Univerza v Mariboru

FERI
Fakulteta za elektrotehniko, računalništvo in informatiko

Milan Dojchinovski

RAZVOJ SESTAVLJANK Z JEZIKOM EMML

Diplomsko delo

Maribor, september 2010

Diplomsko delo univerzitetnega študijskega programa

RAZVOJ SESTAVLJANK Z JEZIKOM EMML

Študent:	Milan Dojchinovski
Študijski program:	UN ŠP − Računalništvo in informatika
Smer:	Informatika
Mentor:	red. prof. dr. Marjan Heričko
Somentor:	asist. dr. Luka Pavlič
Lektorica:	Darinka Verdonik

Maribor, september 2010

Univerza v Mariboru

FERI
Fakulteta za elektrotehniko, računalništvo in informatiko
Smetanova ulica 17
2000 Maribor

Številka: RI-566
Datum in kraj: 09. 09. 2010, Maribor

Na osnovi 330. člena Statuta Univerze v Mariboru (Ur. l. RS, št. 1/2010)

SKLEP O DIPLOMSKEM DELU

1. **Milanu Dojchinovskemu,** študentu univerzitetnega študijskega programa RAČUNALNIŠTVO IN INFORMATIKA, smer Informatika, se dovoljuje izdelati diplomsko delo pri predmetu Gradnja informacijskih sistemov.

2. **MENTOR: red. prof. dr. Marjan Heričko**
   SOMENTOR: asist. dr. Luka Pavlič

3. **Naslov diplomskega dela:**
   RAZVOJ SESTAVLJANK Z JEZIKOM EMML

4. **Naslov diplomskega dela v angleškem jeziku:**
   DEVELOPING MASHUP APPLICATIONS USING EMML

5. Diplomsko delo je potrebno izdelati skladno z "Navodili za izdelavo diplomskega dela" in ga oddati v treh izvodih (en vezan izvod in dva nevezana izvoda) ter en izvod elektronske verzije do 09. 09. 2011 v referatu za študentske zadeve.

Pravni pouk: Zoper ta sklep je možna pritožba na senat članice v roku 3 delovnih dni.

Dekan:

Obvestiti:
- kandidata,
- mentorja,
- somentorja,
- odložiti v arhiv.

ZAHVALA

Zahvaljujem se mentorju rednemu profesorju doktorju Marjanu Heričku za pomoč in vodenje pri opravljanju diplomskega dela. Prav tako se zahvaljujem somentorju asistentu doktorju Luki Pavliču za strokovno pomoč pri opravljanju diplomskega dela.

Posebna zahvala velja staršem, Dragiju in Ljubici, ter sestri Mariji za neizmerno motivacijo in podporo med študijem.

Hvala Hanki, ki mi je vedno stala ob strani.

Prijatelji vedo, zakaj se jim zahvaljujem. Hvala tudi njim.

RAZVOJ SESTAVLJANK Z JEZIKOM EMML

Ključne besede: podjetniške sestavljanke, EMML, platforma Presto

UDK: 004.434(043.2)

Povzetek

V diplomskem delu podrobno predstavimo podjetniške sestavljanke in jezik EMML. Obdelamo arhitekturo podjetniških sestavljank za lažje identificiranje izzivov, ki jih le-te prinašajo, in izpostavimo potrebo po vpeljavi podjetniških sestavljank v podjetjih. Sledi podroben opis jedra jezika EMML kot standarda za razvoj podjetniških sestavljank. Izpostavimo prednosti, ki jih jezik EMML prinaša, ter identificiramo morebitne ovire. Razvoj sestavljank z jezikom EMML prikažemo na praktičnem primeru z izdelavo sestavljank za nadzor mednarodne izmenjave študentov.

DEVELOPING MASHUP APPLICATIONS USING EMML

Keywords: Enterpise mashups, EMML, Presto platform

UDK: 004.434(043.2)

Abstract

In this final work we present in details the enterprise mashups and the Enterprise Mashup Markup Language. We go through the enterprise mashups architecture for easier identificaton of the challenges they bring and we stress the need for implementation of enterprise mashups in enterprises. After that follows a detailed description of the core of the EMML as an enterprise mashup development standard. We present the advantages the EMML language brings and we identify possible obstacles. Developing enterprise mashups using EMML is presented on a practical case with development of a mashups for supervision of international student exchanges.

Kazalo

Kratice

EMML	Enterprise Mashup Markup Language
XML	eXtensible Markup Language
EMP	Enterprise Mashup Platform
SOAP	Simple Object Access Protocol
WSDL	Web Service Description Language
SOA	Service Oriented Architecture
API	Application Programming Interface
BPEL	Business Process Execution Language
CRM	Customer Relationship Management
OMA	Open Mashup Alliance
BPM	Business Process Management
ERP	Enterprise Resource Planning
ESB	Enterprise Service Bus
AAD	Information Access, Augmentation and Delivery
ECM	Enterprise Content Management
IT	Information Technology
BI	Business Intelligence

Poglavje 1

UVOD

Obdobje, v katerem smo, povzroča generiranje podatkov z eksponentno hitrostjo [1]. Tako generirani podatki prihajajo iz več različnih podatkovnih virov in v različnih strukturiranih in nestrukturiranih formatih. Sestavljanje teh podatkov pa je pogojeno z njihovo zmožnostjo za sestavljanje. Tehnologija podjetniških sestavljank izplava na površino kot nov strateški pregled nad podjetniškimi informacijami, s hitrim in enostavnim povezovanjem ter kombiniranjem podjetniških informacijskih virov za hitro pridobivanje kritičnih informacij za nosilce odločitev v podjetju.

Podjetniške sestavljanke ponujajo uporabniško orientiran model za sestavljanje podatkov v realnem času iz več notranjih in zunanjih podatkovnih virov [2]. Kljub temu da razcvet tehnologije podjetniških sestavljank šele prihaja, so se razvijalci tudi v preteklosti srečali s tehnologijo sestavljanja podatkov, ne da bi se tega zavedali.

Ponujeni model poveča produktivnost podjetja s prevzemom vloge razvijalca s strani poslovnih uporabnikov. Samopostrežni pristop izdelave aplikacij skrajša čas, ki je potreben za dobavo novih aplikacij, in poveča inovativnost poslovnih uporabnikov. Znotraj okolja za razvoj podjetniških sestavljank uporabniki uporabljajo, ustvarjajo, prilagajajo za svoje potrebe in souporabljajo sestavljanke v varnem in nadzorovanem okolju. Do nedavnega je razvoj podjetniških sestavljank temeljil na standardih, kot so XML, HTML/XHTML, JavaScript, Ajax, REST, SOAP, JSON itd., vendar tako zgrajene rešitve ne ponujajo interoperabilnosti in prenosljivosti. Na tem mestu nastopi jezik EMML, ki ponuja

rešitev teh problemov s standardizacijo razvoja podjetniških sestavljank. Ponuja prenosljivost sestavljalskih rešitev, interoperabilnost ter odstranjuje podjetniško zaklepanje navznoter. Standardiziran razvoj sestavljalskih rešitev dodatno zagotovi kompatibilnost, varnost in visoko kakovost.

Cilj diplomskega dela je predstaviti podjetniške sestavljanke in spodbuditi njihovo uporabo v podjetjih. Poleg tega predstavimo jezik EMML in potrebe po standardiziranem razvoju, arhitekturo jezika EMML in njegove jedrne ukaze ter identificiramo morebitne ovire in prednosti za prihodnost jezika EMML.

V drugem poglavju razjasnimo termin podjetniških sestavljank ter opišemo njihovo arhitekturo. Predstavimo njihove prednosti in najbolj aktualne platforme za sestavljanje ter identificiramo varnostne izzive, ki jih sestavljanke prinašajo.

Tretje poglavje je namenjeno jeziku EMML. Predstavimo podporna okolja jezika, ključne lastnosti, arhitekturo ter jedro jezika. Dodatno umestimo jezik v kontekst konkurenčnih jezikov in ocenimo njegovo prihodnost.

V četrtem poglavju na praktičnem primeru prikažemo uporabo in gradnjo podjetniških sestavljank z uporabo jezika EMML. Predstavimo problem, postopek razvoja in implementacijo rešitve.

V zadnjem poglavju povzamemo osrednje ugotovitve diplomskega dela.

Poglavje 2

PODJETNIŠKE SESTAVLJANKE

Število nezdružljivih notranjih in zunanjih virov, ki vsebujejo podatke za kritične odločitve v podjetju, sorazmerno poveča kompleksnost pridobivanja in sestavljanja podatkov. Kritične informacije za nosilce odločitev so običajno razširjene čez repozitorije sistemov za upravljanje odnosov s strankami (*angl. Customer Relationship Managment* – CRM), plačilne sisteme, računovodske sisteme, celovite poslovne informacijske rešitve (*angl. Enterprise Resource Planning* – ERP), sisteme za upravljanje vsebin (*angl. Enterprise Content Managment* – ECM) in druge lastne sisteme. Učinkovito zbiranje in procesiranje teh informacij je vedno predstavljalo izziv za informacijske strokovnjake. Če so sestavljanke (*angl. Mashups*) po definiciji spletne aplikacije, ki uporabljajo in združujejo podatke iz dveh ali več zunanjih virov, da bi kreirali novo storitev, kaj so potem podjetniške sestavljanke (*angl. Enterprise Mashups*)?

Podjetniške sestavljanke so dinamične spletne aplikacije, ki v realnem času združujejo več podatkovnih virov za večjo ozaveščenost in izboljšano odločanje ob izpolnjevanju strogih zahtev upravljanja in varnosti podatkov podjetja [3]. Ponujajo model in stimulirajo medsebojno sodelovanje v smislu neposrednega oblikovanja načina uporabe, interakcije in generiranja informacij v omrežju. Podjetniške sestavljanke predstavljajo velik korak naprej od tradicionalnega modela integracije v poslovnem svetu, »izreži-in-prilepi«. So uporabniško usmerjene in ponujajo možnost za mikrointegracijo podatkov. Ker so zgrajene na osnovi podatkov, smiselno ponujajo možnost prikazovanja podatkov z že obstoječimi

tehnologijami, kot so WSDL, REST in RSS. Preko modela sestavljank dobimo občutek za bogastvo, skrito v neizkoriščenih virih, in informacije znotraj podjetniških omrežij in interneta.

2.1 Tipi sestavljank

Termin sestavljanka se je v preteklosti interpretiral na več načinov, kar je prispevalo k zmedi pri razumevanju pravega pomena in namena sestavljank. Beseda ima korenine v glasbeni industriji kot koncept kombinacije dveh ali več pesmi za ustvarjanje novega doživetja.

Razvoj tehnologije je razširil to definicijo, da bi zajeli aplikacije, ki opisujejo način kombiniranja podatkov iz dveh ali več virov v novo integrirano spletno stran. Pristop hibridizacije[1] v grobem delimo v dve ločeni kategoriji [4]:

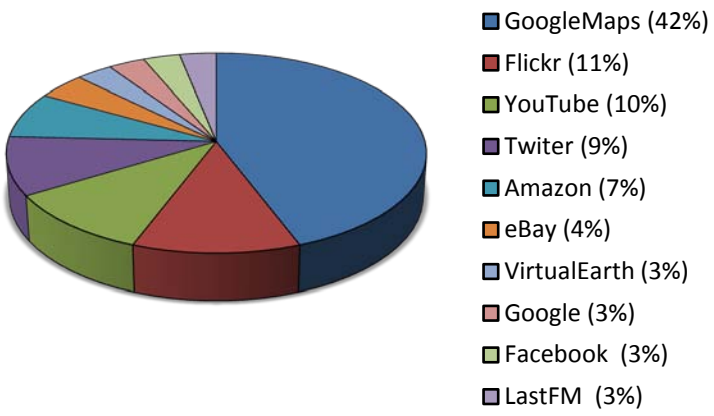
- **potrošniške sestavljanke** (*angl*. *Consumer Mashups*) in
- **podjetniške sestavljanke** (*angl*. *Enterprise Mashups*).

Potrošniške sestavljanke se običajno povezujejo z Web 2.0[2] in zahtevajo manj programerskih sposobnosti, ker temeljijo na dobro definiranih in izpostavljenih programskih vmesnikih (*angl*. *Application Programming Interface* – API) [5] (Slika 2.1) in podajalnikih (*angl*. *feeds*). Rezultat potrošniških sestavljank je v večini primerov generiran s strani enega udeleženca v sestavljanki. Primer generatorja končnega rezultata je Google Maps API, kjer se določene informacije prikažejo na zemljevidu. V tem primeru je Google Maps nosilec končnega rezultata. Potrošniške sestavljanke so aplikacije, ki v brskalniku združujejo podatke iz več različnih javno dostopnih virov.

Primer potrošniške sestavljanke je tudi HousingMaps, ki omogoča vizualno iskanje za najem, prodajo ali nakup stanovanj in sob glede na seznam v Craigslist[3]. Sestavljanka

[1] Križanje, zlasti rastlin: s hibridizacijo so vzgojili odpornejše rastline.
[2] Poslovna revolucija v računalniški industriji, povzročena s prehodom k internetu kot platformi, in poskus razumevanja pravil za uspeh na tej novi platformi.
[3] Craigslist – mreža skupnosti, ki omogoča brezplačno spletno oddajanje oglasov.

HousingMaps prikazuje rezultate iskanja v Google Maps. V tem primeru se sestavljanje javno dostopnih podatkov izvede na strani brskalnika.



Slika 2.1 Najbolj pogosto uporabljeni vmesniki API [5]

V nasprotju s potrošniškimi sestavljankami imajo podjetniške sestavljanke bolj kompleksno ozadje. Podatke kombinirajo iz notranjih in zunanjih javno dostopnih virov in objavijo rezultat v poslovnih portalih, lastnih spletnih aplikacijah ali kot spletno storitev. Podjetniške sestavljanke morajo zagotoviti varnost, upravljanje, spremljanje in razpoložljivost.

Dober primer podjetniških sestavljank je razvita rešitev za Obrambno obveščevalno agencijo, ki spada pod Ameriško ministrstvo za obrambo. Rešitev je razvita na platformi Presto in predstavlja *navidezno operativno središče*, ki združuje več obveščevalnih virov za analizo podatkov v realnem času. Za uporabo rešitve uporabnik najprej izbere področje odgovornosti, ki omeji obveščanje na specifično področje. Nato uporabnik izbere tip obveščanja, in sicer za objekte, osebje, ladje, letala ali dogodke, in dodatno filtrira obveščanje. Tako nastavljeno obveščanje se v realnem času ažurira. Navidezno operativno središče je bilo zasnovano z namenom omogočiti končnim uporabnikom hiter razvoj aplikacij, ki dajo sliko o potencialnih grožnjah [6].

Obrambna obveščevalna agencija je izbrala tehnologijo podjetniških sestavljank, ker na ta način prihranijo čas, potreben za razvoj in postavitev, brez zmanjšanja potrebne stopnje varnosti, še vedno pa se dovoli dostop samo pooblaščenim osebam.

Zaradi tega različna podjetja vzpostavijo različna okolja za izdelavo sestavljank. Glede na pristop izdelave, za katerega se odloči podjetje, podjetniške sestavljanke delimo na [4]:

- Sestavljanke, ki jih izdelajo izključno IT strokovnjaki – Razvijalci uporabljajo tako zunanje kot tudi interne vire podatkov za izdelavo sestavljank in standardne tehnike pri kreiranju vizualne ovojnice. Pri tem načinu izdelave sestavljank uporabniki niso direktno vključeni v proces, vendar izkoriščajo hitro izdelavo aplikacij.
- Sestavljanke, izdelane s strani IT strokovnjakov in uporabnikov – IT strokovnjaki kreirajo množico sestavljivih (*angl*. *mashable*) komponent in ponudijo okolje uporabnikom, ti pa imajo možnost sami sestavljati dele. Če uporabniki potrebujejo nove komponente, potrebujejo podporo s strani IT oddelka.
- Sestavljanke, izdelane od kogarkoli – Podjetje vzpostavi okolje, ki omogoča enostavno in hitro izdelavo sestavljank. Ta pristop je najbolj zahteven za implementacijo in upravljanje, vendar je najbolj učinkovit.

Skozi čas se bodo najverjetneje ti trije pristopi poenotili v en koncept izdelave sestavljank, in sicer v platforme za sestavljanke, principe katerih opišemo v naslednjem poglavju.

2.2 Arhitektura podjetniških sestavljank

Arhitekture današnjih informacijskih tehnologij so običajno razvrščene v treh slojih [6]:

- predstavitveni,
- poslovni in
- integracijski.

Poslovni sloj skrbi za pošiljanje in sprejemanje podatkov iz/od integracijskega sloja ter izvaja potrebne poslovne funkcionalnosti in transformacije podatkov, zahtevane v predstavitvenem sloju.

Poseg v poslovnem sloju in vpeljava nove storitve sta draga in časovno potratna. Potrebno je veliko truda pri planiranju, razvoju, integraciji in testiranju. Dodatno je potrebno še

pridobiti, transformirati, normalizirati in usmerjati podatke z ustreznimi tehnologijami, kot so ESB (*angl*. *Enterprise Service Bus*), BPM (*angl*. *Business Process Management*) in BPEL (*angl*. *Business Process Execution Language*).

Ta pristop integracije ni zaželen, še posebej ne v dinamičnih okoljih. Arhitekturno rešitev tega problema predstavlja vpeljava novega, *t. i.* sestavljalskega sloja, ki se umesti med predstavitveni in poslovni sloj (Slika 2.2).

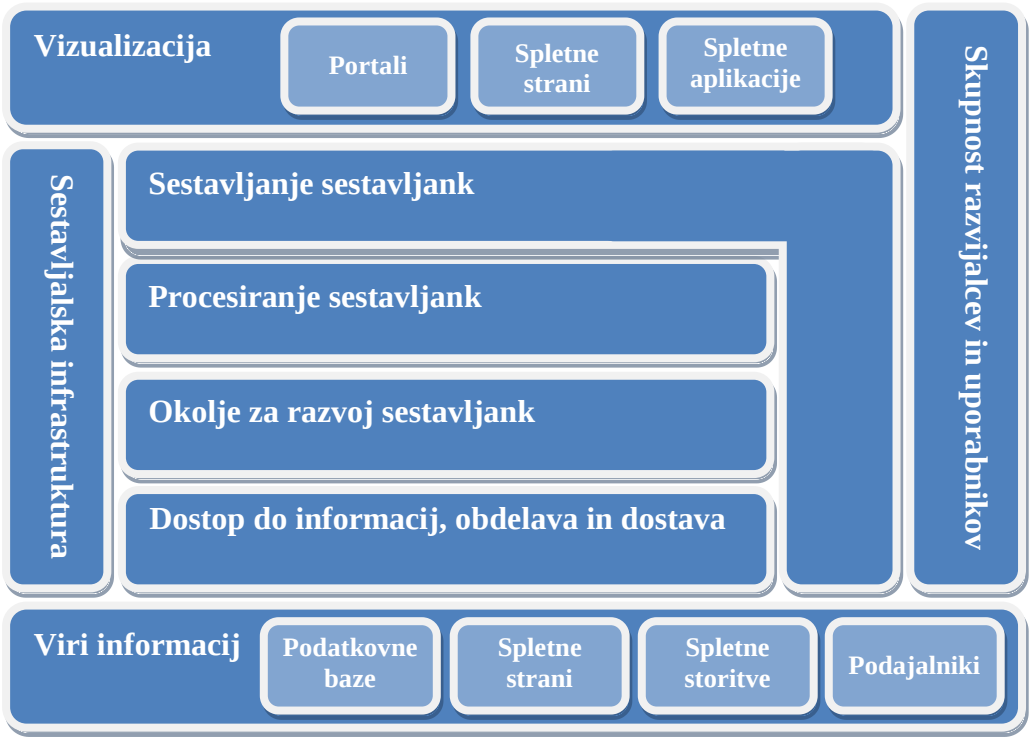
Slika 2.2 Štirislojna arhitektura informacijskih sistemov na osnovi sestavljank

Sestavljalski sloj omogoča uporabnikom izkoristiti prednosti tehnologije bogatih spletnih aplikacij (*angl*. *Rich Internet Application* – RIA) in tehnologije storitveno usmerjenih arhitektur (*angl*. *Service Oriented Architecture* – SOA) [6].

