DOKUMENTACIJA TEHNIČKOG REŠENJA

Sistem za adaptivno praćenje performansi u distribuiranim softverskim sistemima

Autori tehničkog rešenja
Dušan Okanović, Milan Vidaković, Zora Konjović

Godina kada je tehničko rešenje urađeno
2013.

Oblast tehničkog rešenja
Tehničko rešenje po Pravilniku Ministarstva prosvete, nauke i tehnološkog razvoja Republike Srbije pripada kategoriji **M85** (Tehnička i razvojna rešenja - Prototip, nova metoda, softver, standardizovan ili atestiran instrument, nova genska proba, mikroorganizmi). Tehničko rešenje pripada naučnoj oblasti Informatika, uža naučna oblast održavanje softvera.

Apstrakt
Tehničko rešenje je softverski sistem za praćenje performansi distribuiranih softverskih aplikacija. Sistem omogućuje praćenje rada softvera uz minimum uticaja na praćeni sistem i utvrđivanje poštovanja zadatog nivoa performansi. Očekivane performanse se zadaju u okviru XML datoteke, čiji format je deo ovog rešenja. Na osnovu zadatih parametara, sistem za praćenje može da se prilagođava i uključuje praćenje samo u onim delovima praćenog softvera koji pokažu odstupanje od očekivanih performansi. Na taj način se dodatno smanjuje uticaj na performanse softvera koji se prati.

Sistem za praćenje može da radi u dva osnovna režima: obično praćenje i adaptivno praćenje. Kod običnog praćenja, sistem za praćenje meri performanse praćenog softvera - celog ili samo zadatih delova. Ovaj režim rada pogodan je za prikupljanje inicijalnih podataka o performansama, koji kasnije mogu da se iskoriste kao etalon. Adaptivno praćenje podrazumeva da se periodično vrši analiza prikupljenih podataka i da se menjaju parametri praćenja. To podrazumeva da se u delovima u kojima se otkrije odstupanje od očekivanih performansi uključi, a u ostalim isključi merenje.

Sistem je implementiran kao skup JMX ([JMX]) komponenti uz oslonac na Kieker okruženje ([Hoorn2012]). Kieker je izabran zbog minimalnog uticaja na performanse sistema koji se prati. Pomoću dodatnih JMX komponenti, još više se smanjuje uticaj na praćeni sistem. Za definisanje tačaka u kojima se vrši merenje koristi se aspekt-orijentisano programiranje - AOP ([Kiczales96]). Sistem se sastoji od dve grupe komponenti. Prva grupa komponenti su one koje vrše prikupljanje podataka, a koje se nalaze na istom računaru kao i praćeni softver. Druga grupa komponenti vrši analizu dobijenih podataka i formiranje novih parametara sistema za praćenje.

Primena ovog tehničkog rešenja omogućava da se automatizuje proces otkrivanja problema u softveru i da se precizno odredi njegovo poreklo. Zahvaljujući tome, problem može da se otkrije i pre nego što krajnji korisnici praćenog softvera primete njegovo postojanje, npr. kroz pad brzine odziva, periodične zastoje i potpuni prestanak rada. Sistem je koristan pružaocima softverskih usluga, jer dobijene rezultate mogu da koriste za planiranje proširenja kapaciteta, kao i da preduprede nepredviđene zastoje i planiraju preventivno održavanje.

Naučno-istraživački doprinos ovog tehničkog rešenja publikovan je u jednom naučnom radu u međunarodnom časopisu sa SCI liste [Okanovic13], u tri međunarodna časopisa ([Okanovic12a], [Okanovic12b] i [Okanovic12e]), kao i u četiri rada na međunarodnim konferencijama ([Okanovic11a], [Okanovic11b], [Okanovic11c] i [Okanovic12d]).

Tehničko rešenje se koristi u nastavi i u naučno-istraživačke svrhe na Katedri za informatiku na Fakultetu tehničkih nauka Univerziteta u Novom Sadu.

1. Uvod

Tehničko rešenje se odnosi na implementaciju sistema za kontinuirano praćenje performansi softvera. Preciznije, prikazano rešenje je namenjeno za praćenje performansi softvera kad se on već pusti u upotrebu. Po tome se ovo rešenje razlikuje od alata koji se koriste u fazi razvoja softvera, kao što su profajleri i dibageri.

Krajnji cilj praćenja softvera je da obezbedi podatke za proveru poštovanja ugovora o nivou softverskog servisa (eng. *service level agreement*, SLA) [Clifford08]. U okviru ovog ugovora se, pored funkcionalnih zahteva, definišu nefunkcionalni zahtevi, tj. način ispunjavanja funkcionalnih zahteva. U fazi testiranja nemoguće je testirati uticaj okruženja aplikacije i opterećenje od strane krajnjih korisnika. Često se pojavljuju problemi koji nisu primećeni u toku faze testiranja, a koji mogu da utiču na poštovanje ugovora o nivou servisa [Grottke08].

Da bi bili sigurni da aplikacija zadovoljava sve nefunkcionalne zadatke koji su postavljeni pred nju, tj. da zadovoljava parametre definisane ugovorom o nivou servisa, neophodno je da se rad softvera kontinuirano prati. Ako se javi nekakvo odstupanje od zadatih parametara, kontinuirano praćenje može da omogući otkrivanje uzroka. Druga primena rezultata je za planiranje akcija koje mogu da se sprovedu kao privremena mera, pre nego što se isprave greške koje se javljaju u sistemu.

Merenje performansi softvera može da unese izmene u njegovom ponašanju – npr. usporenje u izvršavanju, povećano zauzeće memorije, povećan mrežni saobraćaj, itd. Opterećenje koje merenje unosi u sistem koji se prati (eng. *performance overhead*), a koje se obično manifestuje kao sporiji odziv ili povećana potrošnja memorije i sl, mora da se predvidi i da se njegov uticaj smanji ili eliminiše.

Stvari se dodatno komplikuju u slučaju distribuiranih aplikacija. Distribuirane aplikacije se izvršavaju na heterogenim platformama, počevši od različitog hardvera, operativnih sistema, podrazumevaju upotrebu različitih komunikacionih mreža, itd. Distribuirani sistemi mogu da se sastoje od velikog broja čvorova koji međusobno komuniciraju, pa je praćenje od velikog značaja za pouzdanost sistema. Praćenjem rada mogu da se utvrde odstupanja od predviđenih performansi. Uska grla i greške mogu da se pronađu još u toku razvoja, ali praćenje rada ovih aplikacija mora stalno da se vrši. Vrlo često se dešava da performanse softvera variraju tokom vremena, pa je neophodno da se utvrde i uklone uzroci ovoga.

U ovom rešenju je prikazan alat, pomoću kog može da se vrši kontinuirano praćenje distribuiranih aplikacija, uz mali uticaj na sistem koji se prati. Može da se vrši provera usklađenosti sa SLA ili samo da se vrši merenje (u određenoj metrici). Prikazana je specifikacija i neki detalji implementacije sistema, kao i XML shema dokumenta za definisanje nivoa servisa.

2. Pregled postojećih rešenja

Većina alata za određivanje performansi softvera nije prilagođena za rad sa softverom koji je već u upotrebi. Alati kao što su profajleri mogu da odrede deo koda koji ima probleme sa performansama, ali i oni sami unose veliko opterećenje na sistem. Kod razvoja alata za kontinuirano praćenje o ovome mora da se vodi dosta računa, pa se obično radi na razdvajanju sistema za prikupljanje podataka od sistema za analizu podataka.

Što se tiče Java okruženja, za koje je ovo rešenje prvenstveno namenjeno, veliki broj alata je baziran na upotrebi JVM PI [JVMPI] i JVMTI [JVMTI]. Ovo su interfejsi koji na najnižem nivou pristupaju virtuelnoj mašini (JVM) i preko njih mogu da se prikupljaju informacije o njenom stanju. Alati koji koriste ove interfejse vrše prikupljanje podatak uzorkovanjem. To znači da se periodično aktiviraju i očitavaju potrebne parametre. Na ovaj način se smanjuje opterećenje na praćeni sistem, jer alat nije sve vreme aktivan, ali postoji mogućnost da se propusti neki događaj između dva očitavanja. Agenati, preko kojih se vrši prikupljanje podataka, se pišu u programskom jeziku C++. Ovo je osnovni problem sa ovakvim pristupom, jer se od programera očekuje poznavanje dodatnog programskog jezika. Drugi problem je činjenica da komunikacija ide preko *native* poziva, što može da izazove ozbiljna usporenja i nestabilnost u radu.

