Integracija procesiranja kompleksnih dogodkov in storitveno usmerjenih arhitektur

Martin Potočnik, Matjaž B. Jurič
Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, Tržaška cesta 25, 1000 Ljubljana
martin.potocnik@fri.uni-lj.si; matjaz.juric@fri.uni-lj.si

Izvleček

Inteligentni in agilni moderni informacijski sistemi se morajo v vsakem okolju pravilno in učinkovito odzivati na preproste in kompleksne dogodke, saj so ti pomembni za poslovanje. Zaradi dinamičnosti distribuiranega okolja se povečuje kompleksnost informacijskih sistemov, hkrati pa pomembnost in število dogodkov strmo naraščata. Ker učinkovito procesiranje predvsem kompleksnih dogodkov postaja vedno bolj težavno, a hkrati pomembno, je v članku predstavljena konceptualna rešitev, ki temelji na konvergenci storitveno usmerjenih in dogodkovno gnanih arhitektur. Ker obstoječe platforme omogočajo procesiranje le preprostih dogodkov (kompleksni dogodki se obravnavajo v ločenih informacijskih rešitvah), predlagamo model komponente, ki omogoča celovito procesiranje kompleksnih dogodkov na ravni storitvenega vodila. Predlagana komponenta je povezljiva z obstoječimi sistemi za upravljanje s poslovnimi pravili in sistemi za spremljanje poslovnih aktivnosti ter ponuja zmožnost definiranja, zaznavanja in odzivanja na kompleksne dogodke. Zagotavlja centraliziran nadzor in upravljanje nad vsemi dogodki, kar omogoča učinkovit razvoj naprednih, visoko agilnih in inteligentnih informacijskih sistemov.

Ključne besede: procesiranje kompleksnih dogodkov, storitveno usmerjene arhitekture, dogodkovno gnane arhitekture, storitveno vodilo.

Abstract

Integration of Complex Event Processing into Service Oriented Architecture

Most of today's information systems do not fulfill all goals in sense of event responsiveness. As the dynamism of distributed environments is rapidly increasing, efficient event processing is becoming more and more important. In this paper we present the convergence of Service Oriented Architecture and Event Driven Architecture which are two different paradigms that address complex integration challenges. Their convergence provides necessary capabilities for the development of intelligent, agile and flexible information systems. Because the existing SOA platforms only support processing of simple events (complex event processing is done by external dedicated systems) we propose a new approach to complex event processing in SOA. We define a Service Component Architecture (SCA) compliant component that can be integrated into Enterprise Service Bus (ESB) and can leverage existing Business Rule Management Systems and Business Activity Monitoring systems. The proposed solution offers us a comprehensive platform for complex event processing, centralized event management and event governance.

Key words: Complex Event Processing, Service Oriented Architecture, Event Driven Architecture, Enterprise Service Bus.

1 UVOD

V zadnjih letih je razmah tehnoloških platform omogočil razvoj inteligentnih, agilnih in skalabilnih informacijskih sistemov, s katerimi se lahko organizacije prilagajajo potrebam dinamičnega poslovanja. Dinamičnost distribuiranega okolja predstavljajo številne spremembe oz. dogodki, na katere se morajo informacijski sistemi odzivati pravilno. Danes že uveljavljena in sprejeta pristopa – SOA (Service Oriented Architecture) in EDA (Event Driven Architecture) – delno rešujeta težave agilnosti in kompleksnosti, vendar se z naraščanjem pomembnosti in števila dogodkov, na katere se želimo odzivati pravočasno in pravilno, kompleksnost spet povečuje. Za procesiranje preprostih dogodkov obstajajo pristopi in tehnološke rešitve, ki so danes že uveljavljene; problem je procesiranje kompleksnih dogodkov oz. CEP (Complex Event Processing), ki je zahtevno, a izredno zaželeno, saj ravno kompleksni dogodki vsebujejo pomembne informacije oz. novo znanje. Ker celovite in učinkovite rešitve za definiranje, analizo, procesiranje ter shranjevanje kompleksnih dogodkov na platformi SOA še ni, predlagamo model »komponente CEP«, ki to omogoča. V nadaljevanju so najprej predstavljene storitveno usmerjene arhitekture in dogodkovno gnane arhitekture, nato so opisane dosedanje raziskave in znanstvena dognanja. Tretji razdelek identificira in opisuje problematiko procesiranja kompleksnih dogodkov.

V četrtem razdelku sta predstavljena predlagani pristop k integraciji procesiranja kompleksnih dogodkov v platformo SOA in model predlagane komponente CEP. Peti razdelek vsebuje potencialni znanstveni doprinos in sklep.

1.1 Storitveno usmerjene arhitekture

Danes že uveljavljeni in sprejeti koncept storitveno usmerjenih arhitektur (SOA) predstavlja funkcionalne entitete obstoječih aplikacij, kot šibko sklopljene in ponovno uporabljive storitve. To množico interoperabilnih storitev, ki so uporabljive v različnih poslovnih domenah in informacijskih rešitvah, s kompozicijo ali orkestracijo povezujemo v poslovne procese. Kompozicija in orkestracija storitev se običajno realizira z jeziki, kot je Business Process Execution Language (BPEL) (Alves et al., 2006). Če bi kompozicija ali orkestracija storitev le-te povezovala neposredno, bi arhitekture takšnih informacijskih sistemov postale izredno kompleksne in nepregledne. Zato SOA vpelje storitveno vodilo ali ESB (Enterprise Service Bus), ki je vmesna programska oprema ali platforma, ki omogoča dinamično povezovanje in posredovanje storitev ter nadzor nad njihovimi interakcijami, saj odstrani neposredne povezave med ponudniki in uporabniki storitev. Tako nam zagotavlja šibko sklopljenost, povečano fleksibilnost, celovit pogled na povezljivost, varnost ter primerno dostavo storitev. Storitveno vodilo nam torej ponuja abstrakten pogled na storitve, saj zagotavlja poenostavljen vmesnik za komuniciranje in integracijo storitev.