Arhitektura podjetniških sestavljank (Slika 2.3) zajema vse potrebne možnosti za podjetniško sestavljalsko okolje, kjer imajo uporabniki možnost uporabe visokonivojskih sestavljalskih komponent in razviti sestavljanke znotraj okolja [7]. Okolje podjetniških sestavljank bi moralo omogočiti izdelavo treh temeljnih entitet: sestavljalske komponente, sestavljank in sestavljalske aplikacije. Sestavljanke sestavljajo dve ali več sestavljalskih komponent. Sestavljalske aplikacije pa so sestavljene iz ene ali več sestavljank. Kot je prikazano na sliki 2.3, je arhitektura podjetniških sestavljank zgrajena iz osmih plasti [7]:

- viri sestavljank (informacije in funkcionalnosti),
- sestavljanje sestavljank,
- vizualizacija sestavljank,
- dostop do informacij, obdelava in dostava,
- okolje za razvoj sestavljank,
- procesiranje sestavljank,

- sestavljalska infrastruktura,
- skupnosti avtorjev sestavljank.



Slika 2.3 Arhitektura podjetniških sestavljank

Viri sestavljank hranijo sestavljanke, saj te nimajo svojih lastnih podatkov in funkcionalnosti. V nasprotju s potrošniškimi sestavljankami, ki črpajo informacije izključno iz spleta, preko RESTful (*angl. Representational State Transfer* – RESTful) spletnih storitev, SOAP (*angl. Simple Object Access Protocol* – SOAP) storitev, RSS (*angl. Really Simple Syndication* – RSS) protokola za objavo in distribucijo spletnih vsebin in Atom[4] podajalnikov, podjetniške sestavljanke zagotavljajo širši pogled nad viri, in sicer viri, ki temeljijo na XML, JSON[5], javanskih objektih, primitivnih tipih, POJO[6] ali obrezovanju spletne strani (*angl. Web clipping*).

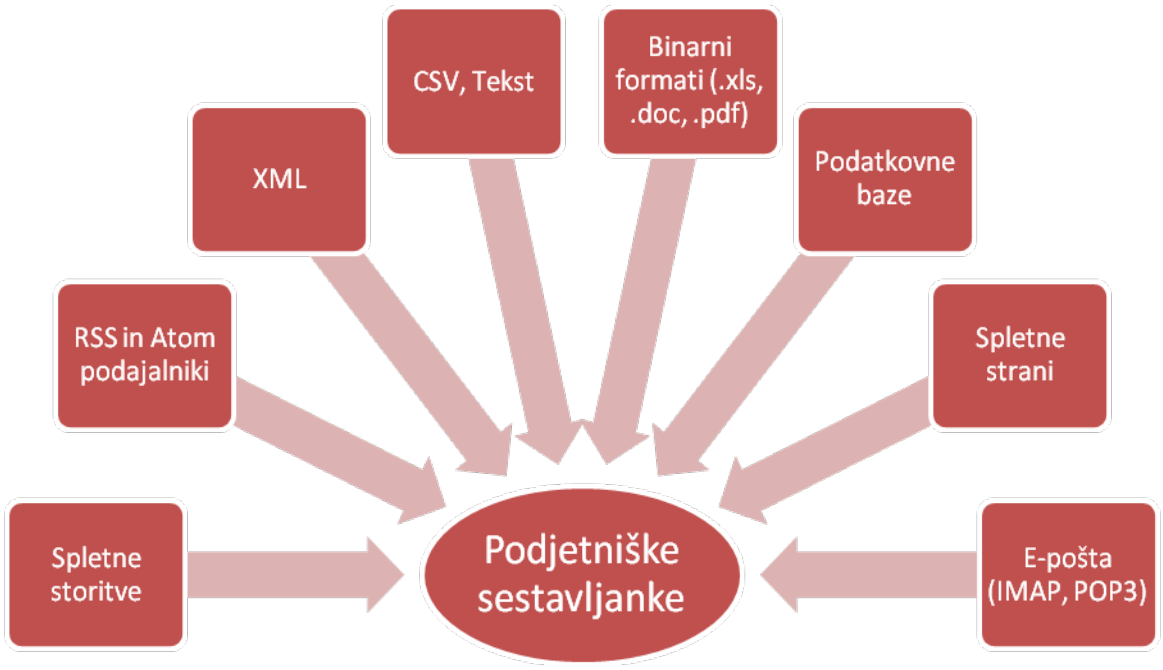
Sloj sestavljanja sestavljank je »jedro« arhitekture sestavljank. Osnovna značilnost tega sloja je zmožnost sestavljanja komponent v sestavljanki in prikaza rezultata. Preden uporabniki začnejo s sestavljanjem komponent, jih morajo najti in razumeti. Zelo pomembne karakteristike, ki jih mora podjetniško sestavljalsko okolje podpreti, je možnost

[4] Spletni vir v zapisu Atom, navadno na voljo na spletnih straneh z novicami ali blogih.
[5] JSON – JavaScript Object Notation – odprti standard, namenjen izmenjavi človeško berljivih podatkov.
[6] POJO – Plain Old Java Object – navaden javanski objekt.

uporabljanja, *razkrivanja* in *objavljanja* sestavljalskih komponent. Le-te stimulirajo večjo uporabniško udeležbo pri ustvarjanju sestavljank. Vključevanje metapodakov o komponenti olajša razumevanje komponent s strani uporabnika v smislu porekla, vrednosti in možnosti povezovanja. Zaželena je tudi možnost ocenjevanja komponent in dodajanje informacij o tem, kako dobro se komponenta povezuje z drugimi komponentami. Sloj sestavljanja sestavljank omogoča objavo sestavljanke za uporabo znotraj okolja skupnosti. To pa predstavlja priložnost za potrjevanje zastavljenih ciljev glede funkcionalnosti sestavljanke. Prikaz rezultatov se lahko prekriva s slojem vizualizacije [7].

Sloj vizualizacije ima za nalogo prikazati rezultat sestavljanke. Prostor vizualizacije so običajno uporabniški vmesniki, ki temeljijo na spletnih brskalnikih, kot so spletne strani, portali in spletne aplikacije. Za gradnjo uporabniškega vmesnika so običajno uporabljene tehnologije HTML/XHTML in JavaScript ter jezik CSS [8].

Ker imajo podjetniške sestavljanke širok razpon svojih podatkovnih virov in podatke iz virov pridobijo tako v nestrukturiranem kot v strukturiranem formatu, npr. v XML, vključno z RSS in Atom, v spletnih storitvah, pa tudi v binarnem formatu, kot sta Excel in PDF dokumenti (Slika 2.4), jih moramo nekako poenotiti, da jih lahko sestavljamo. To odgovornost ima sloj za **dostop do informacij, obdelavo in dostavo** informacij (*angl. Information Access, Augmentation and Delivery* – AAD). Ta sloj zagotovi vire za sestavljanje. Nekateri ponudniki platform se omejijo samo na sloj AAD, te kategoriziramo kot »ponudnike, ki omogočijo sestavljanja« (*angl. mashup enablers*).



Slika 2.4 Viri podjetniških sestavljank

Okolje za razvoj sestavljank ponuja uporabnikom možnost ustvarjanja in upravljanja komponent, potrebnih za nadaljnjo gradnjo sestavljalskih aplikacij. Okolje lahko ima svoje lastno okolje za razvoj sestavljalskih komponent ali pa je integrirano z že obstoječim okoljem za razvoj. Sestavljalske komponente (*angl*. *mashables*) so deli podjetniških podatkov in funkcionalnosti, potrebnih za sestavljanje. Za objavo sestavljalskih komponent so zadolženi IT razvijalci, vendar imajo tudi poslovni razvijalci možnost objavljati svoje lastne podatkovne vire (Excel, Word ali izbrane podajalnike).

Uporaba podjetniških sestavljank kot del že obstoječih informacijskih rešitev ni enostavna. Zaradi tega so potrebni ukrepi, kot je naknadno procesiranje (*angl*. *post-processing*) sestavljank pri njihovi integraciji v informacijsko rešitev. Primer vključevanja sestavljank v že obstoječe informacijske rešitve je uporaba sestavljank kot predstavitvene komponente storitveno usmerjene arhitekture in dogodkovno vodene arhitekture (*angl*. *Event-driven architecture* – EDA). **Sloj za procesiranje sestavljank** lahko podpira še delovni tok sestavljank, pa tudi medsebojno povezovanje, spremljanje poslovnih aktivnosti, poslovno inteligenco in semantično mediacijo [7].

Da bi podprli še varnost, upravljanje, administracijo, upravljanje repozitorija, podporo uporabnikom in kakovost storitve, potrebujemo dobro definirano **sestavljalsko infrastrukturo**.

Skupnost razvijalcev in uporabnikov sestavljank je okolje, kjer udeleženci ustvarjajo, uporabljajo, spreminjajo, ocenjujejo in si medsebojno delijo podjetniške sestavljanke in sestavljalske komponente. S tem je velik delež upravljanja pripisan udeležencem, kar pomeni, da je okolje skupnosti zelo pomemben del okolja sestavljank. Souporaba storitev zmanjšuje podvajanje sestavljank, skrajša se čas razvoja in povečuje učinkovitost. Podjetja se morajo pri vzpostavitvi sestavljalskega okolja potruditi v čim večji meri izkoristiti pridobitve, ki jih skupnost prinaša.

2.3 Pridobitve podjetniških sestavljank

Veliko analitikov trdi, da je vlaganje v tehnologijo prava odločitev za zmanjšanje stroškov in s tem povečanje prihodkov [9]. Izvršni direktorji informatike kot ustvarjalci vrednosti so vedno pod pritiskom povišati prihodke iz obstoječih lastnin, ne da bi dodatno investirali. Če v »lastnino« umestimo tudi »podjetniške podatke«, se zagotovo pogovarjamo o podjetniških sestavljankah kot pristopu za povezovanje podatkov na različne načine in pridobitev novega pogleda nad podatki.

Podjetniške sestavljanke na zelo prijazen način dopolnjujejo dobro poznane tehnologije, kot so BPM, poslovna inteligenca (*angl. Business Intelligence* – BI), integracija informacij (*angl. Enterprise Information Integration* – EII) in storitvena vodila (ESB). Podobnosti in razlike med podjetniškimi sestavljankami in zgoraj navedenimi tehnologijami so [10]:

- BPM – Sestavljanke ne podpirajo dolgoročnih delovnih tokov, ki vključujejo človeška opravila. Sestavljanke se izvajajo naenkrat, vendar lahko imajo velik vpliv v BPM procesu, ko se avtomatsko kličejo podatkovno vodene točke odločitve.
- BI – Podatkovna skladišča niso potrebna. Sestavljanke se direktno povezujejo in črpajo podatke od istih virov, kot jih uporabljajo podatkovna BI skladišča. Torej lahko sestavljanka ponudi visokokakovosten vhod za BI vizualizacijska orodja.

- EII – Sestavljanke ne rešujejo semantičnih problemov, ki potrebujejo zapletene ontologije. Sestavljanke se ustvarjajo in uporabljajo v realnem času. Podobno kot pri BI je tudi EII lahko visokokakovosten vhod za sestavljanke.
- ESB – Sestavljanke običajno ustvarjajo končni uporabniki in jih dostavijo drugim uporabnikom. ESB pa lahko pomaga pri izpostavljanju kompleksnih in težko dostopnih podatkovnih virov kot virov podatkov za sestavljanke.

Sestavljanke so samo dodaten sloj teh tehnologij, ki so dobro sprejete in vpeljane v podjetja [6].

2.3.1 Akterji sestavljalskega okolja

Zelo pogosto se zgodi, da poslovni uporabniki zahtevajo aplikacije, ki jih IT oddelki ne morejo zagotoviti, ker so preobremenjeni z infrastrukturo. Poslovni uporabniki potrebujejo aplikacije »tukaj in zdaj«. Ta problem se še boljše izrazi s frazo »dolgi rep« (*angl. the long tail*).

Teorija dolgega repa pravi, da je samo 20% znanih problemov, ki vplivajo na večino uporabnikov, pokrita s strani IT. Na drugi strani pa 80% potencialnih rešitev, ki služijo majhnemu številu uporabnikov, ni pokritih. Vzrok tega je investiranje v projekte, ki so zahtevani od največje skupine uporabnikov, in zapostavljanje preostalih 80% zahtev uporabnikov. Za veliko število ljudi ni možno izpolniti te zahteve, razen če ponudimo končnim uporabnikom možnost, da sami rešujejo svoje probleme. Sestavljanke so ustrezen pristop zadovoljevanja potreb končnih uporabnikov in povečanja agilnosti IT oddelka.

Za boljše razumevanje vpliva sestavljank najprej predstavimo standardni proces razvoja aplikacije.

V standardnih razvojnih okoljih se proces razvoja nove aplikacije izvaja po naslednjem vrstnem redu:

1. Oblikovanje množice zahtev.
2. Razporejanje odgovornosti med razvijalci, vključenimi v projekt.
3. Razvijanje aplikacije s tradicionalnimi jeziki in platformami.
4. Izvajanje razvojnega procesa skozi cikel zagotavljanja kakovosti.
5. Namestitev in dostava nove aplikacije.

V zgoraj opisani proces je vključeno veliko število udeležencev, in sicer od IT oddelka preko oddelka za razvoj in testiranje do ljudi, zadolženih za varnost. Aktivnosti vseh udeležencev pa morajo biti sinhronizirane.

V nasprotju s tradicionalnim razvojnim ciklom aplikacij v okolju sestavljank velik delež razvoja prevzame končni uporabnik (poslovni analitiki) pod IT nadzorom. Tabela 2.1 kaže odgovornosti posameznih akterjev [11].

Tabela 2.1 Odgovornosti v sestavljalskem okolju

	Glavne naloge IT oddelka	Glavne naloge poslovnih uporabnikov
1.	Nameščanje, konfiguracija, nastavitev varnosti in upravljanja.	Objavljanje osebnih podatkovnih virov (iz Excela, podajalnikov itd.).
2.	Objavljanje jedrnih podatkovnih virov (kot so spletne storitve).	Vizualno ustvarjanje sestavljank, vizualnih gradnikov in sestavljalskih preglednih plošč.
3.	Dodeljevanje vlog in pravic za funkcije in jedrne vire.	Deljenje sestavljank, vizualnih gradnikov in sestavljalskih preglednih plošč.
4.	Ustvarjanje kompleksnih sestavljank, vizualnih gradnikov in sestavljalskih preglednih plošč, ko je potrebno.	Ocenjevanje, uporaba in personalizacija sestavljank, vizualnih gradnikov in sestavljalskih preglednih plošč s strani drugih.

V tabeli 2.1 se vidi, da se poslovni uporabniki znotraj platforme podjetniških sestavljank (*angl*. *Enterprise Mashup Platform* – EMP) osredotočijo na sestavljanje, gradnjo in analiziranje podatkov ter se izogibajo upravljalskih predpisov in podjetniških pravil.

2.3.2 Sestavljanke in storitveno orientirane arhitekture

Storitveno orientirana arhitektura (SOA) je prožen nabor načel, uporabljenih v fazah razvoja informacijskih sistemov in integracije. Sam pristop SOA strogo definira svoje ključne komponente kot šibko sklopljene (*angl*. *loose coupling*) enote funkcionalnosti, ki nimajo vgrajenih medsebojnih klicev [12]. Tako vzpostavljena arhitektura spodbuja zelo zaželeno ponovno uporabo in rekombinacijo že obstoječih storitev. Če dodamo še dejstvo,

da spletne storitve temeljijo na odprtih standardih, kot so XML, WSDL[7], UDDI[8] in SOAP, sklepamo, da se storitve lahko uporabljajo preko neodvisnih razvojnih platform. V sklopu SOA se izpostavljanje spletnih storitev izvaja na infrastrukturi ESB, pri čemer ta poskrbi za zmanjšanje kompleksnosti uporabe storitev s strani uporabnikov. Izpostavljanje obstoječih podjetniških virov (kot so podatkovne baze) kot spletnih storitev je bila samo napoved za prihajajočo tehnologijo podjetniških sestavljank.

Sestavljanke si ne lastijo podatkov, vendar do njih dostopajo. Eden od načinov pridobivanja podatkov v podjetniških okoljih je preko spletnih storitev, kar jasno kaže na povezavo med sestavljankami in storitveno orientiranimi arhitekturami kot najbolj pogostimi dobavitelji storitev za sestavljanje. Končno je vprašanje na temo »prevoznega sredstva za dostavo podatkov znotraj SOA arhitekture« dobilo odgovor: sestavljanke.

Sestavljanke kot razširitev SOA odpirajo velik potencial za podatkovne vire, ki jih izpostavlja SOA, in približajo SOA poslovnim uporabnikom. Podobno kot BPM poskuša izvleči poslovno vrednost iz SOA, imajo enako nalogo poslovne sestavljanke. Podjetniške sestavljanke torej zbudijo SOA in jo pripeljejo na uporabniško namizje.

Učinek skupnega delovanja podjetniških sestavljank in storitveno orientirane arhitekture je neizmerno velik [6]:

- Sestavljanke lahko pomagajo pri kreiranju navideznih storitev iz virov, ki še niso bili kreirani s strani SOA. Napori za kreiranje takšnih storitev s strani SOA lahko trajajo dolgo, v nasprotju s hitrim kreiranjem standardiziranih storitev s pomočjo sestavljank, ki ponudijo uporabnikom možnost začeti prej.
- Sestavljanke omogočajo uporabnikom izpostavljanje prave granularnosti storitev. IT oddelku ni potrebno skrbeti, ali storitve ponujajo podatke, ki so preveč specifični, preveč splošni ali zastareli.
- Sestavljanke omogočajo, da si uporabniki medsebojno delijo izpeljane storitve kot del storitvenega omrežja brez podpore IT oddelka.

[7] WSDL – Web Service Definition Language – je jezik na osnovi XML-a, ki zagotavlja model za opisovanje spletnih storitev.
[8] UDDI – Universal Description Discovery and Integration – je platformno neodvisni register na osnovi XML-a, namenjen podjetjem za objavo, iskanje in uporabo spletnih storitev.

- Sestavljanke ponujajo končnim uporabnikom možnost vizualizacije SOA na grafih, grafikonih, preglednicah in zemljevidih. Končno imajo portali možnost vizualizirati storitev na način, ki si ga uporabnik želi. Uporabniki na svoj način rešujejo lastne, nenehno spreminjajoče se potrebe.
- Sestavljanke uporabnikom omogočajo, da se povežejo s podatkovnimi viri zunaj podjetja. Napori SOA so v veliki meri usmerjeni v notranjost, vendar si uporabniki pogosto želijo vključiti tudi zunanje podatke. Sestavljanke ne skrbijo za poreklo podatkov in v dobro zastavljenem sestavljalskem okolju je dejansko poreklo podatkov nepomembno.