Komorium profajler, predstavljen u [Binder06], vrši transformaciju programa tako da se u toku izvršavanja vrši brojanje izvršenog *byte*-koda. Ovde je reč o alatu koji merenje vrši uzorkovanjem. Kada brojač koda dostigne određenu vrednost, pokreće se agent koji pristupa steku i uzima uzorak, vrši se kreiranje kontekstnog stabla i beleži količina izvršenog *byte*-koda. Transformacija programa se vrši pomoću alata koji ubacuje potrebne instrukcije u programski kod. Pomoću ovih instrukcija kreira se programski stek koji se periodično uzorkuje, jer standardni stek nije dostupan iz Java programa. Prednost ovog pristupa je što se agenti pišu u čistom programskom jeziku Java, tj. ne zahteva nikakve izmene na JVM. Pošto se broji količina izvršenog *byte*-koda (ne meri se vreme), dobijeni rezultati su potpuno nezavisni od JVM,

operativnog sistema i računara na kom se program izvršava. Takođe, dobijeno stablo je precizno, jer Komorium beleži potpuni potpis metoda, što nije podržano kod npr. JVMTI. Komorium je fleksibilan, jer dozvoljava izmenu parametara profajliranja. Nedostaci ovog alata su mogućnost upotrebe isključivo u eksperimentalnim uslovima i nemogućnost profajliranja *native* koda.

Pristup u implementaciji alata za praćenje koji se sve više koristi je instrumentacija - umetanje dodatnog koda koji vrši merenje. Iako se dodavanjem novog koda postojeći kod "prlja", otežava održavanje i, potencijalno, unosi nestabilnost, pažljivim planiranjem i odabirom tačaka u koje će kod biti dodat, mogu da se dobiju kvalitetni rezultati.

Za instrumentaciju može da se koristi i JVMTI, ali je pokazano u nekoliko radova ([Reiss08], [Zhuang06]) da ovaj pristup unosi dosta nestabilnosti u rad softvera. U NetBeans IDE [NetBeans] je uključen JFluid profajler [Dimitriev03], koji koristi JVMTI, ali uključivanjem instrumentacije samo po potrebi redukuje opterećenje na praćeni softver. Na ovaj način se smanjuje velika količina informacija koje se dobijaju profajliranjem. Uključivanje i isključivanje instrumentacije je moguće i u toku rada aplikacije.

Postoji nekoliko alata koji vrše *byte*-kod instrumentaciju. Najpoznatiji je BCEL (eng. *Byte Code Engineering Library*) [BCEL]. Biblioteka omogućava, na binarnom nivou, analizu, kreiranje i manipulaciju datotekama sa Java klasama. Klase se predstavljaju objektima koji sadrže sve informacije o klasi (npr. metode, atribute, *byte*-kod instrukcije). Klase mogu da se menjaju u toku izvršavanja programa, ali postoji i mogućnost da se kreiraju potpuno nove klase. Nedostatak ovog pristupa je što se, slično kao i kod JVMTI, programira na niskom nivou i lako može da se izazove nestabilnost u radu. Za implementaciju HotWave okruženja [Villazon09], autori su iskoristili BCEL biblioteku. Pomoću ovog okruženja mogu da se implementiraju alati za praćenje, ali se ne programira na niskom nivou, nego se koristi jezik sličan AspectJ [AspectJ] jeziku.

Okruženja za aspekt-orijentisano programiranje su implementirana uz oslonac na alate za *byte*-kod instrumentaciju. Pošto AOP okruženja podržavaju programiranje na visokom nivou, uz prevenciju nestabilnosti, ona su ušla u sve širu upotrebu. Mogućnost upotrebe aspekt-orijentisanog programiranja, konkretno alata AspectJ, u procesu profajliranja prikazana je u [Pearce07]. Dok se program izvršava, on ulazi u određene tačke, tj. izaziva određene događaje. Te tačke se u AOP nazivaju *join point*. Pomoću AOP-a, moguće je dodati programski kod pre, posle i oko tih tačaka. Taj programski kod se u AOP naziva *advice*. Tačke mogu da predstavljaju razne konstrukte u programskom jeziku: poziv metode, poziv konstruktora, izvršavanje metode, pristup atributu, itd. Povezivanje *join point*-a sa *advice*-om se vrši preko *pointcut*-a. Pored mogućnosti ubacivanja programskog koda, AspectJ nudi još i mogućnost otkrivanja koja tačka je pokrenula neki *advice* (nešto slično *this* ključnoj reči u programskom jeziku Java), kao i dodavanja novih metoda i atributa postojećim klasama. Sprovedeno testiranje pokazuje da je praćenje pomoću AspectJ alata dovoljno fleksibilno da se pomoću njega vrši praćenje različitih podataka, uz ograničenja opisana u radu – nedostatak rada sa sinhronizovanim blokovima,

nemogućnost prepoznavanja alokacije nizova – i još neke probleme koji su u međuvremenu rešeni u novijim AspectJ implementacijama (npr. upredanje aspekata u toku učitavanja – eng. *load time weaving*). Studija prikazana u [Marwede09] pokazuje da alati, koji su zasnovani na AOP, pružaju dobru osnovu za razvoj alata za praćenje, jer alati zasnovani na JVM TI, ipak unose veće kašnjenje i grešku merenja.

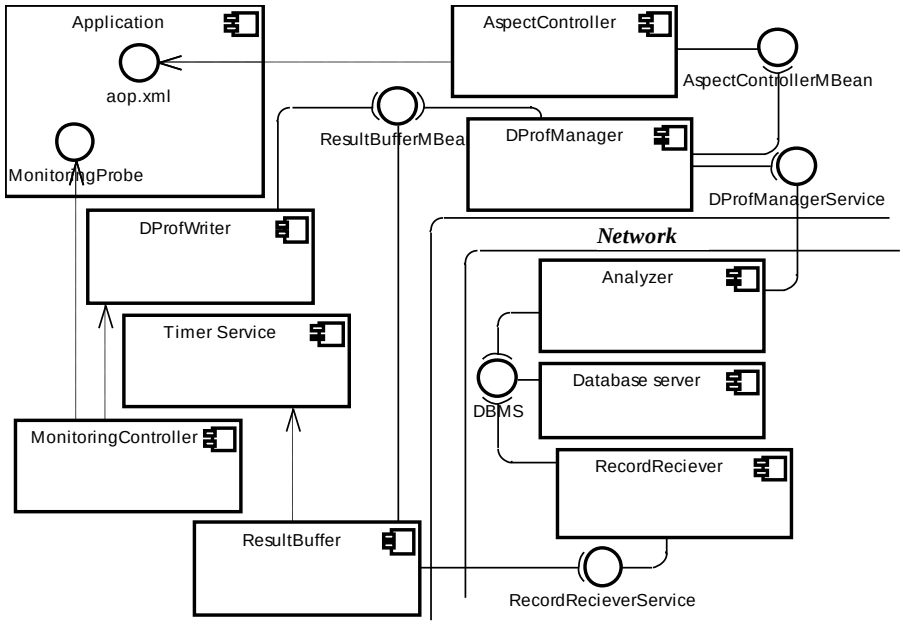
Mogućnosti adaptivnog praćenja aplikacija pomoću AOP prikazane su u [Schade96], [Katchabaw97] i [Liu04], ali nikad nisu implementirane. Oba pristupa u sebi sadrže ideju da se softverska sonda postavi u neku tačku u softveru pomoću AOP-a, ali da se tom sondom upravlja preko upravljačkog interfejsa.

U [Ehlers11], autori predstavljaju sistem za detekciju anomalija, koji je takođe zasnovan na Kieker [Hoorn12] okruženju i posmatranju stabla poziva, kao i sistem opisan u ovom rešenju. Pri praćenju, posmatraju se pojedinačne komponente sistema. Pre početka postavljaju se softverske sonde u metode komponenata koje želimo da pratimo. Na početku se prate samo metode koje čine interfejs komponenata, tj. metode koje su na najvišem nivou stabla poziva. Ako se, tokom praćenja, primeti odstupanje od očekivanih vrednosti, zadatih pomoću OCL-a [OCL], uključuje se sonde za praćenje na sledećem nivou stabla poziva u okviru komponente, itd.