Večina današnjih platform SOA temelji na specifikaciji SCA (Service Component Architecture), ki definira način izgradnje in model združevanja komponent v tehnološko neodvisne kompozite, iz katerih gradimo aplikacije. Vsaka komponenta je atomarna entiteta in vsebuje storitve (predstavljajo poslovno logiko oz. funkcionalnosti), reference (omogočajo dostop do storitev drugih komponent), specifične lastnosti (za definiranje obnašanja ipd.) in povezave (za interakcijo) (Beisiegel & Blohm, 2007).

Ponudniki in uporabniki storitev med seboj komunicirajo z uporabo natančno definiranih vmesnikov, prek katerih lahko storitve pošiljajo in prejemajo sporočila. Klicatelj oz. odjemalec proži specifično operacijo na določeni storitvi ali komponenti, ki obravnava in izvede prejeto zahtevo. Dvosmerne operacije vključujejo tudi odgovor, ki ga storitev kot rezultat vrne odjemalcu. Tak stil interakcije običajno povzroča tesnejšo sklopljenost in nižji nivo ponovne uporabe (Erl, 2005). Te slabosti lahko premostimo z uporabo koncepta EDA, ki je opisan v naslednjem razdelku.

1.2 Dogodkovno gnane arhitekture in procesiranje kompleksnih dogodkov

Koncept dogodkovno gnanih arhitektur (EDA) je arhitekturni pristop, ki temelji na generiranju, zaznavanju, sprejemanju in obravnavanju dogodkov. V splošnem je dogodek vsaka akcija, ki se zgodi zunaj naše aplikacije in jo lahko obravnavamo v njej. V kontekstu večjih informacijskih sistemov pa je dogodek definiran kot sprememba stanja poljubnega elementa poslovnega procesa, ki ima potencialni vpliv na tok izvajanja ali rezultat poslovnega procesa (Levina & Stantchev, 2009). Dogodki so lahko preprosti ali kompleksni. Primeri preprostih dogodkov so npr. izdaja računa, rezervacija letalske vozovnice in sporočilo od temperaturnega senzorja. Kompleksni dogodki so dogodki, ki nastanejo kot rezultat več preprostih dogodkov in običajno niso vidni. Primer je npr. nakup več letalskih vozovnic v kratkem časovnem intervalu za specifično destinacijo ali nepričakovano upadanje naročil za določeni izdelek. Za distribucijo dogodkov potrebujemo dogodkovno vodilo, ki je lahko realizirano kot MOM (Message Oriented Middleware) – JMS (Java Message Service) in MQ (Message Queue), informacijska rešitev za procesiranje dogodkov – Oracle CEP ali IBM WebSphere Business Events, ali kot storitveno vodilo, ki podpira propagacijo dogodkov.

Koncept EDA temelji na šibko sklopljenih komponentah, ki med seboj komunicirajo po vzorcu objavi–naroči, kar pomeni, da je vsak dogodek dostavljen vsem prejemnikom, ki so naročeni na ta dogodek. To implicira strogo delitev med generatorji dogodkov ali dogodkovnimi izvori in prejemniki dogodkov ali dogodkovnimi ponori, ki so poleg dogodkovnega senzorja in sistema za procesiranje dogodkov osnovne komponente dogodkovno gnane arhitekture. Dogodkovni senzor je odgovoren za pravilno zaznavanje dogodkov, sistem za procesiranje dogodkov pa vse dogodke evalvira in se nanje odzove z vnaprej definirano akcijo, ki je običajno definirana s poslovnimi pravili.

EDA definira tri tipe dogodkov s pripadajočimi mehanizmi za njihovo procesiranje:

- preprosti dogodek (SEP – Simple Event Processing) – nanaša se na specifično spremembo stanja,

ki lahko povzroči eno akcijo ali več. V sistemu za procesiranje preprostih dogodkov (SEP) se vedno procesira le en sam dogodek;
- dogodkovni tok (ESP – Event Stream Processing) – sistem za procesiranje toka dogodkov spremlja dogodke v nekem časovnem okviru, ki mu pravimo drsno okno. Dogodki se ne procesirajo individualno, ampak se ob prejemu vsakega novega dogodka ovrednoti množica vseh aktualnih dogodkov (tistih, ki so znotraj drsnega okna);
- kompleksni dogodek (CEP – Complex Event Processing) – sestavljen je iz več preprostih dogodkov, ki v sistem (ne) pridejo bodisi v določenem številu, časovnem intervalu ali določenem zaporedju. Sistem za procesiranje kompleksnih dogodkov poleg ESP omogoča tudi zaznavo kompleksnih vzorcev v množici vseh dogodkov.