Trg podjetniških sestavljank je bil v letu 2008 vreden približno 161 milijonov dolarjev in napovedana je rast na 1,74 milijarde dolarjev do leta 2013. Posebej je pomembno poudariti, da bo imel trg podjetniških sestavljank korist od vse večje razširjenosti SOA. Napovedano je, da bo vrednost SOA na svetovnem trgu z 1,4 milijarde dolarjev v letu 2008 narasla na 2,77 milijarde dolarjev do leta 2014 [13].

2.4 Platforme za sestavljanje

Na področju podjetniških sestavljank se je v zadnjem času pojavilo več platform za sestavljanje. Platforme vključujejo tri elemente, in sicer programske komponente, ki uporabnikom pomagajo prikazovati različne vrste podatkov, orodja, ki omogočijo razvijalcem, da ustvarjajo sestavljanke za druge, da jih uporabljajo, in infrastrukturo za upravljanje, zagotavljanje varnosti in vzdrževanje novih transformacij podatkov.

Veliko število platform zagotovijo orodja, ki pomagajo razvijalcem pri kreiranju komponent, potrebnih za sestavljanje, ali sestavljank, ki se kasneje lahko prilagodijo potrebam poslovnih uporabnikov.

Trenutno so na tržišču med najbolj aktualnimi platformami *JackBe Presto 2.7, IBM Mash-up Center, Convertigo* in *Kapow Web Data Server* [14].

Platforma **JackBe Presto 2.7** je razdeljena na tri produkte, in sicer: *Enterpirse Mash-up Server*, *Mash-up Composer* in *Mash-up Connectors*. Enterprise Mash-up Server zagotavlja platformo za virtualizacijo storitev, varnost in okolje za sodelovanje, ki transformira podatke v formatu, ki se lahko vključi v sestavljanke. Platforma lahko abstrahira več različnih podatkovnih virov, kot so: podatkovne baze, podajalniki tipa RSS in Atom, REST storitve, storitve, opisane v WSDL dokumentu, in Excelove preglednice.

Produkt Mash-up Composer je razdeljen na tri komponente. Komponenta *Presto Wires* je namenjena poslovnim analitikom za vizualno gradnjo sestavljank. Komponenta *Presto Mash-up Studio* je zgrajena na osnovi Eclipse IDE in je namenjena grajenju, testiranju, iskanju napak in postavitvi sestavljank s strani Java razvijalcev. Tretja komponenta, Presto Mashlets, omogoča grafično prikazovanje podatkov s pomočjo vizualnih gradnikov (*angl. mashlets*), ki se lahko vgradijo v portale, spletne dnevnike ali spletne aplikacije. Zgrajene sestavljanke temeljijo na standardnem jeziku za razvoj sestavljank, EMML, ki ga podrobno opišemo v tretjem poglavju.

Platforma Presto podpira sestavljalske priključke (angl. *Mash-up Connectors)*, ki povezujejo Presto z drugimi programskimi produkti, vključno s portali, SOA storitvenimi vodili, HP SOA Systinet, Oracle Fusion ali Excelovimi dokumenti. Presto tudi omogoča povezovanje s programskimi vmesniki (Connect API) za JavaScript, Java in C# [15].

Vse komponente platforme Presto so dostopne v oblaku.

Platforma **IBM Mash-up Center** je največji konkurent platforme Presto. IBM Mash-up Center se izvaja na aplikacijskem strežniku WebSphere. Vključuje orodje za izdelavo vizualnih gradnikov in komponento *Spaces* za gradnjo in delitev sestavljank med poslovnimi uporabniki. Podatke lahko platforma pridobi iz sistemov za upravljanje vsebin (*angl. Content Management System*), kot so SharePoint, WebSphere MQ, Filenet, IBM Content Manager RESTful storitve in WS-Security spletne storitve. Vtičnik za gradnjo vizualnih gradnikov (*angl. Widget Generation plug-in*) omogoča enostavno kreiranje vizualnih gradnikov brez pisanja kode.

Komponenta *Widget Editor* je namenjena razvijalcem za kreiranje novih vizualnih gradnikov iz obstoječih HTML, Javascript, XML ali CSS datotek. Komponenta ponuja nastavitev različnih politik dostopa. IBM Mash-up Center poskrbi za vizualni pregled nad

priljubljenimi vsebinami, dejavnostmi uporabnikov in odvisnostmi med podatkovnimi podajalniki in sestavljankami.

V sklopu produkta IBM Mash-up Center je še komponenta *Mash-up Accelerator*, ki prilagodi produkt Mash-up Centra za integracijo v portal IBM WebSphere Portal. Do programskih produktov IBM Mash-up Centra je možno dostopati tudi v oblaku [16].

Platforma **Convertigo** je namenjena sistemom za upravljanje odnosov s strankami in elektronskemu trgovanju (*angl. e-commerce*), vendar jo podjetja lahko uporabljajo kot vmesnik, ki prikazuje podatke iz različnih lastnih sistemov. Razdeljena je na komponento *C-EMS Vusual studio*, ki temelji na integriranem razvojnem okolju Eclipse 3.1, in strežnik *C-EMS*. Komponenta C-EMS Visual Studio podpira spletne storitve tipa REST, SOAP in Flow RSS 0 ter nestrukturirane spletne strani, kot so HTML, JSP in AJAX, ter SQL podatkovnih baz. Predstavitveni sloj omogoča pretvorbo podatkov in njihovo prikazovanje na prenosnih napravah, kot so smartphone, sub-notesniki in Tablet PC-ji. Projekt, zgrajen s produktom C-EMS Visual Studio, je možno naložiti na strežnik C-EMS ali objaviti v oblaku [17].

Platforma **Kapow Web Data Server** ponuja možnost dostopa do vseh spletnih podatkovnih tipov z zmožnostjo transformirati podatke. Zgrajena je iz komponent, v katerih je proces transformiranja in pridobivanja podatkov opisan s pomočjo *robota*. *Design Studio Client IDE* je vizualno orodje, ki omogoča poslovnim analitikom in razvijalcem graditi, testirati in nameščati robote. Komponenta *Web Based Management* je namenjena za upravljanje, spremljanje in brskanje podatkov. Izvajanje robotov je na strežniku *RoboServer*. Platforma Kapow se osredotoči na hiter dostop do podatkov iz različnih spletnih virov in samodejno dostavo podatkov o poslovanju [18].

2.5 Varnostni izzivi

Vsaka nova tehnologija, tako tudi tehnologija podjetniških sestavljank, prinaša veliko varnostnih izzivov, ki jih morajo podjetja upoštevati in se nanje pripraviti. Samo dejstvo,

da podjetniške sestavljanke obdelujejo podatke dveh ali več različnih virov, nakazuje potencialno nevarnost. Z vsakim novim virom pa tveganje narašča.

Dejstvo, da se podjetniške sestavljanke izvajajo na strani strežnika, pove, da moramo za varnost in upravljanje poskrbeti sami. Sestavljanke moramo vključiti znotraj obstoječe varnosti našega podjetniškega okolja. Da bi zagotovili čim večjo varnost, se moramo držati naslednjih smernic [19]:

- Pravice in razmnoževanje – Sestavljanke morajo znati upravljati avtentikacijo uporabnika znotraj okolja. Sestavljalska okolja morajo tudi ustvarjalcu sestavljanke omogočiti dodeljevati pravice do sestavljanke oz. podpreti avtorizacijo do več različnih podatkovnih virov.
- Standardiziracija in agilnost – Sestavljanke morajo dostaviti uporabniške poverilnice v formatu, ki ga viri storitev zahtevajo. Od vseh tipov storitev imajo samo JDBC/ODBC skladne podatkovne baze in WSDL (preko WS-SecurityPolicy) standardiziran format.
- Prenosljivost in razpoložljivost – Sestavljanke in vizualni gradniki (*angl. widgets*) morajo zagotoviti, da so razpoložljivi za širok nabor uporabnikov oz. da lahko več uporabnikov dostopa do iste sestavljanke ali vizualnega gradnika. Vizualni gradniki sestavljank morajo biti neodvisni od tega, kje so vgrajeni, na Web 1.0 ali Web 2.0 vmesnik, in morajo ohranjati prenosljivo varnost in upravljanje.

Ker se podjetniške sestavljanke povezujejo z več različnimi viri, je potrebno zagotoviti [20]:

- avtentikacijo za več različnih virov z različnimi prijavnimi podatki,
- avtorizacijo za več različnih virov z različnimi prijavnimi podatki,
- točkovno povezovanje z varnimi mehanizmi, kot je SSL,
- zagotavljanje iste značilnosti sestavljank pri izpostavljanju v oblaku.

2.6 Prihodnost sestavljank

Prihodnost sestavljank in rast trga podjetniških sestavljank bosta odvisni od stopnje znanja trga za tehnologijo podjetniških sestavljank. Stopnja zavesti trga za to tehnologijo se lahko dvigne preko oglaševanja, organiziranja sejmov in seminarjev. Trenutno se na trgu sestavljank veliko poslovnih uporabnikov še ne zaveda, da le-te povečujejo donosnost naložbe [21].

Za svetlo prihodnost podjetniških sestavljank je potrebno standardizirati njihov razvoj z enim samim jezikom, kot je npr. jezik EMML. Na ta način zagotovimo večjo berljivost in uporabnost informacij ter spodbujamo interoperabilnost in prenosljivost sestavljank. Tako standardizirane sestavljanke je možno izpostaviti v oblaku in zagotoviti platformo »na zahtevo«, kajti s tem se bo povečala dostopnost in uporaba sestavljank.

Večja dostopnost do javnih in zasebnih podatkov v strukturiranem formatu preko programskih vmesnikov še kako spodbuja uporabo sestavljank. Podjetja morajo torej odpreti svoje podatke navzven.

Ker podjetniške sestavljanke obdelujejo važne podatke znotraj podjetja, bo njihova prihodnost odvisna od dobro zagotovljene varnosti podjetniških sestavljank, kot je opisano v poglavju 2.5.

Poglavje 3

JEZIK EMML

Podjetniške sestavljanke temeljijo na standardih, kljub temu da njihov razvoj do nedavnega ni bil standardiziran. Pomembna vloga pri standardizaciji pripada jeziku EMML (*angl. Enterprise Mashup Markup Language – EMML*) kot odprtemu označevalnemu jeziku, ki omogoča univerzalno berljivost podatkov.

EMML kot odprti format ponuja prenosljivost in interoperabilnost sestavljank, na osnovi česar se pričakuje pospešitev sprejetja podjetniških sestavljank in odstranitev podjetniškega zaklepanja navznoter (*angl. vendor lock-in*). Še dodatno EMML poveča zrelost sestavljank na ta način, da odpira koherentno tržišče, kjer se bo povečalo zaupanje v podatke in rezultate sestavljank. Jezik EMML zmanjša potencialna tveganja pri uporabi podjetniških sestavljank. Dejstvo, da imamo opravka s predvidljivim EMML formatom, odpre vrata enostavni in hitri izdelavi sestavljank, saj spodbuja inovativnost poslovnih uporabnikov.

V nadaljevanju je predstavljen EMML in njegova arhitektura, podane so ključne značilnosti jezika ter poudarjene priložnosti in ovire za uporaba jezika EMML.

3.1 Zgodovina

Korenine jezika EMML segajo v leto 2006. Z namenom omogočiti uporabniško orientiran in uporabniško omogočen način kreiranja sestavljank sta Raj Krishnamurthy in Deepak Alur v podjetju JackBe začela razvijati vmesno programsko opremo, EMP (*angl. Enterprise Mashup Platform*), vključno z jezikom EMML. Kasneje je razvoj jezika EMML, skupaj z implementacijo, prevzela zveza OMA (*angl. Open Mashup Alliance*), ki je bila ustanovljena 23. septembra 2009 z namenom povečati uspešnost uporabe in sprejetja tehnologije podjetniških sestavljank in jezika EMML [22].

Člani, ki sodelujejo v zvezi OMA in podpirajo razvoj jezika EMML, so: *Adobe*, *Bank of America*, *HP*, *Intel*, *JackBe*, *expressFlow*, *Xignite*, *Capgemini*, *Convertigo*, *Delta Resources*, *DreamFace*, *Hinchcliffe & company*, *Kapow technologies*, *MashableLogic*, *Synteractive*, *Enterprise Mashup Summit*, *ProgrammableWeb* in *Zinnia Systems*.

Zveza OMA razvija jezik EMML pod licenco *Creative Commons Attribution – No Derivative Works 3.0 United States License*, kar omogoča širiti implementacijo jezika, in sicer reproducirati, distribuirati in posredovati pod pogoji, ki so navedeni s strani avtorja oz. licencodajalca, pri čemer ni dovoljeno izvesti predelav jezika EMML, kot so spreminjanje, preoblikovanje ali nadgradnja. Dokumentacija jezika EMML in dokumentacija v zvezi z njegovo implementacijo je pod licenco Creative Commons Attribution – Share Alike 3.0 United States License, ki omogoča deljenje in prilagajanje pod pogoji, ki so navedeni s strani licencodajalca, in sicer deljenje predelane dokumentacije pod enakimi ali podobnimi licenci [23].

Zveza OMA za specifikacijo jezika EMML še dodatno zagotavlja shemo, okolje za izvajanje EMML skripte, vzorčne sestavljalske EMML skripte in tehnično dokumentacijo.

3.2 Podporna okolja

Razvoj jezika EMML je pod nadzorom zveze OMA, ki povezuje člane, ki hočejo sodelovati pri razvoju. Kljub dokaj velikemu številu članov samo dva od njih ponujata izdelavo sestavljank na osnovi jezika EMML. To sta *JackBe* s platformo *Presto* in *expressFlow* z orodjem *expressFlow Designer*. Pri ustanovitvi zveze OMA je podjetje JackBe svojo lastno implementacijo (*angl. reference runtime implementation*) jezika kot izvajalno okolje za sestavljalski skripti darovalo zvezi.

Izvajalno referečno okolje, ki predstavlja »*war*« spletno aplikacijo, je prosto dostopno na spletni strani zveze OMA. Preden se postavijo EMML sestvljalske skripte za izvajanje, se war datoteka postavi na katerikoli z Java EE skladen aplikacijski strežnik. Tako nameščene sestavljalske skripte so dostopne oz. izpostavljene kot REST spletna storitev. Več o jeziku EMML navedemo v naslednjih poglavjih [24].

ExpressFlow Designer je vizualno orodje za kreiranje podjetniških sestavljank, ki temelji na jeziku EMML. Podpira samo vizualno generiranje sestavljank, predstavitveni nivo za prikaz podatkov ni podprt in ne omogoča razvijalcem ročnega pisanja sestavljalskih skript. Vse sestavljanke, generirane znotraj orodja, so skladne z izvajalnim referenčnim okoljem. Ker je ExpressFlow Designer prototipna aplikacija, ki uporablja jezik EMML za gradnjo podjetniške sestavljanke, je ni priporočljivo uporabljati v komercialne namene.

Platforma Presto, ki je v lasti podjetja JackBe, je najbolj resna platforma za razvoj podjetniških sestavljank, ki temeljijo na jeziku EMML. Opisana je v poglavju 2.4.

3.3 Konkurenti jezika EMML

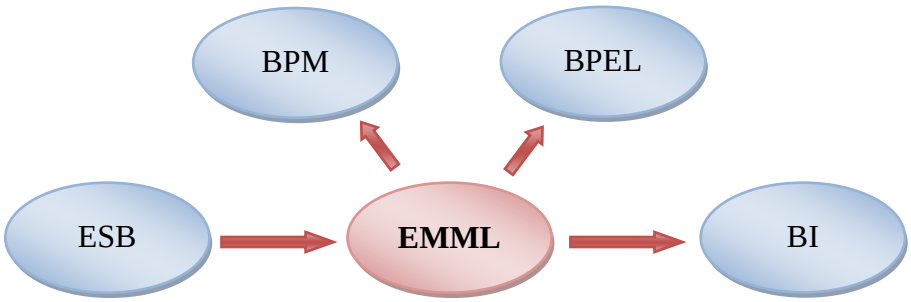
Osnovna definicija podjetniških sestavljank, zgrajenih z jezikom EMML, pravi, da so to aplikacije, ki združujejo več podatkovnih virov. Tako zastavljena definicija navede k razmišljanju o drugih jezikih, podobnih EMML. V ta kontekst spadajo tehnologije za

razvoj aplikacij in integracijskih tehnologij, kot so storitvena vodila, upravljanje poslovnih procesov, poslovna inteligenca, portali, jezik BPEL in programski jeziki, kot je Java.

Storitvena vodila v primerjavi z jezikom EMML in podjetniškimi sestavljankami rešujejo problem »povezovanja dveh aplikacij« in so zelo robustna pri dolgih transakcijah. Jezik EMML in podjetniške sestavljanke rešujejo problem »povezave aplikacija – uporabnik« in se izvajajo v realnem času, jezik EMML pa lahko izkorišča podatkovne vire, izpostavljene s strani storitvenih vodil [25].

Če primerjamo jezika BPEL in EMML, je BPEL procesno orientiran jezik v primerjavi z EMML, ki je podatkovno orientiran. Oba jezika povezujeta različne enote, vendar ne v isti domeni, in se razlikujeta v času, potrebnem za izvajanje. BPEL se lahko izvaja eno sekundo, eno uro ali več let, EMML pa se izvaja v realnem času [25].

Grafično je povezava med EMML in sorodnimi jeziki prikazana na sliki 3.1.



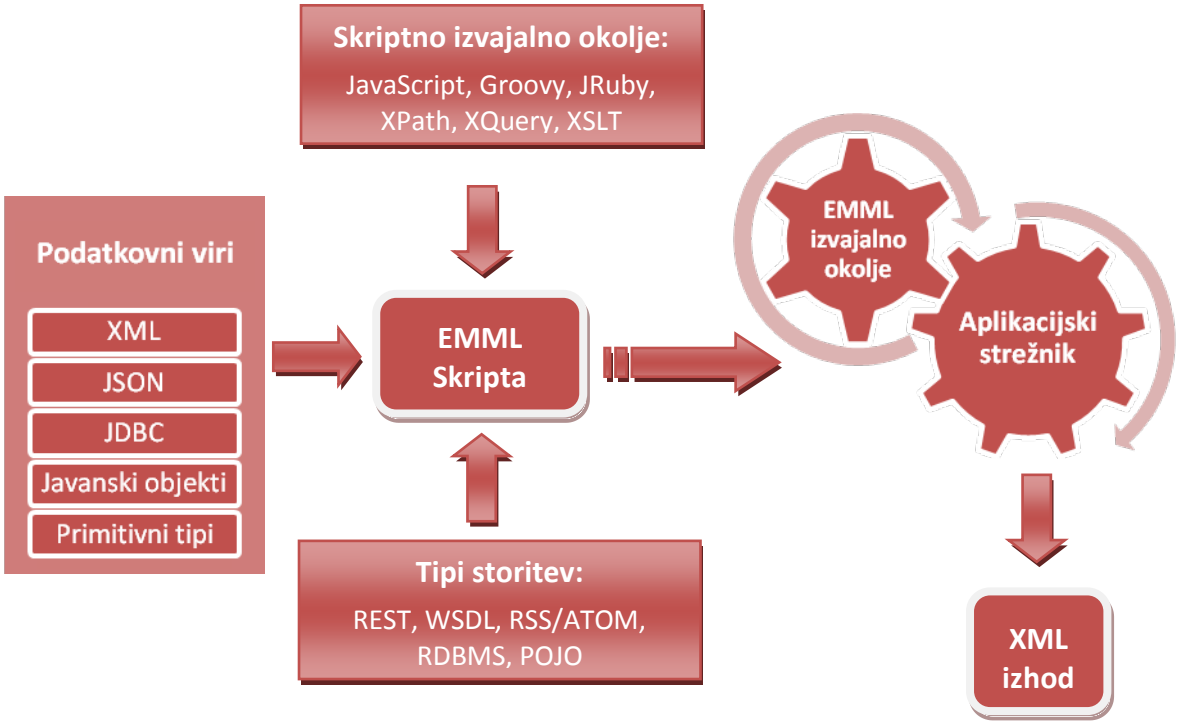
Slika 3.1 Povezava med EMML in konkurenčnimi jeziki

Iz slike 3.1 vidimo, da je lahko rezultat podjetniških sestavljank na osnovi jezika EMML vhod tehnologijam upravljanja poslovnih procesov, jezika BPEL in skladišča poslovne inteligence. Jezik EMML pa lahko izkorišča vire podatkov iz storitvenih vodil.