Rešenje koje je ovde predloženo koristi AOP za instrumentaciju. Podaci se prikupljaju pomoću Kieker okruženja. Podrška za distribuirane sisteme se ogleda u tome da se podaci šalju preko mreže na analizu, umesto da se analiza radi na svakom računaru posebno. Ovo ujedno smanjuje opterećenje u svakom čvoru. Periodičnom analizom dobijenih podataka se dobijaju novi parametri koji se prosleđuju komponentama koje vrše merenje. Na osnovu parametara se uključuje i isključuje praćenje u delovima sistema. Ovako može dodatno da se smanji opterećenje na sistem. Za definisanje očekivanih performansi koristi se XML, što ne predstavlja optrećenje za programere.

3. Arhitektura sistema za praćenje

Arhitektura sistema prikazana je dijagramom komponenti na slici 1.



Slika 1 Komponente DProf sistema

Instrumentacija se vrši pomoću AOP. Merenje može da se vrši pomoću standardnih mehanizama dostupnih u Java platformi ili dodatnih paketa. Prikupljanje dobijenih podataka vrši se pomoću Kieker okruženja koje je prošireno dodatnim komponentama. Podaci se sakupljaju u obliku zapisa koji predstavlja proširenje zapisa koji se distribuira uz Kieker.

Novi *Monitoring Log Writer – DProfWriter* – ne upisuje zapise direktno u *Monitoring Log* (kao što je slučaj kod ostalih, *Kieker*-ovih originalnih, *Monitoring Log Writera*), nego u *ResultBuffer*. Bafer šalje zapise na prijemnu stranu. Zapisi mogu da se šalju periodično (upotrebom JMX *Timer* servisa) ili po zahtevu koji stiže od *Analyzer* komponente (preko *DProfManager* komponente). *RecordReciever* komponenta prihvata zapise i smešta ih u bazu podataka. Praktično, *ResultBuffer*, *RecordReciever* i baza podataka igraju ulogu *Monitoring Log*-a u *Kieker* okruženju. Upotrebom *ResultBuffer* komponente, opterećenje koje DProf sistem izaziva je manje od opterećenja pri upotrebi ostalih Kieker *writer* komponenti. Naime, pošto *ResultBuffer* može da šalje rezultate u paketima, dodatno opterećenje se javlja samo periodično. Sistem je nabolje konfigurisati da pošalje pakete sa rezultatima u periodima kada se očekuje manja aktivnost korisnika sistema.

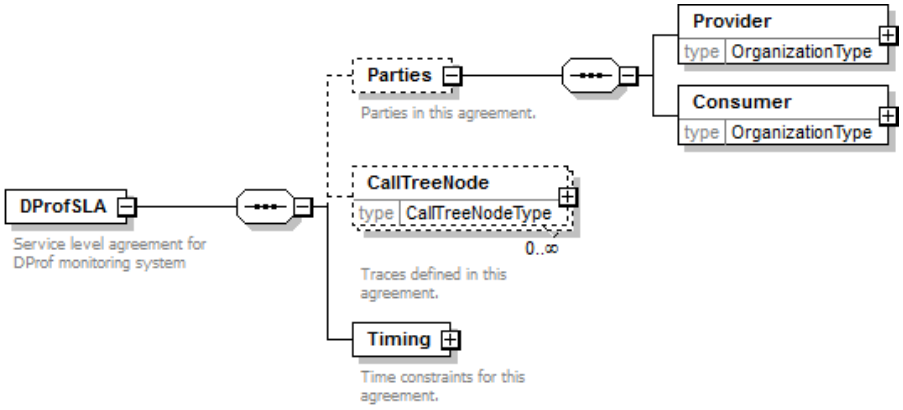
Komponenta *DProfManager* kontroliše rad bafera i *AspectController*-a – komponente koja kontroliše softverske sonde (*Monitoring Probe*) i konfiguriše rad AOP okruženja. Komponenta *Analyzer* vrši analizu pristiglih zapisa. Na osnovu tih zapisa, definiše novu konfiguraciju i šalje je *DProfManager*-u. Konfiguracija praćenja se definiše preko *aop.xml* datoteke – koja se inače koristi za konfigurisanje AOP okruženja. Svaka izmena konfiguracije praćenja rezultuje izmenama u konfiguracionoj datoteci *aop.xml*.

Komunikacija preko mreže se odvija putem veb-servisa. Koriste se JAX-WS veb-servisi. Komponente koje su na istoj strani, komuniciraju kroz virtuelnu mašinu, tj. realizovane su kao JMX *MBean* komponente i komuniciraju kroz *MBeanServer*.

3.1 Definisanje SLA

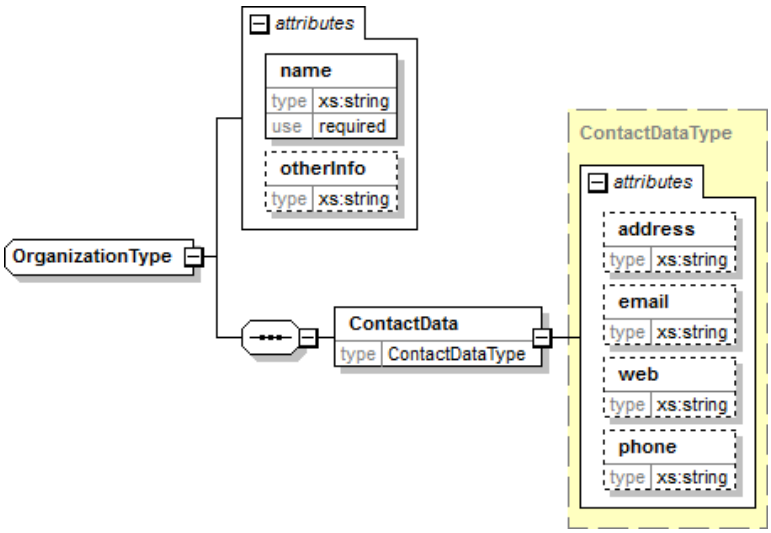
Sistem koji je prikazan u ovom radu koristi jednostavniju XML shemu za definisanje SLA. Nova shema je definisana, jer za potrebe sistema prikazanog u ovom radu nije potrebna sva kompleksnost shema koje se najviše koriste u ovoj oblasti (npr. GXLA [Tebbani06], WSLA [Keller09], SLAng [Lamanna03]). Ova shema je napravljena u skladu sa principima definisanim u [Trienekens04] i predstavlja podskup pomenutih, pa ako je potrebno, dokumenti napisani po ovoj shemi mogu da se prebace u neki drugi standard pomoću XML transformacija. Prema kategorizaciji koja je data u [Paschke06] dokumenti napisani u ovoj shemi spadaju u standardne (prema podeli prema raznovrsnosti primene), operativne ugovore (prema podeli prema nameni), koji se koriste u okviru organizacije (prema podeli prema okviru primene).

Osnovna struktura sheme prikazana je na slici 2.



Slika 2 Korenski element sheme za definisanje SLA u DProf sistemu

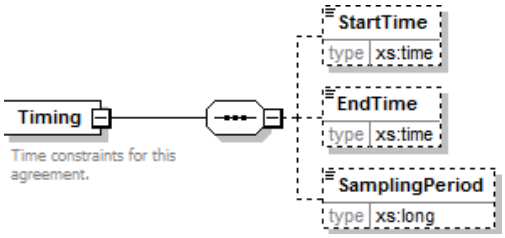
U elementu *Parties* navode se ugovorne strane, a element *Timing* sadrži vremenske parametre za primenu ugovora. Element *CallTreeNode* sadrži sva stabla poziva koja, po ovom ugovoru, treba da se prate. Element *Parties* sadrži podatke o ugovornim stranama. Pružalac usluga je predstavljen elementom *Provider*, a korisnik elementom *Consumer*. Oba elementa su predstavljena kompleksnim tipom *OrganizationType* prikazanim na slici 3.