V članku se usmerjamo na procesiranje kompleksnih dogodkov, saj so funkcionalnosti SEP in ESP že omogočene na današnjih platformah. CEP je tehnologija za filtriranje, koreliranje, agregiranje in analiziranje dogodkovnih podatkov, ki lahko izvirajo iz vseh plasti organizacije, kar nam omogoča, da pridobimo nova znanja iz distribuiranih sistemov (Eckert, Michael & Bry, 2009; Paschke, 2008). Stroji za procesiranje kompleksnih dogodkov za definiranje povezav in odvisnosti med dogodki uporabljajo klasifikacijo, poslovna pravila in filtre. Definiranje kompleksnih dogodkov je običajno realizirano s pomočjo sistemov za upravljanje s poslovnimi pravili (BRMS – Business Rule Management System), ki omogočajo nadzor in upravljanje procesiranja dogodkov v času izvajanja. Ko stroj za procesiranje kompleksnih dogodkov zazna kompleksni dogodek, lahko dogodek posreduje vsem naročenim komponentam ali pa se odzove neposredno (npr. klic specifične storitve, pošiljanje sporočila, zagon poslovnega procesa).

CEP nam omogoča razvoj varnejših in robustnejših informacijskih sistemov, saj se dogodki analizirajo in ovrednotijo pred vplivom na sistem. Tako npr. deluje programska oprema za upravljanje z letalom, ki pilotu ne pusti, da povzroči nevarno stanje v sistemu (dogodke, ki bi lahko povzročili nevarno stanje, stroj za procesiranje dogodkov zavrne).

1.1 Konvergenca SOA in EDA

Klasični pristop SOA ne zadošča potrebam in zahtevam modernega poslovnega okolja, saj njegova dinamičnost zahteva visoko agilno arhitekturo z avtonomnimi, šibko sklopljenimi komponentami in zmožnostjo odzivanja na dogodke. Na drugi strani nam EDA omogoča to dinamičnost, vendar sama ni primerna za celovito informacijsko arhitekturo, saj, prvič, ne omogoča najpogosteje uporabljenega načina komuniciranja – sinhrone dvosmerne komunikacije s sporočili (zahteva–odgovor), in drugič, povezave med komponentami so lahko nejasne in prikrite, kar povzroča težave pri upravljanju in obravnavanju napak.

Konvergenca SOA in EDA je torej smiselna; SOA 2.0 ali dogodkovno gnana SOA združuje dobre lastnosti obeh paradigem, saj razširja storitveno usmerjeni pristop z zmožnostmi dogodkovno gnanega koncepta in tako ponuja te prednosti:
- šibka sklopljenost med komponentami,
- boljša arhitekturna fleksibilnost,
- boljši vpogled v poslovne aktivnosti in poslovne procese,
- nadzor oz. spremljanje poslovnih aktivnosti in odzivnost na okolje (procesiranje dogodkov) v skoraj realnem času,
- zaznavanje in optimalno odzivanje na preproste in kompleksne dogodke.

Zavedati se moramo, da je dogodkovno gnana SOA ali SOA 2.0 arhitekturni pristop oz. koncept in da celovita SOA 2.0 platforma še ne obstaja.

2 PREGLED DOSEDANJIH RAZISKAV

V zadnjem desetletju je veliko avtorjev dognalo, da je za razvoj modernih informacijskih sistemov potrebna konvergenca SOA in EDA oz. razširitev platform SOA z dogodkovno gnanimi koncepti. Woods in Mattern (Woods & Mattern, 2006) sta natančno identificirala razloge in potencialne prednosti združevanja SOA in EDA. Taylor (Taylor, Yochem, Phillips, & Martinez, 2009) je potrdil smiselnost konvergence obeh konceptov ter dobro opisal njune ključne komponente. Michelson (Michelson, 2006) je definiral in opisal razlike med SOA in EDA in izpostavil relacijo med EDA in CEP. Marechaux (Marechaux, 2006) je opisal SOA in EDA kot dva različna koncepta in kot arhitekturni vzorec za podporo dogodkom (proženje, detekcija, distribucija) v arhitekturi SOA predlagal storitveno vodilo. Laliwala in Chaudhary (Laliwala & Chaudhary, 2008) sta prav tako prišla do spoznanja, da obstoječe platforme SOA ne podpirajo vseh konceptov dogodkovno gnanih arhitektur. Predlagala sta dogodkovno gnano SOA, ki temelji

na spletnih storitvah in semantičnem spletu. Tako bi lahko s pomočjo upravljavca dogodkov, stroja za kompozicijo in izvajanje, ontološkega strežnika, mrežnih storitev ter poslovnih pravil avtomatizirala dogodkovno gnane poslovne procese. Wieland (Wieland, Martin, Kopp, & Leymann, 2009) je odkril, da sta SOA in EDA edinstvena arhitekturna stila, ki se v industriji večinoma uporabljata ločeno. Definiral je tudi metodo za razvoj poslovnih procesov SEODA, ki v BPEL propagira dogodke. Jurič (Juric, 2010) je za podporo konceptov EDA v SOA prvi predlagal razširitve WSDL in BPEL. Z razširitvami za definiranje dogodkovnih izvorov in ponorov WSDL lahko definiramo in uporabljamo dogodke na ravni spletnih storitev. BPEL je avtor razširil z dodatnimi aktivnostmi, ki omogočajo dogodkovno gnano izvajanje poslovnih procesov. Vsi predlagani koncepti nam omogočajo asinhrono komunikacijo z vzorcem objavi–naroči in procesiranje preprostih dogodkov.