3.4 Arhitektura EMML

Jezik EMML predstavljajo kot XML označevalen, deklarativen in domensko specifičen jazik za kreiranje podjetniških sestavljank. Ponuja bogat nabor visokonivojskega besednjaka za pridobivanje in sestavljanje različnih podatkov na zelo fleksibilen način.

EMML ponuja enotno sintakso za proženje različnih heterogenih tipov storitev: REST, WSDL, RSS/ATOM, RDBMS, POJO in podatkov, pridobljenih z obrezovanjem iz HTML strani. Sestavljanje podatkov se lahko izvaja nad različnimi podatkovnimi formati, kot so: XML, JSON, JDBC, javanski objekti in primitivni tipi. Kompleksna programska logika je podprta z vgrajenim skriptnim izvajalnim okoljem, ki podpira naslednje jezike: JavaScript, Groovy, JRuby, XPath, XQuery, XSLT in Java. Tako zgrajena EMML skripta se lahko kasneje postavi na katerikoli z Java EE skladen aplikacijski strežnik, na katerega je bilo predhodno nameščeno EMML izvajalno okolje. Tako nameščena sestavljanka je dostopna oz. izpostavljena kot REST spletna storitev, ki vrne rezultat v obliki XML dokumenta. Na sliki 3.2 je prikazana EMML arhitektura.



Slika 3.2 EMML arhitektura

3.5 Jedro jezika

Vse jedrne značilnosti jezika EMML lahko grupiramo v več skupin, in sicer skupine ukazov za:

- proženje storitev oz. pridobivanje podatkov,
- sestavljanje podatkov,
- obogatitev podatkov,
- iskanje napak,
- uporaba makrojev,
- nadzor nad logiko in
- izvajanje transakcij nad podatkovnimi bazami.

Najbolj pomembni ukazi so v nadaljevanju podrobno opisani.

Za boljše razumevanje sintakse jezika EMML so oznake grupirane v več skupin [24]:

Deklaracijska skupina: skupina oznak za definiranje parametrov in metapodatkov, ki se uporabljajo znotraj sestavljank.

- <emml-meta>
- <input>
- <output>
- <user-meta>

Makroji: skupina oznak za definiranje makrojev in njihovo uporabo oz vključevanje.

- <include>
- <macro>

Ukazna skupina: oznake, ki predstavljajo akcije, ki se sprožijo znotraj sestavljank.

- <annotate>
- <appendresult>
- <assign>
- <assert>
- <constructor>
- <directinvoke>
- <display>
- <filter>
- <for>
- <foreach>
- <group>
- <invoke>
- <if>
- <join>
- <merge>
- <parallel>
- <script>
- <select>
- <sort>
- <sql>
- <sqlBeginTransactio
- <sqlCommit>
- <sqlRollback>
- <sqlUpdate>
- <template>
- <while>
- <xslt>

Spremenljivke: skupina dodatnih oznak za definiranje spremenljivk in podatkovnih virov, ki se uporabljajo znotraj sestavljank.

- <datasource>
- <variable>
- <variables>

Med ukaze za proženje storitev oz. pridobivanje podatkov spadajo ukazi invoke, direct invoke, input in output, ki so opisani v poglavju 3.5.2 o deklaraciji spremenljivk in 3.5.3 o proženju storitvenih komponent. Ukazi za sestavljanje podatkov (filter, join, merge, sort, select in group) se opisani v poglavju 3.5.5 o kombiniranje rezultatov storitvenih komponent in 3.5.6 o preoblikovanje vmesnih rezultatov, ukazi za nadzor nad logiko (if, while, for, foreach, parallel) pa v poglavju 3.5.8 o nadzor procesnega toka sestavljank. V poglavju 3.5.4 o povpraševanje po bazah in 3.5.8 o nadzor procesnega toka sestavljank so opisani še ukazi za povpraševanje po bazah in izvajanju transakcij (SQL, SQL update, SQL begin transaction, SQL commit, SQL rollback).

3.5.1 Sestavljalska skripta

Sestavljalsko EMML skripto predstavlja XML datoteka, v kateri se definirajo storitve in operacije, ki se uporabljajo s strani sestavljank, in ukrepi, ki se izvedejo nad dobljenimi rezultati storitev, da se zgradi vmesni ali končni rezultat sestavljank.

Jezik EMML uporablja različne imenske prostore za nove EMML sheme. Trenutni imenski prostor za EMML in imenski prostor za makroje sta naslednja:

- EMML sestavljalski imenski prostor: http://www.openemml.org/2009-04-15/EMMLSchema in
- EMML makro imenski prostor: http://www.openemml.org/2009-04-15/EMMLMacro.

Naslednja shema prikazuje XML shemo elementa <mashup> [26]:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:macro="http://www.openemml.org/2009-04-15/EMMLMacro"
    xmlns="http://www.openemml.org/2009-04-15/EMMLSchema"
    targetNamespace="http://www.openemml.org/2009-04-15/EMMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified"
    version="1.0">
    <xs:element name="mashup">
        <xs:annotation>
            <xs:documentation>Sestavljalska skripta, ki lahko kombinira, filtrira ali na
    drug način obdela rezultate od enega ali več informacijskih virov.
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:choice>
                <xs:group ref="OperationGroup"/>
                <xs:choice maxOccurs="unbounded">
                    <xs:group ref="Declarations"/>
                    <xs:group ref="VariablesGroup"/>
                    <xs:group ref="Statements"/>
                    <xs:group ref="Macroincludes"/>
                    <xs:group ref="MacroRefs"/>
                </xs:choice>
            </xs:choice>
            <xs:attribute name="name" type="xs:string" />
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Shema 3.1 XML shema elementa <mashup>

EMML skripta kot korenski element ima oznako <mashup> z atributom *name* z logičnim imenom sestavljanke. Imena sestavljank morajo biti unikatna, lahko vsebujejo ASCII črke, številke, podčrtaje (_) ali črtice (-) in se morajo začeti s črko. Kot podelement lahko oznaka **<mashup>** vsebuje (logično zapisano) [24]:

((operation) | (deklaracijska skupina | (skupina spremenljivke) | ukazna skupina | skupina makroji | ((macro:ime-po-meri | katerikoli element ni zapisan v EMML imenskega prostora) *)) +)

Oznaka **<operation>** kot podelement oznake <mashup> predstavlja imenovano operacijo sestavljalske storitve (*angl. mashup service*). Kaj vse lahko vsebuje oznaka <operation>, se logično zapiše kot [24]:

(deklaracijska skupina | (skupina spremenljivke) | ukazna skupina | skupina makroji | ((macro:ime-po-meri | katerikoli element ni zapisan v EMML imenskega prostora)*)) +

V nadaljevanju je prikazana XML shema elementa <operation> [26]:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:macro="http://www.openemml.org/2009-04-15/EMMLMacro"
    xmlns="http://www.openemml.org/2009-04-15/EMMLSchema"
    targetNamespace="http://www.openemml.org/2009-04-15/EMMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified"
    version="1.0">
...
    <xs:group name="OperationGroup">
        <xs:sequence>
            <xs:element name="operation" type="operationType"/>
        </xs:sequence>
    </xs:group>

    <xs:complexType name="operationType">
        <xs:choice maxOccurs="unbounded">
            <xs:group ref="Declarations"/>
            <xs:group ref="VariablesGroup"/>
            <xs:group ref="Statements"/>
            <xs:group ref="Macroincludes"/>
            <xs:group ref="MacroRefs"/>
        </xs:choice>
        <xs:attribute name="name" type="xs:string" default="invoke"/>
    </xs:complexType>
</xs:schema>
```

Shema 3.2 XML shema elementa <operation>

3.5.2 Deklaracija spremenljivk in parametrov

Deklaracija spremenljivk in parametrov znotraj sestavljanke, operacij in makrojev se izvaja s pomočjo elementov: **<variable>**, **<input>** in **<output>**. Spremeljivke se lahko grupirajo oz. navedejo znotraj oznake **<variables>**. Sestavljanke, operacije in makroji znotraj sestavljanke imajo en izhodni parameter (<output>), ki vzdržuje rezultat, vendar število vhodnih parametrov ni omejeno. Vsak parameter in spremenljivka ima: *ime* (*name*), *podatkovni tip* (*type*), podatek in obseg, kjer sta ime in podatkovni tip podana kot atributa, ter atributa *default* in *service*.

Imena parametrov in spremenljivk morajo biti unikatna znotraj imenskega prostora, kjer so deklarirana. Morajo se začeti z ASCII črko in lahko vsebujejo številke, črtice (-) ali podčrtaje (_).

Podatkovni tipi parametrov in spremenljivk so lahko primitivni ali kompleksni. Med kompleksne spada tip *document* ali katerikoli tip, ki predstavlja imenovan tip iz specifične storitve, določene z atributom *service*. Dokumentni tipi spremenljivk in parametrov so predstavljeni z XML strukturo. Spremenljivke in parametri kot primitivni tipi so lahko tipa: string, number, data in boolean.

Privzeta vrednost spremenljivk in parametrov se lahko dodeli samo osnovnim podatkovnim tipom s pomočjo atributa *default*, vendar se lahko za parametre in spremenljivke s kompleksnim podatkovnim tipom zgradi podatek z določitvijo XML strukture in podatki znotraj oznak <variable>, <input> ali <output>.

Obseg parametrov in spremenljivk je odvisen od tega, kje so le-ti deklarirani. Parametri imajo globalni obseg. Spremenljivke, deklarirane znotraj oznak <mashup> in <operation>, imajo prav tako globalni obseg, kar pomeni, da lahko do njih dostopamo kjerkoli.

Sklicevanje na parametre in spremenljivke se izvaja z XPath izrazi. Naslednji primer prikazuje uporabo oznak <assign>, pri čemer se del spremenljivke `spremenljivkaA` skopira v novo spremenljivko `novaSpremenljivka`.

```
<assign fromexp="$spremenljivkaA" outputvariable="$doSpremenljivka" />
```

Primer kode 3.1 Določanje vrednosti spremenljivke

Shema, ki definira elemente <input> in <output>, sledi v nadaljevanju [26]:

```
<xs:element name="input">
    <xs:complexType mixed="true">
        <xs:choice>
            <xs:any minOccurs="0" maxOccurs="unbounded" namespace="##any"
processContents="lax"/>
        </xs:choice>
        <xs:attributeGroup ref="nameReq"/>
        <xs:attributeGroup ref="typeReq"/>
        <xs:attributeGroup ref="serviceOpt"/>
        <xs:attributeGroup ref="defaultOpt"/>
    </xs:complexType>
</xs:element>
<xs:element name="output">
    <xs:complexType mixed="true">
        <xs:choice>
            <xs:any minOccurs="0" namespace="##any" processContents="lax"/>
        </xs:choice>
        <xs:attributeGroup ref="nameReq"/>
        <xs:attributeGroup ref="typeReq"/>
        <xs:attributeGroup ref="serviceOpt"/>
        <xs:attributeGroup ref="defaultOpt"/>
    </xs:complexType>
</xs:element>
```

Shema 3.3 XML shema elementov <input> in <output>

V primeru kode 3.2 definiramo elementa <input> in <output>. Element <input> poimenujemo *vhod* in je tipa *string* s privzeto vrednostjo 'Janez'. Element <output> poimenujemo *izhod* in je dokumentnega tipa.

```
<input name="vhod" type="string" default="Janez" />
<output name="izhod" type="document" />
```

Primer kode 3.2 Definiranje elementov <input> in <output>

V nadeljevanju sledi shema elementa <variable> [24]:

```
<xs:element name="variable" type="variableType" />
<xs:complexType name="variableType" mixed="true">
    <xs:choice>
        <xs:any minOccurs="0" maxOccurs="unbounded" namespace="##any"
processContents="lax" />
    </xs:choice>
    <xs:attributeGroup ref="nameReq"/>
    <xs:attributeGroup ref="typeReq"/>
    <xs:attributeGroup ref="serviceOpt"/>
    <xs:attributeGroup ref="defaultOpt"/>
</xs:complexType>
<xs:attributeGroup name="nameReq">
    <xs:attribute name="name" type="xs:string" use="required" />
</xs:attributeGroup>
<xs:attributeGroup name="typeReq">
    <xs:attribute name="type" use="required" type="xs:token" />
</xs:attributeGroup>
<xs:attributeGroup name="serviceOpt">
    <xs:attribute name="service" type="xs:string" />
</xs:attributeGroup>
<xs:attributeGroup name="defaultOpt">
    <xs:attribute  name="default" type="xs:string" />
</xs:attributeGroup>
```

Shema 3.4 XML shema elementa <variable>

3.5.3 Proženje storitvenih komponent

Ena izmed najbolj pomembnih značilnosti jezika EMML je možnost proženja storitvenih komponent in pridobivanja podatkov od javno dostopnih spletnih storitev s pomočjo ukazov <invoke> in <directInvoke>. Ker ukaz <invoke> še vedno ni implementiran, opišemo samo ukaz <directInvoke>.

Spletne storitve, ki se sprožijo, morajo biti tipa REST, SOAP ali sindikacijskih storitev (*angl.* syndication services), kot so RSS in Atom.

Element <directinvoke> vsebuje naslednje atribute [24]:

- **endpoint** (obvezen) – URI do javno dostopne storitve, ki jo zbujamo
- **method** (opcijski) – bodisi GET ali POST
- **requestbody** (opcijski) – ime spremenljivke, ki vsebuje telo zahteve, ko je atribut *method* nastavljen na POST
- **header** (opcijski) – ime spremenljivke, ki vsebuje glavo, ko se storitev sproži
- **feedtype** (opcijski) – določa protokol za normalizacijo sindikacijskih vsebin; lahko ima vrednost *rss* (za RSS 2.0) ali *atom* (za Atom 1.0)

- **outputvariable** (obvezen) – ime spremenljivke, ki sprejme odgovor
- **responseheader** (opcijski) – ime spremenljivke, ki sprejme HTTP glavo od odziva
- **responsecode** (opcijski) – ime spremenljivke, ki sprejeme HTTP kodo stanja odziva
- **cookies** (opcijski) – ime spremenljivke, ki sprejeme piškot od odziva
- **bypassproxy** (opcijski) – določa, ali naj se proženje izogne strežniku, če obstaja; privzeta vrednost je *false*
- **timeout** (opcijski) – maksimalno število minut za čakanje na odgovor; privzeta vrednost je *5 minut*
- **onerror (o**pcijski) – obnašanje sestavljalske skripte, ko pride do napake; vrednost je lahko *abort* – prikinitev nadaljnjega izvajanja skripte ali continue – *izvjanje* skripte se lahko nadaljuje po nastali napaki

Spodnja shema prikazuje XML shemo elementa <directInvoke> [26]:

```
<xs:element name="directinvoke" type="directInvokeType"/>
<xs:complexType name="directInvokeType">
    <xs:attribute name="endpoint" type="xs:anyURI" use="required"/>
    <xs:attribute name="method" type="xs:string"/>
    <xs:attribute name="requestbody" type="xs:string"/>
    <xs:attributeGroup ref="headerAttr"/>
    <xs:attribute name="feedtype">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:enumeration value="rss"/>
                <xs:enumeration value="atom"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
    <xs:attributeGroup ref="outputVarReq"/>
    <xs:attribute name="responseheader" type="xs:string"/>
    <xs:attribute name="responsecode" type="xs:string"/>
    <xs:attribute name="cookies" type="xs:string"/>
    <xs:attribute name="bypassproxy" type="xs:boolean"/>
    <xs:attributeGroup ref="invokeControls"/>
    <xs:anyAttribute processContents="lax">
    </xs:anyAttribute>
</xs:complexType>
<xs:attributeGroup name="outputVarReq">
    <xs:attribute name="outputvariable" type="xs:string" use="required"/>
</xs:attributeGroup>
<xs:attributeGroup name="invokeControls">
    <xs:attribute name="timeout" type="xs:integer" default="5"/>
    <xs:attribute name="onerror" default="abort">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:enumeration value="abort"/>
                <xs:enumeration value="continue"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
</xs:attributeGroup>
<xs:attributeGroup name="headerAttr">
  <xs:attribute name="header" type="xs:string"/>
</xs:attributeGroup>
```

Shema 3.5 XML shema elementa <directInvoke>

V primeru kode 3.3 je prikazana uporaba oznak <directInvoke> z metodo POST, pri čemer se najprej v spremenljivki *post* definira telo zahteve, normalizirano v formatu *Atom*, in se nato pošlje sporočilo za objavo na forumu.

