                                        <Literal Value="true" Type="Boolean"/>
                                      </PrimaryPrefix>
                                    </PrimaryExpression>
                              :::
                    :::
          </InclusiveOrExpression>
        </ConditionalAndExpression>
      </ConditionalOrExpression>
    </ConditionalExpression>
  </Expression>
</StatementExpression>
<StatementExpression>
  <PrimaryExpression>
    <PrimaryPrefix>
      <Name Identifier="b"/>
    </PrimaryPrefix>
  </PrimaryExpression>
  <AssignmentOperator Type="Simple"/>
  <Expression>
    <ConditionalExpression>
      <ConditionalOrExpression>
        <ConditionalAndExpression>
          <InclusiveOrExpression>
                    :::
                             :::
                                    <PrimaryExpression>
                                      <PrimaryPrefix>
                                        <Literal Value="true" Type="Boolean"/>
                                      </PrimaryPrefix>
                                    </PrimaryExpression>
                              :::
                    :::
          </InclusiveOrExpression>
          <InclusiveOrExpression>
                    :::
                             :::
                                    <PrimaryExpression>
                                      <PrimaryPrefix>
                                        <Literal Value="true" Type="Boolean"/>
                                      </PrimaryPrefix>
                                    </PrimaryExpression>
                              :::
                    :::
          </InclusiveOrExpression>
          <InclusiveOrExpression>
                    :::
                             :::
                                    <PrimaryExpression>
                                      <PrimaryPrefix>
                                        <Literal Value="true" Type="Boolean"/>
                                      </PrimaryPrefix>
                                    </PrimaryExpression>
                              :::
                    :::
          </InclusiveOrExpression>
          <InclusiveOrExpression>
             <ExclusiveOrExpression>
                     :::
                              :::
                                    <PrimaryExpression>
                                      <PrimaryPrefix>
                                        <Literal Value="true" Type="Boolean"/>
                                      </PrimaryPrefix>
                                    </PrimaryExpression>
                              :::
                    :::
```

```
                </ExclusiveOrExpression>
                <ExclusiveOrExpression>
                        :::
                                 :::
                                      <PrimaryExpression>
                                        <PrimaryPrefix>
                                          <Literal Value="true" Type="Boolean"/>
                                        </PrimaryPrefix>
                                      </PrimaryExpression>
                                 :::
                        :::
                </ExclusiveOrExpression>
              </InclusiveOrExpression>
            </ConditionalAndExpression>
          </ConditionalOrExpression>
        </ConditionalExpression>
      </Expression>
    </StatementExpression>
    <StatementExpression>
      <PrimaryExpression>
        <PrimaryPrefix>
          <Name Identifier="b"/>
        </PrimaryPrefix>
      </PrimaryExpression>
      <AssignmentOperator Type="Simple"/>
      <Expression>
        <ConditionalExpression>
          <ConditionalOrExpression>
            <ConditionalAndExpression>
              <InclusiveOrExpression>
                       :::
                                :::
                                      <PrimaryExpression>
                                        <PrimaryPrefix>
                                          <Literal Value="true" Type="Boolean"/>
                                        </PrimaryPrefix>
                                      </PrimaryExpression>
                                 :::
                       :::
              </InclusiveOrExpression>
              <InclusiveOrExpression>
                       :::
                                :::
                                      <PrimaryExpression>
                                        <PrimaryPrefix>
                                          <Literal Value="true" Type="Boolean"/>
                                        </PrimaryPrefix>
                                      </PrimaryExpression>
                                 :::
                       :::
              </InclusiveOrExpression>
              <InclusiveOrExpression>
                       :::
                                :::
                                      <PrimaryExpression>
                                        <PrimaryPrefix>
                                          <Literal Value="true" Type="Boolean"/>
                                        </PrimaryPrefix>
                                      </PrimaryExpression>
                                 :::
                       :::
              </InclusiveOrExpression>
              <InclusiveOrExpression>
                       :::
                                :::
                                      <PrimaryExpression>
```

```
                                                  <PrimaryPrefix>
                                                    \_
                                                      ->
\_
   ->
    <Expression>
      <ConditionalExpression>
        <ConditionalOrExpression>
          <ConditionalAndExpression>
                  :::
                           :::
                                   <PrimaryExpression>
                                     <PrimaryPrefix>
                                       <Literal Value="true" Type="Boolean"/>
                                     </PrimaryPrefix>
                                   </PrimaryExpression>
                            :::
                  :::
          </ConditionalAndExpression>
          <ConditionalAndExpression>
                  :::
                           :::
                                   <PrimaryExpression>
                                     <PrimaryPrefix>
                                       <Literal Value="true" Type="Boolean"/>
                                     </PrimaryPrefix>
                                   </PrimaryExpression>
                                 </PostfixExpression>
                               </UnaryExpressionNotPlusMinus>
                           :::
                    :::
              </ExclusiveOrExpression>
            </InclusiveOrExpression>
          </ConditionalAndExpression>
        </ConditionalOrExpression>
      </ConditionalExpression>
    </Expression>
   _/
<-
                                                      _/
                                                    <-
                                                  </PrimaryPrefix>
                                                </PrimaryExpression>
                                              </PostfixExpression>
                                            </UnaryExpressionNotPlusMinus>
                                          :::
                                 :::
                    </ConditionalOrExpression>
                  </ConditionalExpression>
                </Expression>
              </StatementExpression>
              <StatementExpression>
                <PrimaryExpression>
                  <PrimaryPrefix>
                    <Name Identifier="b"/>
                  </PrimaryPrefix>
                </PrimaryExpression>
                <AssignmentOperator Type="Simple"/>
                <Expression>
                  <ConditionalExpression>
                    <ConditionalOrExpression>
                      <ConditionalAndExpression>
                        <InclusiveOrExpression>
                               :::
                                          :::
                                                <PrimaryExpression>
```

```
                                        <PrimaryPrefix>
                                          <Literal Value="true" Type="Boolean"/>
                                        </PrimaryPrefix>
                                      </PrimaryExpression>
                                :::
                          :::
                      </InclusiveOrExpression>
                      <InclusiveOrExpression>
                          :::
                                :::
                                      <PrimaryExpression>
                                        <PrimaryPrefix>
                                          <Literal Value="true" Type="Boolean"/>
                                        </PrimaryPrefix>
                                      </PrimaryExpression>
                                :::
                          :::
                      </InclusiveOrExpression>
                      <InclusiveOrExpression>
                          :::
                                :::
                                      <PrimaryExpression>
                                        <PrimaryPrefix>
                                          <Literal Value="true" Type="Boolean"/>
                                        </PrimaryPrefix>
                                      </PrimaryExpression>
                                :::
                          :::
                      </InclusiveOrExpression>
                      <InclusiveOrExpression>
                          :::
                                :::
                                      <PrimaryExpression>
                                        <PrimaryPrefix>
                                          \_
                                             ->
\_
   ->
    <Expression>
      <ConditionalExpression>
        <ConditionalOrExpression>
          <ConditionalAndExpression>
                 :::
                           :::
                                  <PrimaryExpression>
                                    <PrimaryPrefix>
                                      <Literal Value="true" Type="Boolean"/>
                                    </PrimaryPrefix>
                                  </PrimaryExpression>
                             :::
                  :::
          </ConditionalAndExpression>
          <ConditionalAndExpression>
                 :::
                           :::
                                  <PrimaryExpression>
                                    <PrimaryPrefix>
                                      <Literal Value="true" Type="Boolean"/>
                                    </PrimaryPrefix>
                                  </PrimaryExpression>
                                </PostfixExpression>
                              </UnaryExpressionNotPlusMinus>
                          :::
                  :::
              </ExclusiveOrExpression>
            </InclusiveOrExpression>
```

```
          </ConditionalAndExpression>
        </ConditionalOrExpression>
      </ConditionalExpression>
    </Expression>
   _/
<-
                                                  _/
                                                <-
                                              </PrimaryPrefix>
                                            </PrimaryExpression>
                                          </PostfixExpression>
                                        </UnaryExpressionNotPlusMinus>
                                  :::
                            :::
                        </InclusiveOrExpression>
                        <InclusiveOrExpression>
                               :::
                                         :::
                                                <PrimaryPrefix>
                                                  <Literal Value="true" Type="Boolean"/>
                                                </PrimaryPrefix>
                                              </PrimaryExpression>
                                           :::
                             :::
                  </Expression>
                </StatementExpression>
                <StatementExpression>
                  <PrimaryExpression>
                    <PrimaryPrefix>
                      <Name Identifier="b"/>
                    </PrimaryPrefix>
                  </PrimaryExpression>
                  <AssignmentOperator Type="Simple"/>
                  <Expression>
                    <ConditionalExpression>
                      <ConditionalOrExpression>
                        <ConditionalAndExpression>
                          <InclusiveOrExpression>
                                 :::
                                           :::
                                                 <PrimaryExpression>
                                                   <PrimaryPrefix>
                                                     <Literal Value="true" Type="Boolean"/>
                                                   </PrimaryPrefix>
                                                 </PrimaryExpression>
                                            :::
                                   :::
                          </InclusiveOrExpression>
                          <InclusiveOrExpression>
                                 :::
                                           :::
                                                 <PrimaryExpression>
                                                   <PrimaryPrefix>
                                                     <Literal Value="true" Type="Boolean"/>
                                                   </PrimaryPrefix>
                                                 </PrimaryExpression>
                                            :::
                                   :::
                          </InclusiveOrExpression>
                          <InclusiveOrExpression>
                                 :::
                                           :::
                                                 <PrimaryExpression>
                                                   <PrimaryPrefix>
                                                     <Literal Value="true" Type="Boolean"/>
                                                   </PrimaryPrefix>
```

```
                                                </PrimaryExpression>
                                           :::
                                :::
                        </InclusiveOrExpression>
                        <InclusiveOrExpression>
                               :::
                                          :::
                                                <PrimaryExpression>
                                                  <PrimaryPrefix>
                                                    \_
                                                       ->
\_
   ->
    <Expression>
      <ConditionalExpression>
        <ConditionalOrExpression>
          <ConditionalAndExpression>
                   :::
                              :::
                                    <PrimaryExpression>
                                      <PrimaryPrefix>
                                        <Literal Value="false" Type="Boolean"/>
                                      </PrimaryPrefix>
                                    </PrimaryExpression>
                              :::
                   :::
          </ConditionalAndExpression>
          <ConditionalAndExpression>
                   :::
                              :::
                                    <PrimaryExpression>
                                      <PrimaryPrefix>
                                        <Literal Value="false" Type="Boolean"/>
                                      </PrimaryPrefix>
                                    </PrimaryExpression>
                              :::
                   :::
          </ConditionalAndExpression>
          <ConditionalAndExpression>
                   :::
                              :::
                                    <PrimaryExpression>
                                      <PrimaryPrefix>
                                        <Literal Value="false" Type="Boolean"/>
                                      </PrimaryPrefix>
                                    </PrimaryExpression>
                              :::
                   :::
        </ConditionalOrExpression>
      </ConditionalExpression>
    </Expression>
   _/
<-
                                                       _/
                                                    <-
                                                  </PrimaryPrefix>
                                                </PrimaryExpression>
                                           :::
                                :::
                        </InclusiveOrExpression>
                        <InclusiveOrExpression>
                              :::
                                        :::
                                                <PrimaryExpression>
                                                  <PrimaryPrefix>
                                                    <Literal Value="true" Type="Boolean"/>
```

```
                                  </PrimaryPrefix>
                                </PrimaryExpression>
                        :::
              :::
  </Expression>
</StatementExpression>
<StatementExpression>
  <PrimaryExpression>
    <PrimaryPrefix>
      <Name Identifier="b"/>
    </PrimaryPrefix>
  </PrimaryExpression>
  <AssignmentOperator Type="Simple"/>
  <Expression>
    <ConditionalExpression>
      <ConditionalOrExpression>
        <ConditionalAndExpression>
          <InclusiveOrExpression>
                :::
                          :::
                                <PrimaryExpression>
                                  <PrimaryPrefix>
                                    <Literal Value="true" Type="Boolean"/>
                                  </PrimaryPrefix>
                                </PrimaryExpression>
                           :::
                :::
          </InclusiveOrExpression>
          <InclusiveOrExpression>
                :::
                          :::
                                <PrimaryExpression>
                                  <PrimaryPrefix>
                                    <Literal Value="true" Type="Boolean"/>
                                  </PrimaryPrefix>
                                </PrimaryExpression>
                           :::
                :::
          </InclusiveOrExpression>
          <InclusiveOrExpression>
                :::
                          :::
                                <PrimaryExpression>
                                  <PrimaryPrefix>
                                    <Literal Value="true" Type="Boolean"/>
                                  </PrimaryPrefix>
                                </PrimaryExpression>
                           :::
                :::
          </InclusiveOrExpression>
          <InclusiveOrExpression>
                :::
                          :::
                                <PrimaryExpression>
                                  <PrimaryPrefix>
                                    <Literal Value="true" Type="Boolean"/>
                                  </PrimaryPrefix>
                                </PrimaryExpression>
                           :::
                :::
          </InclusiveOrExpression>
          <InclusiveOrExpression>
                :::
                          :::
                                <PrimaryExpression>
                                  <PrimaryPrefix>
```

```
                                                \_
                                                  ->
\_
   ->
    <Expression>
      <ConditionalExpression>
        <ConditionalOrExpression>
          <ConditionalAndExpression>
                  :::
                            :::
                                    <PrimaryExpression>
                                      <PrimaryPrefix>
                                        <Literal Value="false" Type="Boolean"/>
                                      </PrimaryPrefix>
                                    </PrimaryExpression>
                             :::
                  :::
          </ConditionalAndExpression>
          <ConditionalAndExpression>
                  :::
                            :::
                                    <PrimaryExpression>
                                      <PrimaryPrefix>
                                        <Literal Value="false" Type="Boolean"/>
                                      </PrimaryPrefix>
                                    </PrimaryExpression>
                             :::
                  :::
          </ConditionalAndExpression>
        </ConditionalOrExpression>
      </ConditionalExpression>
    </Expression>
   _/
<-
                                                    _/
                                                  <-
                                                </PrimaryPrefix>
                                              </PrimaryExpression>
                                        :::
                             :::
                </Expression>
              </StatementExpression>
              <StatementExpression>
                <PrimaryExpression>
                  <PrimaryPrefix>
                    <Name Identifier="b"/>
                  </PrimaryPrefix>
                </PrimaryExpression>
                <AssignmentOperator Type="Simple"/>
                <Expression>
                  <ConditionalExpression>
                    <ConditionalOrExpression>
                      <ConditionalAndExpression>
                        <InclusiveOrExpression>
                                :::
                                          :::
                                                <PrimaryExpression>
                                                  <PrimaryPrefix>
                                                    <Literal Value="true" Type="Boolean"/>
                                                  </PrimaryPrefix>
                                                </PrimaryExpression>
                                          :::
                                :::
                        </InclusiveOrExpression>
                        <InclusiveOrExpression>
                                :::
```

```
                                           :::
                                                <PrimaryExpression>
                                                  <PrimaryPrefix>
                                                    <Literal Value="true" Type="Boolean"/>
                                                  </PrimaryPrefix>
                                                </PrimaryExpression>
                                           :::
                                :::
                        </InclusiveOrExpression>
                        <InclusiveOrExpression>
                                :::
                                           :::
                                                <PrimaryExpression>
                                                  <PrimaryPrefix>
                                                    <Literal Value="true" Type="Boolean"/>
                                                  </PrimaryPrefix>
                                                </PrimaryExpression>
                                           :::
                                :::
                        </InclusiveOrExpression>
                        <InclusiveOrExpression>
                                :::
                                           :::
                                                <PrimaryExpression>
                                                  <PrimaryPrefix>
                                                    <Literal Value="true" Type="Boolean"/>
                                                  </PrimaryPrefix>
                                                </PrimaryExpression>
                                           :::
                                :::
                        </InclusiveOrExpression>
                        <InclusiveOrExpression>
                                :::
                                           :::
                                                <PrimaryExpression>
                                                  <PrimaryPrefix>
                                                    \_
                                                       ->
\_
   ->
    <Expression>
      <ConditionalExpression>
        <ConditionalOrExpression>
          <ConditionalAndExpression>
                  :::
                             :::
                                    <PrimaryExpression>
                                      <PrimaryPrefix>
                                        <Literal Value="false" Type="Boolean"/>
                                      </PrimaryPrefix>
                                    </PrimaryExpression>
                             :::
                  :::
          </ConditionalAndExpression>
          <ConditionalAndExpression>
                  :::
                             :::
                                    <PrimaryExpression>
                                      <PrimaryPrefix>
                                        <Literal Value="false" Type="Boolean"/>
                                      </PrimaryPrefix>
                                    </PrimaryExpression>
                             :::
                  :::
          </ConditionalAndExpression>
        </ConditionalOrExpression>
```

```
      </ConditionalExpression>
    </Expression>
   _/
<-
                                                        _/
                                                    <-
                                                  </PrimaryPrefix>
                                                </PrimaryExpression>
                                             :::
                                    :::
                            </InclusiveOrExpression>
                            <InclusiveOrExpression>
                                    :::
                                             :::
                                                  <PrimaryExpression>
                                                    <PrimaryPrefix>
                                                      <Literal Value="true" Type="Boolean"/>
                                                    </PrimaryPrefix>
                                                  </PrimaryExpression>
                                             :::
                                    :::
                            </InclusiveOrExpression>
                          </ConditionalAndExpression>
                        </ConditionalOrExpression>
                      </ConditionalExpression>
                    </Expression>
                  </StatementExpression>
                </Block>
              </ConstructorDeclaration>
            </ClassBody>
          </UnmodifiedClassDeclaration>
        </ClassDeclaration>
      </TypeDeclaration>
    </CompilationUnit>
```

B.2 Zadaci 2 i 3

Ispis B.4: Izvorni kod po predloženoj sintaksi, DotTest.jxj

```
<?xml version="1.0" encoding="UTF-8"?>
<compilation-unit package="jezix.srctest" xmlns="http://tel.fer.hr/jezix/java">
  <import>
    <reference-type package="java.util" name="Set"/>
  </import>
  <import>
    <all-from package="java.io"/>
  </import>
  <class name="DotTest" access="public">
    <extends>
      <reference-type package="java.lang" name="RuntimeException"/>
    </extends>
    <implements>
      <reference-type package="java.lang" name="Cloneable"/>
    </implements>
    <field access="private" name="instanceInt">
      <primitive-type name="int"/>
    </field>
    <field access="private" static="true" name="staticInt">
      <primitive-type name="int"/>
    </field>
    <field access="private" name="instanceInner">
      <reference-type package="jezix.srctest" name="Inner1">
        <enclosing-type name="DotTest"/>
      </reference-type>
      <new-instance-of-class>
        <reference-type package="jezix.srctest" name="Inner1">
          <enclosing-type name="DotTest"/>
        </reference-type>
      </new-instance-of-class>
    </field>
    <field access="private" static="true" name="staticInner">
      <reference-type name="Inner1"/>
    </field>
    <method name="instanceMethod" access="private">
      <instance>
        <body/>
      </instance>
    </method>
    <method name="staticMethod" access="private">
      <static>
        <body/>
      </static>
    </method>
    <class name="Inner1" access="private">
      <extends>
        <reference-type name="DotTest"/>
      </extends>
      <field name="innerInstanceInt" access="private">
        <primitive-type name="int"/>
      </field>
      <method name="getMe" access="private">
        <returns>
          <reference-type name="Inner1">
            <enclosing-type name="DotTest"/>
          </reference-type>
        </returns>
        <instance>
          <body>
            <return>
              <this/>
```

```
      </return>
    </body>
  </instance>
</method>
<method name="inner1Method" access="private">
  <throws>
    <reference-type package="jezix.srctest" name="DotTest"/>
  </throws>
  <parameters>
    <variable name="Inner1">
      <reference-type package="jezix.srctest" name="DotTest"/>
    </variable>
  </parameters>
  <instance>
    <body>
      <variable name="j">
        <primitive-type name="int"/>
      </variable>
      <for>
        <initialize>
          <variables>
            <reference-type name="Inner1"/>
            <variable name="Inner2">
              <field-access name="staticInner">
                <reference-type name="DotTest"/>
              </field-access>
            </variable>
          </variables>
        </initialize>
        <loop>
          <block>
            <try>
              <block>
                <assignment>
                  <variable-access name="j"/>
                  <field-access name="instanceInt">
                    <from-superclass/>
                  </field-access>
                </assignment>
                <method-invocation name="instanceMethod">
                  <from-superclass/>
                </method-invocation>
                <assignment>
                  <variable-access name="j"/>
                  <field-access name="innerInstanceInt">
                    <variable-access name="Inner2"/>
                  </field-access>
                </assignment>
              </block>
              <catch>
                <variable name="DotTest">
                  <reference-type package="jezix.srctest" name="DotTest"/>
                </variable>
                <block>
                  <assignment>
                    <variable-access name="j"/>
                    <field-access name="instanceInt">
                      <variable-access name="DotTest"/>
                    </field-access>
                  </assignment>
                  <assignment>
                    <variable-access name="j"/>
                    <field-access name="instanceInt">
                      <variable-access name="Inner1"/>
                    </field-access>
                  </assignment>
```

```
                <assignment>
                  <variable-access name="j"/>
                  <field-access name="nestedInt">
                    <reference-type name="StaticNested"/>
                  </field-access>
                </assignment>
              </block>
            </catch>
            <catch>
              <variable name="StaticNested">
                <reference-type name="StaticNested"/>
              </variable>
              <block>
                <assignment>
                  <variable-access name="j"/>
                  <field-access name="util">
                    <variable-access name="StaticNested"/>
                  </field-access>
                </assignment>
                <assignment>
                  <variable-access name="j"/>
                  <field-access name="staticInt">
                    <reference-type name="DotTest"/>
                  </field-access>
                </assignment>
              </block>
            </catch>
          </try>
        </block>
      </loop>
    </for>
  </body>
 </instance>
</method>
<class name="Inner2" access="private">
 <field name="inner2InstanceInt" access="private">
   <primitive-type name="int"/>
 </field>
 <constructor access="private">
   <body>
     <super-constructor-invocation/>
     <variable name="c">
       <reference-type package="java.lang" name="Class"/>
     </variable>
     <assignment>
       <variable-access name="c"/>
       <class-literal>
         <reference-type name="DotTest"/>
       </class-literal>
     </assignment>
     <assignment>
       <variable-access name="c"/>
       <class-literal>
         <reference-type name="Inner1">
           <enclosing-type name="DotTest"/>
         </reference-type>
       </class-literal>
     </assignment>
     <assignment>
       <variable-access name="c"/>
       <class-literal>
         <reference-type name="Inner1" package="jezix.srctest">
           <enclosing-type name="DotTest"/>
         </reference-type>
       </class-literal>
     </assignment>
```

```
<assignment>
  <variable-access name="c"/>
  <class-literal>
    <array-type dimensions="1">
      <primitive-type name="int"/>
    </array-type>
  </class-literal>
</assignment>
<assignment>
  <variable-access name="c"/>
  <class-literal>
    <primitive-type name="int"/>
  </class-literal>
</assignment>
<assignment>
  <variable-access name="c"/>
  <void-class-literal/>
</assignment>
<variable name="dt">
  <reference-type name="DotTest"/>
</variable>
<assignment>
  <variable-access name="dt"/>
  <this>
    <reference-type name="DotTest"/>
  </this>
</assignment>
<variable name="o">
  <reference-type name="Object"/>
</variable>
<assignment>
  <variable-access name="o"/>
  <this>
    <reference-type name="Inner1">
      <enclosing-type name="DotTest"/>
    </reference-type>
  </this>
</assignment>
<assignment>
  <variable-access name="o"/>
  <this>
    <reference-type name="Inner1" package="jezix.srctest">
      <enclosing-type name="DotTest"/>
    </reference-type>
  </this>
</assignment>
<variable name="j">
  <primitive-type name="int"/>
</variable>
<variable name="k">
  <primitive-type name="int"/>
  <int value="1"/>
</variable>
<assignment>
  <variable-access name="j"/>
  <field-access name="staticInt">
    <reference-type name="DotTest"/>
  </field-access>
</assignment>
<assignment>
  <variable-access name="j"/>
  <field-access name="staticInt">
    <reference-type package="jezix.srctest" name="DotTest"/>
  </field-access>
</assignment>
<assignment>
```

```
  <variable-access name="j"/>
  <field-access name="instanceInt">
    <variable-access name="dt"/>
  </field-access>
</assignment>
<assignment>
  <variable-access name="j"/>
  <field-access name="innerInstanceInt">
    <field-access name="instanceInner">
      <variable-access name="dt"/>
    </field-access>
  </field-access>
</assignment>
<assignment>
  <variable-access name="j"/>
  <field-access name="instanceInt">
    <field-access name="Arrays">
      <field-access name="util">
        <field-access name="java">
          <this>
            <reference-type name="DotTest"/>
          </this>
        </field-access>
      </field-access>
    </field-access>
  </field-access>
</assignment>
<assignment>
  <variable-access name="j"/>
  <field-access name="nestedInt">
    <reference-type name="StaticNested">
      <enclosing-type name="DotTest"/>
    </reference-type>
  </field-access>
</assignment>
<assignment>
  <variable-access name="j"/>
  <field-access name="inner2InstanceInt">
    <this/>
  </field-access>
</assignment>
<assignment>
  <variable-access name="j"/>
  <field-access name="innerInstanceInt">
    <this>
      <reference-type name="Inner1">
        <enclosing-type name="DotTest"/>
      </reference-type>
    </this>
  </field-access>
</assignment>
<assignment>
  <variable-access name="j"/>
  <field-access name="instanceInt">
    <from-superclass>
      <reference-type name="Inner1">
        <enclosing-type name="DotTest"/>
      </reference-type>
    </from-superclass>
  </field-access>
</assignment>
<variable name="in1">
  <reference-type name="Inner1"/>
</variable>
<assignment>
  <variable-access name="in1"/>
```

```
  <reference-cast>
    <reference-type name="Inner1" package="jezix.srctest">
      <enclosing-type name="DotTest"/>
    </reference-type>
    <variable-access name="o"/>
  </reference-cast>
</assignment>
<assignment>
  <variable-access name="j"/>
  <field-access name="length">
    <reference-cast>
      <array-type dimensions="1">
        <primitive-type name="int"/>
      </array-type>
      <variable-access name="o"/>
    </reference-cast>
  </field-access>
</assignment>
<assignment>
  <variable-access name="in1"/>
  <new-instance-of-class>
    <reference-type name="Inner1">
      <enclosing-type name="DotTest"/>
    </reference-type>
  </new-instance-of-class>
</assignment>
<assignment>
  <variable-access name="o"/>
  <new-instance-of-class>
    <reference-type name="Inner2"/>
    <enclosing-instance>
      <variable-access name="in1"/>
    </enclosing-instance>
    <arguments>
      <variable-access name="k"/>
      <variable-access name="i"/>
    </arguments>
  </new-instance-of-class>
</assignment>
<method-invocation name="staticMethod">
  <reference-type package="jezix.srctest" name="DotTest"/>
</method-invocation>
<method-invocation name="inner1Method">
  <variable-access name="in1"/>
  <arguments>
    <null/>
  </arguments>
</method-invocation>
<method-invocation name="instanceMethod">
  <from-superclass>
    <reference-type name="Inner1">
      <enclosing-type name="DotTest"/>
    </reference-type>
  </from-superclass>
</method-invocation>
<assignment>
  <variable-access name="j"/>
  <field-access name="innerInstanceInt">
    <method-invocation name="getMe">
      <this>
        <reference-type name="Inner1" package="jezix.srctest">
          <enclosing-type name="DotTest"/>
        </reference-type>
      </this>
    </method-invocation>
  </field-access>
```

```
        </assignment>
        <assignment>
          <variable-access name="j"/>
          <field-access name="length">
            <field-access name="IntArray">
              <reference-type name="StaticNested" package="jezix.srctest">
                <enclosing-type name="DotTest"/>
              </reference-type>
            </field-access>
          </field-access>
        </assignment>
        <variable name="DotTest">
          <reference-type name="DotTest"/>
          <variable-access name="dt"/>
        </variable>
        <assignment>
          <variable-access name="j"/>
          <field-access name="instanceInt">
            <variable-access name="DotTest"/>
          </field-access>
        </assignment>
        <method-invocation name="instanceMethod">
          <variable-access name="DotTest"/>
        </method-invocation>
      </body>
    </constructor>
  </class>
</class>
<class name="Inner3" access="private">
  <extends>
    <reference-type name="Inner1">
      <enclosing-type name="DotTest"/>
    </reference-type>
  </extends>
  <constructor access="private">
    <throws>
      <reference-type package="java.lang" name="Exception"/>
    </throws>
    <parameters>
      <variable name="String">
        <reference-type name="DotTest" package="jezix.srctest"/>
      </variable>
    </parameters>
    <body>
      <super-constructor-invocation>
        <enclosing-instance>
          <field-access name="staticInner">
            <reference-type name="DotTest"/>
          </field-access>
        </enclosing-instance>
      </super-constructor-invocation>
      <assignment>
        <field-access name="instanceInt">
          <variable-access name="String"/>
        </field-access>
        <field-access name="instanceInt">
          <variable-access name="String"/>
        </field-access>
      </assignment>
    </body>
  </constructor>
</class>
<class name="StaticNested" access="private" static="true">
  <extends>
    <reference-type name="RuntimeException"/>
  </extends>
```

```
      <field name="util">
        <reference-type name="DotTest"/>
      </field>
      <field static="true" name="nestedInt">
        <primitive-type name="int"/>
        <int value="1"/>
      </field>
      <field static="true" name="IntArray">
        <array-type dimensions="1">
          <primitive-type name="int"/>
        </array-type>
      </field>
    </class>
    <field access="private" name="java">
      <reference-type name="StaticNested"/>
    </field>
    <field name="Arrays" access="private">
      <reference-type name="DotTest"/>
    </field>
  </class>
</compilation-unit>
```

Ispis B.5: Izvorni kod po Badrosovoj sintaksi, DotTest.jmlb

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE java-source-program SYSTEM "java-ml.dtd">
<java-source-program>
  <java-class-file name="DotTest.java">
    <package-decl name="jezix.srctest"/>
    <import module="java.util.Set"/>
    <import module="java.io.*"/>
    <class name="DotTest" visibility="public">
      <superclass name="java.lang.RuntimeException"/>
      <implement interface="java.lang.Cloneable"/>
      <field visibility="private" name="instanceInt">
        <type name="int" primitive="true"/>
      </field>
      <field visibility="private" static="true" name="staticInt">
        <type name="int" primitive="true"/>
      </field>
      <field visibility="private" name="instanceInner">
        <type name="jezix.srctest.DotTest.Inner1"/>
        <new>
          <type name="jezix.srctest.DotTest.Inner1"/>
          <arguments/>
        </new>
      </field>
      <field visibility="private" static="true" name="staticInner">
        <type name="Inner1"/>
      </field>
      <method name="instanceMethod" visibility="private" id="a">
        <type name="void" primitive="true"/>
        <formal-arguments/>
        <block/>
      </method>
      <method name="staticMethod" visibility="private" static="true" id="b">
        <type name="void" primitive="true"/>
        <formal-arguments/>
        <block/>
      </method>
      <class name="Inner1" visibility="private">
        <superclass name="DotTest"/>
        <field name="innerInstanceInt" visibility="private">
          <type name="int" primitive="true"/>
```

```
</field>
<method name="getMe" visibility="private" id="d">
  <type name="DotTest.Inner1"/>
  <formal-arguments/>
  <block>
    <return>
      <this/>
    </return>
  </block>
</method>
<method name="inner1Method" visibility="private" id="e">
  <type name="void" primitive="true"/>
  <formal-arguments>
    <formal-argument name="Inner1" id="f">
      <type name="jezix.srctest.DotTest"/>
    </formal-argument>
  </formal-arguments>
  <throws exception="jezix.srctest.DotTest"/>
  <block>
    <local-variable name="j" id="g">
      <type name="int" primitive="true"/>
    </local-variable>
    <loop kind="for">
      <init>
        <local-variable name="Inner2" id="Inner2">
          <type name="Inner1"/>
          <field-access field="staticInner">
            <var-ref name="DotTest"/>
          </field-access>
        </local-variable>
      </init>
      <try>
        <block>
          <assignment-expr op="=">
            <lvalue>
              <var-set name="j"/>
            </lvalue>
            <field-access field="instanceInt">
              <super/>
            </field-access>
          </assignment-expr>
          <send message="instanceMethod">
            <target>
              <super/>
            </target>
            <arguments/>
          </send>
          <assignment-expr op="=">
            <lvalue>
              <var-set name="j"/>
            </lvalue>
            <field-access field="innerInstanceInt">
              <var-ref name="Inner2"/>
            </field-access>
          </assignment-expr>
        </block>
        <catch>
          <formal-argument name="DotTest" id="dtdt">
            <type name="jezix.srctest.DotTest"/>
          </formal-argument>
          <block>
            <assignment-expr op="=">
              <lvalue>
                <var-set name="j"/>
              </lvalue>
              <field-access field="instanceInt">
```

```
                <var-ref name="DotTest"/>
              </field-access>
            </assignment-expr>
            <assignment-expr op="=">
              <lvalue>
                <var-set name="j"/>
              </lvalue>
              <field-access field="instanceInt">
                <var-ref name="Inner1"/>
              </field-access>
            </assignment-expr>
            <assignment-expr op="=">
              <lvalue>
                <var-set name="j"/>
              </lvalue>
              <field-access field="nestedInt">
                <var-ref name="StaticNested"/>
              </field-access>
            </assignment-expr>
          </block>
        </catch>
        <catch>
          <formal-argument name="StaticNested" id="dtddt">
            <type name="StaticNested"/>
          </formal-argument>
          <block>
            <assignment-expr op="=">
              <lvalue>
                <var-set name="Inner1"/>
              </lvalue>
              <field-access field="util">
                <var-ref name="StaticNested"/>
              </field-access>
            </assignment-expr>
            <assignment-expr op="=">
              <lvalue>
                <var-set name="j"/>
              </lvalue>
              <field-access field="staticInt">
                <var-ref name="DotTest"/>
              </field-access>
            </assignment-expr>
          </block>
        </catch>
      </try>
    </loop>
  </block>
</method>
<class name="Inner2" visibility="private">
  <field name="inner2InstanceInt" visibility="private">
    <type name="int" primitive="true"/>
  </field>
  <constructor visibility="private" name="Inner2" id="h">
    <formal-arguments/>
    <super-call>
      <arguments/>
    </super-call>
    <block>
      <local-variable name="c" id="i">
        <type name="java.lang.Class"/>
      </local-variable>
      <assignment-expr op="=">
        <lvalue>
          <var-set name="c"/>
        </lvalue>
        <field-access field="class">
```

```
    <var-ref name="DotTest"/>
  </field-access>
</assignment-expr>
<assignment-expr op="=">
  <lvalue>
    <var-set name="c"/>
  </lvalue>
  <field-access field="class">
    <var-ref name="DotTest.Inner1"/>
  </field-access>
</assignment-expr>
<assignment-expr op="=">
  <lvalue>
    <var-set name="c"/>
  </lvalue>
  <field-access field="class">
     <var-ref name="jezix.srctest.DotTest.Inner1"/>
  </field-access>
</assignment-expr>
<assignment-expr op="=">
  <lvalue>
    <var-set name="c"/>
  </lvalue>
  <field-access field="class">
    <var-ref name="int[]"/>
  </field-access>
</assignment-expr>
<assignment-expr op="=">
  <lvalue>
    <var-set name="c"/>
  </lvalue>
  <field-access field="class">
    <var-ref name="int"/>
  </field-access>
</assignment-expr>
<assignment-expr op="=">
  <lvalue>
    <var-set name="c"/>
  </lvalue>
  <field-access field="class">
    <var-ref name="void"/>
  </field-access>
</assignment-expr>
<local-variable name="dt" id="ja">
  <type name="DotTest"/>
</local-variable>
<assignment-expr op="=">
  <lvalue>
    <var-set name="dt"/>
  </lvalue>
  <field-access field="this">
    <var-ref name="DotTest"/>
  </field-access>
</assignment-expr>
<local-variable name="o" id="j">
  <type name="Object"/>
</local-variable>
<assignment-expr op="=">
  <lvalue>
    <var-set name="o"/>
  </lvalue>
  <field-access field="this">
     <var-ref name="DotTest.Inner1"/>
  </field-access>
</assignment-expr>
<assignment-expr op="=">
```

```
  <lvalue>
    <var-set name="o"/>
  </lvalue>
  <field-access field="this">
     <var-ref name="jezix.srctest.DotTest.Inner1"/>
  </field-access>
</assignment-expr>
<local-variable name="j" id="k">
  <type name="int" primitive="true"/>
</local-variable>
<local-variable name="k" id="l">
  <type name="int" primitive="true"/>
  <literal-number kind="integer" value="1"/>
</local-variable>
<assignment-expr op="=">
  <lvalue>
    <var-set name="j"/>
  </lvalue>
  <field-access field="staticInt">
    <var-ref name="DotTest"/>
  </field-access>
</assignment-expr>
<assignment-expr op="=">
  <lvalue>
    <var-set name="j"/>
  </lvalue>
  <field-access field="staticInt">
    <var-ref name="jezix.srctest.DotTest"/>
  </field-access>
</assignment-expr>
<assignment-expr op="=">
  <lvalue>
    <var-set name="j"/>
  </lvalue>
  <field-access field="instanceInt">
    <var-ref name="dt"/>
  </field-access>
</assignment-expr>
<assignment-expr op="=">
  <lvalue>
    <var-set name="j"/>
  </lvalue>
  <field-access field="innerInstanceInt">
    <field-access field="instanceInner">
      <var-ref name="dt"/>
    </field-access>
  </field-access>
</assignment-expr>
<assignment-expr op="=">
  <lvalue>
    <var-set name="j"/>
  </lvalue>
  <field-access field="instanceInt">
    <field-access field="Arrays">
      <field-access field="util">
        <var-ref name="java"/>
      </field-access>
    </field-access>
  </field-access>
</assignment-expr>
<assignment-expr op="=">
  <lvalue>
    <var-set name="j"/>
  </lvalue>
  <field-access field="nestedInt">
    <var-ref name="DotTest.StaticNested"/>
```

```
  </field-access>
</assignment-expr>
<assignment-expr op="=">
  <lvalue>
    <var-set name="j"/>
  </lvalue>
  <field-access field="inner2InstanceInt">
    <this/>
  </field-access>
</assignment-expr>
<assignment-expr op="=">
  <lvalue>
    <var-set name="j"/>
  </lvalue>
  <field-access field="innerInstanceInt">
    <field-access field="this">
      <var-ref name="DotTest.Inner1"/>
    </field-access>
  </field-access>
</assignment-expr>
<assignment-expr op="=">
  <lvalue>
    <var-set name="j"/>
  </lvalue>
  <field-access field="instanceInt">
    <field-access field="super">
      <var-ref name="DotTest.Inner1"/>
    </field-access>
  </field-access>
</assignment-expr>
<local-variable name="in1" id="m">
  <type name="Inner1"/>
</local-variable>
<assignment-expr op="=">
  <lvalue>
    <var-set name="in1"/>
  </lvalue>
  <cast-expr>
    <type name="jezix.srctest.DotTest.Inner1"/>
    <var-ref name="o"/>
  </cast-expr>
</assignment-expr>
<assignment-expr op="=">
  <lvalue>
    <var-set name="j"/>
  </lvalue>
  <field-access field="length">
    <paren>
      <cast-expr>
        <type name="int[]"/>
        <var-ref name="o"/>
      </cast-expr>
    </paren>
  </field-access>
</assignment-expr>
<assignment-expr op="=">
  <lvalue>
    <var-set name="in1"/>
  </lvalue>
  <new>
    <type name="DotTest.Inner1"/>
    <arguments/>
  </new>
</assignment-expr>
<assignment-expr op="=">
  <lvalue>
```

```
    <var-set name="o"/>
  </lvalue>
  <new> <!--Should have been in1.new"-->
    <type name="Inner2"/>
    <arguments/>
  </new>
</assignment-expr>
<send message="staticMethod">
  <target>
    <var-ref name="jezix.srctest.DotTest"/>
  </target>
  <arguments/>
</send>
<send message="inner1Method">
  <target>
    <var-ref name="in1"/>
  </target>
  <arguments>
    <literal-null/>
  </arguments>
</send>
<send message="instanceMethod">
  <target>
    <field-access field="super">
       <var-ref name="DotTest.Inner1"/>
    </field-access>
  </target>
  <arguments/>
</send>
<assignment-expr op="=">
  <lvalue>
    <var-set name="j"/>
  </lvalue>
  <field-access field="innerInstanceInt">
    <send message="getMe">
      <target>
        <field-access field="this">
          <var-ref name="jezix.srctest.DotTest.Inner1"/>
        </field-access>
      </target>
      <arguments/>
    </send>
  </field-access>
</assignment-expr>
<assignment-expr op="=">
  <lvalue>
    <var-set name="j"/>
  </lvalue>
  <field-access field="length">
    <field-access field="IntArray">
      <var-ref name="jezix.srctest.DotTest.StaticNested"/>
    </field-access>
  </field-access>
</assignment-expr>
<local-variable name="DotTest" id="df">
  <type name="DotTest"/>
  <var-ref name="dt"/>
</local-variable>
<assignment-expr op="=">
  <lvalue>
    <var-set name="j"/>
  </lvalue>
  <field-access field="instanceInt">
    <var-ref name="DotTest"/>
  </field-access>
</assignment-expr>
```

```
        <send message="instanceMethod">
          <target>
            <var-ref name="DotTest"/>
          </target>
          <arguments/>
        </send>
      </block>
    </constructor>
    <constructor visibility="private" name="Inner2" id="p">
      <formal-arguments>
        <formal-argument name="i" id="q">
          <type name="int" primitive="true"/>
        </formal-argument>
        <formal-argument name="j" id="r">
          <type name="int" primitive="true"/>
        </formal-argument>
      </formal-arguments>
      <super-call>
        <arguments/>
      </super-call>
      <block/>
    </constructor>
  </class>
</class>
<class name="Inner3" visibility="private">
  <superclass name="DotTest.Inner1"/>
  <constructor visibility="private" name="Inner3" id="s">
    <formal-arguments>
      <formal-argument name="String" id="stsr">
        <type name="jezix.srctest.DotTest"/>
      </formal-argument>
    </formal-arguments>
    <throws exception="java.lang.Exception"/>
    <block>
      <!-- this should have been a superclass constructor call -->
      <send message="super">
        <target>
          <field-access field="staticInner">
            <var-ref name="DotTest"/>
          </field-access>
        </target>
        <arguments/>
      </send>
      <assignment-expr op="=">
        <lvalue>
          <field-set field="instanceInt">
            <var-ref name="String"/>
          </field-set>
        </lvalue>
        <field-access field="instanceInt">
          <var-ref name="String"/>
        </field-access>
      </assignment-expr>
    </block>
  </constructor>
</class>
<class name="StaticNested" visibility="private" static="true">
  <superclass name="RuntimeException"/>
  <field name="util">
    <type name="DotTest"/>
  </field>
  <field static="true" name="i">
    <type name="int" primitive="true"/>
    <literal-number kind="integer" value="1"/>
  </field>
  <field static="true" name="IntArray">
```

```
          <type name="int[]"/>
        </field>
      </class>
      <field visibility="private" name="java">
        <type name="StaticNested"/>
      </field>
      <field visibility="private" name="Arrays">
        <type name="DotTest"/>
      </field>
    </class>
  </java-class-file>
</java-source-program>
```

Ispis B.6: Izvorni kod po Mamasovoj sintaksi, DotTest.jmlm

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE CompilationUnit SYSTEM "javaml.dtd">
<CompilationUnit>
  <PackageDeclaration>
    <Name Identifier="jezix.srctest"/>
  </PackageDeclaration>
  <ImportDeclaration>
    <Name Identifier="java.util.Set"/>
  </ImportDeclaration>
  <ImportDeclaration>
    <Name Identifier="java.io.*"/>
  </ImportDeclaration>
  <TypeDeclaration>
    <ClassDeclaration isPublic="True">
      <UnmodifiedClassDeclaration Identifier="DotTest" Extends="True">
        <Name Identifier="java.lang.RuntimeException"/>
        <Name Identifier="java.lang.Cloneable"/>
        <ClassBody>
          <FieldDeclaration isPrivate="True">
            <Type ArraySize="0">
              <PrimitiveType Type="int"/>
            </Type>
            <VariableDeclarator>
              <VariableDeclaratorId Identifier="instanceInt" ArraySize="0"/>
            </VariableDeclarator>
          </FieldDeclaration>
          <FieldDeclaration isPrivate="True" isStatic="True">
            <Type ArraySize="0">
              <PrimitiveType Type="int"/>
            </Type>
            <VariableDeclarator>
              <VariableDeclaratorId Identifier="staticInt" ArraySize="0"/>
            </VariableDeclarator>
          </FieldDeclaration>
          <FieldDeclaration isPrivate="True">
            <Type ArraySize="0">
              <Name Identifier="jezix.srctest.DotTest.Inner1"/>
            </Type>
            <VariableDeclarator>
              <VariableDeclaratorId Identifier="instanceInner" ArraySize="0"/>
              <VariableInitializer>
                <Expression>
                  <ConditionalExpression>
                    <ConditionalOrExpression>
                      <ConditionalAndExpression>
                        <InclusiveOrExpression>
                          <ExclusiveOrExpression>
                            <AndExpression>
                              <EqualityExpression>
```

```
                        <InstanceOfExpression>
                          <RelationalExpression>
                            <ShiftExpression>
                              <AdditiveExpression>
                                <MultiplicativeExpression>
                                  <UnaryExpression>
                                    <UnaryExpressionNotPlusMinus>
                                      <PostfixExpression>
                                        <PrimaryExpression>
                                          <PrimaryPrefix>
                                            <AllocationExpression>
                                              <Name Identifier="jezix.srctest.DotTest.Inner1"/>
                                            </AllocationExpression>
                                          </PrimaryPrefix>
                                          <PrimarySuffix>
                                            <Arguments/>
                                          </PrimarySuffix>
                                        </PrimaryExpression>
                                      </PostfixExpression>
                                    </UnaryExpressionNotPlusMinus>
                                  </UnaryExpression>
                                </MultiplicativeExpression>
                              </AdditiveExpression>
                            </ShiftExpression>
                          </RelationalExpression>
                        </InstanceOfExpression>
                      </EqualityExpression>
                    </AndExpression>
                  </ExclusiveOrExpression>
                </InclusiveOrExpression>
              </ConditionalAndExpression>
            </ConditionalOrExpression>
          </ConditionalExpression>
        </Expression>
      </VariableInitializer>
    </VariableDeclarator>
  </FieldDeclaration>
  <FieldDeclaration isPrivate="True" isStatic="True">
    <Type ArraySize="0">
      <Name Identifier="Inner1"/>
    </Type>
    <VariableDeclarator>
      <VariableDeclaratorId Identifier="staticInner" ArraySize="0"/>
    </VariableDeclarator>
  </FieldDeclaration>
  <MethodDeclaration isPrivate="True">
    <ResultType/>
    <MethodDeclarator Identifier="instanceMethod" ArraySize="0"/>
    <Block/>
  </MethodDeclaration>
  <MethodDeclaration isPrivate="True" isStatic="True">
    <ResultType/>
    <MethodDeclarator Identifier="staticMethod" ArraySize="0">
      <FormalParameter>
        <Type ArraySize="0">
          <PrimitiveType Type="boolean"/>
        </Type>
        <VariableDeclaratorId Identifier="b" ArraySize="0"/>
      </FormalParameter>
    </MethodDeclarator>
    <Block/>
  </MethodDeclaration>
  <NestedClassDeclaration isPrivate="True">
    <UnmodifiedClassDeclaration Identifier="Inner1" Extends="True">
      <Name Identifier="DotTest"/>
      <ClassBody>
```

```
<FieldDeclaration isPrivate="True">
  <Type ArraySize="0">
    <PrimitiveType Type="int"/>
  </Type>
  <VariableDeclarator>
    <VariableDeclaratorId ArraySize="0" Identifier="innerInstanceInt"/>
  </VariableDeclarator>
</FieldDeclaration>
<MethodDeclaration isPrivate="True">
  <ResultType>
    <Type ArraySize="0">
      <Name Identifier="DotTest.Inner1"/>
    </Type>
  </ResultType>
  <MethodDeclarator ArraySize="0" Identifier="getMe"/>
  <Block>
    <ReturnStatement>
      <Expression>
                    :::
                                :::
                                      <PrimaryExpression>
                                        <PrimaryPrefix isThis="True"/>
                                      </PrimaryExpression>
                                :::
                    :::
      </Expression>
    </ReturnStatement>
  </Block>
</MethodDeclaration>
<MethodDeclaration isPrivate="True">
  <ResultType/>
  <MethodDeclarator ArraySize="0" Identifier="inner1Method">
    <FormalParameter>
      <Type ArraySize="0">
        <Name Identifier="jezix.srctest.DotTest"/>
      </Type>
      <VariableDeclaratorId ArraySize="0" Identifier="Inner1"/>
    </FormalParameter>
  </MethodDeclarator>
  <Name Identifier="jezix.srctest.DotTest"/>
  <Block>
    <LocalVariableDeclaration>
      <Type ArraySize="0">
        <PrimitiveType Type="int"/>
      </Type>
      <VariableDeclarator>
        <VariableDeclaratorId ArraySize="0" Identifier="j"/>
      </VariableDeclarator>
    </LocalVariableDeclaration>
    <ForStatement>
      <ForInit>
        <LocalVariableDeclaration>
          <Type ArraySize="0">
            <Name Identifier="Inner1"/>
          </Type>
          <VariableDeclarator>
            <VariableDeclaratorId ArraySize="0" Identifier="Inner2"/>
            <VariableInitializer>
              <Expression>
                        :::
                                    :::
                                              <PrimaryExpression>
                                                <PrimaryPrefix>
                                                  <Name Identifier="DotTest"/>
                                                </PrimaryPrefix>
                                                <PrimarySuffix Identifier="staticInner"/>
```

```
                                        </PrimaryExpression>
                                    :::
                        :::
          </Expression>
        </VariableInitializer>
      </VariableDeclarator>
    </LocalVariableDeclaration>
  </ForInit>
  <Block>
    <TryStatement>
      <Block>
        <StatementExpression>
          <PrimaryExpression>
            <PrimaryPrefix>
              <Name Identifier="j"/>
            </PrimaryPrefix>
          </PrimaryExpression>
          <AssignmentOperator Type="Simple"/>
          <Expression>
              :::
                      :::
                          <PrimaryExpression>
                            <PrimaryPrefix isSuper="True"
                              Identifier="instanceInt"/>
                          </PrimaryExpression>
                      :::
              :::
          </Expression>
        </StatementExpression>
        <StatementExpression>
          <PrimaryExpression>
            <PrimaryPrefix isSuper="True" Identifier="instanceMethod"/>
            <PrimarySuffix>
              <Arguments/>
            </PrimarySuffix>
          </PrimaryExpression>
        </StatementExpression>
        <StatementExpression>
          <PrimaryExpression>
            <PrimaryPrefix>
              <Name Identifier="j"/>
            </PrimaryPrefix>
          </PrimaryExpression>
          <AssignmentOperator Type="Simple"/>
          <Expression>
              :::
                      :::
                          <PrimaryExpression>
                            <PrimaryPrefix>
                              <Name Identifier="Inner2"/>
                            </PrimaryPrefix>
                            <PrimarySuffix Identifier="innerInstanceInt"/>
                          </PrimaryExpression>
                      :::
              :::
          </Expression>
        </StatementExpression>
      </Block>
      <FormalParameter>
        <Type ArraySize="0">
          <Name Identifier="jezix.srctest.DotTest"/>
        </Type>
        <VariableDeclaratorId ArraySize="0" Identifier="DotTest"/>
      </FormalParameter>
      <Block>
        <StatementExpression>
```

```
    <PrimaryExpression>
      <PrimaryPrefix>
        <Name Identifier="j"/>
      </PrimaryPrefix>
    </PrimaryExpression>
    <AssignmentOperator Type="Simple"/>
    <Expression>
          :::
                      :::
                            <PrimaryExpression>
                              <PrimaryPrefix>
                                <Name Identifier="DotTest"/>
                              </PrimaryPrefix>
                              <PrimarySuffix Identifier="instanceInt"/>
                            </PrimaryExpression>
                      :::
          :::
    </Expression>
  </StatementExpression>
  <StatementExpression>
    <PrimaryExpression>
      <PrimaryPrefix>
        <Name Identifier="j"/>
      </PrimaryPrefix>
    </PrimaryExpression>
    <AssignmentOperator Type="Simple"/>
    <Expression>
          :::
                      :::
                            <PrimaryExpression>
                              <PrimaryPrefix>
                                <Name Identifier="Inner1"/>
                              </PrimaryPrefix>
                              <PrimarySuffix Identifier="instanceInt"/>
                            </PrimaryExpression>
                      :::
          :::
    </Expression>
  </StatementExpression>
  <StatementExpression>
    <PrimaryExpression>
      <PrimaryPrefix>
        <Name Identifier="j"/>
      </PrimaryPrefix>
    </PrimaryExpression>
    <AssignmentOperator Type="Simple"/>
    <Expression>
          :::
                      :::
                            <PrimaryExpression>
                              <PrimaryPrefix>
                                <Name Identifier="StaticNested"/>
                              </PrimaryPrefix>
                              <PrimarySuffix Identifier="nestedInt"/>
                            </PrimaryExpression>
                      :::
          :::
    </Expression>
  </StatementExpression>
</Block>
<FormalParameter>
  <Type ArraySize="0">
    <Name Identifier="StaticNested"/>
  </Type>
  <VariableDeclaratorId ArraySize="0" Identifier="StaticNested"/>
</FormalParameter>
```

```
                <Block>
                  <StatementExpression>
                    <PrimaryExpression>
                      <PrimaryPrefix>
                        <Name Identifier="Inner1"/>
                      </PrimaryPrefix>
                    </PrimaryExpression>
                    <AssignmentOperator Type="Simple"/>
                    <Expression>
                          :::
                                    :::
                                          <PrimaryExpression>
                                            <PrimaryPrefix>
                                              <Name Identifier="StaticNested"/>
                                            </PrimaryPrefix>
                                            <PrimarySuffix Identifier="util"/>
                                          </PrimaryExpression>
                                    :::
                              :::
                    </Expression>
                  </StatementExpression>
                  <StatementExpression>
                    <PrimaryExpression>
                      <PrimaryPrefix>
                        <Name Identifier="j"/>
                      </PrimaryPrefix>
                    </PrimaryExpression>
                    <AssignmentOperator Type="Simple"/>
                    <Expression>
                          :::
                                    :::
                                          <PrimaryExpression>
                                            <PrimaryPrefix>
                                              <Name Identifier="DotTest"/>
                                            </PrimaryPrefix>
                                            <PrimarySuffix Identifier="staticInt"/>
                                          </PrimaryExpression>
                                    :::
                              :::
                    </Expression>
                  </StatementExpression>
                </Block>
              </TryStatement>
            </Block>
          </ForStatement>
        </Block>
      </MethodDeclaration>
    </ClassBody>
  </UnmodifiedClassDeclaration>
</NestedClassDeclaration>
<NestedClassDeclaration isPrivate="True">
  <UnmodifiedClassDeclaration Identifier="Inner2">
    <ClassBody>
      <FieldDeclaration>
        <Type ArraySize="0">
          <PrimitiveType Type="int"/>
        </Type>
        <VariableDeclarator>
          <VariableDeclaratorId ArraySize="0" Identifier="inner2InstanceInt"/>
        </VariableDeclarator>
      </FieldDeclaration>
      <ConstructorDeclaration Identifier="Inner2" isPrivate="True">
        <ExplicitConstructorInvocation Type="super">
          <Arguments/>
        </ExplicitConstructorInvocation>
        <Block>
```

```
<LocalVariableDeclaration>
  <Type ArraySize="0">
    <Name Identifier="java.lang.Class"/>
  </Type>
  <VariableDeclarator>
    <VariableDeclaratorId ArraySize="0" Identifier="c"/>
  </VariableDeclarator>
</LocalVariableDeclaration>
<StatementExpression>
  <PrimaryExpression>
    <PrimaryPrefix>
      <Name Identifier="c"/>
    </PrimaryPrefix>
  </PrimaryExpression>
  <AssignmentOperator Type="Simple"/>
  <Expression>
                :::
                            :::
                                  <PrimaryExpression>
                                    <PrimaryPrefix>
                                      <ResultType>
                                        <Type ArraySize="0">
                                          <Name Identifier="DotTest"/>
                                        </Type>
                                      </ResultType>
                                    </PrimaryPrefix>
                                  </PrimaryExpression>
                            :::
                :::
  </Expression>
</StatementExpression>
<StatementExpression>
  <PrimaryExpression>
    <PrimaryPrefix>
      <Name Identifier="c"/>
    </PrimaryPrefix>
  </PrimaryExpression>
  <AssignmentOperator Type="Simple"/>
  <Expression>
                :::
                            :::
                                  <PrimaryExpression>
                                    <PrimaryPrefix>
                                      <ResultType>
                                        <Type ArraySize="0">
                                          <Name Identifier="DotTest.Inner1"/>
                                        </Type>
                                      </ResultType>
                                    </PrimaryPrefix>
                                  </PrimaryExpression>
                            :::
                :::
  </Expression>
</StatementExpression>
<StatementExpression>
  <PrimaryExpression>
    <PrimaryPrefix>
      <Name Identifier="c"/>
    </PrimaryPrefix>
  </PrimaryExpression>
  <AssignmentOperator Type="Simple"/>
  <Expression>
        :::
                    :::
                  <PrimaryExpression>
                    <PrimaryPrefix>
```

```
                    <ResultType>
                      <Type ArraySize="0">
                        <Name Identifier="jezix.srctest.DotTest.Inner1"/>
                      </Type>
                    </ResultType>
                  </PrimaryPrefix>
                </PrimaryExpression>
                  :::
        :::
  </Expression>
</StatementExpression>
<StatementExpression>
  <PrimaryExpression>
    <PrimaryPrefix>
      <Name Identifier="c"/>
    </PrimaryPrefix>
  </PrimaryExpression>
  <AssignmentOperator Type="Simple"/>
  <Expression>
                :::
                          :::
                                <PrimaryExpression>
                                  <PrimaryPrefix>
                                    <ResultType>
                                      <Type ArraySize="0">
                                        <Name Identifier="void"/>
                                      </Type>
                                    </ResultType>
                                  </PrimaryPrefix>
                                </PrimaryExpression>
                          :::
              :::
  </Expression>
</StatementExpression>
<LocalVariableDeclaration>
  <Type ArraySize="0">
    <Name Identifier="DotTest"/>
  </Type>
  <VariableDeclarator>
    <VariableDeclaratorId ArraySize="0" Identifier="dt"/>
  </VariableDeclarator>
</LocalVariableDeclaration>
<StatementExpression>
  <PrimaryExpression>
    <PrimaryPrefix>
      <Name Identifier="dt"/>
    </PrimaryPrefix>
  </PrimaryExpression>
  <AssignmentOperator Type="Simple"/>
  <Expression>
                :::
                          :::
                                <PrimaryExpression>
                                  <PrimaryPrefix>
                                    <Name Identifier="DotTest"/>
                                  </PrimaryPrefix>
                                  <PrimarySuffix isThis="True"/>
                                </PrimaryExpression>
                          :::
              :::
  </Expression>
</StatementExpression>
<LocalVariableDeclaration>
  <Type ArraySize="0">
    <Name Identifier="Object"/>
  </Type>
```

```
  <VariableDeclarator>
    <VariableDeclaratorId ArraySize="0" Identifier="o"/>
  </VariableDeclarator>
</LocalVariableDeclaration>
<StatementExpression>
  <PrimaryExpression>
    <PrimaryPrefix>
      <Name Identifier="o"/>
    </PrimaryPrefix>
  </PrimaryExpression>
  <AssignmentOperator Type="Simple"/>
  <Expression>
                :::
                            :::
                                  <PrimaryExpression>
                                    <PrimaryPrefix>
                                      <Name Identifier="DotTest.Inner1"/>
                                    </PrimaryPrefix>
                                    <PrimarySuffix isThis="True"/>
                                  </PrimaryExpression>
                            :::
              :::
  </Expression>
</StatementExpression>
<StatementExpression>
  <PrimaryExpression>
    <PrimaryPrefix>
      <Name Identifier="o"/>
    </PrimaryPrefix>
  </PrimaryExpression>
  <AssignmentOperator Type="Simple"/>
  <Expression>
    :::
              :::
                <PrimaryExpression>
                  <PrimaryPrefix>
                    <Name Identifier="jezix.srctest.DotTest.Inner1"/>
                  </PrimaryPrefix>
                  <PrimarySuffix isThis="True"/>
                </PrimaryExpression>
              :::
    :::
  </Expression>
</StatementExpression>
<LocalVariableDeclaration>
  <Type ArraySize="0">
    <PrimitiveType Type="int"/>
  </Type>
  <VariableDeclarator>
    <VariableDeclaratorId ArraySize="0" Identifier="j"/>
  </VariableDeclarator>
</LocalVariableDeclaration>
<LocalVariableDeclaration>
  <Type ArraySize="0">
    <PrimitiveType Type="int"/>
  </Type>
  <VariableDeclarator>
    <VariableDeclaratorId ArraySize="0" Identifier="k"/>
    <VariableInitializer>
      <Expression>
                :::
                            :::
                                    <PrimaryExpression>
                                      <PrimaryPrefix>
                                        <Literal Value="1"/>
                                      </PrimaryPrefix>
```

```
                                    </PrimaryExpression>
                              :::
                  :::
      </Expression>
    </VariableInitializer>
  </VariableDeclarator>
</LocalVariableDeclaration>
<StatementExpression>
  <PrimaryExpression>
    <PrimaryPrefix>
      <Name Identifier="j"/>
    </PrimaryPrefix>
  </PrimaryExpression>
  <AssignmentOperator Type="Simple"/>
  <Expression>
                  :::
                              :::
                                    <PrimaryExpression>
                                      <PrimaryPrefix>
                                        <Name Identifier="DotTest"/>
                                      </PrimaryPrefix>
                                      <PrimarySuffix Identifier="staticInt"/>
                                    </PrimaryExpression>
                              :::
                  :::
  </Expression>
</StatementExpression>
<StatementExpression>
  <PrimaryExpression>
    <PrimaryPrefix>
      <Name Identifier="j"/>
    </PrimaryPrefix>
  </PrimaryExpression>
  <AssignmentOperator Type="Simple"/>
  <Expression>
        :::
                    :::
                          <PrimaryExpression>
                            <PrimaryPrefix>
                              <Name Identifier="jezix.srctest.DotTest"/>
                            </PrimaryPrefix>
                            <PrimarySuffix Identifier="staticInt"/>
                          </PrimaryExpression>
                    :::
        :::
  </Expression>
</StatementExpression>
<StatementExpression>
  <PrimaryExpression>
    <PrimaryPrefix>
      <Name Identifier="j"/>
    </PrimaryPrefix>
  </PrimaryExpression>
  <AssignmentOperator Type="Simple"/>
  <Expression>
                  :::
                              :::
                                    <PrimaryExpression>
                                      <PrimaryPrefix>
                                        <Name Identifier="dt"/>
                                      </PrimaryPrefix>
                                      <PrimarySuffix Identifier="instanceInt"/>
                                    </PrimaryExpression>
                              :::
                  :::
  </Expression>
```

```
</StatementExpression>
<StatementExpression>
  <PrimaryExpression>
    <PrimaryPrefix>
      <Name Identifier="j"/>
    </PrimaryPrefix>
  </PrimaryExpression>
  <AssignmentOperator Type="Simple"/>
  <Expression>
        :::
                    :::
                          <PrimaryExpression>
                            <PrimaryPrefix>
                              <Name Identifier="dt"/>
                            </PrimaryPrefix>
                            <PrimarySuffix Identifier="instanceInner"/>
                            <PrimarySuffix Identifier="innerInstanceInt"/>
                          </PrimaryExpression>
                    :::
        :::
  </Expression>
</StatementExpression>
<StatementExpression>
  <PrimaryExpression>
    <PrimaryPrefix>
      <Name Identifier="j"/>
    </PrimaryPrefix>
  </PrimaryExpression>
  <AssignmentOperator Type="Simple"/>
  <Expression>
        :::
                    :::
                          <PrimaryExpression>
                            <PrimaryPrefix>
                              <Name Identifier="java"/>
                            </PrimaryPrefix>
                            <PrimarySuffix Identifier="util"/>
                            <PrimarySuffix Identifier="Arrays"/>
                            <PrimarySuffix Identifier="instanceInt"/>
                          </PrimaryExpression>
                    :::
        :::
  </Expression>
</StatementExpression>
<StatementExpression>
  <PrimaryExpression>
    <PrimaryPrefix>
      <Name Identifier="j"/>
    </PrimaryPrefix>
  </PrimaryExpression>
  <AssignmentOperator Type="Simple"/>
  <Expression>
        :::
                    :::
                          <PrimaryExpression>
                            <PrimaryPrefix>
                              <Name Identifier="DotTest.StaticNested"/>
                            </PrimaryPrefix>
                            <PrimarySuffix Identifier="nestedInt"/>
                          </PrimaryExpression>
                    :::
        :::
  </Expression>
</StatementExpression>
<StatementExpression>
  <PrimaryExpression>
```

```
    <PrimaryPrefix>
      <Name Identifier="j"/>
    </PrimaryPrefix>
  </PrimaryExpression>
  <AssignmentOperator Type="Simple"/>
  <Expression>
        :::
                    :::
                          <PrimaryExpression>
                            <PrimaryPrefix isThis="True"/>
                            <PrimarySuffix Identifier="inner2InstanceInt"/>
                          </PrimaryExpression>
                    :::
        :::
  </Expression>
</StatementExpression>
<StatementExpression>
  <PrimaryExpression>
    <PrimaryPrefix>
      <Name Identifier="j"/>
    </PrimaryPrefix>
  </PrimaryExpression>
  <AssignmentOperator Type="Simple"/>
  <Expression>
        :::
                    :::
                          <PrimaryExpression>
                            <PrimaryPrefix>
                              <Name Identifier="DotTest.Inner1"/>
                            </PrimaryPrefix>
                            <PrimarySuffix isThis="True"/>
                            <PrimarySuffix Identifier="innerInstanceInt"/>
                          </PrimaryExpression>
                    :::
        :::
  </Expression>
</StatementExpression>
<StatementExpression>
  <PrimaryExpression>
    <PrimaryPrefix>
      <Name Identifier="j"/>
    </PrimaryPrefix>
  </PrimaryExpression>
  <AssignmentOperator Type="Simple"/>
  <Expression>
              :::
                          :::
                                <PrimaryExpression>
                                  <PrimaryPrefix>
                                    <Name Identifier="DotTest.Inner1"/>
                                  </PrimaryPrefix>
                                  <PrimarySuffix Identifier="super"/>
                                  <PrimarySuffix Identifier="instanceInt"/>
                                </PrimaryExpression>
                          :::
              :::
  </Expression>
</StatementExpression>
<LocalVariableDeclaration>
  <Type ArraySize="0">
    <Name Identifier="Inner1"/>
  </Type>
  <VariableDeclarator>
    <VariableDeclaratorId ArraySize="0" Identifier="in1"/>
  </VariableDeclarator>
</LocalVariableDeclaration>
```

```
<StatementExpression>
  <PrimaryExpression>
    <PrimaryPrefix>
      <Name Identifier="j"/>
    </PrimaryPrefix>
  </PrimaryExpression>
  <AssignmentOperator Type="Simple"/>
  <Expression>
        :::
                    :::
                        <CastExpression>
                          <Type ArraySize="0">
                            <Name Identifier="jezix.srctest.DotTest.Inner1"/>
                          </Type>
                          <UnaryExpressionNotPlusMinus>
                            <PostfixExpression>
                              <PrimaryExpression>
                                <PrimaryPrefix>
                                  <Name Identifier="o"/>
                                </PrimaryPrefix>
                              </PrimaryExpression>
                            </PostfixExpression>
                          </UnaryExpressionNotPlusMinus>
                        </CastExpression>
                    :::
        :::
  </Expression>
</StatementExpression>
<StatementExpression>
  <PrimaryExpression>
    <PrimaryPrefix>
      <Name Identifier="in1"/>
    </PrimaryPrefix>
  </PrimaryExpression>
  <AssignmentOperator Type="Simple"/>
  <Expression>
               :::
                           :::
                                 <PrimaryExpression>
                                   <PrimaryPrefix>
                                     <AllocationExpression>
                                       <Name Identifier="DotTest.Inner1"/>
                                       <Arguments/>
                                     </AllocationExpression>
                                   </PrimaryPrefix>
                                 </PrimaryExpression>
                        :::
            :::
  </Expression>
</StatementExpression>
<StatementExpression>
  <PrimaryExpression>
    <PrimaryPrefix>
      <Name Identifier="in1"/>
    </PrimaryPrefix>
  </PrimaryExpression>
  <AssignmentOperator Type="Simple"/>
  <Expression>
            :::
                        :::
                                  <PrimaryExpression>
                                    <PrimaryPrefix>
                                        <Name Identifier="in1"/>
                                    </PrimaryPrefix>
                                    <PrimarySuffix>
                                      <AllocationExpression>
```

```
                                        <Name Identifier="Inner2"/>
                                        <Arguments/>
                                      </AllocationExpression>
                                    </PrimarySuffix>
                                  </PrimaryExpression>
                          :::
              :::
    </Expression>
  </StatementExpression>
  <StatementExpression>
    <PrimaryExpression>
      <PrimaryPrefix>
        <Name Identifier="jezix.srctest.DotTest"/>
      </PrimaryPrefix>
      <PrimarySuffix Identifier="staticMethod"/>
      <PrimarySuffix>
        <Arguments/>
      </PrimarySuffix>
    </PrimaryExpression>
  </StatementExpression>
  <StatementExpression>
    <PrimaryExpression>
      <PrimaryPrefix>
        <Name Identifier="in1"/>
      </PrimaryPrefix>
      <PrimarySuffix Identifier="inner1Method"/>
      <PrimarySuffix>
        <Arguments>
          <Expression>
                        :::
                                  :::
                                        <PrimaryExpression>
                                          <PrimaryPrefix>
                                            <Literal Type="Null" Value="null"/>
                                          </PrimaryPrefix>
                                        </PrimaryExpression>
                                      </PostfixExpression>
                                  :::
                        :::
          </Expression>
        </Arguments>
      </PrimarySuffix>
    </PrimaryExpression>
  </StatementExpression>
  <StatementExpression>
    <PrimaryExpression>
      <PrimaryPrefix>
        <Name Identifier="DotTest.Inner1"/>
      </PrimaryPrefix>
      <PrimarySuffix Identifier="super"/>
      <PrimarySuffix Identifier="instanceMethod"/>
      <PrimarySuffix>
        <Arguments/>
      </PrimarySuffix>
    </PrimaryExpression>
  </StatementExpression>
  <StatementExpression>
    <PrimaryExpression>
      <PrimaryPrefix>
        <Name Identifier="j"/>
      </PrimaryPrefix>
    </PrimaryExpression>
    <AssignmentOperator Type="Simple"/>
    <Expression>
          :::
                        :::
```

```
                        <PrimaryExpression>
                          <PrimaryPrefix>
                            <Name Identifier="jezix.srctest.DotTest.Inner1"/>
                          </PrimaryPrefix>
                          <PrimarySuffix isThis="True"/>
                          <PrimarySuffix Identifier="getMe"/>
                          <PrimarySuffix>
                            <Arguments/>
                          </PrimarySuffix>
                          <PrimarySuffix Identifier="innerInstanceInt"/>
                        </PrimaryExpression>
                  :::
      :::
  </Expression>
</StatementExpression>
<StatementExpression>
  <PrimaryExpression>
    <PrimaryPrefix>
      <Name Identifier="j"/>
    </PrimaryPrefix>
  </PrimaryExpression>
  <AssignmentOperator Type="Simple"/>
  <Expression>
      :::
                  :::
                        <PrimaryExpression>
                          <PrimaryPrefix>
                            <Name
                            Identifier="jezix.srctest.DotTest.StaticNested"/>
                          </PrimaryPrefix>
                          <PrimarySuffix Identifier="IntArray"/>
                          <PrimarySuffix Identifier="length"/>
                        </PrimaryExpression>
                  :::
      :::
  </Expression>
</StatementExpression>
<LocalVariableDeclaration>
  <Type ArraySize="0">
    <Name Identifier="DotTest"/>
  </Type>
  <VariableDeclarator>
    <VariableDeclaratorId ArraySize="0" Identifier="DotTest"/>
    <VariableInitializer>
      <Expression>
                  :::
                           :::
                                 <PrimaryExpression>
                                   <PrimaryPrefix>
                                     <Name Identifier="dt"/>
                                   </PrimaryPrefix>
                                 </PrimaryExpression>
                           :::
                  :::
      </Expression>
    </VariableInitializer>
  </VariableDeclarator>
</LocalVariableDeclaration>
<StatementExpression>
  <PrimaryExpression>
    <PrimaryPrefix>
      <Name Identifier="j"/>
    </PrimaryPrefix>
  </PrimaryExpression>
  <AssignmentOperator Type="Simple"/>
  <Expression>
```

```
                        :::
                                  :::
                                            <PrimaryExpression>
                                              <PrimaryPrefix>
                                                <Name Identifier="DotTest"/>
                                              </PrimaryPrefix>
                                              <PrimarySuffix Identifier="instanceInt"/>
                                            </PrimaryExpression>
                                  :::
                        :::
              </Expression>
            </StatementExpression>
            <StatementExpression>
              <PrimaryExpression>
                <PrimaryPrefix>
                  <Name Identifier="DotTest"/>
                </PrimaryPrefix>
                <PrimarySuffix Identifier="instanceMethod"/>
                <PrimarySuffix>
                  <Arguments/>
                </PrimarySuffix>
              </PrimaryExpression>
            </StatementExpression>
          </Block>
        </ConstructorDeclaration>
      </ClassBody>
    </UnmodifiedClassDeclaration>
  </NestedClassDeclaration>
  <NestedClassDeclaration isPrivate="True">
    <UnmodifiedClassDeclaration Identifier="Inner3" Extends="True">
      <Name Identifier="DotTest.Inner1"/>
      <ClassBody>
        <ConstructorDeclaration Identifier="Inner3" isPrivate="True">
          <FormalParameter>
            <Type ArraySize="0">
              <Name Identifier="jezix.srctest.DotTest"/>
            </Type>
            <VariableDeclaratorId ArraySize="0" Identifier="String"/>
          </FormalParameter>
          <Name Identifier="java.lang.Exception"/>
          <ExplicitConstructorInvocation Type="super">
            <PrimaryExpression>
              <PrimaryPrefix>
                <Name Identifier="DotTest"/>
              </PrimaryPrefix>
              <PrimarySuffix Identifier="staticInner"/>
            </PrimaryExpression>
            <Arguments/>
          </ExplicitConstructorInvocation>
          <Block>
            <StatementExpression>
              <PrimaryExpression>
                <PrimaryPrefix>
                  <Name Identifier="String"/>
                </PrimaryPrefix>
                <PrimarySuffix Identifier="instanceInt"/>
              </PrimaryExpression>
              <AssignmentOperator Type="Simple"/>
              <Expression>
                        :::
                                  :::
                                            <PrimaryExpression>
                                              <PrimaryPrefix>
                                                <Name Identifier="String"/>
                                              </PrimaryPrefix>
                                              <PrimarySuffix Identifier="instanceInt"/>
```

```
                                        </PrimaryExpression>
                                :::
                        :::
                    </Expression>
                  </StatementExpression>
                </Block>
              </ConstructorDeclaration>
            </ClassBody>
          </UnmodifiedClassDeclaration>
        </NestedClassDeclaration>
        <NestedClassDeclaration isPrivate="True" isStatic="True">
          <UnmodifiedClassDeclaration Identifier="StaticNested" Extends="True">
            <Name Identifier="RuntimeException"/>
            <ClassBody>
              <FieldDeclaration>
                <Type ArraySize="0">
                  <Name Identifier="DotTest"/>
                </Type>
                <VariableDeclarator>
                  <VariableDeclaratorId ArraySize="0" Identifier="util"/>
                </VariableDeclarator>
              </FieldDeclaration>
              <FieldDeclaration isStatic="True">
                <Type ArraySize="0">
                  <PrimitiveType Type="int"/>
                </Type>
                <VariableDeclarator>
                  <VariableDeclaratorId ArraySize="0" Identifier="nestedInt"/>
                  <VariableInitializer>
                    <Expression>
                              :::
                                        :::
                                                <PostfixExpression>
                                                  <PrimaryExpression>
                                                    <PrimaryPrefix>
                                                      <Literal Value="1"/>
                                                    </PrimaryPrefix>
                                                  </PrimaryExpression>
                                        :::
                              :::
                    </Expression>
                  </VariableInitializer>
                </VariableDeclarator>
              </FieldDeclaration>
              <FieldDeclaration>
                <Type ArraySize="1">
                  <PrimitiveType Type="int"/>
                </Type>
                <VariableDeclarator>
                  <VariableDeclaratorId ArraySize="0" Identifier="IntArray"/>
                </VariableDeclarator>
              </FieldDeclaration>
            </ClassBody>
          </UnmodifiedClassDeclaration>
        </NestedClassDeclaration>
        <FieldDeclaration isPrivate="True">
          <Type ArraySize="0">
            <Name Identifier="StaticNested"/>
          </Type>
          <VariableDeclarator>
            <VariableDeclaratorId ArraySize="0" Identifier="java"/>
          </VariableDeclarator>
        </FieldDeclaration>
        <FieldDeclaration isPrivate="True">
          <Type ArraySize="0">
            <Name Identifier="DotTest"/>
```

```
            </Type>
            <VariableDeclarator>
              <VariableDeclaratorId ArraySize="0" Identifier="Arrays"/>
            </VariableDeclarator>
          </FieldDeclaration>
        </ClassBody>
      </UnmodifiedClassDeclaration>
    </ClassDeclaration>
  </TypeDeclaration>
</CompilationUnit>
```

Dodatak C

Zadaci: rezultati upita

C.1 Zadatak 1

Ispis C.1: Predložena sintaksa

```
<?xml version="1.0" encoding="UTF-8"?>
<and-operations xmlns="http://tel.fer.hr/jezix/java">
  <fourth-member>
    <conditional-or>
      <boolean value="true"/>
      <boolean value="true"/>
    </conditional-or>
  </fourth-member>
  <fourth-member>
    <conditional-or>
      <boolean value="true"/>
      <boolean value="true"/>
    </conditional-or>
  </fourth-member>
</and-operations>
```

Ispis C.2: Badrosova sintaksa

```
<?xml version="1.0" encoding="UTF-8"?>
<and-operations>
  <fourth-member>
    <binary-expr op="||">
      <literal-boolean value="true"/>
      <literal-boolean value="true"/>
    </binary-expr>
  </fourth-member>
  <fourth-member>
    <binary-expr op="||">
      <literal-boolean value="true"/>
      <literal-boolean value="true"/>
    </binary-expr>
  </fourth-member>
</and-operations>
```

Ispis C.3: Mamasova sintaksa

```
<?xml version="1.0" encoding="UTF-8"?>
<and-operations>
  <fourth-member>
    <ConditionalOrExpression>
      <ConditionalAndExpression>
               :::
                      :::
                               <PrimaryExpression>
                                 <PrimaryPrefix isSuper="False" isThis="False">
                                   <Literal Value="true" Type="Boolean"/>
                                 </PrimaryPrefix>
                               </PrimaryExpression>
                      :::
               :::
      </ConditionalAndExpression>
      <ConditionalAndExpression>
               :::
                      :::
                               <PrimaryExpression>
                                 <PrimaryPrefix isSuper="False" isThis="False">
                                   <Literal Value="true" Type="Boolean"/>
                                 </PrimaryPrefix>
                               </PrimaryExpression>
                      :::
               :::
      </ConditionalAndExpression>
    </ConditionalOrExpression>
  </fourth-member>
  <fourth-member>
    <ConditionalOrExpression>
      <ConditionalAndExpression>
               :::
                      :::
                               <PrimaryExpression>
                                 <PrimaryPrefix isSuper="False" isThis="False">
                                   <Literal Value="true" Type="Boolean"/>
                                 </PrimaryPrefix>
                               </PrimaryExpression>
                      :::
               :::
      </ConditionalAndExpression>
      <ConditionalAndExpression>
               :::
                      :::
                               <PrimaryExpression>
                                 <PrimaryPrefix isSuper="False" isThis="False">
                                   <Literal Value="true" Type="Boolean"/>
                                 </PrimaryPrefix>
                               </PrimaryExpression>
                      :::
               :::
      </ConditionalAndExpression>
    </ConditionalOrExpression>
  </fourth-member>
</and-operations>
```

C.2 Zadatak 2

Ispis C.4: Predložena sintaksa

```
<?xml version="1.0" encoding="UTF-8"?>
<static-accesses>
  <access package="" enclosing="" type="DotTest" field="staticInner"/>
  <access package="" enclosing="" type="DotTest" field="staticInner"/>
  <access package="" enclosing="" type="DotTest" field="staticInt"/>
  <access package="" enclosing="" type="DotTest" field="staticInt"/>
  <access package="" enclosing="" type="StaticNested" field="nestedInt"/>
  <access package="" enclosing="DotTest" type="StaticNested" field="nestedInt"/>
  <access package="jezix.srctest" enclosing="" type="DotTest" field="staticInt"/>
  <access package="jezix.srctest" enclosing="DotTest" type="StaticNested" field="IntArray"/>
</static-accesses>
```

Ispis C.5: Badrosova i Mamasova sintaksa

```
<?xml version="1.0" encoding="UTF-8"?>
<static-accesses>
  <access package-and-enclosing="" type="DotTest" field="staticInner"/>
  <access package-and-enclosing="" type="DotTest" field="staticInner"/>
  <access package-and-enclosing="" type="DotTest" field="staticInt"/>
  <access package-and-enclosing="" type="DotTest" field="staticInt"/>
  <access package-and-enclosing="" type="StaticNested" field="nestedInt"/>
  <access package-and-enclosing="DotTest" type="StaticNested" field="nestedInt"/>
  <access package-and-enclosing="jezix.srctest" type="DotTest" field="staticInt"/>
  <access package-and-enclosing="jezix.srctest.DotTest" type="StaticNested" field="IntArray"/>
</static-accesses>
```

C.3 Zadatak 3

Ispis C.6: Predložena sintaksa

```
<?xml version="1.0" encoding="UTF-8"?>
<reference-types>
  <reference-type package="" enclosing="" type="DotTest"/>
  <reference-type package="" enclosing="" type="DotTest"/>
  <reference-type package="" enclosing="" type="DotTest"/>
  <reference-type package="" enclosing="" type="DotTest"/>
  <reference-type package="" enclosing="" type="DotTest"/>
  <reference-type package="" enclosing="" type="DotTest"/>
  <reference-type package="" enclosing="" type="DotTest"/>
  <reference-type package="" enclosing="" type="DotTest"/>
  <reference-type package="" enclosing="" type="DotTest"/>
  <reference-type package="" enclosing="" type="DotTest"/>
  <reference-type package="" enclosing="" type="DotTest"/>
  <reference-type package="" enclosing="" type="DotTest"/>
  <reference-type package="" enclosing="" type="DotTest"/>
  <reference-type package="" enclosing="" type="Inner1"/>
  <reference-type package="" enclosing="" type="Inner1"/>
  <reference-type package="" enclosing="" type="Inner1"/>
  <reference-type package="" enclosing="" type="Inner2"/>
  <reference-type package="" enclosing="" type="Object"/>
  <reference-type package="" enclosing="" type="RuntimeException"/>
  <reference-type package="" enclosing="" type="StaticNested"/>
  <reference-type package="" enclosing="" type="StaticNested"/>
  <reference-type package="" enclosing="" type="StaticNested"/>
  <reference-type package="" enclosing="DotTest" type="Inner1"/>
  <reference-type package="" enclosing="DotTest" type="Inner1"/>
```

```
  <reference-type package="" enclosing="DotTest" type="Inner1"/>
  <reference-type package="" enclosing="DotTest" type="Inner1"/>
  <reference-type package="" enclosing="DotTest" type="Inner1"/>
  <reference-type package="" enclosing="DotTest" type="Inner1"/>
  <reference-type package="" enclosing="DotTest" type="Inner1"/>
  <reference-type package="" enclosing="DotTest" type="Inner1"/>
  <reference-type package="" enclosing="DotTest" type="StaticNested"/>
  <reference-type package="java.lang" enclosing="" type="Class"/>
  <reference-type package="java.lang" enclosing="" type="Cloneable"/>
  <reference-type package="java.lang" enclosing="" type="Exception"/>
  <reference-type package="java.lang" enclosing="" type="RuntimeException"/>
  <reference-type package="java.util" enclosing="" type="Set"/>
  <reference-type package="jezix.srctest" enclosing="" type="DotTest"/>
  <reference-type package="jezix.srctest" enclosing="" type="DotTest"/>
  <reference-type package="jezix.srctest" enclosing="" type="DotTest"/>
  <reference-type package="jezix.srctest" enclosing="" type="DotTest"/>
  <reference-type package="jezix.srctest" enclosing="" type="DotTest"/>
  <reference-type package="jezix.srctest" enclosing="" type="DotTest"/>
  <reference-type package="jezix.srctest" enclosing="DotTest" type="Inner1"/>
  <reference-type package="jezix.srctest" enclosing="DotTest" type="Inner1"/>
  <reference-type package="jezix.srctest" enclosing="DotTest" type="Inner1"/>
  <reference-type package="jezix.srctest" enclosing="DotTest" type="Inner1"/>
  <reference-type package="jezix.srctest" enclosing="DotTest" type="Inner1"/>
  <reference-type package="jezix.srctest" enclosing="DotTest" type="Inner1"/>
  <reference-type package="jezix.srctest" enclosing="DotTest" type="StaticNested"/>
</reference-types>
```

Ispis C.7: Badrosova i Mamasova sintaksa

```
<?xml version="1.0" encoding="UTF-8"?>
<reference-types>
  <reference-type package-and-enclosing="" type="DotTest"/>
  <reference-type package-and-enclosing="" type="DotTest"/>
  <reference-type package-and-enclosing="" type="DotTest"/>
  <reference-type package-and-enclosing="" type="DotTest"/>
  <reference-type package-and-enclosing="" type="DotTest"/>
  <reference-type package-and-enclosing="" type="DotTest"/>
  <reference-type package-and-enclosing="" type="DotTest"/>
  <reference-type package-and-enclosing="" type="DotTest"/>
  <reference-type package-and-enclosing="" type="DotTest"/>
  <reference-type package-and-enclosing="" type="DotTest"/>
  <reference-type package-and-enclosing="" type="DotTest"/>
  <reference-type package-and-enclosing="" type="Inner1"/>
  <reference-type package-and-enclosing="" type="Inner1"/>
  <reference-type package-and-enclosing="" type="Inner1"/>
  <reference-type package-and-enclosing="" type="Inner2"/>
  <reference-type package-and-enclosing="" type="Object"/>
  <reference-type package-and-enclosing="" type="RuntimeException"/>
  <reference-type package-and-enclosing="" type="StaticNested"/>
  <reference-type package-and-enclosing="" type="StaticNested"/>
  <reference-type package-and-enclosing="" type="StaticNested"/>
  <reference-type package-and-enclosing="DotTest" type="Inner1"/>
  <reference-type package-and-enclosing="DotTest" type="Inner1"/>
  <reference-type package-and-enclosing="DotTest" type="Inner1"/>
  <reference-type package-and-enclosing="DotTest" type="Inner1"/>
  <reference-type package-and-enclosing="DotTest" type="Inner1"/>
  <reference-type package-and-enclosing="DotTest" type="Inner1"/>
  <reference-type package-and-enclosing="DotTest" type="Inner1"/>
  <reference-type package-and-enclosing="DotTest" type="Inner1"/>
  <reference-type package-and-enclosing="DotTest" type="StaticNested"/>
  <reference-type package-and-enclosing="java.lang" type="Class"/>
  <reference-type package-and-enclosing="java.lang" type="Cloneable"/>
  <reference-type package-and-enclosing="java.lang" type="Exception"/>
  <reference-type package-and-enclosing="java.lang" type="RuntimeException"/>
```

```
  <reference-type package-and-enclosing="java.util" type="Set"/>
  <reference-type package-and-enclosing="jezix.srctest" type="DotTest"/>
  <reference-type package-and-enclosing="jezix.srctest" type="DotTest"/>
  <reference-type package-and-enclosing="jezix.srctest" type="DotTest"/>
  <reference-type package-and-enclosing="jezix.srctest" type="DotTest"/>
  <reference-type package-and-enclosing="jezix.srctest" type="DotTest"/>
  <reference-type package-and-enclosing="jezix.srctest" type="DotTest"/>
  <reference-type package-and-enclosing="jezix.srctest.DotTest" type="Inner1"/>
  <reference-type package-and-enclosing="jezix.srctest.DotTest" type="Inner1"/>
  <reference-type package-and-enclosing="jezix.srctest.DotTest" type="Inner1"/>
  <reference-type package-and-enclosing="jezix.srctest.DotTest" type="Inner1"/>
  <reference-type package-and-enclosing="jezix.srctest.DotTest" type="Inner1"/>
  <reference-type package-and-enclosing="jezix.srctest.DotTest" type="Inner1"/>
  <reference-type package-and-enclosing="jezix.srctest.DotTest" type="StaticNested"/>
</reference-types>
```

Dodatak D

Definicija sheme za predloženu sintaksu

Ispis D.1: Glavni dokument, jezix-java.xsd

```
<xs:schema targetNamespace="http://tel.fer.hr/jezix/java" xmlns="http://tel.fer.hr/jezix/java"
    xmlns:j="http://tel.fer.hr/jezix/java" xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:include schemaLocation="jezix-java-expression.xsd"/>
  <xs:element name="compilation-unit">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="import" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:choice>
              <xs:element name="reference-type">
                <xs:complexType>
                  <xs:attribute name="name" type="Identifier" use="required"/>
                  <xs:attribute name="package" type="PackageIdentifier" use="required"/>
                </xs:complexType>
              </xs:element>
              <xs:element name="all-from">
                <xs:complexType>
                  <xs:attribute name="package" type="PackageIdentifier" use="required"/>
                </xs:complexType>
              </xs:element>
            </xs:choice>
          </xs:complexType>
        </xs:element>
        <xs:choice maxOccurs="unbounded">
          <xs:element name="class">
            <xs:complexType>
              <xs:complexContent>
                <xs:extension base="ClassDeclaration">
                  <xs:attribute name="access" type="AccessTypeTopLevelClass" use="optional"
                    default="package-private"/>
                  <xs:attribute name="abstract" type="xs:boolean" use="optional" default="false"/>
                </xs:extension>
              </xs:complexContent>
            </xs:complexType>
            <xs:unique name="ClassField">
              <xs:selector xpath="j:field"/>
              <xs:field xpath="@name"/>
            </xs:unique>
```

```
          <xs:unique name="ClassMethod">
            <xs:selector xpath="j:method"/>
            <xs:field xpath="@name"/>
          </xs:unique>
          <xs:unique name="ClassClass">
            <xs:selector xpath="j:class"/>
            <xs:field xpath="@name"/>
          </xs:unique>
          <xs:unique name="ClassInterface">
            <xs:selector xpath="j:interface"/>
            <xs:field xpath="@name"/>
          </xs:unique>
        </xs:element>
        <xs:element name="interface">
          <xs:complexType>
            <xs:complexContent>
              <xs:extension base="InterfaceDeclaration">
                <xs:attribute name="access" type="AccessTypeTopLevelClass"
                  use="optional" default="package-private"/>
              </xs:extension>
            </xs:complexContent>
          </xs:complexType>
          <xs:unique name="InterfaceField">
            <xs:selector xpath="j:field"/>
            <xs:field xpath="@name"/>
          </xs:unique>
          <xs:unique name="InterfaceMethod">
            <xs:selector xpath="j:method"/>
            <xs:field xpath="@name"/>
          </xs:unique>
          <xs:unique name="InterfaceClass">
            <xs:selector xpath="j:class"/>
            <xs:field xpath="@name"/>
          </xs:unique>
          <xs:unique name="InterfaceInterface">
            <xs:selector xpath="j:interface"/>
            <xs:field xpath="@name"/>
          </xs:unique>
        </xs:element>
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="package" type="PackageIdentifier" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:element name="initializer-block">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="do"/>
    </xs:sequence>
    <xs:attribute name="static" type="xs:boolean" use="optional" default="false"/>
  </xs:complexType>
</xs:element>
<xs:element name="constructor">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="ParameterAndThrows"/>
      <xs:element name="do">
        <xs:complexType>
          <xs:sequence>
            <xs:choice>
              <xs:element name="super-constructor-invocation">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="enclosing-instance" type="ReferenceExpressionContainer"
                      minOccurs="0"/>
                    <xs:element name="arguments" type="Arguments" minOccurs="0"/>
```

```
                </xs:sequence>
              </xs:complexType>
            </xs:element>
            <xs:element name="alternate-constructor-invocation">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="arguments" type="Arguments" minOccurs="0"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:choice>
          <xs:group ref="BlockStatementGroup" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
      <xs:unique name="CxVariable">
        <xs:selector xpath="j:variable"/>
        <xs:field xpath="@name"/>
      </xs:unique>
      <xs:unique name="CxClass">
        <xs:selector xpath="j:class"/>
        <xs:field xpath="@name"/>
      </xs:unique>
      <xs:unique name="CxInterface">
        <xs:selector xpath="j:interface"/>
        <xs:field xpath="@name"/>
      </xs:unique>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="access" type="AccessType" use="optional" default="package-private"/>
</xs:complexType>
<xs:unique name="CxParam">
  <xs:selector xpath="j:variable"/>
  <xs:field xpath="@name"/>
</xs:unique>
</xs:element>
<xs:group name="ParameterAndThrows">
  <xs:sequence>
    <xs:element name="throws" type="ReferenceTypeMultiContainer" minOccurs="0"/>
    <xs:element name="parameters" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="variable" maxOccurs="unbounded">
            <xs:complexType>
              <xs:complexContent>
                <xs:extension base="TypeNameContainer">
                  <xs:attribute name="name" type="Identifier" use="required"/>
                  <xs:attribute name="final" type="xs:boolean" use="optional" default="false"/>
                </xs:extension>
              </xs:complexContent>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:group>
<xs:complexType name="VariableDeclarationTemplate">
  <xs:choice>
    <xs:sequence>
      <xs:element ref="primitive-type"/>
      <xs:group ref="PrimitiveExpressionGroup" minOccurs="0"/>
    </xs:sequence>
    <xs:sequence>
      <xs:element ref="reference-type"/>
      <xs:group ref="ReferenceExpressionGroup" minOccurs="0"/>
    </xs:sequence>
```

```
    <xs:sequence>
      <xs:element ref="array-type"/>
      <xs:choice minOccurs="0">
        <xs:group ref="ReferenceExpressionGroup"/>
        <xs:element ref="array-initializer"/>
      </xs:choice>
    </xs:sequence>
  </xs:choice>
  <xs:attribute name="name" type="Identifier" use="required"/>
</xs:complexType>
<xs:complexType name="ConstructWithLabel">
  <xs:attribute name="label" type="Identifier"/>
</xs:complexType>
<xs:complexType name="DoWhile">
  <xs:complexContent>
    <xs:extension base="ConstructWithLabel">
      <xs:sequence>
        <xs:element name="condition" type="BooleanExpressionContainer"/>
        <xs:element ref="do"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:group name="BlockStatementGroup">
  <xs:choice>
    <xs:element ref="assignment"/>
    <xs:element ref="numeric-compound-assignment"/>
    <xs:element ref="boolean-compound-assignment"/>
    <xs:element ref="concat-compound-assignment"/>
    <xs:element ref="method-invocation"/>
    <xs:element ref="new-instance-of-class"/>
    <xs:element ref="new-instance-of-anonymous"/>
    <xs:group ref="IncDecGroup"/>
    <xs:element name="break" type="ConstructWithLabel"/>
    <xs:element name="continue" type="ConstructWithLabel"/>
    <xs:element name="return">
      <xs:complexType>
        <xs:sequence minOccurs="0">
          <xs:group ref="ExpressionGroup"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="throw" type="ReferenceExpressionContainer"/>
    <xs:element name="if">
      <xs:complexType>
        <xs:complexContent>
          <xs:extension base="ConstructWithLabel">
            <xs:sequence>
              <xs:element name="condition" type="ExpressionContainer"/>
              <xs:element ref="do" maxOccurs="2"/>
            </xs:sequence>
          </xs:extension>
        </xs:complexContent>
      </xs:complexType>
    </xs:element>
    <xs:element name="while" type="DoWhile"/>
    <xs:element name="do-while" type="DoWhile"/>
    <xs:element name="for">
      <xs:complexType>
        <xs:complexContent>
          <xs:extension base="ConstructWithLabel">
            <xs:sequence>
              <xs:element name="initialize" minOccurs="0">
                <xs:complexType>
                  <xs:choice>
                    <xs:group ref="ExpressionGroup" maxOccurs="unbounded"/>
```

```
                <xs:element name="variables">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:choice>
                        <xs:element ref="primitive-type"/>
                        <xs:element ref="reference-type"/>
                        <xs:element ref="array-type"/>
                      </xs:choice>
                      <xs:element name="variable" maxOccurs="unbounded">
                        <xs:complexType>
                          <xs:choice minOccurs="0">
                            <xs:group ref="ExpressionGroup"/>
                            <xs:element ref="array-initializer"/>
                          </xs:choice>
                          <xs:attribute name="name" type="Identifier" use="required"/>
                        </xs:complexType>
                      </xs:element>
                    </xs:sequence>
                    <xs:attribute name="final" type="xs:boolean" use="optional"
                      default="false"/>
                  </xs:complexType>
                  <xs:unique name="ForVariables">
                    <xs:selector xpath="j:variable"/>
                    <xs:field xpath="@name"/>
                  </xs:unique>
                </xs:element>
              </xs:choice>
            </xs:complexType>
          </xs:element>
          <xs:element name="condition" type="ExpressionContainer" minOccurs="0"/>
          <xs:element name="update" minOccurs="0">
            <xs:complexType>
              <xs:group ref="ExpressionGroup" maxOccurs="unbounded"/>
            </xs:complexType>
          </xs:element>
          <xs:element ref="do"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
<xs:element name="switch">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="ConstructWithLabel">
        <xs:sequence>
          <xs:element name="test" type="ExpressionContainer"/>
          <xs:element name="switch-block">
            <xs:complexType>
              <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="case" type="NumericExpressionContainer"/>
                <xs:element name="default" type="Empty"/>
                <xs:group ref="BlockStatementGroup"/>
              </xs:choice>
            </xs:complexType>
            <xs:unique name="SwitchVar">
              <xs:selector xpath="j:variable"/>
              <xs:field xpath="@name"/>
            </xs:unique>
            <xs:unique name="SwitchClass">
              <xs:selector xpath="j:class"/>
              <xs:field xpath="@name"/>
            </xs:unique>
            <xs:unique name="SwitchInterface">
              <xs:selector xpath="j:interface"/>
              <xs:field xpath="@name"/>
```

```
            </xs:unique>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
<xs:element name="try">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="ConstructWithLabel">
        <xs:sequence>
          <xs:element ref="do"/>
          <xs:choice>
            <xs:sequence>
              <xs:element name="catch" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="variable">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element ref="reference-type"/>
                          <xs:group ref="ReferenceExpressionGroup" minOccurs="0"/>
                        </xs:sequence>
                        <xs:attribute name="name" type="Identifier" use="required"/>
                        <xs:attribute name="final" type="xs:boolean"/>
                      </xs:complexType>
                    </xs:element>
                    <xs:element ref="do"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element ref="finally" minOccurs="0"/>
            </xs:sequence>
            <xs:element ref="finally"/>
          </xs:choice>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
<xs:element name="do">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="ConstructWithLabel">
        <xs:group ref="BlockStatementGroup" minOccurs="0" maxOccurs="unbounded"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:unique name="BlockVar">
    <xs:selector xpath="j:variable"/>
    <xs:field xpath="@name"/>
  </xs:unique>
  <xs:unique name="BlockClass">
    <xs:selector xpath="j:class"/>
    <xs:field xpath="@name"/>
  </xs:unique>
  <xs:unique name="BlockInterface">
    <xs:selector xpath="j:interface"/>
    <xs:field xpath="@name"/>
  </xs:unique>
</xs:element>
<xs:element name="synchronized">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="ReferenceExpressionGroup"/>
```

```
          <xs:element ref="do"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="variable">
      <xs:complexType>
        <xs:complexContent>
          <xs:extension base="VariableDeclarationTemplate">
            <xs:attribute name="final" type="xs:boolean" use="optional" default="false"/>
          </xs:extension>
        </xs:complexContent>
      </xs:complexType>
    </xs:element>
    <xs:element name="class" type="ClassDeclaration"/>
  </xs:choice>
</xs:group>
<xs:element name="finally">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="do"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="do">
  <xs:complexType>
    <xs:group ref="BlockStatementGroup" minOccurs="0" maxOccurs="unbounded"/>
  </xs:complexType>
  <xs:unique name="DoVar">
    <xs:selector xpath="j:variable"/>
    <xs:field xpath="@name"/>
  </xs:unique>
  <xs:unique name="DoClass">
    <xs:selector xpath="j:class"/>
    <xs:field xpath="@name"/>
  </xs:unique>
  <xs:unique name="DoInterface">
    <xs:selector xpath="j:interface"/>
    <xs:field xpath="@name"/>
  </xs:unique>
</xs:element>
<xs:complexType name="MethodDeclarationBase">
  <xs:sequence>
    <xs:element name="returns" type="TypeNameContainer" minOccurs="0"/>
    <xs:group ref="ParameterAndThrows"/>
  </xs:sequence>
  <xs:attribute name="name" type="Identifier" use="required"/>
</xs:complexType>
<xs:complexType name="StaticSpecifics">
  <xs:choice>
    <xs:element ref="do"/>
    <xs:element name="native" type="Empty"/>
  </xs:choice>
  <xs:attribute name="synchronized" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="strictfp" type="xs:boolean" use="optional" default="false"/>
</xs:complexType>
<xs:complexType name="InstanceSpecifics">
  <xs:complexContent>
    <xs:extension base="StaticSpecifics">
      <xs:attribute name="final" type="xs:boolean" use="optional" default="false"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="FieldDeclaration">
  <xs:complexContent>
    <xs:extension base="VariableDeclarationTemplate">
      <xs:attribute name="access" type="AccessType" use="optional" default="package-private"/>
```

```
      <xs:attribute name="static" type="xs:boolean" use="optional" default="false"/>
      <xs:attribute name="final" type="xs:boolean" use="optional" default="false"/>
      <xs:attribute name="transient" type="xs:boolean" use="optional" default="false"/>
      <xs:attribute name="volatile" type="xs:boolean" use="optional" default="false"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="ClassDeclaration">
  <xs:sequence>
    <xs:element name="extends" type="ReferenceTypeContainer" minOccurs="0"/>
    <xs:element name="implements" type="ReferenceTypeMultiContainer" minOccurs="0"/>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="field" type="FieldDeclaration"/>
      <xs:element ref="initializer-block"/>
      <xs:element ref="constructor"/>
      <xs:element name="method">
        <xs:complexType>
          <xs:complexContent>
            <xs:extension base="MethodDeclarationBase">
              <xs:choice>
                <xs:element name="abstract" type="Empty"/>
                <xs:element name="static" type="StaticSpecifics"/>
                <xs:element name="instance" type="InstanceSpecifics"/>
              </xs:choice>
              <xs:attribute name="access" type="AccessType" use="optional" default="package-private"/>
            </xs:extension>
          </xs:complexContent>
        </xs:complexType>
        <xs:unique name="ClMetPar">
          <xs:selector xpath="j:variable"/>
          <xs:field xpath="@name"/>
        </xs:unique>
      </xs:element>
      <xs:element name="interface">
        <xs:complexType>
          <xs:complexContent>
            <xs:extension base="InterfaceDeclaration">
              <xs:attribute name="access" type="AccessType" use="optional" default="package-private"/>
              <xs:attribute name="static" type="xs:boolean" use="optional" default="false"/>
            </xs:extension>
          </xs:complexContent>
        </xs:complexType>
        <xs:unique name="InterfaceInClassField">
          <xs:selector xpath="j:field"/>
          <xs:field xpath="@name"/>
        </xs:unique>
        <xs:unique name="InterfaceInClassMethod">
          <xs:selector xpath="j:method"/>
          <xs:field xpath="@name"/>
        </xs:unique>
        <xs:unique name="InterfaceInClassClass">
          <xs:selector xpath="j:class"/>
          <xs:field xpath="@name"/>
        </xs:unique>
        <xs:unique name="InterfaceInClassInterface">
          <xs:selector xpath="j:interface"/>
          <xs:field xpath="@name"/>
        </xs:unique>
      </xs:element>
      <xs:element name="class">
        <xs:complexType>
          <xs:complexContent>
            <xs:extension base="ClassDeclaration">
              <xs:attribute name="access" type="AccessType" use="optional" default="package-private"/>
              <xs:attribute name="static" type="xs:boolean" use="optional" default="false"/>
            </xs:extension>
```

```
          </xs:complexContent>
        </xs:complexType>
        <xs:unique name="ClassInClassField">
          <xs:selector xpath="j:field"/>
          <xs:field xpath="@name"/>
        </xs:unique>
        <xs:unique name="ClassInClassMethod">
          <xs:selector xpath="j:method"/>
          <xs:field xpath="@name"/>
        </xs:unique>
        <xs:unique name="ClassInClassClass">
          <xs:selector xpath="j:class"/>
          <xs:field xpath="@name"/>
        </xs:unique>
        <xs:unique name="ClassInClassInterface">
          <xs:selector xpath="j:interface"/>
          <xs:field xpath="@name"/>
        </xs:unique>
      </xs:element>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="name" type="Identifier" use="required"/>
  <xs:attribute name="strictfp" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="final" type="xs:boolean" use="optional" default="false"/>
</xs:complexType>
<xs:complexType name="InterfaceDeclaration">
  <xs:sequence>
    <xs:element name="extends" type="ReferenceTypeMultiContainer" minOccurs="0"/>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="class">
        <xs:complexType>
          <xs:complexContent>
            <xs:extension base="ClassDeclaration">
              <xs:attribute name="access" type="AccessType" use="optional" fixed="public"/>
              <xs:attribute name="static" type="xs:boolean" use="optional" fixed="true"/>
            </xs:extension>
          </xs:complexContent>
        </xs:complexType>
        <xs:unique name="ClassInInterfaceField">
          <xs:selector xpath="j:field"/>
          <xs:field xpath="@name"/>
        </xs:unique>
        <xs:unique name="ClassInInterfaceMethod">
          <xs:selector xpath="j:method"/>
          <xs:field xpath="@name"/>
        </xs:unique>
        <xs:unique name="ClassInInterfaceClass">
          <xs:selector xpath="j:class"/>
          <xs:field xpath="@name"/>
        </xs:unique>
        <xs:unique name="ClassInInterfaceInterface">
          <xs:selector xpath="j:interface"/>
          <xs:field xpath="@name"/>
        </xs:unique>
      </xs:element>
      <xs:element name="interface">
        <xs:complexType>
          <xs:complexContent>
            <xs:extension base="InterfaceDeclaration">
              <xs:attribute name="access" type="AccessType" use="optional" fixed="public"/>
              <xs:attribute name="static" type="xs:boolean" use="optional" fixed="true"/>
            </xs:extension>
          </xs:complexContent>
        </xs:complexType>
        <xs:unique name="InterfaceInInterfaceField">
          <xs:selector xpath="j:field"/>
```

```
          <xs:field xpath="@name"/>
        </xs:unique>
        <xs:unique name="InterfaceInInterfaceMethod">
          <xs:selector xpath="j:method"/>
          <xs:field xpath="@name"/>
        </xs:unique>
        <xs:unique name="InterfaceInInterfaceClass">
          <xs:selector xpath="j:class"/>
          <xs:field xpath="@name"/>
        </xs:unique>
        <xs:unique name="InterfaceInInterfaceInterface">
          <xs:selector xpath="j:interface"/>
          <xs:field xpath="@name"/>
        </xs:unique>
      </xs:element>
      <xs:element name="method">
        <xs:complexType>
          <xs:complexContent>
            <xs:extension base="MethodDeclarationBase">
              <xs:attribute name="access" type="AccessTypeAbstractMethod" use="optional" fixed="public"/>
            </xs:extension>
          </xs:complexContent>
        </xs:complexType>
        <xs:unique name="InterfaceMethodParam">
          <xs:selector xpath="j:variable"/>
          <xs:field xpath="@name"/>
        </xs:unique>
      </xs:element>
      <xs:element name="field" type="VariableDeclarationTemplate"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="name" type="Identifier" use="required"/>
  <xs:attribute name="strictfp" type="xs:boolean" use="optional" default="false"/>
 </xs:complexType>
</xs:schema>
```

Ispis D.2: Pomoćni dokument 1 (izrazi), jezix-java-expression.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://tel.fer.hr/jezix/java" xmlns="http://tel.fer.hr/jezix/java"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:jml="http://tel.fer.hr/jezix/java"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:include schemaLocation="jezix-java-simplestuff.xsd"/>
  <xs:complexType name="MemberAccess">
    <xs:choice>
      <xs:element ref="reference-type"/>
      <xs:group ref="ReferenceExpressionGroup"/>
      <xs:element name="from-superclass" type="ReferenceTypeOptContainer"/>
    </xs:choice>
    <xs:attribute name="name" type="Identifier" use="required"/>
  </xs:complexType>
  <xs:complexType name="Arguments">
    <xs:sequence>
      <xs:group ref="ExpressionGroup" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="array-initializer">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:group ref="ExpressionGroup"/>
        <xs:element ref="array-initializer"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>
```

```
<xs:complexType name="Assignment">
  <xs:sequence>
    <xs:group ref="AssignableGroup"/>
    <xs:group ref="ExpressionGroup"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="assignment" type="Assignment"/>
<xs:element name="numeric-compound-assignment">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="AssignableGroup"/>
      <xs:group ref="NumericExpressionGroup"/>
    </xs:sequence>
    <xs:attribute name="type" type="NumericCompoundAssignmentType"/>
  </xs:complexType>
</xs:element>
<xs:element name="boolean-compound-assignment">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="AssignableGroup"/>
      <xs:group ref="BooleanExpressionGroup"/>
    </xs:sequence>
    <xs:attribute name="type" type="BooleanCompoundAssignmentType"/>
  </xs:complexType>
</xs:element>
<xs:element name="concat-compound-assignment" type="Assignment"/>
<xs:element name="method-invocation">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="MemberAccess">
        <xs:sequence>
          <xs:element name="arguments" type="Arguments" minOccurs="0"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
<xs:group name="AssignableGroup">
  <xs:choice>
    <xs:element name="variable-access">
      <xs:complexType>
        <xs:attribute name="name" type="Identifier" use="required"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="field-access" type="MemberAccess"/>
    <xs:element name="array-access">
      <xs:complexType>
        <xs:sequence>
          <xs:group ref="ReferenceExpressionGroup"/>
          <xs:element name="array-index" type="NumericExpressionContainer" minOccurs="0"
            maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:choice>
</xs:group>
<xs:complexType name="NumericRelation">
  <xs:group ref="NumericExpressionGroup" minOccurs="2" maxOccurs="2"/>
</xs:complexType>
<xs:complexType name="Equality">
  <xs:group ref="ExpressionGroup" minOccurs="2" maxOccurs="2"/>
</xs:complexType>
<xs:complexType name="BinaryBooleanExpr">
  <xs:group ref="BooleanExpressionGroup" minOccurs="2" maxOccurs="unbounded"/>
</xs:complexType>
<xs:group name="PureBooleanExpressionGroup">
```

```
  <xs:choice>
    <xs:element ref="boolean"/>
    <xs:element name="less-than" type="NumericRelation"/>
    <xs:element name="less-or-equal" type="NumericRelation"/>
    <xs:element name="greater-than" type="NumericRelation"/>
    <xs:element name="greater-or-equal" type="NumericRelation"/>
    <xs:element name="instance-of">
      <xs:complexType>
        <xs:sequence>
          <xs:group ref="ReferenceExpressionGroup"/>
          <xs:choice>
            <xs:element ref="reference-type"/>
            <xs:element ref="array-type"/>
          </xs:choice>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="equal" type="Equality"/>
    <xs:element name="not-equal" type="Equality"/>
    <xs:element name="not" type="BooleanExpressionContainer"/>
    <xs:element name="and" type="BinaryBooleanExpr"/>
    <xs:element name="xor" type="BinaryBooleanExpr"/>
    <xs:element name="or" type="BinaryBooleanExpr"/>
    <xs:element name="conditional-and" type="BinaryBooleanExpr"/>
    <xs:element name="conditional-or" type="BinaryBooleanExpr"/>
    <xs:element ref="boolean-compound-assignment"/>
  </xs:choice>
</xs:group>
<xs:group name="IncDecGroup">
  <xs:choice>
    <xs:element name="pre-increment" type="NumericExpressionContainer"/>
    <xs:element name="pre-decrement" type="NumericExpressionContainer"/>
    <xs:element name="post-increment" type="NumericExpressionContainer"/>
    <xs:element name="post-decrement" type="NumericExpressionContainer"/>
  </xs:choice>
</xs:group>
<xs:complexType name="BinaryBitwiseOperation">
  <xs:group ref="NumericExpressionGroup" minOccurs="2" maxOccurs="unbounded"/>
</xs:complexType>
<xs:group name="PureNumericExpressionGroup">
  <xs:choice>
    <xs:element ref="char"/>
    <xs:element ref="int"/>
    <xs:element ref="long"/>
    <xs:element ref="float"/>
    <xs:element ref="double"/>
    <xs:element name="bitwise-not" type="NumericExpressionContainer"/>
    <xs:element name="bitwise-and" type="BinaryBitwiseOperation"/>
    <xs:element name="bitwise-xor" type="BinaryBitwiseOperation"/>
    <xs:element name="bitwise-or" type="BinaryBitwiseOperation"/>
    <xs:element name="primitive-cast">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="primitive-type">
            <xs:complexType>
              <xs:attribute name="name" type="NumericTypeName"/>
            </xs:complexType>
          </xs:element>
          <xs:group ref="NumericExpressionGroup"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element ref="numeric-compound-assignment"/>
    <xs:element name="unary-plus" type="NumericExpressionContainer"/>
    <xs:element name="unary-minus" type="NumericExpressionContainer"/>
    <xs:element name="multiplicative-expression">
```

```
      <xs:complexType>
        <xs:sequence>
          <xs:group ref="NumericExpressionGroup"/>
          <xs:sequence maxOccurs="unbounded">
            <xs:element name="operator" type="MultiplicativeOperator"/>
            <xs:group ref="NumericExpressionGroup"/>
          </xs:sequence>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="additive-expression">
      <xs:complexType>
        <xs:sequence>
          <xs:group ref="NumericExpressionGroup"/>
          <xs:sequence maxOccurs="unbounded">
            <xs:element name="operator" type="AdditiveOperator"/>
            <xs:group ref="NumericExpressionGroup"/>
          </xs:sequence>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="shift-expression">
      <xs:complexType>
        <xs:sequence>
          <xs:group ref="NumericExpressionGroup"/>
          <xs:sequence maxOccurs="unbounded">
            <xs:element name="operator" type="ShiftOperator"/>
            <xs:group ref="NumericExpressionGroup"/>
          </xs:sequence>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:choice>
</xs:group>
<xs:complexType name="NewInstanceOfClass">
  <xs:sequence>
    <xs:element ref="reference-type"/>
    <xs:element name="enclosing-instance" type="ReferenceExpressionContainer" minOccurs="0"/>
    <xs:element name="arguments" type="Arguments" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="new-instance-of-class" type="NewInstanceOfClass"/>
<xs:element name="new-instance-of-anonymous">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="NewInstanceOfClass">
        <xs:sequence>
          <xs:element name="anonymous-class">
            <xs:complexType>
              <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element ref="initializer-block"/>
                <xs:element name="method">
                  <xs:complexType>
                    <xs:complexContent>
                      <xs:extension base="MethodDeclarationBase">
                        <xs:choice>
                          <xs:element name="instance" type="InstanceSpecifics"/>
                        </xs:choice>
                        <xs:attribute name="access" type="AccessType" use="optional"
                          default="package-private"/>
                      </xs:extension>
                    </xs:complexContent>
                  </xs:complexType>
                </xs:element>
                <xs:element ref="field"/>
              </xs:choice>
```

```
            </xs:complexType>
            <xs:unique name="FieldAnon">
              <xs:selector xpath="field"/>
              <xs:field xpath="@name"/>
            </xs:unique>
            <xs:unique name="MethodAnon">
              <xs:selector xpath="method"/>
              <xs:field xpath="@name"/>
            </xs:unique>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
<xs:group name="PureReferenceExpressionGroup">
  <xs:choice>
    <xs:element ref="new-instance-of-class"/>
    <xs:element ref="new-instance-of-anonymous"/>
    <xs:element name="this" type="ReferenceTypeOptContainer"/>
    <xs:element name="class-literal">
      <xs:complexType>
        <xs:choice>
          <xs:element ref="reference-type"/>
          <xs:element ref="primitive-type"/>
          <xs:element ref="array-type"/>
        </xs:choice>
      </xs:complexType>
    </xs:element>
    <xs:element name="void-class-literal" type="Empty"/>
    <xs:element name="new-instance-of-array">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="array-type"/>
          <xs:element name="allocated-dim" type="NumericExpressionContainer" minOccurs="0"
            maxOccurs="unbounded"/>
          <xs:element ref="array-initializer" minOccurs="0"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="reference-cast">
      <xs:complexType>
        <xs:choice>
          <xs:sequence>
            <xs:element ref="reference-type"/>
            <xs:group ref="ReferenceExpressionGroup"/>
          </xs:sequence>
          <xs:sequence>
            <xs:element ref="array-type"/>
            <xs:group ref="ReferenceExpressionGroup"/>
          </xs:sequence>
        </xs:choice>
      </xs:complexType>
    </xs:element>
    <xs:element ref="string"/>
    <xs:element ref="null"/>
    <xs:element name="concatenation">
      <xs:complexType>
        <xs:group ref="ExpressionGroup" minOccurs="2" maxOccurs="unbounded"/>
      </xs:complexType>
    </xs:element>
    <xs:element ref="concat-compound-assignment"/>
  </xs:choice>
</xs:group>
<xs:group name="AmbiguousExpressionGroup">
  <xs:choice>
```

```
      <xs:group ref="AssignableGroup"/>
      <xs:element ref="method-invocation"/>
      <xs:element name="conditional">
        <xs:complexType>
          <xs:sequence>
            <xs:group ref="BooleanExpressionGroup"/>
            <xs:group ref="ExpressionGroup" minOccurs="2" maxOccurs="2"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element ref="assignment"/>
    </xs:choice>
  </xs:group>
  <xs:group name="BooleanExpressionGroup">
    <xs:choice>
      <xs:group ref="AmbiguousExpressionGroup"/>
      <xs:group ref="PureBooleanExpressionGroup"/>
    </xs:choice>
  </xs:group>
  <xs:group name="NumericExpressionGroup">
    <xs:choice>
      <xs:group ref="AmbiguousExpressionGroup"/>
      <xs:group ref="IncDecGroup"/>
      <xs:group ref="PureNumericExpressionGroup"/>
    </xs:choice>
  </xs:group>
  <xs:group name="PrimitiveExpressionGroup">
    <xs:choice>
      <xs:group ref="AmbiguousExpressionGroup"/>
      <xs:group ref="PureBooleanExpressionGroup"/>
      <xs:group ref="IncDecGroup"/>
      <xs:group ref="PureNumericExpressionGroup"/>
    </xs:choice>
  </xs:group>
  <xs:group name="ReferenceExpressionGroup">
    <xs:choice>
      <xs:group ref="AmbiguousExpressionGroup"/>
      <xs:group ref="PureReferenceExpressionGroup"/>
    </xs:choice>
  </xs:group>
  <xs:group name="ExpressionGroup">
    <xs:choice>
      <xs:group ref="AmbiguousExpressionGroup"/>
      <xs:group ref="IncDecGroup"/>
      <xs:group ref="PureNumericExpressionGroup"/>
      <xs:group ref="PureBooleanExpressionGroup"/>
      <xs:group ref="PureReferenceExpressionGroup"/>
    </xs:choice>
  </xs:group>
  <xs:complexType name="ExpressionContainer">
    <xs:group ref="ExpressionGroup"/>
  </xs:complexType>
  <xs:complexType name="BooleanExpressionContainer">
    <xs:group ref="BooleanExpressionGroup"/>
  </xs:complexType>
  <xs:complexType name="NumericExpressionContainer">
    <xs:group ref="NumericExpressionGroup"/>
  </xs:complexType>
  <xs:complexType name="PrimitiveExpressionContainer">
    <xs:group ref="PrimitiveExpressionGroup"/>
  </xs:complexType>
  <xs:complexType name="ReferenceExpressionContainer">
    <xs:group ref="ReferenceExpressionGroup"/>
  </xs:complexType>
</xs:schema>
```

Ispis D.3: Pomoćni dokument 2, jezix-java-simplestuff.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://tel.fer.hr/jezix/java" xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://tel.fer.hr/jezix/java" elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="array-type">
    <xs:complexType>
      <xs:group ref="SimpleTypeNameGroup"/>
      <xs:attribute name="dimensions">
        <xs:simpleType>
          <xs:restriction base="xs:int">
            <xs:minInclusive value="1"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
  <xs:group name="SimpleTypeNameGroup">
    <xs:choice>
      <xs:element ref="primitive-type"/>
      <xs:element ref="reference-type"/>
    </xs:choice>
  </xs:group>
  <xs:group name="TypeNameGroup">
    <xs:choice>
      <xs:group ref="SimpleTypeNameGroup"/>
      <xs:element ref="array-type"/>
    </xs:choice>
  </xs:group>
  <xs:complexType name="TypeName">
    <xs:group ref="TypeNameGroup"/>
  </xs:complexType>
  <xs:complexType name="ReferenceTypeContainer">
    <xs:sequence>
      <xs:element ref="reference-type"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ReferenceTypeOptContainer">
    <xs:sequence>
      <xs:element ref="reference-type" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ReferenceTypeMultiContainer">
    <xs:sequence>
      <xs:element ref="reference-type" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="reference-type">
    <xs:complexType>
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:element name="enclosing-type">
          <xs:complexType>
            <xs:attribute name="name" type="Identifier" use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="name" type="Identifier" use="required"/>
      <xs:attribute name="package" type="PackageIdentifier" use="optional"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="primitive-type">
    <xs:complexType>
      <xs:attribute name="name" type="PrimitiveTypeName" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="Empty"/>
```

```
<xs:simpleType name="Identifier">
  <xs:restriction base="xs:NCName">
    <xs:pattern value="[^.\-]+"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="PackageIdentifier">
  <xs:restriction base="xs:NCName">
    <xs:pattern value="[^\-]+"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="AccessTypeTopLevelClass">
  <xs:restriction base="xs:string">
    <xs:enumeration value="public"/>
    <xs:enumeration value="package-private"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="AccessTypeStaticMethod">
  <xs:union memberTypes="AccessTypeTopLevelClass">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="private"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>
<xs:simpleType name="AccessTypeAbstractMethod">
  <xs:union memberTypes="AccessTypeTopLevelClass">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="protected"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>
<xs:simpleType name="AccessType">
  <xs:union memberTypes="AccessTypeStaticMethod AccessTypeAbstractMethod"/>
</xs:simpleType>
<xs:simpleType name="NumericTypeName">
  <xs:restriction base="xs:string">
    <xs:enumeration value="byte"/>
    <xs:enumeration value="short"/>
    <xs:enumeration value="char"/>
    <xs:enumeration value="int"/>
    <xs:enumeration value="long"/>
    <xs:enumeration value="float"/>
    <xs:enumeration value="double"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="PrimitiveTypeName">
  <xs:union memberTypes="NumericTypeName">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="boolean"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>
<xs:element name="boolean">
  <xs:complexType>
    <xs:attribute name="value" type="xs:boolean"/>
  </xs:complexType>
</xs:element>
<xs:element name="string">
  <xs:complexType>
    <xs:attribute name="value" type="xs:string"/>
  </xs:complexType>
```

```
</xs:element>
<xs:element name="null">
  <xs:complexType/>
</xs:element>
<!--xs:element name="byte">
  <xs:complexType>
    <xs:attribute name="value" type="xs:byte"/>
  </xs:complexType>
</xs:element>
<xs:element name="short">
  <xs:complexType>
    <xs:attribute name="value" type="xs:short"/>
  </xs:complexType>
</xs:element-->
<xs:element name="char">
  <xs:complexType>
    <xs:attribute name="value">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:length value="1"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>
<xs:element name="int">
  <xs:complexType>
    <xs:attribute name="value" type="xs:int"/>
  </xs:complexType>
</xs:element>
<xs:element name="long">
  <xs:complexType>
    <xs:attribute name="value" type="xs:long"/>
  </xs:complexType>
</xs:element>
<xs:element name="float">
  <xs:complexType>
    <xs:attribute name="value" type="xs:float"/>
  </xs:complexType>
</xs:element>
<xs:element name="double">
  <xs:complexType>
    <xs:attribute name="value" type="xs:double"/>
  </xs:complexType>
</xs:element>
<xs:simpleType name="MultiplicativeOperatorType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="multiplication"/>
    <xs:enumeration value="division"/>
    <xs:enumeration value="remainder"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="AdditiveOperatorType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="addition"/>
    <xs:enumeration value="subtraction"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="ShiftOperatorType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="left-shift"/>
    <xs:enumeration value="right-shift"/>
    <xs:enumeration value="right-shift-unsigned"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="MultiplicativeOperator">
```

```
    <xs:attribute name="type" type="MultiplicativeOperatorType" use="required"/>
  </xs:complexType>
  <xs:complexType name="AdditiveOperator">
    <xs:attribute name="type" type="AdditiveOperatorType" use="required"/>
  </xs:complexType>
  <xs:complexType name="ShiftOperator">
    <xs:attribute name="type" type="ShiftOperatorType" use="required"/>
  </xs:complexType>
  <xs:simpleType name="NumericCompoundAssignmentType">
    <xs:union memberTypes="MultiplicativeOperatorType AdditiveOperatorType ShiftOperatorType">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="bitwise-and"/>
          <xs:enumeration value="bitwise-xor"/>
          <xs:enumeration value="bitwise-or"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:union>
  </xs:simpleType>
  <xs:simpleType name="BooleanCompoundAssignmentType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="and"/>
      <xs:enumeration value="xor"/>
      <xs:enumeration value="or"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

Dodatak E

XSLT-ove transformacije za ortogonalna proširenja

Ispis E.1: Transformacija za generiranje dokumentacije

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns="http://www.w3.org/1999/xhtml"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:jd="http://tel.fer.hr/jezix/java/doc"
    xmlns:j="http://tel.fer.hr/jezix/java">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
  <xsl:template match="jd:*"/>
  <xsl:template match="j:*"/>
  <xsl:template match="/j:compilation-unit">
    <html>
      <head>
        <title>API documentation</title>
      </head>
      <body>
        <xsl:apply-templates select="@package | *"/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="@package">
    <p>Package <big>
        <b>
          <code>
            <xsl:value-of select="."/>
          </code>
        </b>
      </big>
    </p>
  </xsl:template>
  <xsl:template match="j:class">
    <h2>Class <xsl:value-of select="@name"/>
    </h2>
    <p>
      <xsl:apply-templates select="jd:description"/>
    </p>
    <xsl:call-template name="Version"/>
    <xsl:call-template name="Authors"/>
    <xsl:apply-templates select="j:field"/>
    <xsl:apply-templates select="j:constructor"/>
    <xsl:apply-templates select="j:method" mode="Class"/>
```

```
</xsl:template>
<xsl:template match="j:interface">
  <h2>Interface <xsl:value-of select="@name"/>
  </h2>
  <xsl:call-template name="Version"/>
  <xsl:call-template name="Authors"/>
  <p>
    <xsl:apply-templates select="jd:description"/>
  </p>
  <xsl:apply-templates select="j:field"/>
  <xsl:apply-templates select="j:method" mode="Interface"/>
</xsl:template>
<xsl:template match="j:field">
  <h4>Field: <code>
      <xsl:call-template name="TypeNameGroup"/>
      <xsl:text> </xsl:text>
      <xsl:value-of select="@name"/>
    </code>
  </h4>
  <xsl:apply-templates select="jd:description"/>
</xsl:template>
<xsl:template match="j:constructor">
  <h4>Constructor</h4>
  <xsl:call-template name="MethodDetail"/>
</xsl:template>
<xsl:template match="j:method" mode="Interface">
  <h4>
    <xsl:text>Method </xsl:text>
    <code>
      <xsl:value-of select="@name"/>
    </code>
  </h4>
  <xsl:call-template name="MethodDetail"/>
</xsl:template>
<xsl:template match="j:method" mode="Class">
  <h4>
    <xsl:choose>
      <xsl:when test="j:instance">
        <xsl:text>Instance</xsl:text>
      </xsl:when>
      <xsl:when test="j:abstract">
        <xsl:text>Abstract</xsl:text>
      </xsl:when>
      <xsl:when test="j:static">
        <xsl:text>Class</xsl:text>
      </xsl:when>
    </xsl:choose>
    <xsl:text> method </xsl:text>
    <code>
      <xsl:value-of select="@name"/>
    </code>
  </h4>
  <xsl:apply-templates mode="MethodDetail"
    select="j:instance | j:abstract | j:static"/>
  <xsl:apply-templates select="j:*/j:returns"/>
</xsl:template>
<xsl:template match="j:returns">
  <p>Returns
    <code>
      <xsl:call-template name="TypeNameGroup"/>
    </code>
    <xsl:text> -- </xsl:text>
    <xsl:apply-templates select="jd:description"/>
  </p>
</xsl:template>
<xsl:template match="jd:description">
```

```
  <xsl:copy-of select="node()"/>
</xsl:template>
<xsl:template name="CommaSeparator">
  <xsl:if test="not( position() = last() )">
    <xsl:text>, </xsl:text>
  </xsl:if>
</xsl:template>
<xsl:template name="MethodDetail" mode="MethodDetail" match="j:*">
  <p>Signature: <code>
      <xsl:for-each select="j:parameter">
        <xsl:call-template name="TypeNameGroup"/>
        <xsl:call-template name="CommaSeparator"/>
      </xsl:for-each>
    </code>
  </p>
  <p>
    <xsl:apply-templates select="jd:description"/>
  </p>
  <p>Parameters:
    <ul>
      <xsl:for-each select="j:parameter">
        <li>
          <code>
            <xsl:call-template name="TypeNameGroup"/>
            <xsl:text> </xsl:text>
            <xsl:value-of select="@name"/>
            <xsl:text> </xsl:text>
          </code>
          <xsl:apply-templates select="jd:description"/>
        </li>
      </xsl:for-each>
    </ul>
  </p>
  <xsl:if test="j:throws">
    <p>Throws:
      <ul>
        <xsl:for-each select="j:throws">
          <li>
            <code>
              <xsl:call-template name="ReferenceTypeName"/>
            </code>
            <xsl:text> -- </xsl:text>
            <xsl:apply-templates select="jd:description"/>
          </li>
        </xsl:for-each>
      </ul>
    </p>
  </xsl:if>
</xsl:template>
<xsl:template name="Version">
  <xsl:if test="jd:version">
    <p>Version: <xsl:value-of select="jd:version/@value"/>
    </p>
  </xsl:if>
</xsl:template>
<xsl:template name="Authors">
  <xsl:if test="jd:author">
    <p>Author(s):
      <xsl:for-each select="jd:author">
        <xsl:value-of select="@value"/>
        <xsl:call-template name="CommaSeparator"/>
      </xsl:for-each>
    </p>
  </xsl:if>
</xsl:template>
<xsl:template mode="TypeName" match="*">
```

```
    <xsl:choose>
      <xsl:when test="primitive-type">
        <xsl:value-of select="primitive-type/@name"/>
      </xsl:when>
      <xsl:when test="reference-type">
        <xsl:value-of select="primitive-type/@name"/>
      </xsl:when>
    </xsl:choose>
  </xsl:template>
  <xsl:template name="ReferenceTypeName" mode="ReferenceTypeName"
      match="*">
    <xsl:if test="@package">
      <xsl:value-of select="@package"/>
      <xsl:text>.</xsl:text>
    </xsl:if>
    <xsl:for-each select="j:enclosing-type">
      <xsl:value-of select="@name"/>
      <xsl:text>.</xsl:text>
    </xsl:for-each>
    <xsl:value-of select="@name"/>
  </xsl:template>
  <xsl:template mode="SimpleTypeNameGroup" match="j:primitive-type">
    <xsl:value-of select="@name"/>
  </xsl:template>
  <xsl:template mode="SimpleTypeNameGroup" match="j:reference-type">
    <xsl:call-template name="ReferenceTypeName"/>
  </xsl:template>
  <xsl:template mode="SimpleTypeNameGroup" match="j:array-type">
    <xsl:apply-templates mode="SimpleTypeNameGroup" select="*"/>
    <xsl:call-template name="Brackets">
      <xsl:with-param name="count" select="@dimensions"/>
    </xsl:call-template>
  </xsl:template>
  <xsl:template name="Brackets">
    <xsl:param name="count"/>
    <xsl:if test="$count > 0">
      <xsl:text>[]</xsl:text>
      <xsl:call-template name="Brackets">
        <xsl:with-param name="count" select="$count - 1"/>
      </xsl:call-template>
    </xsl:if>
  </xsl:template>
  <xsl:template name="TypeNameGroup" mode="TypeNameGroup" match="*">
    <xsl:apply-templates mode="SimpleTypeNameGroup"
      select="j:primitive-type | j:reference-type | j:array-type"/>
  </xsl:template>
</xsl:stylesheet>
```

Ispis E.2: Transformacija za generiranje metode `equals`

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://tel.fer.hr/jezix/java"
    xmlns:j="http://tel.fer.hr/jezix/java"
    xmlns:t="http://tel.fer.hr/jezix/java/template-equals">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
  <xsl:variable name="thatVar">that</xsl:variable>
  <xsl:variable name="paramName">o</xsl:variable>
  <xsl:template match="@* | node()">
    <xsl:copy>
      <xsl:apply-templates select="@* | node()"/>
    </xsl:copy>
  </xsl:template>
  <xsl:template match="j:compilation-unit">
```

```
  <compilation-unit>
    <xsl:apply-templates select="@* | node()"/>
  </compilation-unit>
</xsl:template>
<xsl:template match="j:method[ t:template-equals ]">
  <method name="equals" access="public">
    <xsl:apply-templates select="t:template-equals" mode="InsideEquals"/>
  </method>
</xsl:template>
<xsl:template match="t:template-equals" mode="InsideEquals">
  <xsl:copy-of select="."/>
  <returns>
    <primitive-type name="boolean"/>
  </returns>
  <parameters>
    <variable>
      <xsl:attribute name="name"><xsl:value-of select="$paramName"/></xsl:attribute>
      <reference-type name="Object" package="java.lang"/>
    </variable>
  </parameters>
  <instance>
    <body>
      <xsl:variable name="typeName">
        <xsl:value-of select="ancestor::j:class[ 1 ]/@name"/>
      </xsl:variable>
      <variable>
        <xsl:attribute name="name"><xsl:value-of select="$thatVar"/></xsl:attribute>
        <reference-type>
          <xsl:attribute name="name"><xsl:value-of select="$typeName"/></xsl:attribute>
        </reference-type>
        <reference-cast>
          <reference-type>
            <xsl:attribute name="name"><xsl:value-of select="$typeName"/></xsl:attribute>
          </reference-type>
          <variable-access>
            <xsl:attribute name="name"><xsl:value-of select="$paramName"/></xsl:attribute>
          </variable-access>
        </reference-cast>
      </variable>
      <return>
        <xsl:apply-templates select="t:conditions" mode="InsideEquals"/>
      </return>
    </body>
  </instance>
</xsl:template>
<xsl:template match="t:conditions" mode="InsideEquals">
  <conditional-and>
    <xsl:apply-templates select="t:*" mode="InsideEquals"/>
  </conditional-and>
</xsl:template>
<xsl:template name="accessField">
  <xsl:param name="where"/>
  <field-access>
    <xsl:attribute name="name"><xsl:value-of select="@name"/></xsl:attribute>
    <xsl:choose>
      <xsl:when test="$where = 'here'">
        <this/>
      </xsl:when>
      <xsl:when test="$where = 'there'">
        <variable-access>
          <xsl:attribute name="name"><xsl:value-of select="$thatVar"/></xsl:attribute>
        </variable-access>
      </xsl:when>
      <xsl:otherwise>
        <xsl:text>@@Error in XSLT: wrong value for parameter "where"</xsl:text>
      </xsl:otherwise>
```

```
      </xsl:choose>
    </field-access>
  </xsl:template>
  <xsl:template name="Same">
    <equal>
      <xsl:call-template name="accessField">
        <xsl:with-param name="where">here</xsl:with-param>
      </xsl:call-template>
      <xsl:call-template name="accessField">
        <xsl:with-param name="where">there</xsl:with-param>
      </xsl:call-template>
    </equal>
  </xsl:template>
  <xsl:template name="Equal">
    <method-invocation name="equals">
      <xsl:call-template name="accessField">
        <xsl:with-param name="where">
          <xsl:text>here</xsl:text>
        </xsl:with-param>
        <xsl:with-param name="name">
          <xsl:value-of select="@name"/>
        </xsl:with-param>
      </xsl:call-template>
      <arguments>
        <xsl:call-template name="accessField">
          <xsl:with-param name="where">
            <xsl:text>there</xsl:text>
          </xsl:with-param>
          <xsl:with-param name="name">
            <xsl:value-of select="@name"/>
          </xsl:with-param>
        </xsl:call-template>
      </arguments>
    </method-invocation>
  </xsl:template>
  <xsl:template match="t:automatic" mode="InsideEquals">
    <xsl:for-each select="ancestor::j:class[ 1 ]/j:field">
      <xsl:choose>
        <xsl:when test="j:primitive-type">
          <xsl:call-template name="Same"/>
        </xsl:when>
        <xsl:when test="j:reference-type">
          <xsl:call-template name="Equal"/>
        </xsl:when>
      </xsl:choose>
    </xsl:for-each>
  </xsl:template>
  <xsl:template match="t:same" mode="InsideEquals">
    <xsl:for-each select="t:field">
      <xsl:call-template name="Same"/>
    </xsl:for-each>
  </xsl:template>
  <xsl:template match="t:equal" mode="InsideEquals">
    <xsl:for-each select="t:field">
      <xsl:call-template name="Equal"/>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Ispis E.3: Transformacija za generiranje koda za prolazak po kolekciji

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:t="http://tel.fer.hr/jezix/java/template-iterator"
```

```
  xmlns:j="http://tel.fer.hr/jezix/java"
  xmlns="http://tel.fer.hr/jezix/java">
<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
<xsl:template match="@* | node()">
  <xsl:copy>
    <xsl:apply-templates select="@* | node()"/>
  </xsl:copy>
</xsl:template>
<xsl:template match="j:*">
  <xsl:param name="itemVar"/>
  <xsl:copy>
    <xsl:apply-templates select="@* | node()">
      <xsl:with-param name="itemVar" select="$itemVar"/>
    </xsl:apply-templates>
  </xsl:copy>
</xsl:template>
<xsl:template match="j:compilation-unit">
  <compilation-unit>
    <xsl:apply-templates select="@* | node()"/>
  </compilation-unit>
</xsl:template>
<xsl:template match="t:iterate-collection" mode="InsideFor">
  <xsl:variable name="itemVar">
    <xsl:choose>
      <xsl:when test="t:item-var/@name">
        <xsl:value-of select="t:item-var/@name"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:text>item</xsl:text>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable>
  <xsl:copy-of select="."/>
  <initialize>
    <variables>
      <reference-type name="Iterator" package="java.util"/>
      <variable name="iter">
        <method-invocation name="iterator">
          <xsl:copy-of select="t:collection-access/j:*"/>
        </method-invocation>
      </variable>
    </variables>
  </initialize>
  <condition>
    <method-invocation name="hasNext">
      <variable-access name="iter"/>
    </method-invocation>
  </condition>
  <loop>
    <block>
      <variable>
        <xsl:attribute name="name"><xsl:value-of select="$itemVar"/></xsl:attribute>
        <xsl:copy-of select="t:item-type/j:*"/>
        <reference-cast>
          <xsl:copy-of select="t:item-type/j:*"/>
          <method-invocation name="next">
            <variable-access name="iter"/>
          </method-invocation>
        </reference-cast>
      </variable>
      <xsl:apply-templates select="t:do/*">
        <xsl:with-param name="itemVar" select="$itemVar"/>
      </xsl:apply-templates>
    </block>
  </loop>
</xsl:template>
```

```
  <xsl:template match="j:for[ t:iterate-collection ]">
    <for>
      <xsl:apply-templates mode="InsideFor" select="t:iterate-collection"/>
    </for>
  </xsl:template>
  <xsl:template match="t:item-access">
    <xsl:param name="itemVar"/>
    <variable-access>
      <xsl:attribute name="name"><xsl:value-of select="$itemVar"/></xsl:attribute>
    </variable-access>
  </xsl:template>
</xsl:stylesheet>
```

Kratki sadržaj

Okosnica svakog okružja za razvoj programske podrške je *izvorni kod* koji predstavlja ključnu kariku u sučelju između čovjeka i računala. Gotovo od samih početaka računalske ere udomaćio se pristup da čovjek zapisuje *tekst* koji se kodira prema jednom od svjetskih standarda te namjene, a računalo potpuno automatskim procesom taj tekst dekodira, utvrđuje njegovo značenje i proizvodi kod spreman za izravno izvršavanje na svojoj središnjoj obradbenoj jedinici. Tekst izvornog koda ugrubo podsjeća na pisani prirodni jezik i to njegovo svojstvo ključno je za uspješnost sučelja prema čovjeku.

U novije vrijeme, prije svega porastom kompleksnosti programskih sustava koji se razvijaju, pojavili su se novi zahtjevi na izvorni kod. On postaje ključan izvor različitih oblika informacija o razvojnom projektu i mjesto idealno za smještanje dodatnih informacija koje nisu potrebne za dobivanje samog izvršnog koda. Tradicionalni oblik izvornog koda pokazao se nedovoljno fleksibilnim da podrži ugradnju dodatnih informacija bez negativnih posljedica na njegovu primarnu namjenu.

Spomenute informacije potrebno je, zbog veličine i kompleksnosti projekata, izvlačiti automatskim putem. Također, kako projekt evoluira, potrebno je provoditi i neke sustavne zahvate nad velikim količinama izvornog koda kao cjelinom, što je ponovo isplativo jedino ako se može obaviti automatski. Zbog tih potreba automatizacije obrade izvornog koda pojavila se nova stavka u pitanju sučelja prema čovjeku. Postalo je bitno da zadatke obrade koda bude praktično formalno opisati i time omogućiti njihovo automatsko obavljanje. Rješenje je vrlo nepraktično formalizirati kao zahvate nad samim tekstom: mnogo je jednostavnije opisati ih koristeći se samim pojmovima kojima barata dotični programski jezik.

Za svaku automatizaciju obrade izvornog koda stoga je zajednički prvi korak njegovog automatskog razumijevanja do oblika (zvanog *stablo apstraktne sintakse*) kojim je eksplicitno opisano kombiniranje osnovnih jezičnih pojmova u opis programa. Znatno bi se olakšao i ubrzao razvoj automatskih zadataka obrade kad bi taj korak bio standardiziran i široko dostupan. To je motivacija za uvođenje novog

oblika tekstualnog zapisa izvornog koda koji će omogućiti takvu standardizaciju.

XML je standardni jezik za opis stabla proizvoljne strukture. Ako se njime opiše stablo ekvivalentno stablu apstraktne sintakse (u disertaciji ono se naziva *infosetom izvornog koda*), bit će moguće do njega doći standardiziranim procesom. Postojanje standardnog procesa izgradnje infoseta na osnovu XML-ovskog teksta stvara i uvjete za standardiziranje jezika za automatiziranje njegove obrade.

Ova disertacija prvenstveno se bavi problemom oblikovanja sintakse infoseta izvornog koda. Istraživavanjem dosadašnjih rezultata uočeno je da u zajednici koja se bavi pitanjima primjene XML-a na izvorni kod izostaje dovoljno sustavan pristup i posebice da se već donekle i udomaćila pogrešna interpretacija pojma "apstraktna sintaksa". Pod time se često podrazumijeva sintaksa vrlo srodna konkretnoj sintaksi tradicionalnog izvornog koda. Nju se često navodi i kao uzor prema kojem treba oblikovati sintaksu infoseta izvornog koda. Negativne posljedice ovog pogrešnog shvaćanja uočene su i u analiziranim sintaksama. Zaključeno je da je radi daljnjeg napretka na području primjene XML-a na izvorni kod neophodno sustavnim pristupom utvrditi **zahtjeve** na sintaksu infoseta izvornog koda koje ona treba zadovoljiti da bi omogućila maksimalno iskorištavanje tih prednosti te **načela** kojima se treba voditi prilikom oblikovanja sintakse da bi zadovoljila identificirane zahtjeve.

Kao najosjetljiviji zahtjevi (istovremeno vrlo značajni i vrlo zahtjevni za ostvarivanje) identificirane su prilagođenost sintakse upitima i stroga formalna definicija sheme u jeziku WXS. Najvažniji dio istraživanja bilo je oblikovanje sintakse za programski jezik Java. Sustavnim praćenjem tog procesa izdvojeni su najosjetljiviji aspekti oblikovanja i formulirani u obliku načela. Uspješnost sintakse provjerena je razvojem nekoliko ortogonalnih proširenja koja zahtijevaju automatsku obradu. Formulacija njihove obrade u jeziku XSLT pokazala je da je uz relativno malo napora moguće riješiti prilično kompleksne zadatke. Razvijena sintaksa također je uspoređena s dvije postojeće sintakse iste namjene. Definirano je nekoliko zadataka koje je trebalo riješiti upitom u jeziku *XQuery* i izvorni kod nad kojim treba obaviti upite. Upiti nad dokumentom po predloženoj sintaksi bili su znatno lakši za razvoj, kraći i razumljiviji. Potraženi su točni uzroci problema u formulaciji upita nad alternativnim sintaksama i pokazalo se da oni leže u nepridržavanju istaknutih načela oblikovanja sintakse. To je najizravniji pokaz njihove valjanosti i opravdanosti.

Abstract

XML-based Framework for Telecommunicatins Software Development

The cornerstone of every software development environment is the *source code*—the most important component of the machine-to-human interface. From the very beginning of the computer era the approach was adopted where the human writes down *text*, encoded according to one of the global text-encoding standards, and the computer uses a fully automated process to decode it, determine its meaning, and produce code ready for direct execution on its central processing unit. The source code text is structured according to rules abstracted from natural languages and this feature is the key to the success of the human interface based on it.

In recent times, primarily due to the increasing complexity of the software systems, new requirements on the source code have arisen. It has become an important source of various kinds of information about the software project and a place ideal for storing additional information, not needed to produce the executable code. The traditional form of source code has shown weaknesses in supporting the inclusion of this information without negative effects on its primary purpose.

Due to the size and complexity of the projects, the extraction of this information has to be supported by an automatic process. Also, as the project evolves, systematic overhauls involving large amounts of source code need to be performed. This again is feasible only if it can be automated. The situation results in the rise of new requirements on the human interface. It has become important for the processing tasks to be easily formalized and so made ready for automatic execution. These tasks are very impractical to formalize as text-processing tasks. It is much simpler to describe them in terms of the concepts of the programming language.

Because of this, every automated code-processing task begins with its automatic interpretation into the form known as the *abstract syntax tree* which explicitly shows the combining of basic language concepts into a description of the program. The development of automated tasks would be greatly simplified if this step was

standardized and widely available. This is the motivation to introduce a new form of the textual encoding of the source code that will enable such standardization.

XML is the standard language for the description of trees of arbitrary structure. If it is employed to describe a tree equivalent to the abstract syntax tree (this dissertation uses the term *source code infoset* for it), it is possible to attain it using a standardized automatic process. The existence of the standard process for the construction of the infoset makes it possible to develop standard infoset-processing lanugages which can be employed to automate the code-processing tasks.

This dissertation is primarily concerned with the problem of the design of the source code infoset syntax. The study of the related work has shown that in the community dealing with application of XML to source code there is a lack of a sufficiently systematic approach. Especially, it was found that the key concept of the abstract syntax is being misinterpreted. Under this term, a syntax very similar to the concrete syntax of the traditional source code is often assumed. It is often mentioned as the proper basis for the infoset syntax. The negative effects of this misinterpretion have been identified in the analyzed syntaxes as well. The conclusion was that before further advances in this field are made, a systematic approach is needed to determine the **requirements** on the source code infoset syntax it has to meet in order to leverage the potentials of XML, and further to determine the **principles** in syntax design that need to be followed in order to meet the requirements.

Identified as the most sensitive requirements (at the same time very important and very involving) were "query-friendliness" and a strict formalization in the WXS schema language. The key part of the research was the development of an infoset syntax for the Java programming language. Through a systematic monitoring of the process the most critical aspects of it were isolated and formulated as design principles. The success of the syntax was tested by developing several orthogonal extensions that require automatic processing. The formulation of their processing in XSLT has shown that it is possible to solve quite complex tasks with a reasonable amount of effort. The syntax was also compared with two of the most relevant existing syntaxes with the same purpose. Several processing tasks to be solved in the XQuery language were defined and also the source code on which to perform the queries. The queries for the proposed syntax were substantially easier to develop, shorter, and more readable. The exact sources of the inferiority of the other syntaxes were searched for and were found in the divergence from the stated design principles. This is the most direct justification of their validity.

Sažetak

Okosnica svakog okružja za razvoj programske podrške je tekstualni izvorni kod koji predstavlja ključnu kariku u sučelju između čovjeka i računala. On je strukturiran prema pravilima apstrahiranim iz prirodnih jezika, što mu daje čitljivost i lakoću formuliranja. Povećanjem kompleksnosti softverskih projekata pojavili su se novi zahtjevi na izvorni kod: postalo je važno da on bude pogodan za automatsku obradu, točnije da čovjeku bude omogućena njena laka formalizacija. Također je postala bitna praktičnost uključivanja i obrade dodatnih podataka u izvorni kod, nepotrebnih za njegovo kompiliranje. Tim novim zahtjevima moguće je udovoljiti prelaskom na novi tekstualni oblik zapisa prema standardu XML. Glavni problem pri tom prelasku je oblikovanje nove sintakse izvornog koda, a uočeno je da u zajednici koja se bavi tim problemom nedostaje dovoljno sustavan pristup i da se pogrešno shvaća ključan pojam *apstraktne sintakse*. Stoga je proveden sustavan postupak identificiranja točnih zahtjeva na sintaksu i načela u njenom oblikovanju kojih se treba držati radi ispunjenja tih zahtjeva. Razvijena je sintaksa za programski jezik Java i uspoređena je s dvije postojeće sintakse. Pokazalo se da predložena sintaksa u znatno većoj mjeri ispunjava zahtjeve, a uzroci slabosti alternativnih sintaksi nađeni su u njihovom nepridržavanju utvrđenih načela oblikovanja, čime je potvrđena njihova valjanost i opravdanost.

Summary

XML-based Framework for Telecommunicatins Software Development

The cornerstone of every software development environment is the *source code*—the most important component of the machine-to-human interface. It is structured according to rules abstracted from natural languages and this feature is the key to the success of the human interface based on it. Due to the increasing complexity of the software systems, new requirements on the source code have arisen because there is an increasing need to automate its processing in various ways. Therefore it is important for it to allow the processing tasks to be easily formalized and so made ready for automatic execution. Also, it has to allow easy addition and processing of arbitrary metadata needed by the development process. The new requirements can be met by adopting a new textual format that complies with the XML standard. The main problem in the transition is the design of the new source code syntax and it was found that in the community dealing with this problem there is a lack of a sufficiently systematic approach and that the key concept of the *abstract syntax* is being misinterpreted. Therefore a systematic process was conducted which identified the exact requirements on the syntax and the principles that need to be followed in its design to meet those requirements. A syntax for the Java programming language was developed and compared to two existing syntaxes. It was shown that the proposed syntax meets the requirements to a significantly higher proportion. The source of the deficiencies of the alternative syntaxes was identified as lying in their divergence from the mentioned design principles, which confirms their validity.

Ključne riječi

- programski jezik
- izvorni kod
- XML
- gramatika
- sintaksa
- apstraktna sintaksa
- AST
- semantika
- automatizacija
- ortogonalno proširenje

Keywords

- programming language
- source code
- XML
- grammar
- syntax
- abstract syntax
- AST
- semantics
- automatization
- orthogonal extension

Životopis

Rođen sam 14. kolovoza 1974. u Zagrebu. Na Klasičnoj gimnaziji u Zagrebu maturirao sam 1993, a diplomirao 1998. godine na Fakultetu elektrotehnike i računarstva, smjer Telekomunikacije i informatika, s izvrsnim uspjehom. Dodijeljena mi je Brončana plaketa "Josip Lončar" za najbolji ukupni uspjeh u mojoj generaciji diplomanada na smjeru. Srednja ocjena ispita položenih na dodiplomskom studiju iznosi 4.16. Na Zavodu za osnove elektrotehnike i električka mjerenja zaposlio sam se 15. srpnja 1998. godine kao znanstveni novak na projektu Ministarstva znanosti i tehnologije broj 036036, "Razvoj elektroničkog okidača za NN prekidače". Poslijediplomski studij započeo sam 1998. godine na području Primjena računarstva. U znanstvenoistraživačkom radu bavio sam se razvojem i primjenom numeričkih metoda analize elektromagnetskog polja. Magistarski rad pod naslovom "Proračun svezanih elektromagnetskih polja i pomaka strukture primjenom metode konačnih elemenata" obranio sam 5. prosinca 2000. s izvrsnim uspjehom. Srednja ocjena ispita na poslijediplomskom magistarskom studiju iznosi 5.00. Od 1. veljače 2001. zaposlen sam na Zavodu za telekomunikacije kao znanstveni novak na projektu Ministarstva znanosti i tehnologije broj 036004, "Inteligentna i širokopojasna telekomunikacijska mreža". Doktorski studij započeo sam 2001. godine na području Telekomunikacije i informatika. U nastavi sudjelujem na predmetima Seminar Java+XML, Konkurentno programiranje i Telematičke usluge.

Curriculum Vitae

I was born on August 14th, 1974 in Zagreb. I graduated from the Classical Gymnasium, Zagreb in 1993. In 1998 I received a B.Sc. degree at the University of Zagreb, Faculty of Electrical Engineering and Computing (FER). I was awarded the Bronze plaque "Josip Loncar" for the best overall achievement in my generation. My average score on undergraduate courses was 4.16. I started working at FER's Department of Electrical Engineering Fundamentals and Measurements on July 15th, 1998 as a research assistant on the project number 036036 of the Ministry of Science and Technology, named "Development of an electronic actuator for LV circuit breakers". I began my graduate study in 1998 in the field Applied computing. My scientific research was in the field of development and application of numeric methods for electromagnetic field analysis. I defended magna cum laude my Master's thesis under the title "Coupled Electromagnetic and Structural Field Calculation Using Finite Element Method" on December 5th, 2000. My average score on graduate courses was 5.00. As of February 1st, 2001 I work at the Department of Telecommunications as a research assistant on the project number 036004 of the Ministry of Science and Technology, named "Intelligent and Broadband Telecommunication Network". I started my Ph.D. study in 2001 in the field Telecommunications and Information Technology. I am involved in the following educational courses on FER: Seminar Java+XML, Concurrent Programming and Telematic Services.