V zadnjih letih so bile predlagane, razvite in izdane številne specifikacije, kot sta WS-Eventing (Box et al., 2004) in WS-Notification (P. Niblett & S. Graham, 2005). WS-Eventing definira dogodkovno storitev z naborom operacij, ki spletni storitvi omogočajo asinhrono oz. dogodkovno gnano komunikacijo z drugimi storitvami. WS-Notification sestoji iz treh specifikacij, s pomočjo katerih si lahko spletne storitve izmenjujejo sporočila: WS-BaseNotification (Steve Graham, Hull, Murray, & Event-driven, 2006), ki definira interakcijo med generatorji in prejemniki obvestil, WS-BrokeredNotification (Liu, 2006), ki definira vmesnike za posrednike obvestil, in WS-Topics (Peter Niblett, 2006), ki definira hierarhično področje za teme. Niblett in Graham sta naredila celovit pregled specifikacije WS-Notificaton (P. Niblett & S. Graham, 2005), Huang in Gannin pa sta kmalu za tem primerjala WS-Eventing in WS-Notification (Huang, 2006). Leta 2006 so HP, IBM, Intel in Microsoft predlagali specifikacijo WS-EventNotification, ki združuje WS--Eventing in WS-Notification (Kevin Cline et al., 2006). Procesiranje preprostih dogodkov in asinhrono komunikacijo na ravni SCA je s pomočjo razširitev predlagal Beisiegel (Beisiegel & Vorthmann, 2009).

Vse omenjene specifikacije nam omogočajo uporabo in procesiranje le preprostih dogodkov. Celovit pristop k vpeljavi procesiranja kompleksnih dogodkov v SOA še ni bil opisan ali predlagan.

3 PREDSTAVITEV PROBLEMA

Veliko število današnjih informacijskih sistemov predstavljajo distribuirani informacijski sistemi, kar pomeni visoko raven kompleksnosti, ki je posledica kompleksnosti samih poslovnih procesov in okolja, ki vpliva na izvajanje le-teh. Procesna usmerjenost oz. upravljanje poslovnih procesov (BPM – Business Process Management) in SOA zmanjšujeta to kompleksnost, pomembnost kompleksnih dogodkov, želja po optimizaciji, dinamičnosti in učinkovitem odzivanju informacijskih sistemov na okolje (dogodke), pa jo, žal, zvišujejo. Z večanjem števila storitev in senzorskih naprav (Internet of Things) se število dogodkov povečuje, kar implicira pomembnost integracije, konsolidacije in inteligentne obravnave distribuiranih dogodkov. Problem današnjih platform SOA je, da delno podpirajo koncepte dogodkovne usmerjenosti (SOA 2.0 ali Event-driven SOA), ne podpirajo pa konceptov procesiranja kompleksnih dogodkov (podpirajo SEP). CEP zajema metode, tehnike in orodja za procesiranje dogodkov v času izvajanja in omogoča izpeljavo kompleksnih dogodkov, ki nosijo obogatene informacije in znanje. Pri realizaciji sodobnih dinamičnih in distribuiranih informacijskih sistemov, ki podpirajo dogodkovno vodene koncepte, se tako še vedno soočamo z izzivi:

- Kako upravljati z velikim številom distribuiranih dogodkov?
- Kako prepoznati kompleksne vzorce iz množice preprostih dogodkov?
- Kako pridobiti znanje iz meddogodkovnih povezav?
- Kako pridobiti zmožnost napovedovanja dogodkov?
- Kako učinkovito implementirati in uporabljati poslovna pravila, ki bi podpirala tudi CEP?
- Kako učinkovito integrirati CEP, sisteme za upravljanje s poslovnimi pravili (BRMS) in sisteme za spremljanje poslovnih aktivnosti (BAM)?

Zaradi omenjenih problemov danes vse platforme SOA rešujejo CEP z ločenimi produkti (npr. IBM WebSphere Business Events ali Oracle Complex Event Processing). Kljub omenjenim informacijskim rešitvam, ki omogočajo CEP, je razvoj inteligentnih in agilnih informacijskih sistemov SOA, ki uporabljajo CEP, izredno zahtevno. Za opisane probleme predlagamo rešitev, ki je opisana v naslednjem razdelku.

4 PREDLAGANA REŠITEV

Predlagamo inovativen pristop k integraciji CEP in SOA. Naš cilj je vpeljati zmožnost celovitega procesiranja kompleksnih dogodkov, med katere ta semantično spada – na hrbtenico vsake platforme SOA oz. na storitveno vodilo (ESB). Ko govorimo o celovitem procesiranju kompleksnih dogodkov, mislimo na definiranje, zaznavanje, analiziranje, obravnavanje, predobdelavo, obdelavo, konsolidiranje, filtriranje, klasificiranje, agregiranje, integriranje, usmerjenje in posredovanje dogodkov.