```
<constructor outputvariable="body" >
    <entry xmlns='http://www.w3.org/2005/Atom'>
        <title type='text'>Moj prvi post</title>
        <content type='xhtml'>
            <div xmlns="http://www.w3.org/1999/xhtml">
                <p>"Zdravo vsem!"</p>
                <p>"To sem jaz, EMML!"</p>
            </div>
        </content>
        <author>
            <name>EMML</name>
            <email>EMML@oma.org</email>
        </author>
    </entry>
</constructor>

<directInvoke endpoint="http://www.feri.uni-mb.si/forum/putPost"
    outputvariable="$result" method="POST" requesbody="$body" />
```

Primer kode 3.3 Uporaba elementa <directInvoke>

Obrezovanje iz HTML strani se tudi lahko izvede s pomočjo ukaza <directinvoke>, pri čemer se pridobljeni rezultat v izvajalnem okolju EMML pretvori v XHTML formo. Obrezovanje se izvede enako kot navadno proženje spletne storitve in se dobljeni rezultat kasneje obdela.

Prenos parametrov za določeno storitev je možno izvesti tako, da se uporabi katerokoli ime atributa, ki ni definirano v jeziku EMML. V naslednjem primeru sta atributa *informations* in *studentRegistrationNumber* definirana v navedenem imenskem področju in bosta prenesena do storitve:

```
<directinvoke endpoint="http://www.feri.uni-mb.si/getStudent"
    method="GET" outputVariable="$result"
    xmlns:sn="http://www.feri.uni-mb.si/"
    sn:informations="all" sn:studentRegistrationNumber="ES869375" />
```

Primer kode 3.4 Pošiljanje parametrov z <directinvoke>

3.5.4 Povpraševanje po bazah

Jezik EMML podpira tudi poizvedbe v eni ali več podatkovnih bazah. SQL poizvedbe se izvedejo s pomočjo ukaza <sql> ali kateregakoli drugega SQL stavka, ki ni poizvedba, s pomočjo ukaza <sqlUpdate>, kot so na primer stavki za ažuriranje podatkov (*angl. update*) in shranjene procedure brez povratnih odgovorov. EMML ponuja tudi možnost za

izvajanje transakcij z ukazi <sqlBeginTransaction>, <sqlCommit> in <sqlRollback> ali klic shranjenih procedur.

Ukaz <sql> se uporablja za poizvedbe po podatkovnih virih, deklariranih z elementom <datasource>. Element <sql> vsebuje naslednje atribute [24]:

- **name** (opcijski) – ime podatkovnega vira; če ime ni navedeno, potem se poizvedba pošilje do privzetega podatkovnega vira
- **query** (obvezen) – poizvedba za izvedbo; SQL sintaksa je odvisna od podatkovne baze
- **startrow** (opcijski) – indeks prve vrstice rezultata za vrnitev; privzeta vrednost je ena
- **rowcount** (opcijski) – število vrnjenih vrstic; privzeto vrne vse vrstice
- **outputvariable** (obvezen) – ime spremenljivke, ki sprejeme odgovor

Struktura vrnjenega odgovora je naslednja:

```
<records>
    <record>
        <stolpecA-ime> vrednost </stolpecA-ime>
        <stolpecB-ime> vrednost </stolpecB-ime>
...
    <record>
</records>
```

Primer kode 3.5 Struktura vrnenjenega odgovora

Element <sqlUpdate> vsebuje naslednje atribute [24]:

- **name** (opcijski) – ime podatkovnega vira; če ime ni navedeno, se izvede nad privzetim podatkovnim virom
- **statement** (obvezen) – SQL stavek za izvedbo; SQL sintaksa je odvisna od podatkovne baze
- **outputvariable** (opcijski) – ime spremenljivke, ki sprejeme odgovor

Pošiljanje parametrov za poizvedbo se izvede v obliki *`:ime-parametra`*. Preprost primer pošiljanja parametrov sledi v nadaljevanju:

```
<variable name="studentId" type="string" default="1" />
<sql query="Select * from STUDENT Where STUDENT_ID = :studentId"
    outputvariable="$result" />
```

Primer kode 3.6 Pošiljanje parametrov

Za elementa <sql> kot tudi za elementa <sqlUpdate> je najprej treba definirati podatkovni vir <datasource> s potrebnimi informacijami za podatkove baze.

Vsak definiran podatkovni vir mora biti ali definiran URL do JDBC povezave in s potrebnimi podatki za prijavo v podatkovno bazo ali pa JNDI ime za povezavo do baze podatkov. Primer kode za oba načina definiranja podatkovnega vira je prikazan v naslednjih dveh primerih:

```
<datasource
 url="jdbc:name:protocol//host:port"
 username="admin" password="admin" />
```

Primer kode 3.8 Povezovanje z URL za JDBC

```
<datasource
 jndiname="java:/comp/env/jdbc/mydatasource"
/>
```

Primer kode 3.7 Povezovanje z JNDI imenom

Element <datasource> vsebuje naslednje atribute [24]:

- **name** (opcijski) – ime podatkovnega vira; če ime ni navedeno, se izvede nad privzetim podatkovnim virom
- **url** (opcijski) – URL za JDBC povezavo do podatkovnega vira; če URL ni naveden, se mora uporabiti atribut *jndiname*
- **driverClassName** (opcijski) – ime razreda (Class) za gonilnik JDBC povezave do podatkovnega vira; uporabi se samo, če je določen atribut *url*
- **username** (opcijski) – uporabniško ime za JDBC povezave do podatkovnega vira; potreben, če je atribut *url* določen
- **password** (opcijski) – geslo za prijavo v podatkovni vir; potreben, če je atribut *url* določen
- **jndiname** (opcijski) – JNDI ime za podatkovni vir; če je izpuščen, se morajo določiti informacije za JDBC povezavo

3.5.5 Kombiniranje rezultatov storitvenih komponent

Ker je eden glavnih namenov jezika EMML kombinirati pridobljene podatke iz enega ali več podatkovnih virov, je logično podpirati tudi ukaze za njihovo kombiniranje. Ukaza, ki ju EMML ponuja za kombiniranje, sta <join> in <merge>, in sicer je <join> podoben ukazu *join* pri podatkovnih bazah in <merge> ukazu *union*.

Z elementom <join> se določa način, kako se podatki iz ene ali več spremenljivk združujejo pod definiranim pogojem oz. se upoštevajo samo vozlišča, ki zadoščajo pogoju.

Podatki iz spremenljivk morajo imeti ponavljajoče se strukture, ki se medsebojno povezujejo na določen način (npr. tuji ključ). Atributa, ki ju element <join> podpira, sta [24]:

- **outputvariable** (obvezen) – izhodna spremenljivka, ki sprejeme odgovor
- **joincondition** (obvezen) – XPath 2.0 izraz, ki določa, katere spremenljivke se združujejo in pod katerimi pogoji

Končni rezultat združitve se lahko še dodatno obdela in zgradi se končna struktura izhoda s pomočjo podelementa <select>, ki ima obvezen atribut *name* za določanje korenskega vozlišča. Znotraj elementa <select> se običajno definira XML struktura s korenskim elementom, ki ovija vsak element iz ukaza *join*.

Ukaz <merge> spaja homogene podatke iz ene ali več spremenljivk v enotno strukturo. Strukture morajo biti enake, razen imena korenskih elementov. Atributi, ki se lahko nastavijo za element <merge>, so [24]:

- **inputvariables** (obvezen) – seznam vhodnih spremenljivk, ločenih z vejico
- **outputvatiable** (obvezen) – izhodna spremenljivka, ki sprejme odgovor
- **select** (opcijski) – izraz, s katerim se določi, katera vozlišča naj se vključijo v rezultat

3.5.6 Preoblikovanje vmesnih rezultatov

Manipulacija podatkov, shranjenimi v spremenljivkah, se v sklopu jezika EMML lahko izvede s pomočjo več različnih ukazov [24]:

- **<assign>** – ovrednotenje ali kopiranje dela ali vseh podatkov iz spremenljivke
- **<filter>** – filtriranje spremenljivke
- **<group>** – grupiranje podatkov spremenljivke ali opcijsko filtriranje ali sortiranje
- **<sort>** – sortiranje spremenljivke
- **<annotate>** – dodajanje vozlišč ali podatkov spremenljivke
- **<script>** – preoblikovanje podatkov z uporabo uporabniško določene skriptne kode
- **<xslt>** – preoblikovanje podatkov z XSLT

Z uporabo elementa <assign> lahko ovrednotimo spremenljivko z literalom, kopiranjem dela druge spremenljivke ali celo spremenljivko. Naslavljanje pri uporabi elementa <assign> se izvede z XPath izrazom. Element <assign> vsebuje naslednje atribute [24]:

- **fromvariable** (opcijski) – spremenljivka za kopiranje
- **fromexpr** (opcijski) – XPath izraz, ki določa spremenljivko ali del spremenljivke in funkcije, ki se izvedejo, preden se rezultat skopira
- **literal** (opcijski) – vrednost literala za kopiranje
- **toexpr** (opcijski) – XPath izraz, ki določa vozlišče spremenljivke, kamor se skopira rezultat; nastavljen mora biti ali atribut *toexpr* ali atribut *outputvariable*
- **outputvariable** (opcijski) – spremenljivka, kamor se skopira rezultat

V elementu <assign> mora biti nastavljen atribut *fromvariable, fromexp* ali *literal.*

Element <filter> se uporablja za filtriranje vsebine spremenljivke pod določenim pogojem in kopiranje rezultata v novo spremenljivko. Pogoj filtriranja se zapiše z XPath izrazom, kjer se lahko zapiše več pogojev. Za uporabo ukaza <filter> je potrebno dobro poznavanje jezika XPath 2.0. Element <filter> definira naslednje atribute [24]:

- **inputvariable** (obvezen) – vhodna spremenljivka za filtriranje
- **filterexpr** (obvezen) – izraz, ki definira pogoj filtriranja; vsa vozlišča, ki zadostujejo pogojem, se vključijo v rezultat
- **outputvariable** (obvezen) – izhodna spremenljivka, ki sprejme rezultat filtriranja

Grupiranje ponavljajočih se vozlišč, opcijsko filtriranje in sortiranje je v jeziku EMML podprto z elementom <group>. Z ukazom <group> oz. z XPath izrazom v atributu *by* se lahko izvede selekcija ponavljajočih se skupin vozlišč in se določi vrednost, po kateri se vozlišča grupirajo. Pridobljen rezultat se shranjuje v spremenljivki, navedeni v atributu *outputvariable*. V atributu *having* se poda pogoj, pod katerim se izberejo vozlišča. Element <group> vsebuje naslednje atribute [24]:

- **by** (obvezen) – XPath izraz, s katerim se definira vrednost, po kateri se vozlišča grupirajo
- **having** (opcijski) – XPath izraz, ki poda kriterij, po katerem se izberejo vozlišča
- **outputvariable** (obvezen) – spremenljivka, ki sprejme rezultat

Gnezdenje elementov tipa <group> je tudi dovoljeno, vendar takrat atribut *outputvariable* ni potreben v gnezdenih skupinah.

Primer uporabe elementa <group> sledi v nadaljevanju:

```
<group by="$list//faculty/course" outputvariable="$result">
    <courses>
        <course>
            <name>{$courseName}</name>
            <group by="students/student" having="avarage &gt; 8.5">
                <numberOfStudents>{count(student)}</numberOfStudents>
            </group>
            <group by="students/student">
                <numberOfAllStudents>{count(student)}</numberOfStudents>
            </group>
        </course>
    </courses>
</group>
```

Primer kode 3.9 Uporaba elementa <group>

V zgornjem primeru je izvedeno grupiranje ponavljajočih se vozlišč tipa *course* in z gnezdenim grupiranjem vozlišč tipa *student*. Rezultat se shrani v spremenljivko *result* s korenskim elementom *courses*, ki vsebuje seznam elementov tipa *course* s podelementom *name*, *numberOfStudents* in *numberOfAllStudents*. Vrednost elementov numberOfStudents se določi v gnezdenem elementu <group> in s štetjem študentov s povprečjem, višjim od 8.5. Vrednost elementov numberOfAll students se tudi določi v gnezdenem elementu <group>, s štetjem vseh študentov.

Ponavadi je potrebno podatke razvrščati naraščajoče ali padajoče glede na določeno vozlišče. Razvrščanje je podprto z elementom <sort>. Ta element vsebuje naslednje atribute [24]:

- **inputvariable** (obvezen) – spremenljivka za razvrščanje
- **sortexpr** (obvezen) – XPath izraz, pot do vozlišča za razvrščanje
- **sortkeys** (obvezen) – vozlišče, po katero se razvršča
- **sortdir** (opcijski) – način razvrščanja, vrednost atributa je lahko *ascending* (naraščajoče) ali *descending* (padajoče)
- **outputvariable** (obvezen) – spremenljivka, ki sprejme rezultat

Atribut *sortkeys* ima lahko več ključev oz. elementov, po katerih se razvršča, ločenih z vejico.

Element <annotate> doda atribute ali vozlišče otrok določenemu vozlišču. Dodajanje se izvede tako, da se v telesu elementa <annotate> doda besedilo v obliki:

```
[element | attribute] name value
```

Za ime in vrednost ni nujno, da sta statično določena. Lahko sta tudi dinamična. Element <annotate> vsebuje naslednje atribute [24]:

- **variable** (obvezen) – vhodna spremenljivka
- **expr** (obvezen) – relativna pot glede na vhodne spremenljivke do vozlišča za urejanje

Dodajanje več elementov in atributov ciljnega vozlišča je prikazano v naslednjem primeru kode:

```
<annotate variable="$person" expr="/person">
    element faculty {"FERI"}
    element university {"UM"}
    attirbute id {$studentRegistrationNumber}
</annotate>
```

Primer kode 3.10 Upraba elementa <annotate>

V zgornjem primeru elementu *person*, definiranemu v spremenljivki *person*, dodamo element *faculty* z vrednostjo "FERI", element *university* z vrednostjo "UM" in atribut *id* z vrednostjo iz spremenljivke *studentRegistrationNumber*.

Preoblikovanje podatkov s pomočjo skriptne kode se izvede z elementom <script>. Ta element vsebuje naslednje atribute [24]:

- **type** (obvezen) – MIME tip uporabljenega skriptnega jezika
- **src** (opcijski) – pot do lokalne datoteke, ki vsebuje skriptno kodo
- **application** (opcijski) – pot do mape, ki vsebuje javanske razrede, uporabljene v tej skriptni kodi
- **inputvariables** (opcijski) – seznam spremenljivk
- **outputvariable** (opcijski) – spremenljivka, ki sprejme rezultat skripte

XSLT transformacija nad podatki se izvede z elementom <xslt>. Ta element vsebuje naslednje atribute [24]:

- **script** (obvezen) – ime XSLT datoteke za izvajanje

- **inputvariable** (obvezen) – spremenljivka dokumentnega tipa, ki kaže na vhodni dokument za transformacijo
- **outputvariable** (obvezen) – spremenljivka, ki sprejme rezultat transformacije

3.5.7 Gradnja rezultatov

Struktura končnega rezultata sestavljanke, vhodnih spremenljivk ali vmesnih rezultatov se lahko definira s pomočjo elementov [24]:

- **<constructor>** – gradnja dobro oblikovanega (*angl. well-formed*) dokumenta
- **<appendresult>** – dodajanje enega ali več elementov spremenljivki
- **<select>** – izbira določenih vozlišč iz vhodnega dokumenta in dodajanje v izhodne spremenljivke; uporaba znotraj elementov <mashup>, <operation>, <macro> in <join>
- **<group>** – grupiranje podatkov spremenljivke ali opcijsko filtriranje ali sortiranje

Z elementom <construct> se zgradi dobro oblikovan dokument, ovit s korenskim elementom. Oblikovan dokument se shrani v vhodni ali izhodni parameter ali spremenljivko. Strukturo elementa definiramo s statičnimi podatki ali z uporabo dinamičnih sestavljalskih izrazov za dinamično vstavljanje podatkov. Element <construct> vsebuje samo en atribut [24]:

- **outputvariable** (obvezen) – spremenljivka, ki sprejme zgrajeni rezultat

Primer uporabe elementa <construct>, pri čemer se najprej zgradi vsebina dokumenta, nato pa se podatki vnašajo v dokumentu, je prikazan v naslednjem delčku kode:

```
<mashup
    xmlns:res="http://www.feri.uni-mb.si/results">
...
<construct outputvariable="$result">
    <res:students/>
</construct>
...
<foreach variable="$student">
    items="$list//students/student">
    <appendresult outputvariable="result">
        <res:student>
            <res:name>{$student//name/string()}</res:name>
            <res:surname>{$student//surname/string()}</res:surname>
         </res:student>
    </appendresult>
</foreach>
```

Primer kode 3.11 Uporaba elementa <construct>

V zgornjem primeru je uporabljen tudi element <appendresult>, s katerim k spremenljivki `result` dodajamo vsebino, definirano znotraj elementa <appendresult>. Najpogosteje se element uporabi skupaj s ponavljajočimi se ukazi <for> in <foreach>. V primeru s pomočjo ukaza <foreach> procesiramo vse študente in z ukazom <appendresult> k elementu *student* dodamo elementa *name* in *surname*. Vsebina elementa <appendresult> je XML dokument s statičnimi podatki ali dinamičnimi sestavljalskimi izrazi. Element <appendresult> ima podobno kot element <construct> le en atribut [24]:

- **outputvariable** (obvezen) – spremenljivka, ki sprejme rezultat ukaza

Element <appendresult> dodaja ponavljajoče se vozlišče h korenskemu elementu izhodne spremenljivke, definirane v atributu *outputvariable*. Če spremenljivka ne vsebuje korenskega elementa, se vstavi element <xml> kot privzeti korenski element.

Element <select> je že omenjen v sklopu razlage elementa <group>. Element <select> se uporabi za izbiro določenih vozlišč iz ponavljajočih se vozlišč iz vhodnega dokumenta in za dodajanje v izhodne spremenljivke. Atributi, ki jih element <select> ponuja, so [24]:

- **inputvariable** (obvezen) – vhodna spremenljivka
- **selectexpr** (opcijski) – XPath izraz, ki definira pot do ponavljajočih se vozlišč iz vhodne spremenljivke; relativna pot glede na pot, nastavljeno v atributu *inputvariable*
- **outputvariable** (obvezen) – spremenljivka, ki sprejeme rezultat ukaza

Element <select> lahko vsebuje element <columns>, ki pa lahko vsebuje več elementov <column>. Element <column> ima kot vsebino XPath izraz, ki definira pot do vozlišča za vključitev v izhodno spremenljivko. Pot je relativna glede na pot, nastavljeno v atributu *selectexpr*. Primer uporabe elementa <select> sledi v nadaljevanju:

```
<select inputvariable="$source" outputvariable="$result"
    selectexpr="//students/student[@name='Janez']">
    <columns>
        <column>name<column>
        <column>surname<column>
        <column>registrationNumber<column>
        <column>university<column>
        <column>faculty<column>
    </columns>
</select>
```

Primer kode 3.12 Uporaba elementa <select>

Rezultat predhodnega primera je množica vozlišč <student>, ki se imenujejo 'Janez'. V izhodu so vključeni samo podelementi name, surname, registrationNumber, university in faculty za vsakega študenta z imenom 'Janez'.

3.5.8 Nadzor procesnega toka sestavljank

Nadzor nad izvajanjem sestavljank znotraj sestavljalskih skript je omogočen s pomočjo ukazov [24]:

- <if> – pogojno procesiranje
- <for> – procesiranje določene množice vozlišč
- <foreach> – procesiranje množice vozlišč
- <while> – izvajanje zanke, dokler velja pogoj
- <break> – prekinitev izvajanja zanke
- <parallel> – vzporedno izvajanje ukazov

Element <if> temelji na pogosto uporabljenem konstruktu if-then-else, ki procesira določeno zaporedje ukazov na podlagi enega ali več pogojev. V sklopu elementa <if> so tudi elementi <elseif> in <else>. Element <if> vsebuje en atribut [24]:

- **condition** (obvezen) – XPath 2.0 pogojni izraz; če je pogoj izpolnjen, se izvajajo neposredni ukazi otroci elementa <if> (brez ukazov <elseif> in <else>)

Element <elseif> je opcijski element z množico ukazov, ki se izvedejo, če je izpolnjen pogoj, definiran v atributu *condition*. Element <elseif> vsebuje en atribut [24]:

- **condition** (obvezen) – XPath 2.0 pogojni izraz; če je pogoj izpolnjen, se izvaja množica ukazov elementa <elseif> (brez ukazov <if> in <else>)

Element <else> je brez atributov in se njegova množica ukazov izvede, če noben pogoj ni izpolnjen.

Jezik EMML podpira oba načina iteracije (*for* in *foreach*) čez množico vozlišč.

Element <for> izvaja skupino ukazov nad številčno določenim naborom vozlišč. Atributi, ki jih element <for> podpira, so [24]:

- **variable** (obvezen) – ime števca (spremenljivka); obseg te spremenljivke je omejen na <for> zanke
- **startcountervalue** (obvezen) – število ali XPath 2.0 izraz, ki določa začetno vrednost števca
- **finalcountervalue** (obvezen) – število ali XPath 2.0 izraz, ki določa končno vrednost števca

Uporaba elementov <for>, <if> in <break> je prikazana v primeru kode 3.13:

```
<for variable="$i" startcountervalue="1" finalcountervalue="$students//student">
    <if condition="$students//student[$i]/name='Janez' and
                $students//student[$i]/surname='Novak'">
        <directinvoke endpoint="http://www.feri.uni-mb.siQ/registrateStudent"
            method="POST" outputVariable="$result"
            xmlns:sn="http://www.feri.uni-mb.si/"
            sn:studentRegistrationNumber="$students//student[$i]/regNumber " />
        <break/>
    </if>
</for>
```

Primer kode 3.13 Uporaba elementov <for>, <if> in <break>

V zgornjem primeru se zanka for izvede tolikokrat, kolikor ima študentov. Preveri se, če je ime študenta 'Janez' in priimek 'Novak', in če najdemo študenta 'Janez Novak', se študent registrira ter izvajanje prekine z elementom <break/>.

Kot podelement elementov <if>, <elseif> ali <else> se lahko uporabi element <break>, vendar le, če so v zanki, kot je prikazano v predhodnem primeru.

Element <foreach> vzporedno ali sekvenčno procesira vsako vozlišče v množico vozlišč. Pot do množice vozlišč se definira s pomočjo XPath 2.0 izraza. Element <foreach> vsebuje naslednje atribute [24]:

- **variable** (obvezen) – spremenljivka, ki vzdržuje vsako vozlišče ponavljajočega se elementa; obseg dostopa je omejen samo na zanko <foreach>
- **items** (obvezen) – XPath 2.0 izraz, ki določa pot do množice vozlišč
- **parallel** (opcijski) – določa način procesiranja; vrednost atributa je lahko:
    - *yes* : vzporedno procesiranje
    - *no* : sekvenčno procesiranje, privzeta vrednost

- **tasks** (opcijski) – določa, koliko zank se bo procesiralo, če se procesiranje izvaja vzporedno (parallel=*"yes"*) – vse zanke ali se bo procesiranje končalo po dokončanju prve zanke; vrednost atributa je lahko:
  - *invokeall* : procesiranje vseh zank
  - *invokeany* : dokončanje prve zanke prekine procesiranje
- **merge** (opcijski) – določa način združitve rezultata vzporednih zank v eno ali več spremenljivk; vrednost atributa je lahko:
  - *true* : izhodna spremenljivka, definirana v elementu <fuse>, ki vzdržuje vseh rezultatov *"po vrednost"*
  - *false* : izhodna spremenljivka, definirana v elementu <fuse>, ki vzdržuje reference za vsako vozlišče v vzporedni zanki *"po referenci"*; atribut je smiselno nastaviti, ko je *"parallel=yes"* in *tasks="invokeall"*

Element <fuse> kot podelement elementa <parallel> definira XML strukturo z določenimi vozlišči ali izračuni, ki se vključijo v rezultatu. Brez elementa <fuse> se procesirajo vse zanke, vendar brez tega, da se rezultat veže na katerokoli spremenljivko. Element <fuse> vsebuje naslednji atribut [24]:

- **outputvariable** (obvezen) – spremenljivka, ki sprejme rezultat

Uporaba elementa <foreach> je prikazana v primeru kode 3.14:

```
<foreach variable="$student" items="students/student" parallel="yes"
        tasks="invokeall" merge="true">
    <directinvoke endpoint="http://www.feri.uni-mb.si/studentStatus"
                xmlns:sn="http://www.feri.uni-mb.si/"
                sn:studentRegistrationNumber="$student//regNumber/string()"
                outputvariable="$temp"/>
    <if condition="$temp='student'">
    <fuse outputvariable="$summary">
        <student>
            <name>{$student//name}</name>
            <surname>{$student//surname}</surname>
            <regNumber>{$student//regNumber}</regNumber>
        </student>
    </fuse>
</foreach>
```

Primer kode 3.14 Uporaba elementa <foreach>

V prejšnem primeru so paralelno procesirani vsi elementi *student* in za vsakega študenta preverimo njegov status. Če študent še vedno ima status študenta, se v spremenljivki *summary* shranijo njegovi podatki. Končni rezultat je shranjen v spremenljivki *summary* s

korenskim elementom <taskresults> in podelementi <taskresult>, ki se kreirajo za vsako zanko, kjer se shranjuje element <student>, definiran v elementu <fuse>.

Za vzporedno procesiranje je poleg tega, da je potrebno določiti atribut *parallel="yes"* in zgraditi izhod elementa <foreach> z elementom <fuse>, potrebno določiti še atributa *items* in *variable*.

Procesiranje skupine ukazov v zanki pod določenim pogojem, dokler je pogoj izpolnjen ali je ukaz </break> izvršen, se izvaja s pomočjo elementa <whilie>. Element <while> vsebuje naslednji atribut [24]:

- **condition** (obvezen) – XPath 2.0 pogojni izraz; pogoj se preveri pred vsakim izvajanjem zanke

Procesiranje se izvaja, dokler je pogoj resničen. Možno je nastaviti atribut *condition="true"* in kasneje eksplicitno prekiniti nadaljnje izvajanje z elementom </break>.

Element <break> nasilno prekine izvajanje stavkov <for>, <foreach> ali <while>. V elementih <if>, <elseif> ali <else> ga lahko uporabimo le, če je podelement prej naštetih iteracijskih elementov.

3.6 Prihodnost jezika EMML

Večina strokovnjakov trdi, da razširitev jezika EMML na trgu šele prihaja, vendar po Gartnerju[9] ne moremo pričakovati široke podpore za EMML, dokler se vsaj eno veliko podjetje, kot je IBM, SAP, Oracle ali Microsoft, ne vključi v zvezo OMA. Uspeh zveze OMA je torej odvisen od sposobnosti prepričati kupce podjetniških tehnologij o prednostih uporabe jezika EMML [27].

[9] Gartner – družba, ki se ukvarja z raziskavami in svetovanjem na področju informacijskih tehnologij s sedežem v Stamfordu, Connecticut.

Vodstvo IBM pravi, da je razlog, zakaj ne podpirajo EMML, v licenci Creative Commons, s katero pride specifikacija jezika EMML, ki prepoveduje izpeljana dela. To pomeni, da ni dovoljeno razširjati oz. prilagajati jezika brez dovoljenja lastnika, tj. OMA [28].

Kakorkoli, jezik EMML je edinstven sestavljalski jezik na trgu in ponuja standard za razvoj sestavljank. Razlogi, zakaj so EMML in sestavljalske platforme s podporo EMML nujno potrebni, so [29]:

- na tržišču je veliko število ponudnikov sestavljalskih platform brez enega dominantnega
- vse aplikacije ne ponujajo API, ki so glavni vir podatkov za sestavljanke
- potreben je konsistenten dostop do podatkovnih virov

Ker tržišče sestavljank še vedno narašča, brez dominantne platforme za razvoj sestaljank, je standardni način razvoja sestavljank potreben za pospešitev njihove uporabe. Kljub morebitnim oviram bo imel jezik EMML svetlo prihodnost.

Poglavje 4

STORITEV ZA NADZOR MEDNARODNE IZMENJAVE ŠTUDENTOV

Mednarodna izmenjava študentov, učnega osebja in drugih zaposlenih iz upravičenih institucij je vedno bolj popularna. Dejstvo, da se pričakuje, da bo število izmenjav študentov in učiteljev doseglo več kot 3 milijone do leta 2012, kaže, da se moramo zavedati, da iz 3 milijonov izmenjav sledi veliko število podatkov.

Podatki, ki jih dobimo, so iz različnih področij, in sicer od najbolj občutljivih podatkov o osebah do podatkov o sami izmenjavi in partnerskih institucijah. Zaradi tega mora biti oskrba podatkov na zelo visokem nivoju. Tudi če je vzpostavljena varnost nad podatki na izredno visokem nivoju, se zelo pogosto zgodi, da pri njihovi uporabi pride do neželenih izpostavitev podatkov nepooblaščenim osebam.

Še posebej je pomembno, da upravljanje in vpogled v podatke izvedemo na hiter in učinkovit način. Strokovnjaki se tako trudijo na različne načine približiti podatke uporabnikom za njihovo hitro analizo.

V tem problemskem področju vidimo tudi mednarodne izmenjave študentov, kjer sicer imamo podatke na namizju, vendar jih ne moremo na enostaven način uporabiti in dobiti boljšega pregleda nad izmenjavo študentov.

Naša rešitev ponuja možnost uporabe podjetniških sestavljank na področju mednarodnih izmenjav. V nadaljevanju podamo razloge za izbiro tega problemskega področja in opišemo vzpostavljeno arhitekturo s ključnimi elementi.

4.1 Ideja

Razvoj rešitve se je začel z idejo ponuditi in prikazati nov standardiziran način prikazovanja podatkov. Rešitev je namenjena vsem udeležencem v procesu izmenjave študentov, in sicer koordinatorjem izmenjav, študentom, upraviteljem financ in vsemu drugemu osebju, ki je tako ali drugače vključeno v proces izmenjave ali ima interes za vpogled v podatke o mednarodni izmenjavi študentov.

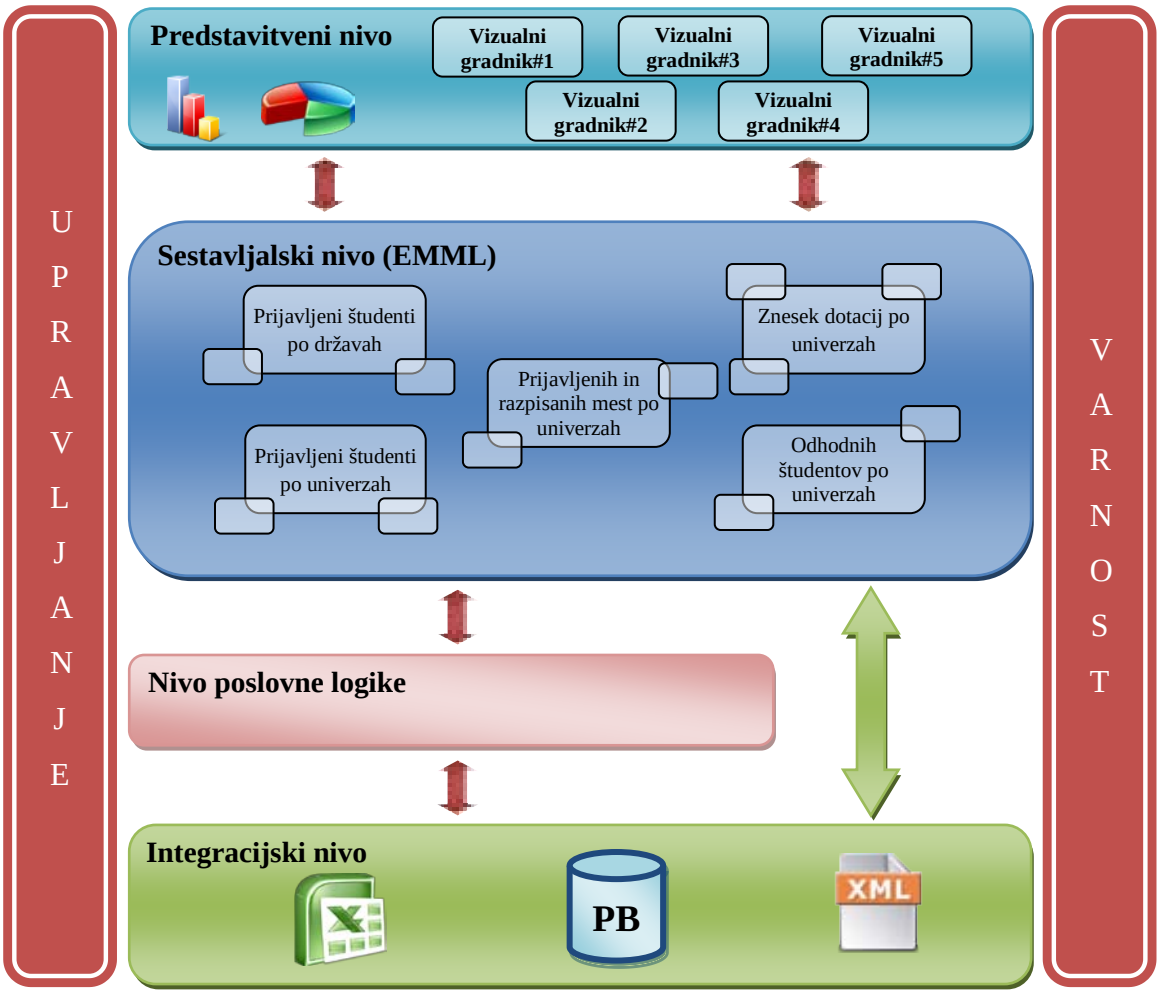
Projekt smo poimenovali *cDeep*, kar se nanaša na globok vpogled v podatke o mednarodni izmenjavi študentov. Ponuja nam možnost boljšega in bolj fleksibilnega vpogleda v izmenjavo študentov v povezavi s pridobljenim denarjem za financiranje izmenjav. Omogočili smo lastno nastavitev parametrov ponujenih storitev ter možnost kasnejše ponovne uporabe in rekombinacije vzpostavljenih storitev.

Storitev za nadzor mednarodne izmenjave študentov je zamišljena kot nabor izpostavljenih storitev v obliki sestavljank in vizualnih gradnikov, ki so dostopni neodvisno od lokacije in časa dostopa. Nabor sestavljank ni dokončno določen, saj prepustimo rekombinacijo in izdelavo novih sestavljank inovativnosti posameznega uporabnika.

4.2 Arhitektura rešitve cDeep

Arhitekturo rešitve cDeep smo zgradili, kot priporočajo strokovnjaki, na štirih nivojih (Slika 4.1). Prvi je t. i. integracijski nivo, ki abstrahira dostop do podatkov iz podatkovne baze, Excelovih in XML dokumentov. Integracija podatkovne baze je izvedena tako, da so tabele, pogledi in procedure za dostop do podatkov izpostavljeni kot storitve. Tretji nivo, nivo sestavljank, je vmesni nivo med predstavitvenim nivojem in nivojem poslovne logike,

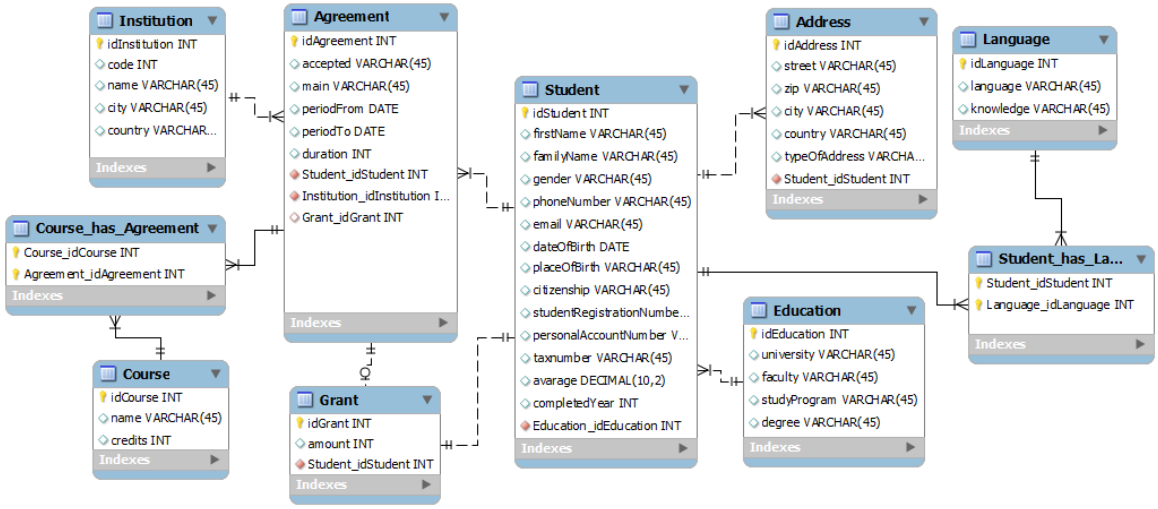
ki vsebuje definiran nabor sestavljank. Predstavitveni nivo sestavlja nabor vizualnih gradnikov (*angl. mashlets*) za predstavitev podatkov, ki se lahko kasneje uporabijo v različnih podjetniških portalih, spletnih dnevnikih (*angl. blogs*) ali pa jih lahko vgradimo v lastne spletne aplikacije [6].



Slika 4.1 Arhitektura rešitve cDeep

4.2.1 Podatkovni model

Razvoj rešitve cDeep se prične z načrtovanjem podatkovnega modela. Podatkovni model pokriva področje mednarodne izmenjave študentov in ga sestavlja deset entitet, ki vsebujejo osnovne podatke o študentih, njihovem doseženem izboraževanju, učnem načrtu s seznamom izbranih in potrjenih predmetov, ciljnih inštitucijah, izmenjavah, poznavanju tujih jezikov in odobrenih dotacijah (Slika 4.2).



Slika 4.2 Podatkovni model

Podatkovni model modeliramo na osnovi pridobljenih informacij o izmenjavi študentov v mednarodnem programu *Erasmus*. Pridobljene informacije so v obliki konkretnih vzorcev vlog za izmenjavo in pridobitev dotacij.

Podatkovni model modeliramo in upravljamo z uporabo produkta *MySQL Workbench 5.2*. Pretvorba logičnega modela v fizičnega je izvedena tudi z uporabo produkta *MySQL Workbench 5.2*.

4.2.2 Integracijski nivo

Razvoj rešitve nadaljujemo na integracijskem nivoju, v katerem abstrahiramo dostop do podatkov na varen in nadzorovan način. Izpostavljanje podatkov iz podatkovne baze in iz Excelovega dokumenta izvedemo na načina, opisana v nadaljevanju.

Abstrahiranje tabel, pogledov in procedure za dostop do podatkov iz podatkovne baze izpostavimo kot storitve z uporabo *Presto Service Explorer* in z ustrezno nastavitvijo gonilnika za povezovanje s podatkovno bazo MySQL.

Izpostavljanje podatkov iz Excelovega dokumenta v obliki REST storitev izvedemo iz vtičnika *Presto Excel Connector* za Microsoft Excel. Na ta način omogočimo dostop do podatkov v Excelovem dokumentu iz platforme Presto.

4.2.3 Sestavljalski nivo

Po objavljanju storitve, s katero dostopamo do podatkov, začnemo graditi sestavljanko na sestavljalskem sloju. Vsaka sestavljanka je grafično oblikovana z orodjem *Presto Wires* za modeliranje podjetniških sestavljank. Izhod vsake sestavljanke je v XML formatu.

Tabela 4.1 Opisi sestavljank

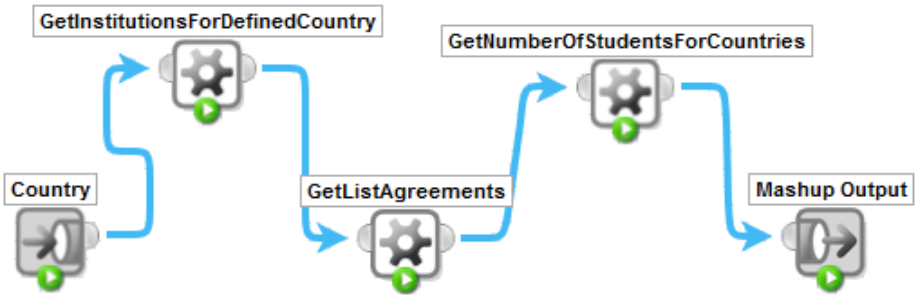
Ime sestavljanke	Opis sestavljanke
PrijavljeniŠtudentiPo Državah	Sestavljanka ima za vhod ime države, za katero dobimo podatke o številu tujih študentov po državah.
PrijavljeniŠtudentiPo Univerzah	Sestavljanka ima za vhod ime države, za katero dobimo podatke o številu tujih študentov, razporejenih po univerzah.
PrijavljenihInRazpisanih MestPoUniverzah	Sestavljanka ima za vhod ime države, za katero dobimo podatke o številu prijavljenih študentov in številu razpisanih mest po posameznih univerzah v državi.
ZnesekDotacijPoUniverzah	Sestavljanka ima za vhod ime države, za katero dobimo podatke o skupnem znesku dotacij za vsako univerzo posebej. Znesek dotacij eni univerzi je vsota vseh dotacij študentov, prijavljenih na tej univerzi.
OdhodnihStudentovPo Univerzah	Sestavljanka ima za vhod ime univerze, za katero dobimo podatke, na katere univerze so se študentje iz te univerze prijavili in v kolikšnem številu.

Poleg grafičnega modeliranja sestavljank je potrebno še ročno napisati podporne sestavljanke. Ročno pisanje sestavljank je nujno, ker orodje Presto Wires ne podpira grafičnega modeliranja bolj zapletenih situacij. V našem primeru je bilo potrebno modelirati procesiranje nabora vozlišč s pomočjo ukaza <foreach> ali <for>. Presto Wires trenutno omogoča grafično predstavitev ukazov za proženje storitev (Direct Invoke), kopiranje dela podatkov iz spremenljivke (Extract), filtriranje vsebine spremenljivke (Filter), določitev vhodnega parametra (Input), združevanje pod definiranim pogojem (Join), spajanje homogenih podatkov (Merge), razvrščanje podatkov (Sort) in formatiranje

dokumentov v CSV[10] formatu (FormatAsCSV). V nadaljevanju opišemo modeliranje vsake sestavljanke in njihovo strukturo.

4.2.3.1 Sestavljanka PrijavljeniŠtudentiPoDržavah

Namen sestavljanke *PrijavljeniŠtudentiPoDržavah* je pridobiti podatke o številu prijavljenih študentov po državah za definirano državo kot vhodni parameter sestavljanke. Za modeliranje sestavljanke uporabimo in ustrezno povežemo dve ročno napisani sestavljanki v Presto Mashup Studio in eno izpostavljeno storitev (Slika 4.3).



Slika 4.3 Diagram sestavljanke PrijavljeniŠtudentiPoDržavah

Prva komponenta sestavljanke je ime države, ki je vhodni parameter za storitev *GetInstitutionsForDefinedCountry*. Dobljeni rezultat je seznam institucij, ki se naprej posreduje k ročno napisani sestavljanki *GetListAgreements* (XML dokument 7.1).

Sestavljanka GetListAgreements kot vhodni parameter pridobi seznam institucij. S pomočjo ukaza <foreach> za vsako institucijo pridobimo seznam učnih načrtov, ki jih dodamo k izhodu. Na osnovi seznama učnih načrtov ročno napisana sestavljanka *GetNumberOfStudentsForCountries* generira XML dokument s podatki o imenu države in številu prijavljenih študentov iz te države (Primer kode 4.1).

[10] Format za besedilno datoteko, ki vsebuje z vejico ločene vrednosti.