Slika 3 Kompleksni tip *OrganizationType*

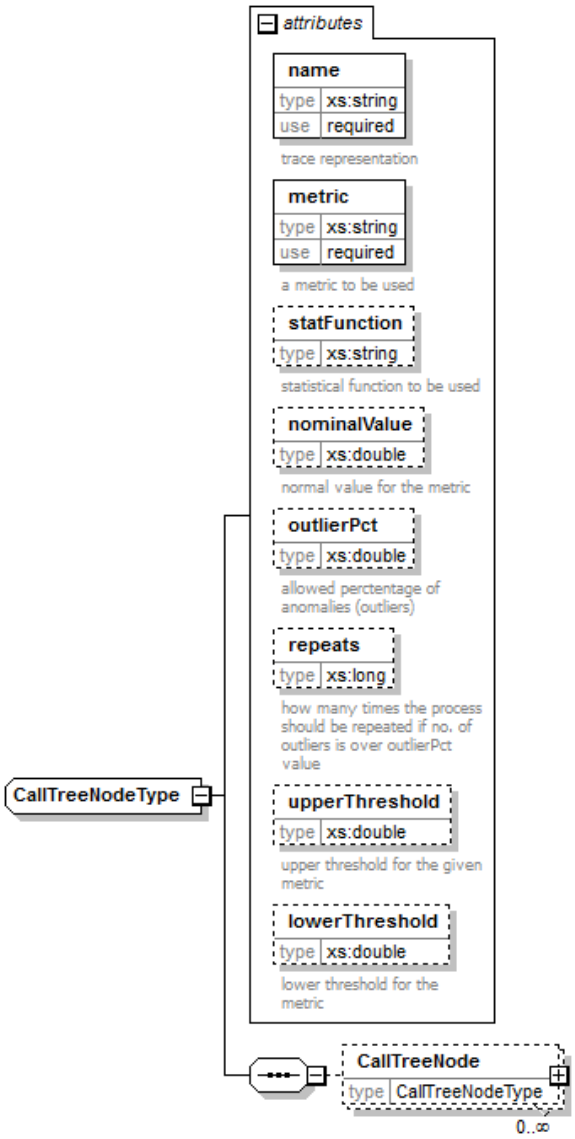
Kompleksni tip *OrganizationType* sadrži osnovne podatke o organizaciji. Atribut name predstavlja naziv organizacije. Element *ContactData* (predstavljen kompleksnim tipom *ContactDataType*) sadrži kontakt podatke o organizaciji. Atributi definisani u ovom tipu su: *adress* (adresa organizacije), *email* (kontakt e-mail organizacije), *web* (veb adresa organizacije) i *phone* (kontakt telefon organizacije). Svi atributi su predstavljeni prostim tipom *string*.

Element *Timing* (slika 4) sadrži tri pod-elementa. *StartTime* i *EndTime* su predstavljeni prostim tipom *time*. Podelement *StartTime* označava vreme kada treba da počne praćenje rada aplikacije. *EndTime* predstavlja vreme kad praćenje treba da se završi. Element *SamplingPeriod* predstavlja vreme između dve rekonfiguracije procesa praćenja. JMX *Timer* okida *Analyzer* komponentu i pokreće se proces analize nakon isteka ovog perioda. Sva tri elementa su prostog tipa *long* i predstavljaju vreme u milisekundama. *StartTime* i *EndTime* predstavljaju vreme u milisekundama koje je proteklo od ponoći, 1. januara 1970. (kao u Unix i Java specifikaciji).



Slika 4 Element *Timings*

U elementu *CallTreeNode* navode se stabla poziva koja će biti praćena. Elementi *CallTreeNode* su kompleksnog tipa *CallTreeNodeType* (slika 5).

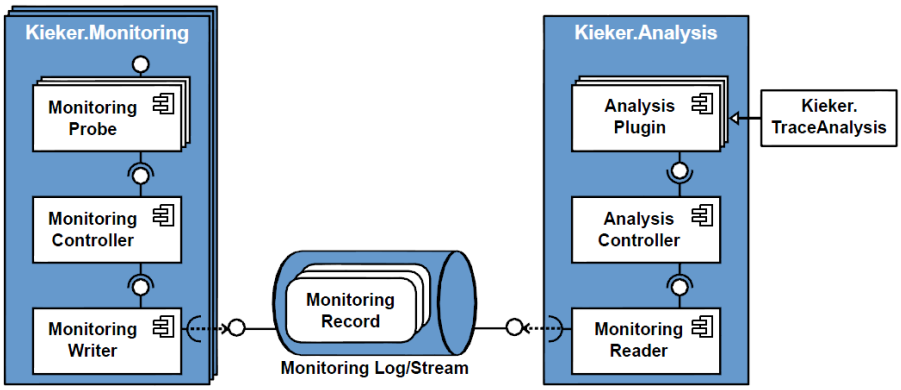


Slika 5 Element *CallTreeNode*

Atribut *metric* definiše meru koja se koristi u merenjima. Atribut *statFunction* predstavlja statističku funkciju koja se koristi u obradi podataka (npr. srednja vrednost, standardna devijacija, maksimum, minimum). *outlierPct* je tolerisani procenat vrednosti koje se izbacuju iz posmatranog skupa podataka. Ako je dobijeni procenat veći od zadate vrednosti, merenje treba da se ponovi. Ako se prekorači dozvoljen broj ponavljanja, definisan atributom *repeats*, prijavljuje se problem. U zavisnosti od izabrane statističke funkcije, koriste se atributi *nominalValue* (nominalna vrednost), *upperThreshold* (gornja granična vrednost) i *lowerThreshold* (donja granična vrednost).

3.2 Kieker okruženje

Arhitektura Kieker-a je data na slici 7.



Slika6 Arhitektura Kieker okruženja (preuzeto iz [Ehmke12])

Dve osnovne komponente okruženja su Kieker.Monitoring, koja smešta zapise (Monitoring Log Record) u Monitori ng Log i Kieker.Analysis, koja obezbeđuje infrastrukturu za analizu i vizualizaciju dobijenih podataka. Obe komponente okruženja rade nezavisno jedna od druge. Ovo omogućava da, npr. grupa servera (eng. *cluster*) izvršava softver koji se prati. Podaci dobijeni praćenjem mogu da se čuvaju na nekom drugom, a analiziraju na trećem serveru, koji nije deo osnovne grupe servera. Instrumentacija softvera se vrši preko softverskih sondi. Može da se vrši na različite načine, najčešće preko različitih alata za AOP. Referentna implementacija je urađena uz upotrebu AspectJ alata. Presretanje se najčešće vrši pomoću *around advice*-a, tako što se zabeleži stanje sata pre i posle poziva metode. Zapisi koje generišu sonde, prosleđuju se kontroleru (Monitoring Controller), a on ih upisuje u skladište pomoću Monitoring Log Writter-a. AOP okruženje može da se podesi da se presreću samo klase iz određenih paketa ili sve. U aspektima može da se definiše da li da se prate sve metode ili samo one koje su označene određenom anotacijom. Pomoću ove dve tehnike se fino bira koji delovi aplikacije se prate, što dalje omogućava da se bira odnos između željene količine podataka i *overhead*-a.

Čitanje podataka iz skladišta i njihovo pretvaranje u zapise vrši Monitoring Log Reader. Dobijeni zapisi se prosleđuju Monitoring Log Consumer-u, koji se obično pravi tako da prihvata samo zapise određenog tipa. Zadatak Consumer-a je da izvrši analizu ili vizualizaciju komponenti. Kontrolu rada Kieker.Analysis komponente vrši Analysis Controller.

Vizualizacija podataka u Kieker okruženju može da se vrši na više načina [Rohr08]. Kieker.Analysis komponenta, može da generiše UML dijagrame sekvenci, lance Markova, dijagrame zavisnosti komponenti (UML dijagram komponenti proširen težinskim faktorima zavisnosti između komponenti, koji označavaju broj poziva komponente) i dijagrame vremenskih rasporeda poziva (predstavlja odnose trajanja izvršavanja u jednoj grani stabla konteksta poziva).

Skladište u kom se čuvaju zapisi može da bude baza podataka, datoteka, JMS red (eng. *queue*) ili nešto drugo. Svaki tip zapisa nasleđuje klasu *AbstractMonitoringRecord*. Referentna implementacija obezbeđuje još i *OperationExecutionMonitoringRecord*, zapis koji sadrži informaciju o izvršenoj operaciji, računaru na kom se izvršava posmatrana aplikacija i

vremenske podatke. Vremenski podaci se beleže u rezoluciji nanosekunde, pošto se vreme očitava pozivom *System.nanoTime()*.

Ako je potrebno da se vrši merenje nekih drugih parametara, onda je potrebno da se definiše novi tip softverske sonde (nova aspekt klasa u okviru koje se vrše merenja) i novi tip zapisa (nova klasa koja nasleđuje *AbstractMonitoringRecord*). Za praćenje rada jedne aplikacije mogu da se koriste različiti aspekti i anotacije, tako da je moguće da se prati više različitih parametara istovremeno.

3.3 Proces praćenja

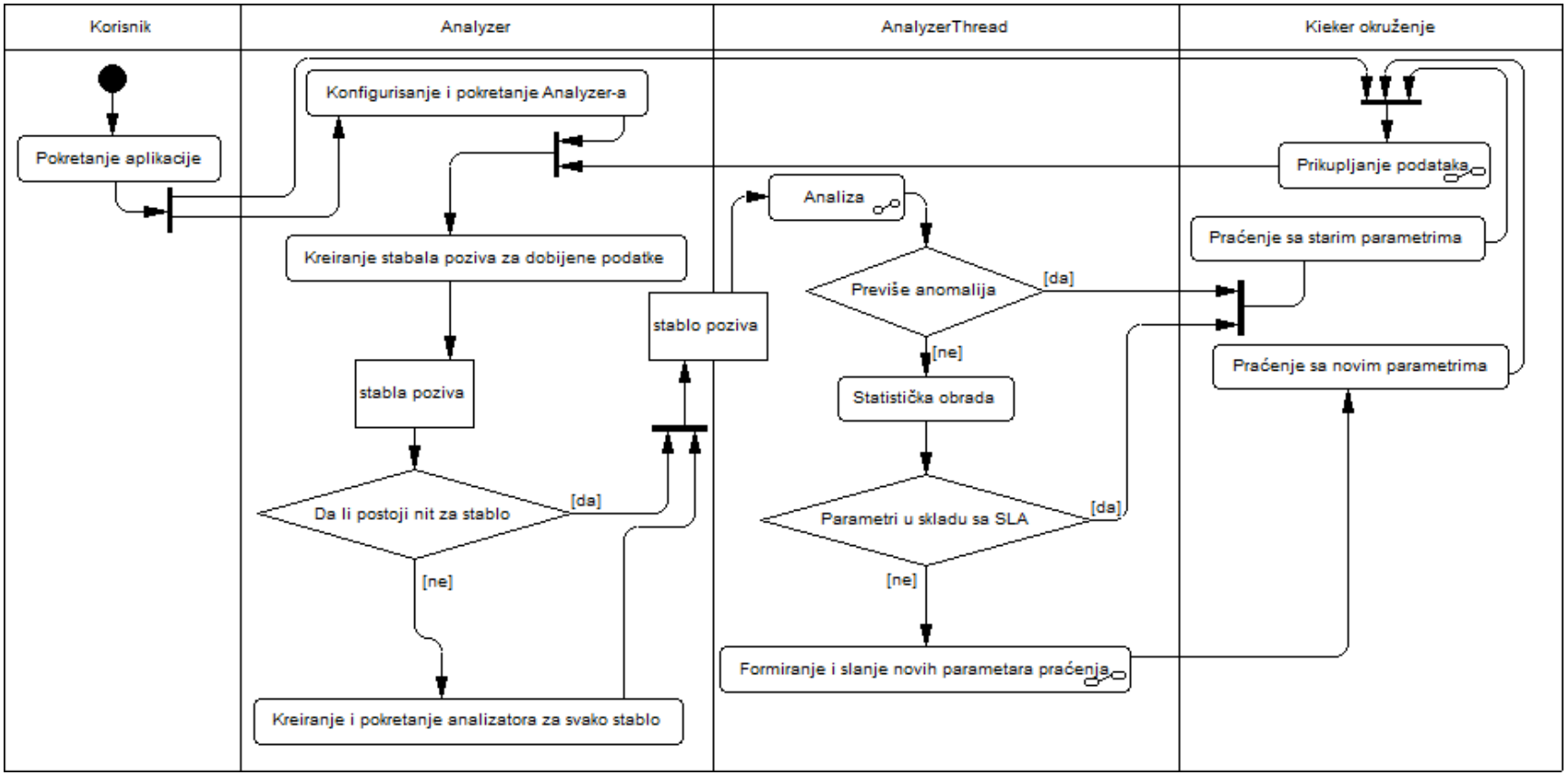
Globalni dijagram aktivnosti koji prikazuje celokupan proces praćenja dat je na slici 8.

Pre nego što se pokrene aplikacija, inicijalna konfiguracija se zadaje anotiranjem metoda koje želimo da pratimo i/ili pomoću include/exclude stavki u aop.xml konfiguracionoj datoteci AspectJ okruženja.

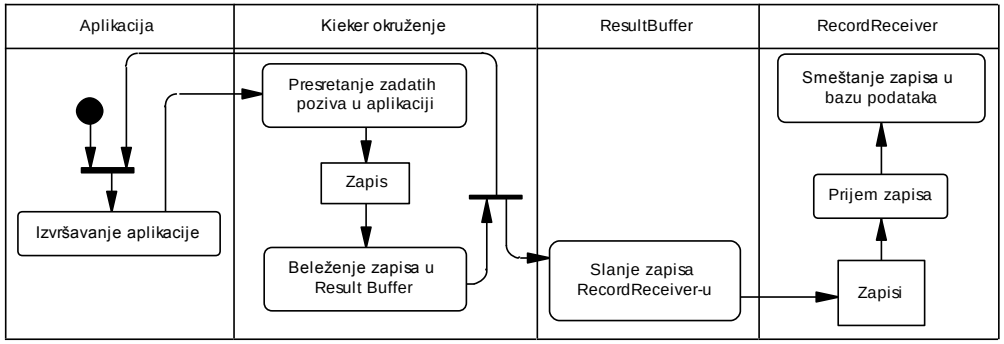
Ceo proces počinje pokretanjem aplikacije. Pri pokretanju aplikacije, pokreću se i okruženje i *Analyzer* komponenta. Proces se grana na prikupljanje i analizu podataka. Kieker okruženje sa DProf proširenjem vrši prikupljanje podataka u toku izvršavanja aplikacije.

Analyzer komponenta periodično uzima prikupljene podatke iz baze podataka i od njih kreira stabla poziva. Za svako stablo predviđeno SLA ugovorom se kreira programska nit koja vrši analizu dobijenih stabala. Moguće je da nit za takvo stablo već postoji, pa se kreiranje i u tom slučaju preskače, nego se toj već postojećoj niti prosleđuje odgovarajuće stablo. Niti vrše statističku analizu podataka. Prvo se broj anomalija poredi sa brojem zadatim u SLA dokumentu. Ako je u podacima pronađeno previše anomalija, proces praćenja za to stablo se ponavlja sa istim parametrima. Ovo se radi da bi se dobila realna slika rezultata. Ako se i nakon ponavljanja dobije prevelik broj anomalija, smatra se da u sistemu postoji problem. Ako je broj anomalija prihvatljiv, proverava se da li su parametri u skladu sa SLA. Ako jesu praćenje se nastavlja sa parametrima koji obuhvataju samo koren stabla. Ako parametri nisu u skladu sa SLA, proces se nastavlja dalje sa novim parametrima koji će obuhvatiti nove nivoe u stablu poziva.

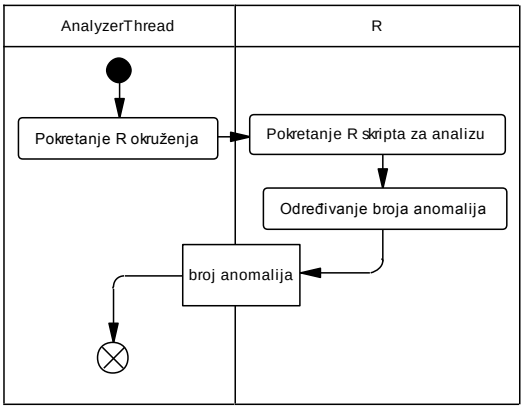
Prikupljanje podataka prikazano je dijagramom aktivnosti datim na slici 9. Tokom izvršavanja aplikacije, Kieker okruženje presreće pozive u izvršavanju aplikacije i vrši potrebna merenja. Dobijene vrednosti se šalju *ResultBuffer* komponenti. *ResultBuffer* dalje prosleđuje podatke u *RecordReceiver*-u.



Slika 8 Dijagram aktivnosti koji opisuje rad DProf sistema



Slika 9 Dijagram aktivnosti za proces praćenja i beleženja zapisa



Slika 10 Dijagram aktivnosti za proces analize podataka

Dijagram aktivnosti za proces analize dat je na slici 10.

Programske niti koriste R programski jezik [RLanguage10] i dodatni paket *extremevalues* [Loo11], za statističku analizu dobijenih rezultata. Paket *extremevalues* sadrži funkcije za detekciju anomalija (eng. *outlier detection*). U statistici se za anomalije smatraju pojave koje su numerički drastično udaljene od ostatka skupa podataka. Najčešći uzrok pojave anomalija su greške merenja. Paket extremevalues sadrži funkcije koje procenjuju koje vrednosti su anomalije. Za zadati skup podataka, regresijom se utvrđuju parametri raspodele kojoj pripada skup posmatranih vrednosti. Prema prvom metodu, vrednosti iz skupa podataka koje su izvan zadatog intervala, a u odnosu na raspodelu, su ekstremne vrednosti. Drugi metod uzima odstupanja reziduala posmatranih vrednosti od zadatih vrednosti. Ako je odstupanje veće od zadate granice, vrednost je anomalija. Paketi *rJava* i *JRI* se koriste za povezivanje R i Java okruženja [Urbanek11]. Pomoću funkcija definisanih u rJava, moguće je izvršavanje Java programa u okviru R okruženja, a pomoću JRI iz Java programa mogu da se pozivaju funkcije napisane u R jeziku.

Pomoću R okruženja prvo se određuje broj anomalija. Ove anomalije su obično posledica različitih spoljašnjih uticaja: učitavanja klasa, pokretanje pozadinskih procesa dok radi aplikacija ili hardverskih poremećaja. Nakon što se anomalije uklone iz skupa posmatranih vrednosti, vrši se proračun u skladu sa statističkom operacijom zadatom u SLA.

U zavisnosti od rezultata statističke obrade, moguća su tri scenarija:

1. Ako statistička obrada pokaže da su rezultati u skladu sa SLA, praćenje se nastavlja sa starim parametrima.
2. Ako je broj anomalija prevelik, merenje se ponavlja sa starim parametrima, pošto se smatra da dobijeni rezultati ne daju pravu sliku sistema. Ako se ova situacija ponovi još jednom, onda se smatra da postoji problem.

3. Ako statistička obrada da rezultate koji nisu u skladu sa SLA, kreiraju se novi parametri za proces praćenja: utvrđuje se da je u trenutnom čvoru greška ili se uključuje praćenje u sledećem nivou stabla poziva.

4. Primena rešenja

Verifikacija rešenja je sprovedena je u nekoliko scenarija [Okanovic12c]. Prvo je sprovedeno praćenje brzine odziva aplikacije opisane u [Okanovic08] na pokrenute na više servera u cilju merenja normalnih vrednosti. Drugi scenario obuhvata praćenje rada aplikacije i traženje odstupanja od definisanog SLA. Svi scenariji su zasnovani na praćenju rada aplikacije koja je distribuirana na nekoliko JBoss servera. Na osnovu dobijenih rezultata pokazano je da ovo rešenje unosi manje opterećenje u sistem u odnosu na komponente koje su deo standardne Kieker distribucije. Ovo je postignuto izmenama u arhitekturi, prvenstveno uvođenjem ResultBuffer komponente koja podatke šalje na analizu samo u trenucima kad manji broj korisnika pristupa aplikaciji. Arhitekturalno poboljšanje kroz uvođenje adaptivnosti sistema dodatno je smanjilo ukupan uticaj na praćenu aplikaciju.

Definisanje SLA ugovora i praćenje njegove primene prikazano je u [Okanovic11c], [Okanovic12a] i [Okanovic12b]. Ugovori su napravljeni prema predloženoj XML shemi i prikazano je kako ovo rešenje može da se iskoristi za praćenje rada aplikacije koje je već u upotrebi, a da korisnici ne osete probleme u radu. U [Okanovic12b] je prikazano praćenje potrošnje memorije, kao i mogućnost upotrebe ovog rešenja za praćenje i drugih parametara performansi, ne samo brzine odziva, koja se obično meri.

Na osnovu podataka dobijenih praćenjem, u [Okanovic12d] i [Okanovic12e] prikazano je kako ovo rešenje može da obezbedi rezultate koji dalje mogu da se upotrebe za planiranje održavanja praćenog softvera. U [Okanovic12d], primenom statističkih metoda nad rezultatima pronađena je zavisnost na osnovu koje može da se proceni kada je potrebno u sistem dodati novi server da bi se odgovorilo na povećane zahteve sve većeg broja korisnika. Na ovaj način održava se potrebna brzina odziva aplikacije. Rezultati dobijeni praćenje su u [Okanovic12e] upotrebljeni za planiranje preventivnog restartovanja aplikacije. Ovaj postupak se koristi u situacijama kad aplikacija ima grešku zbog koje ne oslobađa resurse koje više ne koristi, pa može dođe u situaciju da potroši resurse (npr. "curenje" memorije, konekcija prema bazi podataka). Ako greška ne može odmah da se ukloni, planira se preventivno gašenje i pokretanje aplikacije, nakon čega ona oslobađa resurse i može normalno da radi. Krajnji korisnici mogu da budu unapred obavešteni o ovom zastoju, ili restartovanje može da se izvede kada je najmanji broj korisnika aktivan.

5. Zaključak

Tehničko rešenje je alat za adaptivno praćenje performansi distribuiranih aplikacija. Željene performanse se definišu preko XML dokumenta, za koji je, u okviru rešenja, definisana XML shema. Ovo rešenje omogućava automatizaciju procesa otkrivanja problema u softveru uz

minimizaciju opterećenja na sam softver. Minimizacija opterećenja je ovde od posebne važnosti, jer se ovaj sistem ne koristi u toku razvoja, nego dok sistemu pristupaju krajnji korisnici. Od posebne važnosti je činjenica da sistem može da se koristi i za praćenje softvera koji nije napisan u programskom jeziku Java. Rešenje je implementirano u programskom jeziku Java, uz upotrebu Kieker okruženja i JMX tehnologije.

Tehničko rešenje obezbeđuje automatizaciju procesa praćenja softvera, omogućava otkrivanje problema i pre nego što ih krajnji korisnici osete i obezbeđuje rezultate koji mogu da se upotrebe za planiranje daljih akcija u održavanju sistema. Opis tehničkog rešenja, kao i neke od njegovih konkretnih primena prikazani su u 8 naučnih radova, od kojih jedan pripada kategoriji M23, tri su kategoriji M53, a preostala četiri kategoriji M33.

Literatura

[AspectJ] AspectJ, http://www.eclipse.org/aspectj/

[BCEL] Byte Code Engineering Library. Apache Software Foundation. [Online] http://jakarta.apache.org/bcel/index.html

[Binder06] Binder, W., Portable and Accurate Sampling Profiling for Java, Software – Practice & Experience, v.36 n.6, p.615-650, 2006.

[Clifford08] Clifford, D., van Bon, J.: Implementing ISO/IEC 20000 Certification: The Roadmap, ITSM Library, Van Haren Publishing, 2008. ISBN 908753082X

[Dimitriev03] M. Dimitriev, Design of JFluid, Technical Report: SERIES13103, Sun Microsystems Inc., USA (2003)

[Ehlers11] Ehlers, J., van Hoorn, A., Waller, J., Hasselbring, W.: Self-Adaptive Software System Monitoring for Performance Anomaly Localization. In Proceedings of the 8th IEEE/ACM International Conference on Autonomic Computing (ICAC 2011). ACM, Karlsruhe, Germany. p. 197-200. (2011)

[Ehmke12] Kieker 1.5 User Guide. Ehmke, N. C., Hoorn, A. v., Jung, R. (2012) [online] Available: http://netcologne.dl.sourceforge.net/project/kieker/kieker/kieker-1.5/kieker-1.5_userguide.pdf

[Grottke06] Grottke, M.: Analysis of Software Aging in a Web Server. IEEE Transactions on Reliability, v. 55, n. 3. p. 411-420. (2006)

[Hoorn12] Hoorn, A. v., Hasselbring, W., Waller, J.: Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis. Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE 2012). ACM, Boston, Massachusetts, USA. p. 247-248. (2012)

[JVMPI] Java Virtual Machine Profiler Interface (JVMPI). Oracle. [Online] http://download.oracle.com/javase/1.4.2/docs/guide/jvmpi/jvmpi.html (current April 2012)

[JVMTI] Java Virtual Machine Tool Interface (JVMTI). Oracle. [Online] http://download.oracle.com/javase/6/docs/technotes/guides/jvmti/ (current April 2012)

[JMX] Java Management Extensions Technology. Oracle. [Online] http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/

[Katchabaw97] Katchabaw, M., Howard, S., Lutfiyya, H., Marshall, A., Bauer, M.: Making Distributed Applications Manageable Through Instrumentation. Journal of Systems and Software, v.45, n.2. Elsevier Science, New York, USA. p. 81-97. (1997)

[Keller03] Keller, A., Ludwig, H.: The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. Journal of Network and Systems Management, v. 11 n. 1, p. 57-81. (2003)

[Kiczales96] Kiczales, G.: Aspect-Oriented Programming. ACM Computing Surveys (CSUR), v.28 n.4. (1996)

[Lamanna03] Lamanna, D., Skene, J., Emmerich, W.: SLAng: A Language for Defining Service Level Agreements. In Proceedings of the 9th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS '03). IEEE Computer Society, Washington, DC, USA. p. 100-107. (2003)

[Liu04] Liu, R., Gibbs, C., Coady, Y.: MADAPT: Managed Aspects for Dynamic Adaptation Based on Profiling Techniques. In Proceedings of the 3rd Workshop on Adaptive and Reflective Middleware, Toronto, Ontario, Canada. p. 214 - 219. (2004)

[Loo11] extremevalues: Univariate Outlier Detection. Mark van der Loo. (2011) [Online] Available: http://cran.r-project.org/web/packages/extremevalues/ (current September 2011)

[Marwede09] Marwede, N., Rohr, M., van Hoorn, A., Hasselbring, W.: Automatic Failure Diagnosis Support in Distributed Large-Scale Software Systems Based on Timing Behavior Anomaly Correlation. In Proceedings of the 2009 European Conference on Software Maintenance and Reengineering (CSMR '09). IEEE Computer Society, Kaiserslautern, Germany. p. 47-58. (2009)

[NetBeans] NetBeans, http://netbeans.org/index.html

[OCL] Object Constraint Language (OCL) 2.0. OMG. [Online] Available: http://www.omg.org/spec/MOF/2.0 (Septembar 2011)

[Okanovic08] Okanović, D., Vidaković, M.: One Implementation of the System for Application Version Tracking and Automatic Updating. Proceedings of the IASTED International Conference on Software Engineering, Innsbruck, Austria. p. 80-85. (2008)

[Okanovic11a] Okanović, D., Vidaković, M.: Performance Profiling of Java Enterprise Applications. In Proceedings of the 1st International Conference on Information Society Technology and Management. Kopaonik, Serbia. p. 60-64. (2011)

[Okanovic11b] Okanović, D., Hoorn, A. v., Konjović, Z., Vidaković, M.: Towards Adaptive Monitoring of Java EE Applications. In Proceedings of the 5th International Conference on Information Technology - ICIT. Amman, Jordan. CD. (2011)

[Okanovic11c] Okanović, D., Konjović, Z., Vidaković, M.: Continuous Monitoring System for Software Quality Assurance. In Proceedings of the 15th International Scientific Conference on Industrial Systems (IS11). Novi Sad, Srbija. CD. (2011)

[Okanovic12a] Okanović, D., Vidaković, M., Konjović, Z.: Service Level Agreement XML Schema for Software Quality Assurance. Acta Technica Corviniensis, vol. 5, n. 1. p. 123-128. (2012)

[Okanovic12b] Okanović, D., Konjović, Z., Vidaković, M.: Low-Overhead Continuous Monitoring of Service Level Agreements. International Journal of Industrial Engineering and Management (IJIEM), v.3, n.4. p. 25-31. (2012)

[Okanovic12c] Okanović, D.: Model adaptivnog sistema za praćenje i predikciju rada distribuiranih aplikacija. Doktorska disertacija, Univerzitet u Novom Sadu. (2012)

[Okanovic12d] Okanović, D., Vidaković, M.: Software Performance Prediction Using Linear Regression. In Proceedings of the 2nd International Conference on Information Society Technology and Management. Kopaonik, Serbia. p. 60-64. (2012)

[Okanovic12e] Okanović, D., Vidaković, M., Konjović, Z.: Monitoring of JEE Applications and Performance Prediction. Journal of Information Technology and Applications, v.1, n.2. p. 136-143. (2012)

[Okanovic13] Okanović D., van Hoorn A., Konjović Z., Vidaković M.: SLA-Driven Adaptive Monitoring of Distributed Applications for Performance Problem Localization, Computer Science and Information Systems (ComSIS), 2013.

[Paschke06] Paschke, A., Schnappinger-Gerull, E.: A Categorization Scheme for SLA Metrics. In Proceedings of Multi-Conference Information Systems (MKWI 2006), Passau, Germany. (2006)

[Pearce07] Pearce, D., Webster, M., Berry, R., Kelly, P.: Profiling with AspectJ. Software—Practice & Experience, v.37 n.7, p. 747-777. (2007)

[Reiss08] Reiss, S.: Controlled Dynamic Performance Analysis. In Proceedings of the 7th international workshop on Software and performance. Princeton, USA. (2008)

[RLanguage10] R Development Core Team. R: A language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. (2010)

[Rohr08] Rohr, M, Hoorn, A. v., Matevska, J., Sommer, N., Stoever, L., Giesecke, S., Hasselbring, W.:Kieker: Continuous Monitoring and on Demand Visualization of Java Software Behavior. In Proceedings of the IASTED International Conference on Software Engineering (SE '08). ACTA Press, Anaheim, CA, USA. p. 80-85. (2008)

[Schade96] Schade, A., Trommler, P., Kaiserswerth, M.: Object Instrumentation for Distributed Applications Management. Proceedings of the IFIP/IEEE International Conference on Distributed Platforms (ICDP'96), Dresden, Germany. IFIP, Chapman and Hall. (1996)

[Tebbani06] Tebbani, B., Aib, I.: GXLA a Language for the Specification of Service Level Agreements. Lecture Notes in Computer Science, v. 4195. Springer-Verlag, Berlin Heidelberg New York. p. 201-214. (2006)

[Trienekens04] Trienekens, J., Bouman, J., Zwan, M. v. d.: Specification of Service Level Agreements: Problems, Principles and Practices. Software Quality Control, v. 12 n. 1. p. 43-57. (2004)

[Urbanek11] Urbanek, S.: Low-level R to Java interface. [Online] http://www.rforge.net/rJava

[Villazon09] Villazon, A., Binder, W., Ansaloni, D., Moret, P.: HotWave: Creating Adaptive Tools with Dynamic Aspect-Oriented Programming in Java. Proceedings of the 8th International Conference on Generative Programming and Component Engineering, Denver, USA. p. 95-98 (2009)

[Zhuang06] X. Zhuang, M. J. Serrano, H. W. Cain, J. Choi, Accurate, efficient, and adaptive calling context profiling, Proceedings of the 2006 ACM SIGPLAN conference on Programming language design and implementation, 2006, Ottawa, Canada

Рецензија техничког решења

На основу достављеног материјала, у складу са одредбама *Правилника о поступку и начину вредновања, и квантитавном исказивању научноистраживачких резултата истраживача,* који је донео Национални савет за научни и технолошки развој Републике Србије («Службени гласник РС», бр. 38/2008,) рецензент др Срђан Шкрбић оценио је да су испуњени услови за признање својства техничког решења следећем резултату научноистраживачког рада:

Назив: Решење за адаптивно праћење перформанси у дистрибуираним софтверским системима

Аутори: др Душан Окановић, др Милан Видаковић, др Зора Коњовић
Категорија техничког решења: (М85) Прототип, нова метода, софтвер, стандардизован или атестиран инструмент, новагенска проба, микроорганизми

Образложење

Предложено решење урађено је за: Факултет техничких наука у Новом саду (За потребе пројекта са ев. бр. ИИИ 44010: „Интелигентни системи за развој софтверских производа и подршку пословања засновани на моделима“, руководилац пројекта: проф. др Иван Луковић). Предложено решење је урађено: 2013. год.

Субјект који је решење прихватио и примењује: Факултет техничких наука.

Област на коју се техничко решење односи је: софтвер за аутоматско праћење рада софтвера и локализацију проблема у раду

Проблем који се техничким решењем решава:

- локализација извора проблема са перформансама
- утврђивање поштовања уговора о нивоу сервиса
- аутоматизација процеса праћења

Стање решености тог проблема у свету

У свету постоји велики број решења за праћење рада софтвера. Већина решења је намењена за праћење перформанси током развоја и нису погодна за примену у фази употребе софтвера. Наиме, већина ових решења заузима превелику количину ресурса и додатно оптерећује систем, изнад прихватљиве границе.

Суштина техничког решења.

Основу алата чине компоненте урађене у JMX технологији које врше анализу података и Kieker алат помоћу ког се врши скупљање података. Техничко решење састоји се од следећих модула:

1. модул за мерење и прикупљање података о извршавању апликације
2. модул за слање података на анализу,
3. модул за анализу података,
4. модул за управљање процесом праћења.

Карактеристике предложеног техничког решења су следеће:

За имплементацију решења употребљен је програмски језик Java, Kieker алат, JMX технологија за имплементацију додатних компоненти и аспект-оријентисано програмирање за прикупљање података. Оваквим избором постигнута је платформска независност решења. Дефинисање очекиваних перформанси се ради преко XML докумената за које је дефинисана XML схема.

Могућности примене предложеног техничког решења:

Систем за адаптивно праћење перформанси у дистрибуираним системима намењен је за примену од стране произвођача софтвера, као и корисника. Помоћу њега може да се врши утврђивање порекла проблема, као и утврђивање испуњености уговора о нивоу софтверског сервиса.

На основу свега наведеног као рецензент оцењујем да је резултат научноистраживачког рада под називом: „Решење за адаптивно праћење перформанси у дистрибуираним системима“ успешно реализован, да представља истраживачки резултат који је практично употребљив и предлажем да се прихвати у категорију научно-истраживачких резултата као софтверски производ (**М85**).

У Новом Саду, 24.12.2013.

Рецензент:

др Срђан Шкрбић

Природно-математички факултет Нови Сад

Рецензија техничког решења

На основу достављеног материјала, у складу са одредбама *Правилника о поступку и начину вредновања, и квантитавном исказивању научноистраживачких резултата истраживача,* који је донео Национални савет за научни и технолошки развој Републике Србије («Службени гласник РС», бр. 38/2008,) рецензент др Милош Рацковић оценио је да су испуњени услови за признање својства техничког решења следећем резултату научноистраживачког рада:

Назив: Решење за адаптивно праћење перформанси у дистрибуираним софтверским системима

Аутори: др Душан Окановић, др Милан Видаковић, др Зора Коњовић
Категорија техничког решења: (М85) Прототип, нова метода, софтвер, стандардизован или атестиран инструмент, новагенска проба, микроорганизми

Образложење

Предложено решење урађено је за: Факултет техничких наука у Новом саду (За потребе пројекта са ев. бр. ИИИ 44010: „Интелигентни системи за развој софтверских производа и подршку пословања засновани на моделима“, руководилац пројекта: проф. др Иван Луковић). Предложено решење је урађено: 2013. год.

Субјект који је решење прихватио и примењује: Факултет техничких наука.

Област на коју се техничко решење односи је: алат за аутоматску локализацију проблема у раду софтвера и праћење перформанси софтвера

Проблем који се техничким решењем решава:

- аутоматизација процеса праћења
- локализација извора проблема са перформансама
- утврђивање поштовања уговора о нивоу сервиса

Стање решености тог проблема у свету

Већина доступних решења за праћење перформанси софтвера је намењена за праћење током развоја и нису погодна за примену на софтверу који је већ у употреби. Код оваквих система мора озбиљно да се води рачуна о ресурсима које алат за праћење заузима, да би што мање утицао на рад и перформансе праћеног система.

Суштина техничког решења.

Алат се састоји од компонената урађених у JMX технологији и Kieker алата за сакупљање података. Техничко решење састоји се од следећих модула:

1. модул за мерење и прикупљање података о извршавању апликације
2. модул за управљање процесом праћења,
3. модул за слање података на анализу,
4. модул за анализу података.

Карактеристике предложеног техничког решења су следеће:

Систем је имплементиран у програмском језику Java, али може да подржи и рад са софтвером у неком другом програмском језику, уз мање додатке. Употребљени су и Kieker алат, JMX технологија за имплементацију додатних компоненти, као и аспект-оријентисано програмирање за прикупљање података. Тиме је постигнута платформска независност решења. Дефинисање очекиваних перформанси се ради преко XML докумената за које је дефинисана XML схема.

Могућности примене предложеног техничког решења:

Систем за адаптивно праћење перформанси у дистрибуираним системима намењен је за примену од стране произвођача софтвера, као и корисника. Помоћу њега може да се врши утврђивање порекла проблема, као и утврђивање испуњености уговора о нивоу софтверског сервиса.

На основу свега наведеног као рецензент оцењујем да је резултат научноистраживачког рада под називом: „Решење за адаптивно праћење перформанси у дистрибуираним системима“ успешно реализован, да представља истраживачки резултат који је практично употребљив и предлажем да се прихвати у категорију научно-истраживачких резултата као софтверски производ (**М85**).

У Новом Саду, 24.12.2013.

Рецензент:

др Милош Рацковић

Природно-математички факултет Нови Сад

УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
Трг Доситеја Обрадовића 6, 21000 Нови Сад, Република Србија
Деканат: 021 6350-413; 021 450-810; Централа: 021 485 2000
Рачуноводство: 021 458-220; Студентска служба: 021 6350-763
Телефакс: 021 458-133; e-mail: ftndean@uns.ac.rs
ИНТЕГРИСАНИ СИСТЕМ МЕНАЏМЕНТА СЕРТИФИКОВАН ОД:

Наш број: 01.сл
Ваш број:
Датум: 2013-12-27

ИЗВОД ИЗ ЗАПИСНИКА

Наставно-научног већа Факултета техничких наука у Новом Саду, на 18. редовној седници одржаној дана 26.12.2013. године, донело је следећу одлуку:

-непотребно изостављено-

Тачка 11.1. Верификација нових техничких решења и именовање рецензената

Тачка 11.1.52. у циљу верификације новог техничког решења усвајају се рецензенти:

- Др Срђан Шкрбић, доцент, Природно-математички факултет Нови Сад
- Др Милош Рацковић, редовни професор, Природно-математички факултет Нови Сад

Назив техничког решења:

СИСТЕМ ЗА АДАПТИВНО ПРАЋЕЊЕ ПЕРФОРМАНСИ У ДИСТРИБУИРАНИМ СОФТВЕРСКИМ СИСТЕМИМА

Аутори техничког решења: др Душан Окановић, доцент; Др Милан Видаковић, ванредни професор; др Зора Коњовић, редовни професор

-непотребно изостављено-

Записник водила:
Јасмина Димић, дипл. правник

Тачност података оверава:
Секретар
Иван Нешковић, дипл. правник

Декан
Проф. др Раде Дорословачки

УНИВЕРЗИТЕТ
У НОВОМ САДУ

ФАКУЛТЕТ
ТЕХНИЧКИХ НАУКА

Трг Доситеја Обрадовића 6, 21000 Нови Сад, Република Србија
Деканат: 021 6350-413; 021 450-810; Централа: 021 485 2000
Рачуноводство: 021 458-220; Студентска служба: 021 6350-763
Телефакс: 021 458-133; e-mail: ftndean@uns.ac.rs

ИНТЕГРИСАНИ СИСТЕМ МЕНАЏМЕНТА СЕРТИФИКОВАН ОД:

Наш број: 01.сл
Ваш број:
Датум: 2014-01-16

ИЗВОД ИЗ ЗАПИСНИКА

Наставно-научног већа Факултета техничких наука у Новом Саду, на 19. седници одржаној дана 15.01.2014. године, донело је следећу одлуку:

-непотребно изостављено-

Тачка 3. Верификација техничких решења

Тачка 3.48: На основу позитивног извештаја рецензената верификује се техничко решења под називом:

СИСТЕМ ЗА АДАПТИВНО ПРАЋЕЊЕ ПЕРФОРМАНСИ У ДИСТРИБУИРАНИМ СОФТВЕРСКИМ СИСТЕМИМА

Аутори техничког решења: др Душан Окановић, доцент; Др Милан Видаковић, ванредни професор; др Зора Коњовић, редовни професор

-непотребно изостављено-

Записник водила:

Јасмина Димић, дипл. правник

Тачност података оверава:
Секретар

Иван Нешковић, дипл. правник

Декан

Проф. др Раде Дорословачки