4.1 Vpeljava procesiranja kompleksnih dogodkov v storitveno vodilo s komponento CEP

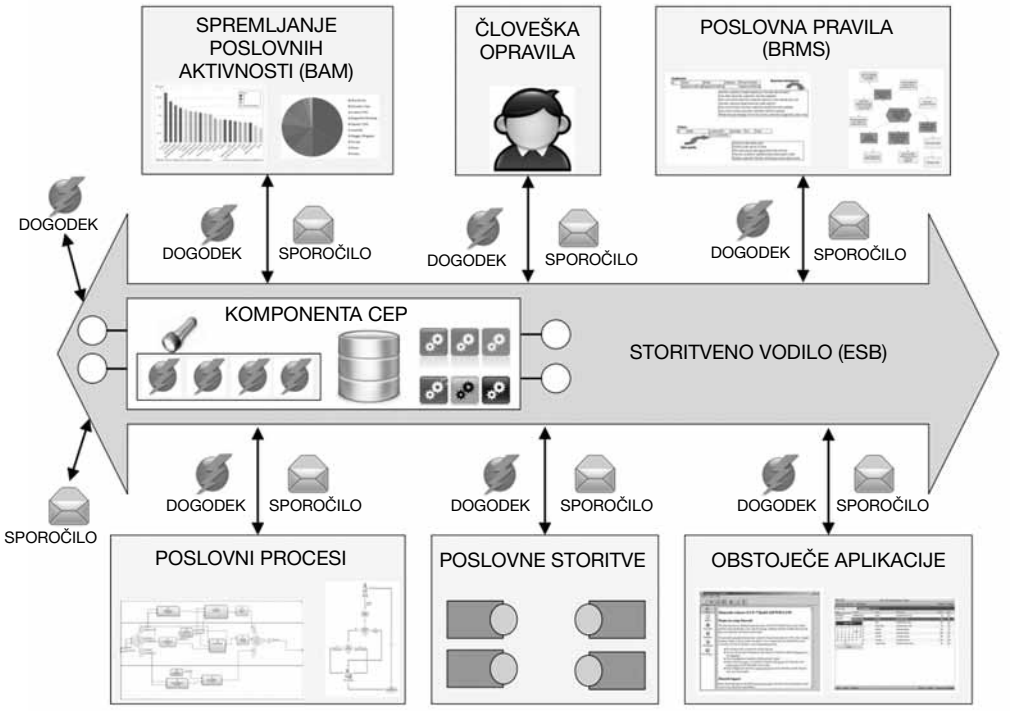
Marechaux (Maréchaux, 2006) je v svojem članku pokazal, da je storitveno vodilo idealen arhitekturni vzorec za procesiranje dogodkov, saj pomeni centralizirano entiteto za komunikacijo med storitvami, procesi, aplikacijami, človeškimi opravili, hkrati pa strogo razdvoji ponudnike in uporabnike storitev, kar je ključnega pomena za dogodkovno gnano (asinhrono objavi–naroči) komunikacijo.

Za vpeljavo procesiranja kompleksnih dogodkov predlagamo t. i. avtonomno in celovito komponento CEP, ki je skladna s specifikacijo SCA (Beisiegel & Blohm, 2007). Skladnost s specifikacijo SCA je pomembna, saj želimo, da se da komponento CEP preprosto integrirati v obstoječe platforme SOA (na storitveno vodilo), ki temeljijo na SCA. Predlagani model je tipičen kompozit SCA, ki vsebuje osem komponent. Vse komponente in njihove medsebojne povezave so opisane v naslednjem razdelku.

Kot lahko vidimo na sliki 1, predlagana komponenta CEP ni povezana na storitveno vodilo kot sistemi za upravljanje s poslovnimi pravili (BRMS) ali sistemi za spremljanje poslovnih aktivnosti (BAM), ampak je popolnoma integrirana v ogrodje storitvenega vodila. Vsebuje:

- lastno vrsto za dogodke,
- podatkovni vir za shranjevanje,
- mehanizme za definiranje in procesiranje kompleksnih dogodkov in
- mehanizme za integracijo z zunanjimi storitvami in sistemi.

Komponenta uporablja interne podatkovne sheme, ki so primerne za učinkovito definiranje in procesiranje kompleksnih dogodkov ter shranjevanje le-teh. Vmesniki so definirani generično, kar omogoča integracijo z eksternimi sistemi prek storitvenega vodila.



Slika 1: **Integracija komponente CEP v storitveno vodilo**

Ker sisteme BRMS in BAM danes pogosto uporabljajo, menimo, da je zmožnost integracije komponente CEP z njimi pomembna. V ta namen komponenta vsebuje posebne vmesnike in programabilne transformacijske module oz. vtičnike, prek katerih jo lahko povežemo z obstoječimi BRMS in BAM. Transformacijski vtičniki so ključnega pomena, saj zagotavljajo pravilno dvosmerno preslikavo definicij kompleksnih dogodkov in drugih podatkov v primerne podatkovne formate. To nam omogoča, da kompleksne dogodke in odzive nanje definiramo z obstoječimi sistemi BRMS in da za spremljanje poslovnih aktivnosti in kompleksnih dogodkov (kompleksni KPI – Key Performance Indicators) lahko uporabljamo obstoječe sisteme BAM.

4.2 Predlagani model komponente CEP

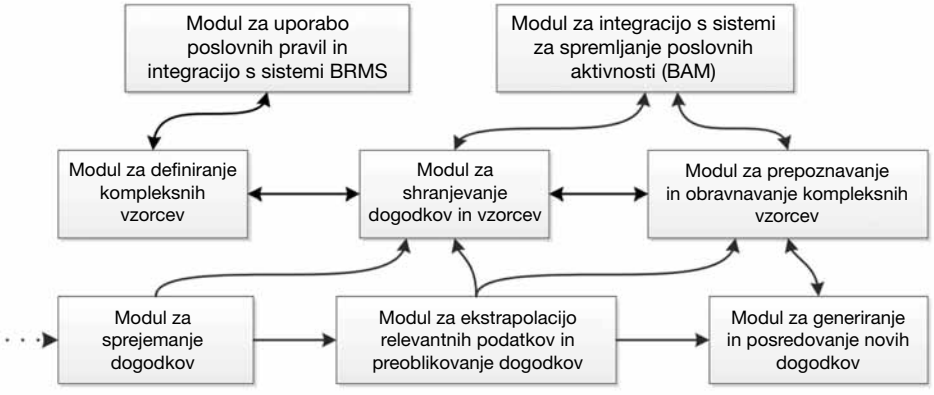
Za učinkovito vpeljavo celovitega procesiranja kompleksnih dogodkov (definiranje, zaznavanje, analiziranje, obravnavanje, predobdelava, obdelava, konsolidiranja, filtriranje, klasificiranje, agregiranje, integriranje, usmerjenje in posredovanje kompleksnih dogodkov) v storitveno vodilo predlagamo konceptualni model kompozita SCA, ki ga sestavlja osem modulov:

- *modul za sprejemanje dogodkov* – ta modul s storitvenega vodila sprejema vse dogodke in jih posreduje potrebnim modulom ali zunanjim sistemom;
- *modul za ekstrapolacijo relevantnih podatkov in preoblikovanje dogodkov* – ta modul iz dogodkovnih podatkov izlušči pomembne podatke, ki so potrebni za optimalno procesiranje. Dogodke za posredovanje ali boljšo obdelavo lahko modul tudi preoblikuje;
- *modul za definiranje kompleksnih vzorcev* – ta modul služi za konstrukcijo vzorcev, s katerimi opišemo kompleksne dogodke (vzorce je mogoče definirati tudi v zunanjih sistemih BRMS). Za preprosto in učinkovito definiranje kompleksnih vzorcev lahko uporabimo posebne strukture. K temu modulu spada tudi grafični vmesnik, ki pa je lahko realiziran poljubno, ker je definiranje vzorca realizirano kot človeško opravilo z natančno specificiranim vmesnikom. Tako za definiranje kompleksnih dogodkov oz. vzorcev dosežemo popolno tehnološko neodvisnost;
- *modul za shranjevanje dogodkov in vzorcev* – ta modul je odgovoren za shranjevanje vseh dogodkov in definiranih vzorcev. Shranjevanje dogodkov bo potekalo na dva načina:
  - kratkoročno shranjevanje bo namenjeno ESP (procesiranje dogodkovnih tokov) z drsečim oknom. Procesirajo se zadnji n-dogodki ali dogodki v določenem časovnem intervalu;
  - dolgoročno shranjevanje (v podatkovno bazo) je namenjeno arhiviranju vseh dogodkov. To omogoča kasnejšo analizo in prepoznavo kompleksnih vzorcev.

  Vzorci (njihovi formalni opisi oz. definicije) se prav tako shranjujejo na dva načina. Aktivni vzorci (tisti, ki jih trenutno prepoznavamo) so shranjeni v pomnilniku, pasivni vzorci pa so arhivirani v podatkovni bazi. Vsi vzorci se shranjujejo na dva načina:
  - v internem formatu, ki ga definira komponenta CEP;
  - v originalnem formatu, ki ga definirajo zunanji sistemi BRMS (original potrebujemo za dvosmerno sinhronizacijo);
- *modul za uporabo poslovnih pravil in integracijo z BRMS sistemi* – ta modul omogoča integracijo z zunanjimi sistemi BRMS, kar pomeni možnost uporabe poslovnih pravil, ki so definirana zunaj našega sistema. Modul je prek vtičnikov sposoben sprejemati poslovna pravila različnih formatov in jih transformirati v interno shemo kompleksnih dogodkov. Tako omogočimo uporabo že definiranih pravil, hkrati pa lahko uporabniki za definiranje kompleksnih dogodkov in njihovih posledic uporabljajo obstoječe sisteme;
- *modul za integracijo s sistemi za spremljanje poslovnih aktivnosti (BAM)* – ker je spremljanje poslovnih aktivnosti za poslovanje izredno pomembno, komponenta CEP omogoča integracijski modul za uporabo zunanjih sistemov BAM. Modul informacije o kompleksnih dogodkih tako lahko posreduje različnim sistemom BAM;
- *modul za prepoznavanje in obravnavanje kompleksnih vzorcev* – ta osrednji modul CEP je odgovoren za prepoznavanje kompleksnih aktivnih vzorcev. Ko je zaznan kompleksni dogodek, modul poskrbi za pravilno obravnavo (npr. sproži določeno storitev, človeško opravilo ali proces, dogodek posreduje v druge sisteme ali ga samo zabeleži);
- *modul za generiranje in posredovanje novih dogodkov* – ta modul generira dogodke in jih posreduje na pravilne končne točke. Iz vhodnih podatkov je sposoben generirati dogodek pravilnega tipa in ga posredovati zunanjim ali internim sistemom.

Največ ga uporablja modul za prepoznavanje in obravnavanje dogodkov.

Na sliki 2 je prikazana shema predlaganega modela komponente CEP. Vhod v komponento je spodaj levo – v modulu za sprejemanje dogodkov. Ta modul zazna dogodek in ga posreduje v modula za shranjevanje dogodkov in za inteligentno predobdelavo (ekstrapolacija relevantnih podatkov in preoblikovanje dogodkov v primernejšo obliko). Zadnji modul shrani preoblikovani dogodek, hkrati pa ga preda modulu za prepoznavanje in obravnavanje kompleksnih vzorcev. Za proženje vnaprej definiranih akcij poskrbi modul za generiranje in posredovanje novih dogodkov. Pomemben del predlagane komponente je definiranje kompleksnih vzorcev, ki se posredno lahko vrši v modulu za uporabo poslovnih pravil in integracijo s sistemi BRMS, neposredno pa v modulu za definiranje kompleksnih vzorcev. Modul za integracijo s sistemi BAM poskrbi za učinkovito in transparentno komunikacijo z zunanjimi sistemi za spremljanje poslovnih aktivnosti. Za boljše razumevanje ponazorimo delovanje modela s primerom. V obstoječem sistemu BRMS definiramo kompleksni vzorec, ki ga sestavlja število transakcij (na uporabnika), ki je za 30 odstotkov višje od povprečja, v časovnem intervalu 90 minut, vsaka od transakcij pa presega znesek 20 evrov. Ko nastopi kompleksni dogodek, želimo poklicati določeno spletno storitev. Ta vzorec se v modulu za definiranje kompleksnih vzorcev transformira v primerno obliko in shrani v podatkovno bazo. Ko v sistem začnejo prihajati dogodki, jih modul za sprejemanje pošlje v obdelavo in jih shrani. Modul za prepoznavanje in obravnavanje kompleksnih vzorcev ob nastopu vsakega dogodka preveri, ali so izpolnjeni pogoji za nastop kompleksnega dogodka – če so, poskrbi za pravilno obravnavno. Ko v sistem prispe prva transakcija, ki presega mejni znesek, se zaradi časovnega intervala zažene časovnik, ki teče največ 90 minut. Če v tem intervalu število transakcij preseže mejo, modul za prepoznavanje in obravnavanje pokliče ustrezno storitev in shrani kompleksni dogodek. Sistem BAM lahko ta dogodek prikaže na poslovnem portalu in tako omogoča učinkovit nadzor in upravljanje.



Slika 2: **Shema predlaganega modela**

Za konvergenco storitveno usmerjenih sekvenčnih poslovnih procesov in naključno zaznanih kompleksnih dogodkov lahko uporabimo mehanizme za obravnavo dogodkov, s katerimi se lahko pravilno odzivamo na dogodke. Predlagana komponenta CEP nam omogoča, da pred dejanskim vplivom dogodkov na poslovni proces na inteligenten način obravnavamo večje število preprostih in kompleksnih dogodkov (neodvisno od poslovnih procesov). S tem omogočimo optimalnejši razvoj poslovnih procesov, saj kompleksno logiko za obravnavanje dogodkov prestavimo v centralizirano komponento, ki nam zagotovi prepoznavanje kompleksnih dogodkov. Poslovni procesi tako postanejo bolj pregledni in inteligentni, saj se lahko odzivajo na kakršne koli dogodke. Ker predlagana komponenta hrani vse dogodke in vzorce, lahko nad temi podatki izvajamo analize (poslovna inteligenca) in pridemo do novih znanj,

s katerimi lahko optimiziramo poslovne procese in definiramo vzorce za nove kompleksne dogodke ali kompleksne kazalnike uspešnosti (KPI).

5 SKLEP

V članku sta predstavljena koncepta storitveno usmerjenih arhitektur in dogodkovno gnanih arhitektur, ki sta dva arhitekturna pristopa za integracijo informacijskih sistemov. Izpostavili smo razlike v komunikaciji in pomanjkljivosti obeh pristopov ter pokazali, da je njuna konvergenca smiselna in nujna za razvoj modernih agilni in inteligentnih sistemov. Predstavili smo dosedanje raziskave s področja SOA in EDA in prišli do spoznanja, da trenutne arhitekture oz. platforme ne omogočajo procesiranja kompleksnih dogodkov. Ker CEP postaja za optimalno izvajanje poslovnih procesov izrednega pomena, menimo, da bi procesiranje kompleksnih dogodkov moralo biti del same platforme SOA 2.0 (Event-Driven SOA). Za integracijsko platformo, ki bi omogočala vpeljavo CEP v SOA, smo identificirali storitveno vodilo (ESB), saj ta ponuja idealno ogrodje za prejemanje in distribucijo podatkov. Predlagali smo avtonomno komponento CEP, ki omogoča celovito procesiranje kompleksnih dogodkov (definiranje, zaznavanje, analiziranje, obravnavanje, predobdelava, obdelava, konsolidiranja, filtriranje, klasificiranje, agregiranje, integriranje, usmerjenje in posredovanje kompleksnih dogodkov). Predlagana komponenta CEP je skladna s specifikacijo SCA, visoko konfigurabilna in preprosta za integracijo v obstoječe platforme SOA oz. njihova storitvena vodila. Model komponente sestavlja osem modulov oz. komponent SCA, ki so odgovorne za posamezne funkcionalne zahteve procesiranja kompleksnih dogodkov. S predlaganim pristopom ponujamo nov model za procesiranje kompleksnih dogodkov v obstoječih in razvijajočih se platformah SOA. S centraliziranim nadzorom in upravljanjem preprostih in kompleksnih dogodkov ter z učinkovitim procesiranjem le-teh so omogočeni krajši čas razvoja zaradi preprostejše arhitekture, višja stopnja agilnosti in odzivnosti na okolje in boljši pregled nad poslovnimi aktivnostmi in procesi.

6 VIRI IN LITERATURA

[1] Alves, A., Arkin, A., Askary, S., Bloch, B., Curbera, F., Goland, Y., Kartha, N. et al. (2006). *Web services business process execution language version 2.0*. *Language* (pp. 1–264). Retrieved from http://www.citeulike.org/group/2540/article/1233272.

[2] Beisiegel, M., & Blohm, I H. (2007). *SCA Service Component Architecture – Assembly Model Specification*. *Citeseer*. Citeseer. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.168.5919&rep=rep1&type=pdf.

[3] Beisiegel, M., & Vorthmann, S. (2009). *SCA Extensions for Event Processing and Pub/Sub*. Retrieved from http://www.osoa.org/download/attachments/35/SCA_Assembly_Extensions_for_Event_Processing_and_PubSub_V1_0.pdf?version=1.

[4] Box, D., Cabrera, L. F., Critchley, C., Curbera, Francisco, Ferguson, D., Graham, Steve, Hull, D. et al. (2004). Web Services Eventing (WS-Eventing). Retrieved from http://www.w3.org/TR/2009/WD-ws-eventing-20090317/.

[5] Eckert, Michael, F. & Bry, C. (2009). Complex Event Processing (CEP). *Informatik-Spektrum*, *32*(2), 163–167.

[6] Erl, T. (2005). *Service-oriented architecture: concepts, technology, and design* (p. 792). Prentice Hall PTR. Retrieved from http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Service-oriented+architecture:+concepts,+technology,+and+design#0.

[7] Graham, Steve, Hull, D., Murray, B. & Event-driven, T. (2006). Web Services Base Notification 1.3. *October*. Retrieved from http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf.

[8] Huang, Y. (2006). A comparative study of web services-based event notification specifications. *2006 International Conference on Parallel Processing Workshops (ICPPW'06)*, 7–14. Ieee. doi:10.1109/ICPPW.2006.5.

[9] Juric, M. B. (2010). WSDL and BPEL extensions for Event Driven Architecture. *Information and Software Technology*, *52*(10), 1023–1043. Elsevier B.V. doi:10.1016/j.infsof.2010.04.005.

[10] Kevin Cline, Cohen, J., Davis, D., Ferguson, D. F., Kreger, H., Mccollum, R., Murray, B. et al. (2006). *Toward Converging Web Service Standards for Resources, Events, and Management*. *Changes* (pp. 1–9).

[11] Laliwala, Z. & Chaudhary, S. (2008). Event-driven Service-Oriented Architecture. *2008 International Conference on Service Systems and Service Management*, 1–6. Ieee. doi:10.1109/ICSSSM.2008.4598452.

[12] Levina, O. & Stantchev, V. (2009). Realizing Event-Driven SOA. *2009 Fourth International Conference on Internet and Web Applications and Services*, 37–42. Ieee. doi:10.1109/ICIW.2009.14.

[13] Liu, L. (2006). Web Services Brokered Notification 1.3. *October*. Retrieved from http://docs.oasis-open.org/wsn/wsn-ws_brokered_notification-1.3-spec-os.pdf.

[14] Maréchaux, J.-L. (2006). *Combining Service-Oriented Architecture and Event-Driven Architecture using an Enterprise Service Bus*. Retrieved from http://www.ibm.com/developerworks/webservices/library/ws-soa-eda-esb/index.html.

[15] Michelson, B. M. (2006). Event-Driven Architecture Overview. *Architecture*, 8. doi:10.1571/bda2-2-06cc.

[16] Niblett, P. & Graham, S. (2005). Events and service-oriented architecture: The OASIS Web Services Notification specification. *IBM Systems Journal*, *44*(4), 869–886. doi:10.1147/sj.444.0869.

[17] Niblett, Peter. (2006). Web Services Topics 1.3. *October*. Retrieved from http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-os.pdf.

[18] Paschke, A. (2008). Design patterns for complex event processing. *2nd International Conference on Distributed Event--Based Systems (DEBS'08)*. Retrieved from http://arxiv.org/abs/0806.1100.

[19] Taylor, H., Yochem, A., Phillips, L. & Martinez, F. (2009). *Event--Driven Architecture: How SOA Enables the Real-Time Enterprise*. (M. Taub, G. Doench, M. Thurston, K. Hart, B. Hassir, & K. Annett, Eds.) (p. 308). Addison-Wesley.

[20] Wieland, M., Martin, D., Kopp, O. & Leymann, F. (2009). SOEDA: A Methodology for Specification and Implementation of Applications on a Service-Oriented Event-Driven Architecture. In W. Abramowicz (Ed.), *Proceedings of the 12th International Conference on Business Information Systems BIS 2009 Poznan Poland April 2729 2009* (Vol. 21, pp. 193–204). Springer Verlag. doi:10.1007/978-3-642-01190-0_17.

[21] Woods, D. & Mattern, T. (2006). *Enterprise SOA: Designing IT for Business Innovation* (p. 452). O'Reilly.

▪

Martin Potočnik je diplomiral na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru in se kot raziskovalec zaposlil v laboratoriju za tehnologije komuniciranja. Kasneje je postal mladi raziskovalec in doktorski študent Fakultete za računalništvo in informatiko Univerze v Ljubljani, kjer je zaposlen v laboratoriju za integracijo informacijskih sistemov. Raziskovalno se ukvarja predvsem z integracijskimi izzivi, storitveno usmerjenimi arhitekturami in računalništvom v oblaku. Sodeluje tudi v aplikativnih industrijskih projektih ter se aktivno udeležuje številnih konferenc s področja informatike.

▪

Matjaž B. Jurič je redni profesor na Fakulteti za računalništvo in informatiko Univerze v Ljubljani, kjer vodi laboratorij za integracijo informacijskih sistemov. Je avtor oz. soavtor štirinajstih knjig. Sodeloval je pri številnih projektih doma in v tujini, med drugim tudi pri razvoju RMI-IIOP, sestavnega dela platforme Java 2, in je član BPEL Advisory Boarda. Leta 2007 je od SOA World Journal dobil nagrado za najboljšo knjigo s področja SOA, leta 2010 pa nagrado za najodmevnejši znanstveni članek s področja storitev iz NMS. Ima naziva Java Champion in Oracle ACE Director.