```
<output>
    <country>
        <name>Država1</name>
        <numberStudents>5</numberStudents>
    </country>
    <country>
        <name>Država2</name>
        <numberStudents>3</numberStudents>
    </country>
    ...
</output>
```

Primer kode 4.1 Struktura izhodnega XML dokumenta sestavljanke PrijavljeniŠtudentiPoDržavah

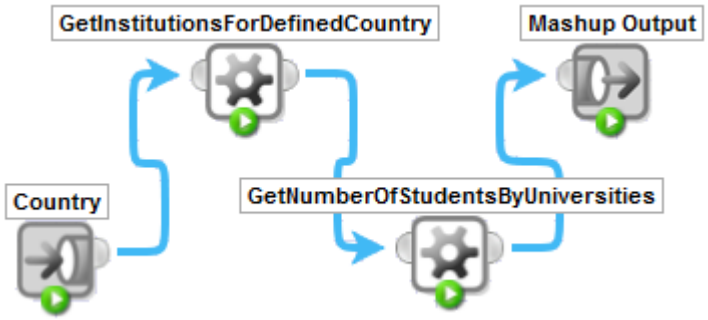
Ročno napisano sestavljanko GetNumberOfStudentsForCountries prikažemo v XML dokumentu 7.2.

Sestavljanka ima za vhodni parameter seznam učnih načrtov, iz katerih dobimo seznam študentov za vsak učni načrt. S pomočjo ukaza <group> grupiramo študente po državljanstvu ter preštejemo študente za vsako državo posebej.

Tako modelirano sestavljanko *PrijavljeniŠtudentiPoDržavah* objavimo kot REST spletno storitev, za katero ustvarimo vizualni gradnik, predstavljen v poglavju 4.3.

4.2.3.2 Sestavljanka PrijavljeniŠtudentiPoUniverzah

Sestavljanka *PrijavljeniŠtudentiPoUniverzah* ima za vhod ime države, za katero dobimo podatke o številu tujih študentov, razporejenih po univerzah. Za modeliranje sestavljanke ustrezno povežemo eno izpostavljeno storitev in eno ročno napisano sestavljanko s Presto Mashup Studiom.



Slika 4.4 Diagram sestavljanke PrijavljeniŠtudentiPoUniverzah

Prva komponenta sestavljanke je ime države, za katero s pomočjo storitve *GetInstitutionsForDefinedCountry* dobimo seznam vseh institucij. Ta seznam je vhod sestavljanke *GetNumberOfStudensByUniversities*, ki procesira vse institucije in kot rezultat vrne seznam institucij s številom prijavljenih študentov.

V XML dokumentu 7.3 prikažemo sestavljanko GetNumberOfStudentsByUniversities, kjer iz seznama institucij pridobimo seznam vseh učnih načrtov, ki jih grupiramo s pomočjo ukaza <group> in ustvarimo XML dokument, ki ima strukturo kot v primeru kode 4.2.

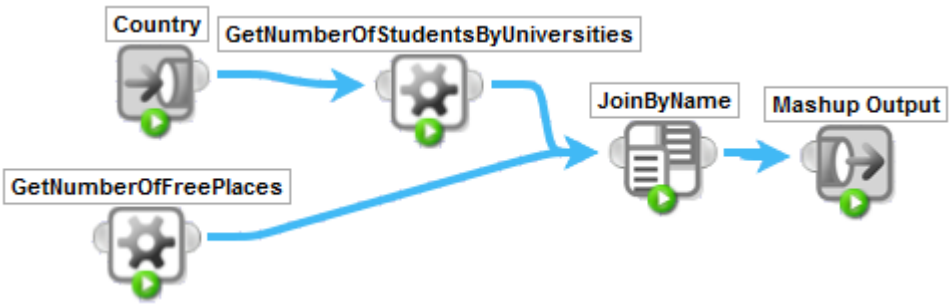
```
<institutions>
    <institution>
        <name>Fakulteta za elektrotehniko, računalništvo in informatiko</name>
        <numberStudents>5</numberStudents>
    </institution>
    <institution>
        <name>Pedagoška fakulteta</name>
        <numberStudents>9</numberStudents>
    </institution>
    ...
</institutions>
```

Primer kode 4.2 Struktura izhodnega XML dokumenta sestavljanke PrijavljeniŠtudentiPoUniverzah

Izhod sestavljanke GetNumberOfStudentsByUniversities se posreduje kot končni rezultat sestavljanki PrijavljeniŠtudentiPoUniverzah, ki jo objavimo kot REST spletno storitev. Vizualni gradnik sestavljanki prikažemo v poglavju 4.3.

4.2.3.3 Sestavljanka PrijavljenihInRazpisanihMestPoUniverzah

Sestavljanka združuje dva vira podatkov, in sicer podatke iz Excelovega dokumenta in iz sestavljanke *GetNumberOfStudentsByUniversities*. Podatke iz Excelovega dokumenta objavimo v obliki REST spletne storitve s pomočjo *Presto Excel Connectorja*. Diagram modelirane sestavljanke prikažemo na sliki 4.5.



Slika 4.5 Diagram sestavljanke PrijavljenihInRazpisanihMestPoUniverzah

Vhodni parameter sestavljanke PrijavljenihInRazpisanihMestPoUniverzah je država, za katero s pomočjo sestavljanke GetNumberOfStudentsByUniversities dobimo seznam vseh institucij v tej državi in število prijavljenih študentov po institucijah. Storitev GetNumberOfFreePlaces vrne seznam institucij s številom razpisanih prostih mest. Ta

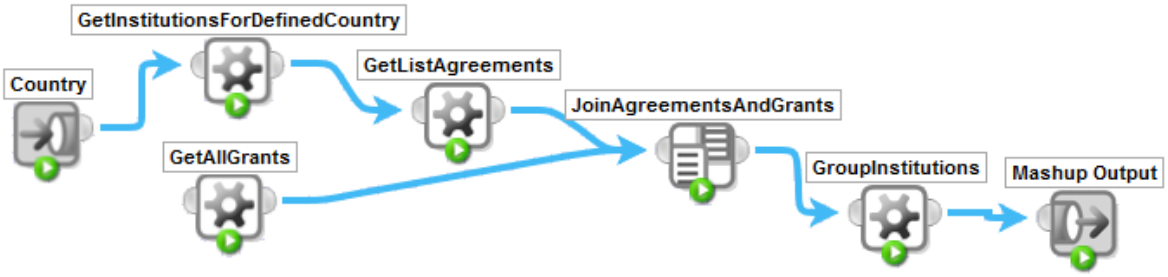
seznama združimo s konstruktom *JoinByName*, ki združi sezname po imenu institucije in posreduje rezultat združitve kot končni rezultat sestavljanke.

Ker je skripta sestavljanke GetNumberOfStudentsByUniversities podobna prejšnjim primerom sestavljank, prikažemo s Presto Wires generirano EMML skripto kot sestavljanko PrijavljenihInRazpisanihMestPoUniverzah (XML dokument 7.4).

Vizualni gradnik sestavljanke prikažemo v poglavju 4.3.

4.2.3.4 Sestavljanka ZnesekDotacijPoUniverzah

Ker so informacije o finančnih sredstvih zelo privlačne, smo modelirali sestavljanko, ki za definirano državo prikaže skupen znesek dotacij, razporejenih po univerzah, ki jih študentje, prijavljeni na teh unverzah, pridobijo (Slika 4.6).



Slika 4.6 Diagram sestavljanke ZnesekDotacijPoUniverzah

Sestavljanko modeliramo tako, da ustrezno povežemo dve storitvi (*GetInstitutionsForDefinedCountry* in *GetAllGrants)* in dve sestavljanki (*GroupInstitutions* in *GetListAgreements*).

Konstukt *JoinAgreementAndGrants* združuje seznam učnih načrtov s seznamom dotacij po ključu *idGrant*. Rezultat konstrukta *JoinAgreementsAndGrants* ustrezno obdelamo s sestavljanko *GroupInstitutions*, pri čemer najprej za vsak učni načrt pridobimo ustrezno institucijo in jo dodamo v seznam. Tako ustvarjen seznam institucij grupiramo po imenu institucij in rezultat posredujemo končnemu rezultatu v sestavljanki ZnesekDotacijPoUniverzah.

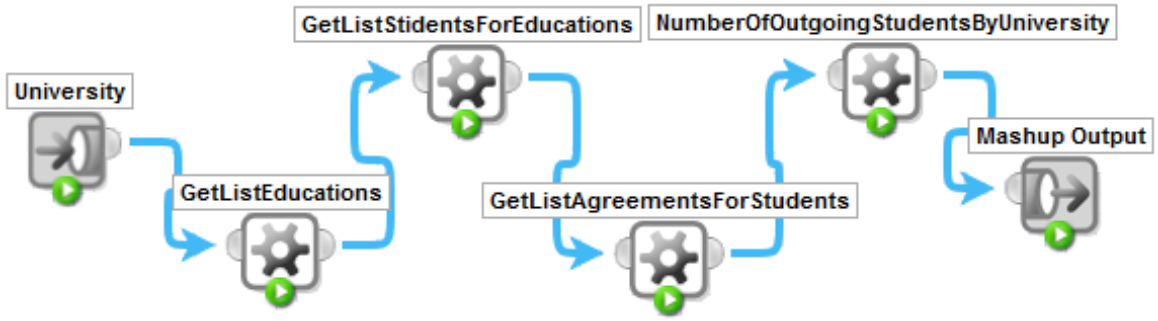
V XML dokumentu 7.5 prikažemo sestavljanko GroupInstitutions, kjer učne načrte grupiramo po ključu *idInstitution* in rezultat strukturiramo. Dobljeni rezultat z grupiranjem

naknadno procesiramo z ukazom <foreach> in za vsako vrednost *idInstitution* pridobimo ustrezno vrednost institucije ter jo dodamo h končnemu rezultatu.

Tako zgrajeno sestavljanko objavimo kot REST spletno storitev.

4.2.3.5 Sestavljanka OdhodnihStudentovPoUniverzah

Namen sestavljanke je pridobiti podatke o številu odhodnih študentov univerze. Ime univerze je podano kot vhodni parameter sestavljanke. Sestavljanke modeliramo tako, da ustrezno povežemo eno storitev (*GetListEducations*) in tri sestavljanke (*GetListStudentsForEducations*, *GetListAgreementsForStudents* in *NumberOfOutgoingStudentsByUniversities*) (Slika 4.7).



Slika 4.7 Diagram sestavljanke OdhodnihStudentovPoUniverzah

V XML dokumentu 7.6 prikažemo sestavljanko NumberOfOutgoingStudentsByUniversities, kjer na osnovi seznama učnih načrtov dobimo seznam institucij, učni načrti katerih se povezujejo. Seznam institucij grupiramo po imenu in zgradimo XML dokument kot končni rezultat sestavljanke (Primer kode 4.3).

```
<output>
    <institution>
        <name>Vienna Technical University</name>
        <numberOfStudents>11<numberOfStudents>
    </institution>
    <institution>
        <name>Budapest University of Technology and Economics</name>
        <numberOfStudents>3<numberOfStudents>
    </institution>
</output>
```

Primer kode 4.3 Struktura izhodnega XML dokumenta sestavljanke OdhodnihStudentovPoUniverzah

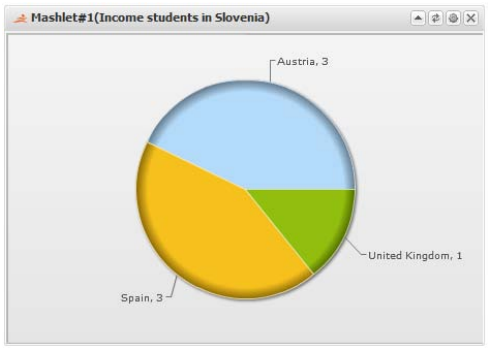
Sestavljanko OdhodnihStudentovPoUniverzah objavimo v obliki REST spletne storitve, za katero zgradimo vizualni gradnik.

4.3 Predstavitveni nivo

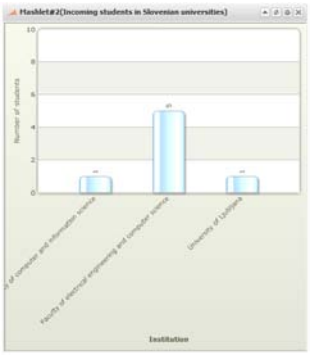
Razvoj rešitve nadaljujemo na predstavitvenem nivoju, kjer z orodjem *Presto Mashlet Maker* zgradimo vizualne gradnike za vsako sestavljanko.

Gradnjo vizualnih gradnikov začnemo z izbiro sestavljanke, za katero si želimo zgraditi vizualni gradnik. Nato sledi definiranje vhodnih parametrov in izbira načina prikazovanja. Lahko izbiramo med vizualnimi gradniki za prikazovanje podatkov v RSS formatu, tabelarično, na grafikonu v obliki strukturnega kroga ali stolpcev, na Yahoo zemljevidu ali v drevesnem prikazu strukture podatkov.

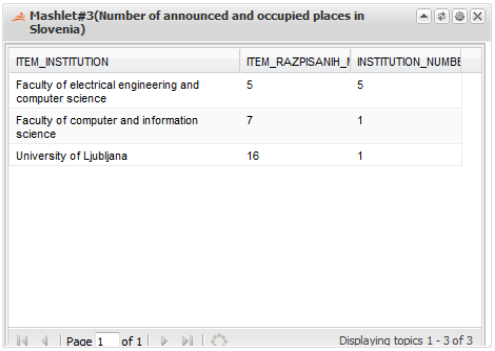
Na sliki 4.8, sliki 4.9, sliki 4.10, sliki 4.11 in sliki 4.12 prikažemo zgrajene vizualne gradnike za vse sestavljanke.



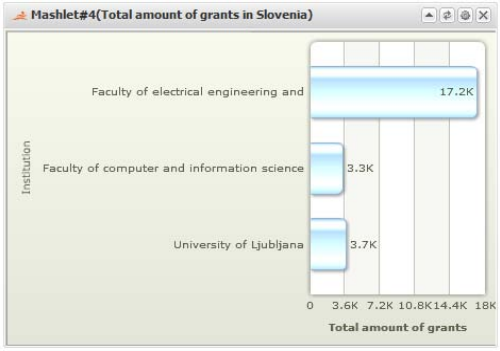
Slika 4.8 Vizualni gradnik za sestavljanko PrijavljeniŠtudentiPoDržavah



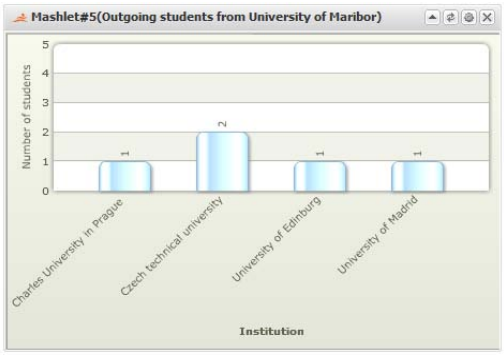
Slika 4.9 Vizualni gradnik za sestavljanko PrijavljeniŠtudentiPoUniverzah



Slika 4.10 Vizualni gradnik za sestavljanko PrijavljenihInRazpisanihMestPoUniverzah



Slika 4.11 Vizualni gradnik za sestavljanko ZnesekDotacijPoUniverzah



Slika 4.12 Vizualni gradnik za sestavljanko OdhodnihStudentovPoUniverzah

Zgrajeni vizualni gradniki se lahko vgradijo v portalih, spletnih dnevnikih ali spletnih aplikacijah.

S praktičnim primerom smo utemeljili vrednost podjetniških sestavljank. Nadaljnji razvoj rešitve bi predstavljal razvoj nivoja poslovne logike, ki ga v tem diplomskem delu ne obravnavamo, in postavitev naše rešitve v oblaku za zagotovitev njene večje dostopnosti in skalabilnosti.

Poglavje 5

SKLEP

Tehnologija podjetniških sestavljank se identificira kot dodaten integracijski sloj, namenjen končnim uporabnikom. Uporabniško orientirana integracija podatkovnih virov, ki ponujajo podatke v različnih formatih, je osrednji namen podjetniških sestavljank. S tem pridobimo pri času, ki je potreben za razvoj sestavljank, in zmanjšamo stroške.

Sklepamo, da se bodo v prihodnosti podjetniške sestavljanke razvijale v smeri gradnje bolj uporabniško usmerjene platforme za razvoj sestavljank, ki bodo zagotovile visok nivo varnosti in upravljanja in s tem večji delež trga uporabe tehnologije podjetniških sestavljank.

Ugotovili smo, da je trenutno na področju podjetniških sestavljank veliko število ponudnikov platform za razvoj sestavljank, vendar ne zagotavljajo prenosljivosti svojih rešitev, to pa je zelo pomembno v poslovnem svetu.

Prihodnost podjetniških sestavljank je pogojena s standardizacijo njihovega razvoja. Jezik EMML kot standard za razvoj podjetniških sestavljank ponuja interoperabilnost in prenosljivost sestavljalskih rešitev z zagotovljeno varnostjo. Prihodnost tega jezika pa je pogojena z vključitvijo vsaj enega velikega podjetja (IBM, SAP, Oracle ali Microsoft) v njegov nadaljnji razvoj in uporaba jezika EMML v tem podjetju pri svojih sestavljalskih rešitvah.

EMML smo identificirali kot zmogljiv jezik z visokonivojsko sposobnostjo za proženje storitev, deklarativno transformacijo podatkov, paralelizacijo, virtualizacijo storitev ipd.

Odgovor na vprašanje, ali je možen razvoj podjetniških sestavljank brez jezika EMML, je DA, vendar ta pristop razvoja ne zagotavlja visokokakovostnih in kompatibilnih rešitev.

S praktičnim primerom potrdimo prednosti razvoja aplikacij tipa podjetniških sestavljank z jezikom EMML. Ugotovimo, da EMML ponuja nov, hiter pristop k pridobivanju, transformiranju, sestavljanju in prikazovanju podatkov iz različnih podatkovnih virov na enem samem mestu.

Samo dejstvo, da je bil trg podjetniških sestavljank v letu 2008 vreden približno 161 milijonov dolarjev in da je napovedana rast 1,74 milijarde dolarjev do leta 2013, kaže, da se bo razvoj tehnologije podjetniških sestavljank in jezika EMML še naprej nadaljeval [13].

Poglavje 6

LITERATURA

1. Kurtzman, Wayne. We Live In Exponential Times. *Media Bullseye.* [Elektronski] Media Bullseye, 14. Maj 2010. [Navedeno: 12. Junij 2010.] http://www.mediabullseye.com/mb/2009/05/we-live-in-exponential-times.html.

2. Wikipedia. Mashup. *Wikipedia.* [Elektronski] Wikipedia, 15. Maj 2010. [Navedeno: 15. Maj 2010.] http://en.wikipedia.org/wiki/Mashup_(web_application_hybrid).

3. Warner, Chris. Defining Enterprise Mashups. *The Enterprise Web 2.0 Blog.* [Elektronski] JackBe, 10. Marec 2009. [Navedeno: 15. Marec 2010.] http://blogs.jackbe.com/2009/03/defining-enterprise-mashups.html.

4. Ogrinz, Michael. *Mashup Patterns: Designs and Examples for the Modern Enterprise.* Boston : Pearson Education, 2009. 978-0-321-57947-8.

5. ProgrammableWeb. ProgrammableWeb. *ProgrammableWeb.* [Elektronski] 19. Avgust 2010. [Navedeno: 19. Avgust 2010.] http://www.programmableweb.com/apis.

6. Crupi, John. A Business Guide to Enterprise Mashups. *JackBe.* [Elektronski] April 2008. [Navedeno: 20. Maj 2010.] http://www.jackbe.com/downloads/Jackbe_business_guide_to_enterprise_mashups.pdf.

7. Bradley, Anthony. Reference architecture for enterprise mashups. *Gartner*. [Elektronski] 7. September 2007. [Navedeno: 14. Maj 2010.] http://www.gartner.com/DisplayDocument?doc_cd=151491.

8. Griffin, Eric. *Foundations of Popfly: Rapid Mashup Development.* Berkeley : Apress, 2008. 978-1-59059-951-8.

9. Cresswell, Anthony M. Return on Investment In Information Technology:A Guide for Managers. *Center for Technology in Government.* [Elektronski] Avgust 2004. [Navedeno: 26. Maj 2010.] http://www.ctg.albany.edu/publications/guides/roi/roi.pdf.

10. Crupi, John. Enterprise mashups part I: Bringing SOA to the People. *The SOA Magazine.* [Elektronski] 16. Maj 2008. [Navedeno: Maj. 18 2010.] http://www.soamag.com/I18/0508-1.pdf.

11. Lane, Cooper. Mashups in the Enterprise IT Environment. *JackBe.* [Elektronski] 2010. [Navedeno: 28. Maj 2010.] http://www.jackbe.com/about/environment_form.php.

12. Wikipedia. Service-oriented architecture. *Wikipedia.* [Elektronski] Wikipedia, 4. Junij 2010. [Navedeno: 4. Junij 2010.] http://en.wikipedia.org/wiki/Service-oriented_architecture.

13. Insights, Business. The Future of Enterprise Mashups: Demand, Challenges and Vendor. *Research and markets.* [Elektronski] Business Insights, Avgust 2009. [Navedeno: 13. Junij 2009.] http://www.researchandmarkets.com/reports/1083799/the_future_of_enterprise_mashups_demand.

14. Courtney, Martin. Mashup tools: enterprise enablers for the mashed age. *The IET Knowledge Network.* [Elektronski] The IET Knowledge Network, 22. Marec 2010. [Navedeno: 26. Juni 2010.] http://kn.theiet.org/magazine/issues/1005/mash-up-makers-1005.cfm.

15. JackBe. Presto: The Enterprise Mashup Platform. *JackBe.* [Elektronski] 2009. [Navedeno: 17. April 2010.] http://www.jackbe.com/downloads/Jackbe_Presto_Overview.pdf.

16. IBM. Work smarter with IBM Mashup Center V2.0. *IBM.* [Elektronski] 27. Oktober 2009. [Navedeno: 25. Avgust 2010.] http://www-01.ibm.com/common/ssi/rep_ca/0/897/ENUS209-360/ENUS209-360.PDF.

17. Convertigo. Enterprise Mashup Development with Eclipse. *Convertigo.* [Elektronski] Convertigo, 2010. [Navedeno: 24. Julij 2010.] http://www.convertigo.com/en/overview/development.html.

18. Technologies, Kapow. Kapow Web Data Server. *Kapow Technologies.* [Elektronski] Kapow Technologies, 2010. [Navedeno: 27. Avgust 2010.] http://kapowtech.com/index.php/products/kapowwebdataserver.

19. Warner, Cris. Enterprise Mashup Security 101. *JackBe.* [Elektronski] JackBe, 13. Marec 2008. [Navedeno: 5. Junij 2010.] http://blogs.jackbe.com/2008/03/mashup-security-101.html.

20. McKendrick, Joe. Five enterprise mashup security issues, and what to do about them. *ZDNet.* [Elektronski] ZDNet, 10. Junij 2010. [Navedeno: 20. Avgust 2010.] http://www.zdnet.com/blog/service-oriented/five-enterprise-mashup-security-issues-and-what-to-do-about-them/4997.

21. Absalom, Richard. The Future of Enterprise Mashups: Demand, challenges and vendor opportunities. *DreamFace Interactive.* [Elektronski] DreamFace Interactive, 26. Oktober 2009. [Navedeno: 19. Avgust 2010.] http://www.dreamface-interactive.com/blog/?p=70.

22. Wikipedia. EMML. *Wikipedia.* [Elektronski] Wikipedia, 15. Maj 2010. [Navedeno: 15. Maj 2010.] http://en.wikipedia.org/wiki/EMML.

23. Alliance, Open Mashup. EMML Schema, EMML Reference Runtime Implementation, and Documentation License Terms. *Open Mashup Alliance.* [Elektronski] Open Mashup Alliance, 23. September 2009. [Navedeno: 5. Avgust 2010.] http://www.openmashup.org/download/agreement.php.

24. —. OMA EMML Documentation. *Open Mashup Alliance.* [Elektronski] Open Mashup Alliance, 23. September 2009. [Navedeno: 5. April 2010.] http://www.openmashup.org/omadocs/v1.0/index.html.

25. JackBe. An Architects's Guide to Enterprise Mashups. *JackBe.* [Elektronski] 2. November 2009. [Navedeno: 3. Julij 2010.] http://www.jackbe.com/downloads/architects_guide_day2_qa.pdf.

26. Alliance, Open Mashup. EMML Schema. *Open Mashup Alliance.* [Elektronski] 15. April 2009. [Navedeno: 10. April 2010.] http://www.openmashup.org/schemas/v1.0/EMMLSpec.xsd.

27. Knipp, , Eric, Valdes, , Ray in Bradley, Anthony. Open Mashup Alliance Needs More Support to Create Standardization. *Gartner.* [Elektronski] 30. September 2009. [Navedeno: 18. April 2010.] http://www.gartner.com/resources/171600/171619/open_mashup_alliance_needs_m_171619.pdf.

28. IBM. What is IBMs official view on Enterprise Mashup Markup Language? *IBM.* [Elektronski] IBM, 17. Marec 2010. [Navedeno: 13. Maj 2010.] http://www.ibm.com/developerworks/forums/thread.jspa?threadID=321569.

29. Viswanathan, Arun. Mashups and the Enterprise Mashup Markup Language (EMML). *Dr.Dobb's.* [Elektronski] Dr.Dobb's, 12. April 2010. [Navedeno: 15. Maj 2010.] http://www.drdobbs.com/java/224300049.

Poglavje 7

PRILOGE

7.1 Seznam slik

7.2 Seznam tabel

7.3 Seznam shem

7.4 Seznam primerov kode

7.5 Seznam XML dokumentov

```
<mashup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.jackbe.com/2008-03-01/EMMLSchema/
                        ../src/schemas/EMMLSpec.xsd"
    xmlns="http://www.jackbe.com/2008-03-01/EMMLSchema"
    xmlns:macro="http://www.jackbe.com/2008-03-01/EMMLMacro"
    name="GetListAgreements">
    <operation name="invoke">
        <input name="listinstitutions" type="document" />
        <output name="result" type="document" />
        <variables>
            <variable name="tempagreements" type="document" />
        </variables>
        <foreach variable="institution"
                 items="$listinstitutions/*:Institution_Array/*:Institution">
            <variable name="tempinstitutionId" type="string"
                default="$institution/*:idInstitution" />
            <assign fromexpr="$institution/*:idInstitution"
                outputvariable="tempinstitutionId" />
            <invoke service="ListAgreementsForDefinedInstitutionId"
                operation="runMashup" inputvariables="tempinstitutionId"
                outputvariable="tempagreements" />
            <foreach variable="agreement1"
                items="$tempagreements/*:Agreement_Array/*:Agreement">
                <appendresult outputvariable="$result">
                    $agreement1
                </appendresult>
            </foreach>
        </foreach>
    </operation>
</mashup>
```

XML dokument 7.1 Sestavljanka GetListAgreements

```
<mashup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.jackbe.com/2008-03-01/EMMLSchema/
        ../src/schemas/EMMLSpec.xsd"
    xmlns="http://www.jackbe.com/2008-03-01/EMMLSchema"
    name="GetNumberOfStudentsForCountries">
    <operation name="invoke">
        <input name="agreements" type="document" />
        <output name="result" type="document" />
        <variables>
            <variable name="tempstudent" type="document" />
            <variable name="liststudentid" type="document" />
        </variables>
        <foreach variable="agreement" items="$agreements/*:xml/*:Agreement">
            <appendresult outputvariable="liststudentid">
                $agreement/*:studentIdstudent
            </appendresult>
        </foreach>
        <foreach variable="studentid" items="$liststudentid/*:xml/*:studentIdstudent">
            <variable name="tempstudentId" type="string" default="{$studentid/text()}"/>
            <assign  fromexpr="$studentid/text()" outputvariable="tempstudentId" />
            <invoke service="GetStudentById" operation="runMashup"
                inputvariables="tempstudentId" outputvariable="tempstudent" />
            <appendresult outputvariable="$result">
                $tempstudent
            </appendresult>
        </foreach>
        <variable name="groupresult" type="document" />
        <group by="$result/*:xml/*:Student/*:citizenship" outputvariable="$groupResult">
            <output>
                <country>
                    <name>{$group_key}</name>
                    <numberStudents>{count(citizenship)}</numberStudents>
                </country>
            </output>
        </group>
        <assign fromvariable="$groupResult" outputvariable="$result" />
    </operation>
</mashup>
```

XML dokument 7.2 Sestavljanka GetNumberOfStudentsForCountries

```
<foreach variable="institution"
items="$listinstitutions/*:Institution_Array/*:Institution">
    <variable name="tempId" type="string" default="$institution/*:idInstitution" />
    <assign fromexpr="$institution/*:idInstitution" outputvariable="tempId" />
    <invoke service="ListAgreementsForDefinedInstitutionId" operation="runMashup"
        inputvariables="tempId" outputvariable="tmpagreements" />
    <foreach variable="agreement1" items="$tmpagreements/*:Agreement_Array/*:Agreement">
        <appendresult outputvariable="$result">
            <institution>{$institution/*:name}</institution>
        </appendresult>
    </foreach>
</foreach>
<group by="$result/*:xml/*:institution/*:name" outputvariable="$groupResult" >
    <institutions>
        <institution>
            <name>{$group_key}</name>
            <numberStudents>{count(name)}</numberStudents>
        </institution>
    </institutions>
</group>
```

XML dokument 7.3 Sestavljanka GetNumberOfStudentsByUniversities

```
<mashup name="cDeepMashup3"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.jackbe.com/2008-03-01/EMMLSchema../src/schemas/
        EMMLSpec.xsd" xmlns=http://www.jackbe.com/2008-03-01/EMMLSchema"
    xmlns:macro="http://www.jackbe.com/2008-03-01/EMMLMacro">
    <operation name="runMashup">
        <output wires_id="output_0" name="output_0" type="document" service=""/>
        <variable name="invoke_1_out" type="document" default="" />
        <invoke wires_id="invoke_1" name="invoke_1" header="" service="Razpis"
            operation="getData" serviceVersion="" outputvariable="invoke_1_out"
            onerror="continue" inputvariables="" />
        <variable name="invoke_5_out" type="document" default="" />
        <inputparam wires_id="inputparam_11" name="country_2_11" type="string"
            default="Slovenia"/>
        <invoke wires_id="invoke_5" name="invoke_5" header=""
            service="NumberOfStudentsInInstitutionsInDefinedCountry"
            operation="runMashup" serviceVersion="" outputvariable="invoke_5_out"
            onerror="continue" inputvariables="country_2_11" />
        <join wires_id="join_6" name="join_6" inputvariables="invoke_1_out,invoke_5_out"
            outputvariable="output_0" joincondition="$invoke_1_out/*:RazpisData/*:
            item/*:Institution=$invoke_5_out/*:output/*:institution/*:institution">
        <select>
        <item_institution>
            <item_Institution>{$invoke_1_out/*:Institution/string()}</item_Institution>
            <item_code>{$invoke_1_out/*:code/string()}</item_code>
            <item_razpisanih_mest>{$invoke_1_out/*:razpisanih_mest/string()}
            </item_razpisanih_mest>
            <institution_institution>{$invoke_5_out/*:institution/string()}
            </institution_institution>
            <institution_numberStudents>{$invoke_5_out/*:numberStudents/string()}
            </institution_numberStudents>
        </item_institution>
        </select>
        </join>
    </operation>
</mashup>
```

XML dokument 7.4 Sestavljanka PrijavljenihInRazpisanihMestPoUniverzah

```
<group by="$agreements/*:result/*:Agreement_Grant/*:Agreement_institutionIdinstitution"
    outputvariable="$groupresult" >
    <institutions>
        <institution>{$group_key}</institution>
        <amount>{sum(Grant_amount)}</amount>
    </institutions>
</group>
<foreach variable="institution" items="$groupresult/*:output/*:institutions" >
    <assign  fromexpr="$institution/*:institution" outputvariable="tempinstitutionId" />
    <invoke service="getInstitutionById" operation="runMashup"
        inputvariables="tempinstitutionId" outputvariable="inst" />
    <appendresult outputvariable="$result">
        <result>
            {$institution/*:amount}
            {$inst}
        </result>
    </appendresult>
</foreach>
```

XML dokument 7.5 Sestavljanka GroupInstitutions

```
<foreach variable="agreement" items="$listagreements/*:xml/*:Agreement" >
    <variable name="tempinstitutionid" type="string"
        default="$agreement/institutionIdinstitution/string()" />
    <assign  fromexpr="$agreement/institutionIdinstitution/string()"
        outputvariable="tempinstitutionid" />
    <invoke service="getInstitutionById" operation="runMashup"
        inputvariables="tempinstitutionid" outputvariable="tempinstitution" />
    <appendresult outputvariable="$result">
        $tempinstitution
    </appendresult>
</foreach>
<variable name="tempresult" type="document" />
<group by="$result/*:xml/*:Institution/*:name" outputvariable="tempresult">
    <institution>
        <name>{$group_key}</name>
        <numberOfStudents>{count(name)}</numberOfStudents>
    </institution>
</group>
```

XML dokument 7.6 Sestavljanka NumberOfOutgoingStudentsByUniversities

7.6 Naslov študenta

Ime in priimek:	Milan Dojchinovski
Naslov:	Vera Jocikj 10/2-2
Pošta:	1000 Skopje
Država:	Makedonija
Telefon študenta:	+386 31 291 498; +389 71 632 310
E-pošta:	dojcinovski.milan@gmail.com

7.7 Kratek življenjepis

Rojen: 25. marec 1986 v Skopju, Makedonija

Šolanje: 1993–2001 OU Kiril Pejčinovikj, Skopje, Makedonija

2001–2005 DSZ Rade Jovčevski-Korčagin, Skopje, Makedonija

2005–2010 Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, Slovenija

FERI
Fakulteta za elektrotehniko, računalništvo in informatiko

Smetanova ulica 17
2000 Maribor

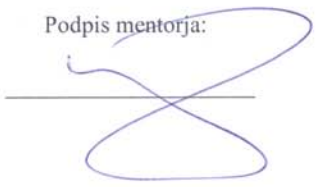
IZJAVA O USTREZNOSTI DIPLOMSKEGA DELA

Podpisani mentor red. prof. dr. Marjan Heričko izjavljam, da je študent Milan Dojchinovski izdelal diplomsko delo z naslovom: Razvoj sestavljank z jezikom EMML v skladu z odobreno temo diplomskega dela, Navodili o pripravi diplomskega dela in mojimi navodili.

Datum in kraj:

9. 9. 2010, Maribor

Podpis mentorja:



FERI
Fakulteta za elektrotehniko, računalništvo in informatiko
Smetanova ulica 17
2000 Maribor

IZJAVA O ISTOVETNOSTI TISKANE IN ELEKTRONSKE VERZIJE DIPLOMSKEGA DELA IN OBJAVI OSEBNIH PODATKOV DIPLOMANTOV

Ime in priimek diplomanta: Milan Dojchinovski

Vpisna številka: 93595710

Študijski program: računalništvo in informatika, smer informatika

Naslov diplomskega dela: Razvoj sestavljank z jezikom EMML

Mentor: red. prof. Marjan Heričko

Somentor: asist. dr. Luka Pavlič

Podpisani Milan Dojchinovski izjavljam, da sem za potrebe arhiviranja oddal elektronsko verzijo zaključnega dela v Digitalno knjižnico Univerze v Mariboru.
Diplomsko delo sem izdelal sam ob pomoči mentorja. V skladu s 1. odstavkom 21. člena Zakona o avtorskih in sorodnih pravicah (Ur. l. RS, št. 16/2007) dovoljujem, da se zgoraj navedeno zaključno delo objavi na portalu Digitalne knjižnice Univerze v Mariboru.

Tiskana verzija diplomskega dela je istovetna elektronski verziji, ki sem jo oddal za objavo v Digitalno knjižnico Univerze v Mariboru.

Podpisani izjavljam, da dovoljujem objavo osebnih podatkov, vezanih na zaključek študija (ime, priimek, leto in kraj rojstva, datum diplomiranja, naslov diplomskega dela), na spletnih straneh in v publikacijah UM.

Datum in kraj:
9. 9. 2010, Maribor

Podpis diplomanta: