Vloga ogrodij pri razvoju sodobnih informacijskih rešitev

Andrej Krajnc, Marjan Heričko
IZUM, Prešernova 17, 2000 Maribor, andrej.krajnc@izum.si
FERI Maribor, Smetanova 17, 2000 Maribor, marjan.hericko@uni-mb.si

Povzetek

Prispevek opisuje uporabo in vlogo ogrodij pri razvoju sodobnih informacijskih rešitev. Prva uporabna ogrodja so bila narejena v osemdesetih letih. Največ prvih ogrodij je bilo uporabnih na nivoju grafičnega vmesnika, vse bolj pa se ogrodja uveljavljajo tudi na nivoju poslovnih komponent. Čeprav koncept ogrodij zelo spominja na druge koncepte ponovne uporabe, kot so komponente, knjižnice razredov in vzorci, obstajajo med njimi določene razlike. Ker so ogrodja uporabna na številnih področjih, obstaja veliko različnih ogrodij, ki jih je možno klasificirati na več načinov. Prispevek predlaga pristop k celoviti klasifikaciji ogrodij. Opredeljena je tudi vloga organizacijskih ogrodij, ki se od ostalih ogrodij razlikujejo predvsem po obsegu in osredotočenosti. Organizacijska ogrodja so veliko bolj kompleksna in poskušajo zaobjeti več vidikov, zato je tudi izgradnja takšnih ogrodij veliko bolj zahtevna. V prispevku so analizirani različni pristopi h gradnji ogrodij, poudarek pa je na definiranju primernega pristopa k razvoju organizacijskih ogrodij.
Ključne besede: objektno-orientirana programska ogrodja, klasifikacija, vzorci, komponente, ponovna uporaba, organizacijska ogrodja

Abstract

The Role of Frameworks in Developing Modern Information Solutions

The paper describes the role and usage of frameworks in developing modern information solutions. The first usable frameworks were developed in 1980's. Initially they meant to be focused on GUI development but nowadays frameworks can be found on the level of business components. Although the concept of frameworks is similar to the other reuse techniques, such as components, class libraries and patterns, they differ from each other. Because frameworks can be applied in many domains, there are many frameworks, which can be classified in different ways. The paper proposes a complete approach how to classify frameworks. The paper also describes enterprise frameworks, which are specific due to their scale and focus. Enterprise frameworks are much more complex and try to embrace many aspects and therefore their development is hard. The paper also analyzes different approaches for building frameworks. The approach how to build enterprise frameworks is described in detail.
Keywords: object-oriented software frameworks, classification, patterns, components, reuse, enterprise frameworks

1 UVOD

V zadnjih letih je pri razvoju programske opreme zelo narasla potreba po ponovni uporabi. Podjetja si ne morejo več privoščiti, da bi informacijske sisteme vedno znova gradila od začetka, saj je to predrago in nekonkurenčno. Napredek v nivoju ponovne uporabe je predstavljal uveljavitev objektne in komponentne tehnologije. Problem knjižnic razredov in komponent pa je v tem, da v večini primerov pri ponovni uporabi uporabimo le programsko kodo, nimamo pa pravih informacij o tem, kako, zakaj in na kakšne zahteve je bila zgrajena ta komponenta ipd. Da bi še izboljšali stopnjo ponovne uporabe informacijskih sistemov, je bil uveden koncept objektno orientiranih ogrodij. Ogrodja so večji gradniki kot komponente in jih nekateri uvrščajo med (že) na pol narejene aplikacije. Ogrodja so namenjena uporabi v več aplikacijah. V splošnem velja, da lahko z ogrodji še izboljšamo ponovno uporabo pri gradnji novih aplikacij [1, 2, 3].

Kljub temu, da so ogrodja prisotna že precej časa, še vedno velikokrat niso najbolje razumljena. Prispevek poskuša narediti korak naprej pri razumevanju ogrodij, predvsem pa želi jasno opredeliti koncept ogrodja in postaviti ločnico glede na ostale tehnike ponovne uporabe. Analizirali smo obstoječe klasifikacije [3, 4, 5, 6, 7], ki se večinoma osredinjajo na posamezne vidike, naš namen pa je bil oblikovati predlog celovite klasifikacije ogrodij. Posebna pozornost v prispevku je namenjena razumevanju organizacijskih ogrodij, ki so posebna vrsta ogrodij in se od ostalih razlikujejo predvsem po obsegu in namenu.

V nadaljevanju povzamemo zgodovinski razvoj ogrodij, v drugem poglavju so predstavljeni osnovni koncepti ogrodij ter primerjava ogrodij z drugimi tehnikami ponovne uporabe, v tretjem poglavju je podan

predlog celovite klasifikacije ogrodij, v četrtem pa so predstavljena organizacijska ogrodja in pristopi k njihovi izgradnji.

1.1 Zgodovinski razvoj ogrodij

Za prvo ogrodje, ki se je začelo uporabljati na več področjih, velja *Model-View-Controller* (MVC). MVC je objektno orientirano ogrodje za razvoj uporabniškega vmesnika. Razvito je bilo v začetku osemdesetih let in je uporabljeno v jeziku Smalltalk-80. Podjetje Apple Inc. je v tistem času razvilo ogrodje MacApp, ki je prav tako ogrodje za razvoj uporabniškega vmesnika, namenjeno pa je gradnji aplikacij za računalnike Macintosh.

Med pomembnejša ogrodja, razvita v devetdesetih, spadajo CommonPoint (množica ogrodij za hitrejši razvoj aplikacij), HotDraw (ogrodje za izgradnjo grafičnih urejevalnikov, napisano v jeziku Smalltalk), ACE (*ADAPTIVE Communication Environment* – objektno orientirano ogrodje, namenjeno za komunikacijsko programsko opremo), JAWS (*Adaptive Web Server* – spletni strežnik in ogrodje za izgradnjo drugih vrst strežnikov) in verjetno eno najbolj uporabljenih ogrodij za platformo Windows MFC (*Microsoft Foundation Classes*). Ogrodje MFC je bilo ob ogrodju OWL (*Object Windows Library*) kar nekaj časa de facto industrijski standard za razvoj grafičnih aplikacij na osebnih računalnikih. V devetdesetih so se pojavila številna nova ogrodja tudi na drugih področjih, kot so multimedija, operacijski sistemi in sistemi za kontrolo procesov [8].

Eno od največjih težav pri uporabi ogrodij je predstavljalo pomanjkanje standardov na področju ogrodij. Tega so se zavedali tudi v večini najpomembnejših podjetij, kar je prispevalo k temu, da veliko novejših ogrodij ni produkt zgolj enega podjetja, temveč pri njihovi izgradnji prek različnih delovnih skupin sodeluje več podjetij.

Med najpomembnejša ogrodja nove generacije prav gotovo spadajo programske arhitekture COM/DCOM (*Component Object Model/Distributed Component Object Model*), CORBA (*Common Object Request Broker Architecture*), EJB/RMI (*Enterprise JavaBeans/Remote Method Invocation*) in Microsoft .NET.

Opazen napredek je vzpodbudil tudi nastanek in uveljavitev programskega jezika java. Večina ogrodij za javo namreč nastaja znotraj delovnih skupin v procesu JCP (*Java Community Process*) [9], ki ga sicer upravlja podjetje Sun, vendar pa v njem poleg korporacije Microsoft sodelujejo skorajda vsa pomembnejša podjetja. Med najpomembnejša ogrodja, ki so v okviru procesa JCP, sodijo poleg že omenjenih EJB in RMI še AWT (*Abstract Window Toolkit*), Swing/JFC (*Java Foundation Classes*), Java Servlet, JSP (JavaServer Pages), JSF (JavaServer Faces), Collection Framework, JMF (*Java Media Framework*), JAF (*JavaBeans Activation Framework*), še veliko več pa jih je v izgradnji. Glede na to, da java postaja eden najpomembnejših programskih jezikov za razvoj aplikacij, lahko najdemo številna ogrodja za javo tudi zunaj procesa JCP. Najpomembnejše, sedaj že precej uveljavljeno in preverjeno ogrodje je prav gotovo IBM SanFrancisco (sedaj *IBM Business Components for WebSphere Application Server*), ki poleg splošnih poslovnih objektov vsebuje še domensko specifične poslovne objekte. Vse več je tudi ogrodij, ki temeljijo na odprti kodi. Primeri takšnih ogrodij so Struts, Avalon in JCorporate Expresso.

V zadnjem času je za uporabo ogrodij zelo pomembno tudi ogrodje Microsoft .NET. S pojavom številnih jezikov, ki jih definira to ogrodje, predvsem s pojavom jezika C#, je nastalo kar nekaj novih ogrodij. Eno od najbolj znanih je ogrodje Windows Forms, ki je namenjeno za gradnjo oken v okolju Windows in predstavlja alternativo zgoraj omenjenemu Swingu. Ostala pomembnejša ogrodja v Microsoft .NET so ASP.NET, ADO.NET, .NET Web Services in BizTalk.

Številna podjetja uporabljajo opisana ogrodja in ob tem gradijo še svoja. Ta ogrodja se uporabljajo večinoma zgolj interno za razvoj drugih produktov in niso namenjena prodaji.

Že iz opisa ugotovimo, da ogrodja naslavljajo različne domene in namene. Da bi lažje identificirali primerna nam ogrodja, je smiselno opredeliti nove kriterije in klasifikacije, ki olajšajo izbiro.

2 DEFINICIJE IN KONCEPTI

2.1 Definicija ogrodja

Kaj sploh so ogrodja? V praksi lahko naletimo na številne produkte, ki so označeni kot ogrodje (*framework*). So vsi ti produkti tudi v resnici ogrodja? Zdi se, da je beseda ogrodje modna beseda in jo je koristno prilepiti k opisu produkta. Tako so nekateri produkti velikokrat ogrodja zgolj na papirju. Vsaka knjižnica razredov namreč še ni ogrodje; podobno se tudi pojmovanje ogrodij razlikuje od pojmovanja komponent in vzorcev.

Najbolj znana definicija ogrodij je iz leta 1988, avtorja pa sta Ralph Johnson in Brian Foote [4]:

"Ogrodje je množica razredov, ki vključuje abstraktni načrt rešitve za družino povezanih problemov."

Po našem mnenju so eno najustreznejših definicij objektno orientiranega ogrodja podali avtorji knjige *Design Patterns* (1995) [10]:

"Ogrodje je množica sodelujočih razredov, ki sestavljajo ponovno uporabljen načrt za specifično vrsto programske opreme. Ogrodje določa arhitekturne smernice z razdelitvijo načrta v abstraktne razrede in z definiranjem njihovih odgovornosti in sodelovanj. Razvijalec prilagaja ogrodje za posamezno aplikacijo z uporabo podrazredov in povezovanjem primerkov razredov ogrodja."

Ogrodje torej ni konkretna aplikacija, temveč predstavlja skelet za aplikacije, ki bodo zgrajene z uporabo tega ogrodja. Določene so osnovne funkcije, ki so skupne za več aplikacij, specifične funkcije za določeno aplikacijo pa dopolnijo uporabniki ogrodja sami – torej razvijalci informacijskih rešitev.

2.2 Primerjava ogrodij z drugimi tehnikami ponovne uporabe

Ogrodja so v današnjem času praviloma objektno orientirana tehnika ponovne uporabe. Imajo lastnosti, ki so skupne vsem tehnikam ponovne uporabe. Vse tehnike neki problem razbijejo na manjše enote, ki so bolj razumljive, obvladljive in ponovno uporabne v drugih situacijah. Cilji ponovne uporabe naj bi bili med drugim večja produktivnost, zmanjšanje stroškov in boljša kakovost.

Čeprav se ogrodja v praksi uporabljajo že nekaj časa, še posebej med objektno orientiranimi razvijalci, jih mnogo ljudi še vedno ne razume najbolje in jih zato napačno uporablja. Velikokrat je ogrodje opredeljeno kot vrsta vzorcev ali zgolj kot posebna vrsta komponent oziroma knjižnic razredov.

2.2.1 Ogrodja in komponente

Obstaja več definicij komponent. Večina avtorjev komponente obravnava kot del programske opreme z natančno določenim vmesnikom, implementacija te komponente pa je za uporabnike skrita [11]. Podobno kot za komponente velja tudi za ogrodja: ta ograjujejo podatke in funkcionalnost z natančno definirano množico vmesnikov, ki infrastrukturi ogrodja daje stabilnost in robustnost.

Z integracijo množic abstraktnih razredov in definiranjem standardnih načinov za sodelovanje med primerki teh razredov ponujajo ogrodja ponovno uporabne komponente za aplikacije. Na splošno komponente niso popolnoma samostojne, saj so običajno odvisne od funkcionalnosti, ki jih ponujajo druge komponente v ogrodju. Zbirke teh komponent tvorijo delno implementacijo, tj. skelet aplikacije. Ta skelet lahko prilagajamo z uporabo dedovanja ali z uporabo primerkov ponovno uporabnih komponent iz ogrodja.

Ker proizvajalci prodajajo ogrodja kot produkte, lahko tudi ogrodja štejemo za komponente. V aplikacijo lahko vključimo več komponent in več ogrodij. Vendar so ogrodja bolj razširljiva kot komponente, definiran pa imajo tudi bolj kompleksen vmesnik. Razvijalci morajo te vmesnike spoznati, preden začnejo uporabljati ogrodja, ker je naknadno spoznavanje novega ogrodja precej težko. Dobra lastnost ogrodij je, da so zelo zmogljiva in jih lahko uporabljamo za skoraj vse vrste aplikacij. Dobro ogrodje lahko v veliki meri zmanjša trud, ki ga je treba vložiti v izgradnjo razširljivih aplikacij [12].

Ogrodja in komponente sta torej različni, a kljub temu komplementarni in sodelujoči tehnologiji.

2.2.2 Ogrodja in knjižnice razredov

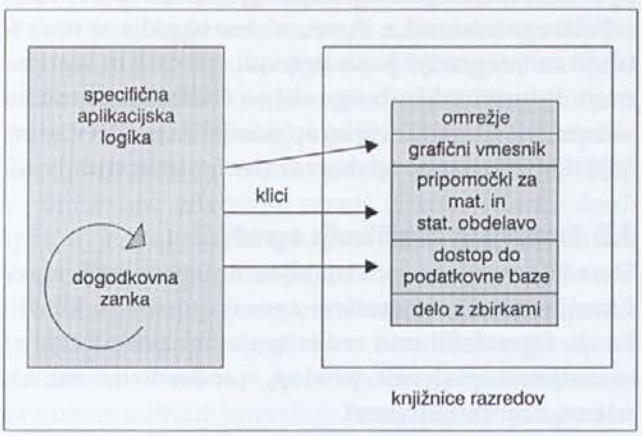
Ogrodja v praksi najpogosteje zamenjujemo s knjižnicami razredov, saj na prvi pogled med njimi ni večjih razlik. Tako kot knjižnice razredov vidijo razvijalci tudi ogrodja kot množico razredov, zato se pogosto zgodi, da uporabniki v resnici uporabljajo ogrodje, govorijo pa o "knjižnici razredov".

Ogrodja se od knjižnic razredov razlikujejo po tem, da ponujajo ponovno uporabo na višjem nivoju abstrakcije in granulacije. Pri knjižnicah razredov lahko v mnogih primerih ponovno uporabimo zgolj en razred, pri ogrodjih pa moramo najprej precej časa nameniti spoznavanju ogrodja, ponavadi spoznavanju cele množice medsebojno povezanih razredov. V večini primerov namreč nekega razreda iz ogrodja ne moremo ponovno uporabiti neodvisno od drugih. Velikokrat je knjižnica razredov ogrodje, če so med komponentami znotraj nje odvisnosti in če se programerji, ki jo spoznavajo, soočajo z veliko kompleksnostjo.

Knjižnice razredov ponujajo relativno majhno zrnatost (*granularity*) ponovne uporabe. Kot prikazuje slika 1, so npr. razredi tipično nizkonivojske, relativno neodvisne in splošne komponente, kot npr. pripomočki za matematično in statistično obdelavo, delo z

zbirkami, razredi za delo z omrežji, dostop do podatkovne baze. V nasprotju s tem komponente v ogrodju sodelujejo in tako ponujajo razširljiv arhitekturni skelet za družino povezanih aplikacij. Kot prikazuje slika 2, to zmanjšuje količino aplikacijsko specifične kode, saj je precej domensko specifičnega procesiranja že vključenega v splošne komponente ogrodja. Za razliko od knjižnic razredov za ogrodja velja, da definirajo napol gotove aplikacije, ki vključujejo domensko specifične objektne strukture in funkcionalnost. Lahko bi rekli, da imamo pri knjižnici razredov praviloma ponovno uporabo zgolj na nivoju programske kode, medtem ko pri ogrodjih ponovno uporabimo vsaj še zasnovo oz. načrt [13].

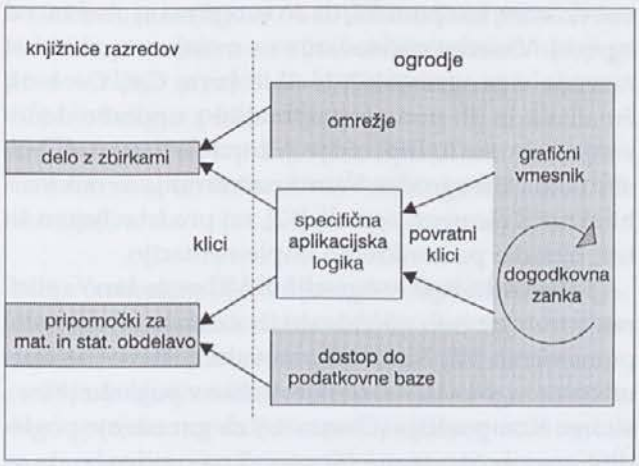
Obseg ponovne uporabe ogrodij je lahko precej večji kot pri uporabi tradicionalnih knjižnic razredov [1, 2, 3].



Slika 1: **Arhitektura knjižnice razredov**

Programiranje, ki temelji na uporabi ogrodij, zahteva nov način razmišljanja. Knjižnice razredov so večinoma "pasivne", kar pomeni, da nimajo glavne kontrole nad tokovi izvajanja aplikacije in izvajanje programa kontrolira koda določene aplikacije (slika 1). Za ogrodja pa velja, da so "aktivna", kar pomeni, da se pri izvajanju aplikacije običajno kontrola izvajanja prenese na ogrodje, ki potem po potrebi kliče programsko kodo za določeno aplikacijo. Takšno arhitekturo, ki temelji na povratnih klicih, prikazuje slika 2. Gre za obrat kontrole (*inversion of control*), ki je največkrat opisan kot "hollywoodsko načelo": ne kličite nas, mi bomo poklicali vas [14].

Tudi ogrodja in knjižice razredov sta komplementarni tehnologiji. Ogrodja pri izvajanju programske



Slika 2: **Arhitektura aplikacijskega ogrodja**

kode uporabljajo knjižnice razredov. Uporaba je lahko interna (kot del ogrodja) ali pa eksterna (prek povratnih, aplikacijsko specifičnih klicev).

2.2.3 Ogrodja in vzorci

Vzorci (*patterns*) so postali popularen način ponovne uporabe informacij o načrtovanju, kar še zlasti velja za objektno orientirano skupnost. Vzorec opisuje problem, rešitev in kontekst, v katerem rešitev deluje. Opisuje tudi tehnike uporabe, stroške in pridobitve. Razvijalci, ki si delijo množico vzorcev, imajo skupen slovar za opisovanje njihovih načrtov. Vzorci naj bi opisovali ponavljajoče se rešitve, ki so nadčasovne, torej vedno aktualne [12].

Ogrodja, ki so bila implementirana večkrat, predstavljajo neke vrste apliciranje določenega vzorca. Tako je npr. Bushmann [12] ogrodje Model-View-Controller opisal kot vzorec. Še več: aplikacije, ki uporabljajo ogrodja, se morajo prilagoditi načrtu ogrodja in modelu sodelovanja v tem ogrodju, tako da so ogrodja tudi razlog za uporabo vzorcev v aplikacijah. Vendar pa pri ogrodjih ne gre le za ideje, temveč tudi za programsko kodo. Če razvijalec razume ogrodje, mu ta koda ponuja način testiranja, primere za spoznavanje ogrodja ipd. Ponovna uporaba te kode omogoča hiter razvoj enostavne aplikacije, ki lahko, s tem ko razvijalec spoznava ogrodje, preraste v "pravo" aplikacijo.

Vzorci, opisani v knjigi *Design Patterns* [10], so z ogrodji tesno povezani na drug način. Ti vzorci so bili odkriti s preiskovanjem več ogrodij in izbrani za tipične primere ponovno uporabne objektno orientirane programske opreme. V ogrodju je običajno apliciranih

več vzorcev, kar pomeni, da so vzorci manj obsežni od ogrodij. Vzorcev načrtovanja ne moremo izraziti kot razrede v programskih jezikih java, C#, C++ ali Smalltalk in jih ponovno uporabiti z uporabo dedovanja oziroma kompozicije. Vzorci so torej tudi bolj abstraktni kot ogrodja. Vzorci načrtovanja so mikroarhitekturni elementi ogrodij [12], saj predstavljajo rešitev, ogrodja pa konkretno implementacijo.

Tako lahko npr. v ogrodju MVC zasledimo apliciranje treh glavnih načrtovalskih vzorcev in več manj pomembnih [12]. Vzorec Opazovalec (*Observer*) se uporablja za zagotovitev aktualnosti slike v pogledu (*View*), vzorec Kompozicija (*Composite*) za gnezdenje pogledov, vzorec Strategija (*Strategy*) pa omogoča, da se odgovornost za upravljanje z uporabniškimi dogodki v pogledu delegira na nadzornika (*Controller*). Podobno so številni načrtovalski vzorci iz knjige GoF [10] uporabljeni tudi v javanskih ogrodjih, ki jih je mogoče najti v razvojnem paketu JDK (*Java Development Kit*). Še zlasti veliko vzorcev lahko najdemo v ogrodju Swing.

Ogrodja so torej ena izmed tehnik ponovne uporabe. So bolj abstraktna in fleksibilna kot komponente, vendar bolj konkretna in lažja kot čisti načrt za ponovno uporabo. Čeprav bi jih lahko šteli za konkretno obliko vzorcev, pa ogrodja predstavljajo tehniko, pri kateri se hkrati ponovno uporabita načrt in programska koda, in so torej neke vrste aplikacijski generatorji oziroma predloge. Vzorci so ponazorjeni s programi, ogrodja pa so programi.

3 KLASIFIKACIJA OGRODIJ

3.1 Obstoječe klasifikacije ogrodij

Čeprav obstaja veliko objektno orientiranih ogrodij, ni bilo do sedaj veliko narejenega na področju klasifikacije ogrodij. V literaturi lahko najdemo le nekaj klasifikacij, pomembnejše so opisane v nadaljevanju.

Prvo klasifikacijo ogrodij sta naredila leta 1988 Johnson in Foote [4]. Ogrodja sta klasificirala glede na način, kako jih razširjamo. Tako sta definirala ogrodja v obliki bele škatle (*whitebox frameworks*) in ogrodja v obliki črne škatle (*blackbox frameworks*). Ogrodja v obliki bele škatle uporabljajo za razširjanje dedovanje in dinamično povezovanje, ogrodja v obliki črne škatle pa za razširjanje uporabljajo definiranje vmesnikov za komponente, ki jih lahko vključujemo v ogrodje z uporabo kompozicije. Fayad, Schmidt in Johnson [3] so leta 1999 razširili klasifikacijo z ogrodji v obliki sive škatle. Ogrodja v obliki sive škatle so načrtovana tako, da se izognemo slabostim, ki jih imajo ogrodja v obliki bele oz. črne škatle. Nekateri avtorji [5] so dodali še ogrodja v obliki steklene škatle, v katere je možen vpogled, spreminjati pa jih ne moremo.

Leta 1994 so v podjetju Taligent (zdaj IBM) ogrodja glede na področje delili v tri kategorije [6]: aplikacijska, domenska in podporna ogrodja. Aplikacijska ogrodja naj bi ponujala funkcionalnost, potrebno za različne aplikacije, npr. uporabniški vmesnik, dostop do baze, ipd. Domenska ogrodja so uporabljena le v eni domeni, npr. bančništvo, zavarovalništvo. Podporna ogrodja naslavljajo domene, kot so porazdeljeno računalništvo, dostop do datotek ipd.

Fayad in Schmidt [7] sta leta 1997 ogrodja glede na področje razdelila v sistemsko infrastrukturna, povezovalna in organizacijsko aplikacijska ogrodja. Sistemsko infrastrukturna ogrodja poenostavljajo razvoj prenosljive in učinkovite sistemske infrastrukture, to so npr. komunikacijska ogrodja, ogrodja za razvoj uporabniškega vmesnika. Povezovalna ogrodja se uporabljajo za integracijo porazdeljenih aplikacij in komponent. Primeri takšnih ogrodij so CORBA, sporočilni sistemi ipd. Organizacijsko aplikacijska ogrodja naslavljajo domene, kot so telekomunikacije, proizvodnja itd.

3.2 Predlagana klasifikacija ogrodij

V nadaljevanju bo predstavljen nov predlog h klasifikaciji ogrodij, ki razširja zgoraj omenjene klasifikacije. Opredelili smo več kriterijev klasifikacije, ki so razširljivost, področje, pristop, standardiziranost, zrnatost, licence in format.

3.2.1 Vrste ogrodij glede na razširljivost

Pri predlagani klasifikaciji glede na razširljivost so združene nekatere zgoraj opisane obstoječe klasifikacije. Tako imamo glede na razširljivost *ogrodja v obliki bele, črne, sive* in *steklene škatle.*

3.2.2 Vrste ogrodij glede na področje

V predlagani klasifikaciji smo kot osnovo vzeli klasifikacijo, ki sta jo predlagala Fayad in Schmidt, nato pa smo naredili nekaj sprememb. Organizacijsko aplikacijska ogrodja delimo v dve vrsti ogrodij: domenska in organizacijska.

Domenska ogrodja vsebujejo znanja o določeni domeni in ne pokrivajo nobenega drugega vidika.

Organizacijska ogrodja so posebna vrsta ogrodij. Od ostalih ogrodij se razlikujejo predvsem po obsegu in osredinjenosti. Glede na druga ogrodja so po obsegu

večja in bolj kompleksna, in za razliko od drugih poskušajo zaobjeti več vidikov, tako infrastrukturnega kot tudi domenskega, poseben poudarek pa je tudi na arhitekturi. Organizacijska ogrodja so podrobneje opisana v četrtem poglavju.

Kot dodatno smo dodali še eno vrsto ogrodij – poslovno-sodelovalna ogrodja. Ta ogrodja omogočajo elektronsko poslovanje in integracijo. Poskušajo premagati ovire tehnologije EDI (*Electronic Data Interchange*), ki je bila zelo draga in omejena na uporabo le v velikih podjetjih. Večina poslovno-sodelovalnih ogrodij temelji na uporabi jezika XML (*eXtensible Markup Language*). Primer takšnega ogrodja je ebXML (*Electronic Business using eXtensible Markup Language*).

Glede na področje smo definirali torej pet vrst ogrodij: *sistemsko infrastrukturna, povezovalna, domenska, organizacijska* in *poslovno sodelovalna ogrodja.*

3.2.3 Vrste ogrodij glede na pristop h gradnji

Z razvojem objektne tehnologije so se začeli uporabljati novi pristopi h gradnji ogrodij. Objektno orientirana analiza in načrtovanje sta vodila k objektnim ogrodjem (npr. MVC, MFC, Swing), s pojavom komponent pa so se pojavila komponentna ogrodja (npr. DCOM, EJB).

Storitveno orientiran razvoj (*Service-oriented development*) je naslednji korak v tem razvoju [15]. Medtem ko prejšnji pristopi temeljijo na uporabi vmesnikov objektov in komponent, pa storitveno orientiran razvoj temelji na uporabi storitev. Storitev je pogodbeno definirano obnašanje, ki je lahko implementirano in ponujeno s strani katerekoli komponente za uporabo v katerikoli komponenti. Pri klasičnih projektih tipa strežnik/odjemalec so implementacije odjemalca in strežnika v veliki meri odvisne ena od druge, implementacija storitve pa naj bi bila neodvisna od implementacije odjemalca. Primera storitveno orientiranih ogrodij sta Jini in Microsoft .NET.

Pred časom je bilo kot nova paradigma v razvoju programske opreme predstavljeno aspektno orientirano programiranje (AOP) [16]. AOP definira nov programski konstrukt, imenovan aspekt (*aspect*), ki ga uporabljamo za zajetje vseh tistih delov, ki se razpredajo po celotnem sistemu, in jih želimo povezati v ločene programske entitete. Tako komponente v aplikacijah ohranijo svojo odgovornost, obnašanje, ki je bilo prej razpredeno po celotnem sistemu, pa je sedaj omejeno le na en aspekt. Aspektno orientirana ogrodja podpirajo delitev komponent in aspektov na različnih nivojih. Gre za tridimenzionalni model, ki je sestavljen iz komponent, aspektov in nivojev. Komponente ponujajo osnovno funkcionalnost, nivoji pa predstavljajo aspekte in komponente, razdeljene v bolj obvladljive podprobleme.

Glede na pristop h gradnji ločimo torej *objektna, komponentna, storitveno orientirana* in *aspektno orientirana* ogrodja.

3.2.4 Vrste ogrodij glede na standardiziranost

Glede na standardiziranost ločimo *standardizirana, nestandardizirana* in *delno standardizirana ogrodja.*

Standardizirana ogrodja so tista, ki jih je kot standard sprejelo neodvisno mednarodno priznano telo za standardizacijo. Primer takšnega ogrodja je CORBA, ki ga je sprejelo združenje OMG (*Object Management Group*). Prednost standardiziranih ogrodij je, da ponujajo najrazličnejšim podjetjem skupna izhodišča, ki so v pomoč pri razvoju informacijskih sistemov. Verjetno pa je največja pomanjkljivost standardov, da se večinoma sprejemajo zelo dolgo.

Nestandardizirana ogrodja so tista, ki so bila največkrat narejena le v okviru enega samega podjetja in se običajno najhitreje odzovejo na spremembe na tržišču. Ena njihovih največjih pomanjkljivosti je, da so večinoma nestabilna. Primer takšnih ogrodij so Microsoftovi MFC, COM in DCOM in IBM SanFrancisco.

Delno standardizirana ogrodja so tista, ki nastanejo kot produkt neke delovne skupine, v katero je vključena večina najpomembnejših proizvajalcev programske opreme. Gre za kompromis med načinom dela različnih teles za standardizacijo in načinom dela podjetij, ki sama postavljajo standarde. Primer delno standariziranih ogrodij so javanska ogrodja podjetja Sun, sprejeta v okviru procesa JCP: EJB, Swing/JFC, Collection Framework, JAF, JSP, Java Servlet in JavaServer Faces.

3.2.5 Vrste ogrodij glede na zrnatost

Glede na zrnatost ločimo *drobnozrnata* in *grobozrnata ogrodja.*

Drobnozrnata ogrodja so v splošnem manjša z manj funkcionalnosti in kot takšna so lažja za implementacijo in ponovno uporabo. Grobozrnata ogrodja so v splošnem večja, sestavljena iz več komponent in vzorcev, imajo kompleksne vmesnike in so težja za implementacijo in ponovno uporabo.

3.2.6 Vrste ogrodij glede na licenco

V zadnjem času prosta programska oprema, še posebej odprtokodna programska oprema, vse bolj pridobiva

na pomenu. Vse več podjetij prepoznava prednosti uporabe proste programske opreme.

Glede na licenco lahko ogrodja razdelimo na *komercialna* in *nekomercialna* ogrodja.

Komercialna ogrodja so tista, za katera moramo kupiti licenco. Primeri takšnih ogrodij so implementacije modela CORBA ter IBM-ov SanFrancisco.

Nekomercialna ogrodja lahko uporabljamo brezplačno. Mednje spada večina ogrodij, sprejetih v okviru procesa JCP. Nastaja pa tudi vse več ogrodij znotraj različnih projektov za odprto kodo (*open source*). Najbolj znana tovrstna ogrodja so JUnit, ogrodja Avalon, Struts in Cocoon v okviru Apachevega projekta Jakarta ter Expresso podjetja JCorporate.

3.2.7 Vrste ogrodij glede na format

Butler Group je predlagal klasifikacijo komponent glede na format [17]. Mi pa smo razširili to klasifikacijo, da je uporabna za ogrodja.

V vsaki fazi razvojnega cikla lahko obstaja ogrodje v različnih oblikah. Lahko obstaja kot *logična specifikacija, fizični načrt, izvorna koda* ali *binarna oz. izvršljiva koda*. Lahko je uporabljena oz. ponovno uporabljena v kateremkoli od teh formatov, seveda odvisno od cilja, ki ga želimo doseči.

Logična specifikacija predstavlja celovit opis tega, kar zagotavljajo storitve ogrodja skupaj s kakršnimikoli omejitvami glede na to, kako naj bi ogrodja te storitve zagotavljale. Primer takšnih specifikacij so specifikacije ebXML.

Fizični načrt vključuje popolne specifikacije razredov, vmesnikov in metod, ki definirajo ogrodje. Neko podjetje lahko naredi fizični načrt, medtem ko drugo podjetje naredi implementacijo tega načrta. Primer takšnega fizičnega načrta je specifikacija za Java Servlet, ki je definirana v procesu JCP, implementirana pa s strani fundacije Apache v produktu Tomcat.

Večina ogrodij je dostavljenih kot izvršljiva koda, kar je tudi format za večino komercialnih ogrodij.

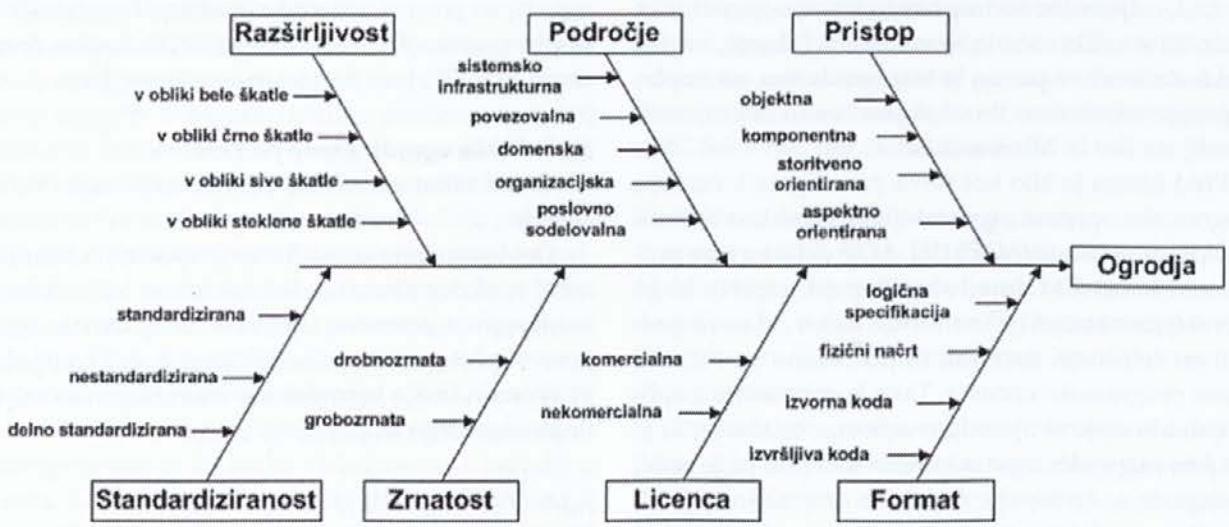
Nekatera ogrodja (npr. odprtokodna ogrodja) so dostavljena skupaj z izvorno kodo.

Slika 3 prikazuje predstavljeno klasifikacijo ogrodij. Poudariti velja, da pri klasifikaciji ne gre za medsebojno izključevanje kategorij, temveč se predvsem odražajo različne dimenzije ogrodij.

Da lahko neko ogrodje uvrstimo v več kategorij, dokazujejo organizacijska ogrodja. Organizacijska ogrodja ne zaobjemajo le enega, temveč več vidikov, zaradi česar je tudi razvoj takšnih ogrodij nekoliko drugačen od razvoja klasičnih ogrodij. Naslednje poglavje podrobneje predstavlja organizacijska ogrodja in pristop k njihovi gradnji.

4 Organizacijska ogrodja

Organizacijska ogrodja (*Enterprise Frameworks*) so posebna vrsta ogrodij. Od ostalih ogrodij se razlikujejo predvsem po obsegu in osredotočenosti. Če jih primerjamo z ostalimi ogrodji, so organizacijska ogrodja po obsegu večja in bolj kompleksna, v njih pa so lahko vsebovani najrazličnejši objekti, komponente in



Slika 3: **Predlog celovite klasifikacije ogrodij**

tudi druga ogrodja. Kar se osredotočenosti tiče, ostala ogrodja večinoma pokrivajo le en poseben vidik, bodisi domenski vidik (npr. finančne transakcije v spletni trgovini), infrastrukturni vidik (npr. porazdeljenost in trajnost objektov) ipd. Posebnost organizacijskih ogrodij je, da poskušajo zaobjeti več vidikov, tako infrastrukturnega kot tudi domenskega, poseben poudarek pa je tudi na arhitekturi [18, 19]. Veliko organizacijskih ogrodij temelji na vzorcu organizacijsko komponentno ogrodje.

4.1 Vzorec organizacijsko komponentno ogrodje

V zadnjem času sta eni od najpomembnejših programskih arhitektur J2EE in .NET. Kljub določenim razlikam lahko med njima najdemo tudi precej podobnosti. Ena od njih je ta, da obe arhitekturi temeljita na skupnem arhitekturnem vzorcu, imenovanem organizacijsko komponentno ogrodje (*Enterprise Component Framework* oz. ECF) [20].

Vzorec ECF opisuje osnovne mehanizme za uporabo komponent v porazdeljenem okolju, predvsem storitve za medprocesno komunikacijo, transakcije in trajnost.

Vzorec vsebuje sedem vlog, ki medsebojno sodelujejo: *Odjemalec*, *Tovarniški namestnik*, *Oddaljeni namestnik*, *Kontekst*, *Komponenta*, *Vsebnik* in *Trajnostna storitev* (slika 4).

Odjemalec (*Client*) je katerakoli entiteta, ki zahteva storitve od komponente (*Component*) v ogrodju. Uporabnik ne komunicira neposredno s komponento, temveč prek dveh posrednikov, tovarniškega namestnika (*Factory Proxy*) in oddaljenega namestnika (*Remote Proxy*), ki delegirata uporabnikove zahteve h komponenti. Ta koncept posrednih klicov omogoča podporo naslednjima zelo pomembnima funkcijama:

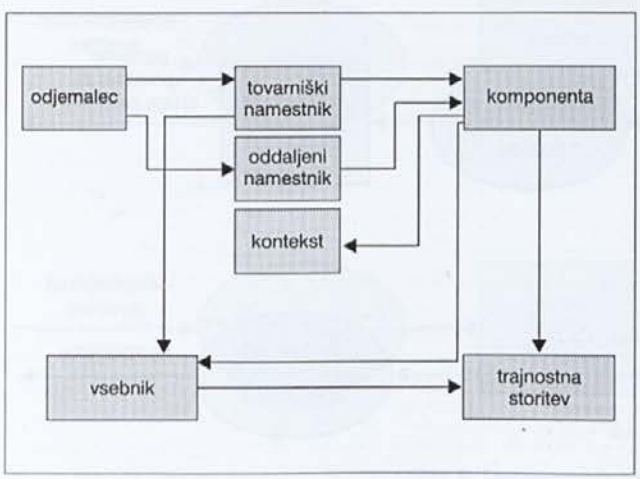
- *transparentnost lokacij*: na nižjem nivoju abstrakcije so lahko posredniki realizirani z uporabo koncepta *stub-skeleton*, ki se uporablja pri arhitekturah CORBA, Java RMI in COM+.
- *prestrezanje sporočil*: uporaba posrednikov omogoča, da okolje, v katerem se uporabljajo komponente, prestreza klice metod in zagotavlja storitve, ki temeljijo na množici atributov, ki so definirani v času izvajanja, in deklarativno opredeljujejo zahteve glede varnosti, transakcij itd.

Posrednika tovarniški namestnik in oddaljeni namestnik sta v bistvu dve izpeljanki vzorca namestnik (*Proxy*) [10] in kot taka predstavljata neke vrste podvzorec v vzorcu ECF. Tovarniški namestnik izvaja operacije za ustvarjanje objektov (npr. operaciji *create* in *find*), oddaljeni namestnik pa izvaja poslovne operacije, specifične za komponente (npr. metode *set* in *get* za lastnosti).

Komponente in oba posrednika se nahajata v vsebniku (*Container*). Vsebnik predstavlja izvajalno okolje za ogrodje, saj omogoča različne porazdeljene storitve, kot so medprocesna komunikacija, varnost, transakcije in trajnost. Vsebnik vsebuje tudi entiteto kontekst (*Context*), ki za vsako komponento vzdržuje specifične kontekstne informacije, kot so stanja transakcij, podatke o trajnosti in varnosti.

Trajnostne storitve (*Persistence Service*) za komponente (npr. shranjevanje in črpanje informacij iz podatkovne baze) se lahko koordinirajo s komponento samo, koordiniranje pa se lahko tudi delegira k vsebniku.

Vzorec ECF lahko ob J2EE in .NET uporabimo pri izgradnji drugih arhitektur. Vzorec ECF namreč vsebuje koncepte, ki so skupni mnogim poslovnim procesom.



Slika 4: **Vzorec ECF**

4.2 Pristop k razvoju ogrodij

Razvojni cikel programskih produktov je tipično sestavljen iz faz, kot so zajemanje zahtev, analiza, načrtovanje, implementacija, itd. Organizacijska ogrodja so svojstveni produkti, saj pokrivajo ostale programske produkte (komponente sistema) in tako je razvojni cikel organizacijskega ogrodja prepleten z razvojnimi cikli ostalih produktov. Najbolje bi bilo, če bi nam uspelo ločiti organizacijsko ogrodje in njegov razvojni cikel od drufih produktov in njihovih razvojnih ciklov. Gledano na široko je organizacijsko ogrodje skupek domensko specifičnih vidikov (domenska funkcionalnost) in domensko neodvisnih vidikov.

Najbolje bi bilo, če bi lahko te vidike obravnavali ločeno karseda dolgo in jih povezali šele zelo pozno. Vendar pa je ločevanje teh vidikov precej težko in prav to predstavlja enega ključnih faktorjev za učinkovito ponovno uporabo organizacijskih ogrodij. Razvoj organizacijskih ogrodij temelji na poznanih pristopih k razvoju ogrodij, ki so predstavljeni v nadaljevanju.

Za razvoj ogrodij se uporabljajo različni načini, žal pa še vedno nimamo splošno sprejetega in uveljavljenega načina razvoja ogrodij. V splošnem lahko rečemo, da se pojavljata dva pristopa [19]:

- analitični pristop (*top-down*): ogrodje razvijamo z izvajanjem procesa domenskega inženirstva, pri čemer začnemo z analizo domene, načrtovanjem, realizacijo itd.,
- sintetični pristop (*bottom-up*): začnemo z razvojem aplikacije znotraj domene ogrodja in potem vpeljujemo točke variabilnosti.

4.2.1 Ogrodja kot produkt domenskega inženirstva

Domensko inženirstvo je množica aktivnosti, katerih namen je razviti ponovno uporabne produkte iz realnega primera v funkcionalni domeni. Različne metode domenskega inženirstva imajo lahko različne rezultate ali predlagajo različne hevristike, vendar se bolj ali manj vse opirajo na te aktivnosti:

- izvajanje neke vrste analize različnosti oz. podobnosti z namenom, da identificiramo tiste vidike, ki so skupni za vse aplikacije znotraj domene in da identificiramo tiste vidike, ki ločijo neko aplikacijo od druge aplikacije,
- izpeljava vedno bolj konkretnih opisov za te vidike, pri čemer začnemo z opisi na nivoju analize, potem pa se spuščamo vse niže do opisov programske kode.

Proces domenskega inženirstva se izvaja inkrementalno, saj gre za kompleksne postopke, ki lahko trajajo precej časa. Prav tako je zaželeno, da se ogrodje oz. arhitektura domene testira dokaj zgodaj, preden gremo v širšo implementacijo komponent.

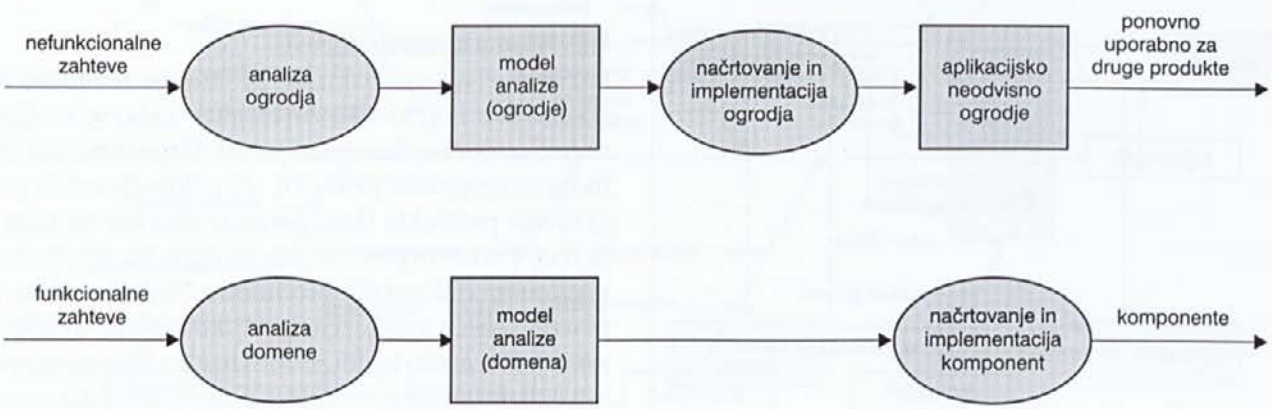
Katere vidike naj obravnavamo najprej? Splošno mnenje je, da moramo začeti s tistimi vidiki, ki imajo največji vpliv na arhitekturo. Tako naj bi se v prvem inkrementu ukvarjali s tistimi funkcijami, ki za delovanje zahtevajo večji del ali celotno infrastrukturo, kasneje bi dodajali nove funkcije, vendar le-te ne bi smele zahtevati nove infrastrukture.

Če npr. želimo razviti ogrodje za spletno bančništvo, potem bodo z vidika infrastrukture pomembni naslednji vidiki poslovnih aplikacij:

- porazdeljenost: aplikacijska logika naj bi bila lokacijsko transparentna, kar pomeni, da moramo uporabljati porazdeljeno infrastrukturo,
- varnost: komunikacija prek mreže mora biti varna, kar zahteva upravljanje z uporabniki, avtorizacijami, certifikati, enkripcijo ipd.,
- transakcijske storitve: uporabljati moramo transakcijske monitorje, ki bodo podpirali porazdeljene transakcije.

Ker so vsi ti vidiki povezani, mora, če želimo razviti ogrodje za spletno bančništvo, prvi inkrement vsebovati najmanjšo množico funkcij, ki za delovanje zahtevajo implementacijo zgoraj omenjenih vidikov.

Slika 5 prikazuje implementacijo aplikacijsko neodvisnega ogrodja, ki je lahko ponovno uporabna. Aplikacijsko neodvisno ogrodje je bilo implementirano



Slika 5: **Implementacija aplikacijsko neodvisnega ogrodja**

pred načrtovanjem in implementacijo domenskih komponent.

4.2.2 Ogrodja kot produkt razvoja aplikacij

Pri tem pristopu razvijamo ogrodja iz podmnožic aplikacij v domeni ogrodja. Tudi tukaj se ogrodje razvija inkrementalno in vzporedno z razvojem aplikacij.

Proces se prične z identifikacijo domene ogrodja in z identifikacijo zahtev za prvo aplikacijo. Implementacija aplikacije se kasneje razvije v prvi inkrement ogrodja, kjer implementiramo prve točke variabilnosti (t.i. *hotspot*). Točke variabilnosti so tisti vidiki, ki se pogosto spreminjajo od ene aplikacije do druge. Potem, ko smo končali z razvojem prve iteracije ogrodja, začnemo z razvojem naslednje druge aplikacije, čemur sledi druga iteracija razvoja ogrodja z novimi točkami variabilnost, sledi tretja aplikacija, pa tretja iteracija ogrodja itd.

V tipičnem inkrementu se konkretni razred (točka variabilnosti) iz iteracije N v iteraciji N+1 zamenja z abstraktnim razredom ali vmesnikom in konkretnimi podrazredi.

V določenih primerih lahko točke variabilnosti obravnavamo tudi z uporabo parametrov.

Slika 6 prikazuje ogrodje, ki je bilo implementirano hkrati z implementacijo domenskih komponent. V tem primeru je končno ogrodje precej v manjši meri ponovno uporabno in je specifično zgolj za to aplikacijo oz. morebiti še za kakšne aplikacije iz te domene.

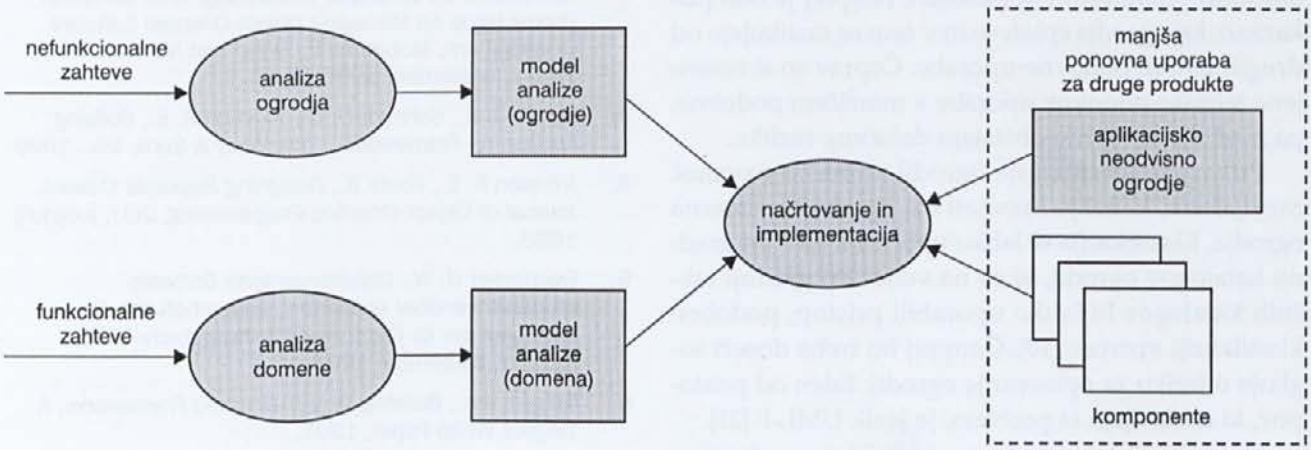
4.3 Razvoj organizacijskih ogrodij

Kot že prej omenjeno, so organizacijska ogrodja posebna vrsta ogrodij. Tako tudi zanje velja, da jih je mogoče razvijati s katerim od dveh zgoraj opisanih pristopov (analitični oz. sintetični pristop). Kateri je primernejši za razvoj organizacijskih ogrodij, če sploh kateri?

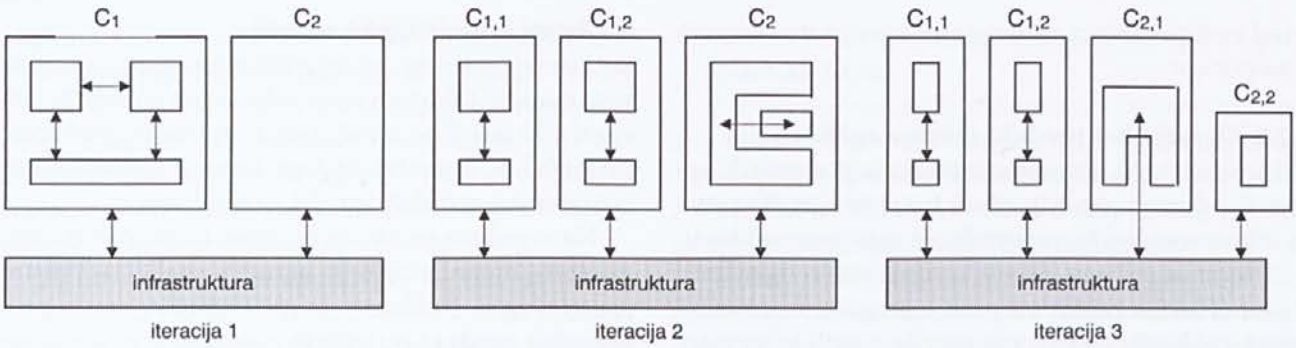
Na eni strani se zdi, da je zaradi truda, potrebnega za razvoj organizacijskega ogrodja, analitičen pristop precej tvegan. Veliko truda gre namreč v razvoj programskih produktov, katerih uporabnost in stroškovna učinkovitost ni zagotovljena. Samo dejanski projekti, ki uporabljajo ogrodje, lahko validirajo arhitekturo in komponente. Na drugi strani pa se zdi potratno priti do arhitekture skozi iteracije, saj je potrebno kar nekaj časa, preden se arhitektura izoblikuje. Sprememba arhitekture v poznih iteracijah je lahko zelo problematična, saj ima lahko to velik vpliv na že delujoče aplikacije.

Verjetno je najboljši hibridni pristop [19], ki združuje lastnosti obeh, tako analitičnega kot tudi sintetičnega pristopa. Arhitekturni vidiki ogrodja bi morali biti implementirani s centraliziranim analitičnim pristopom, pri čemer začnemo z načrtovanjem in nadaljujemo vse do implementacije. Posebno pozornost moramo posvetiti ločitvi infrastrukturnih vidikov od funkcijskih vidikov. Funkcijski vidiki arhitekture bodo razviti v iteracijskem in inkrementalnem načinu. Slika 7 prikazuje ta hibridni pristop.

Infrastruktura je razvita v prvi iteraciji, kjer se izvede omenjeni analitični pristop, začenši z zahtevami, sledi analiza, načrtovanje in dejanska implementacija



Slika 6: **Implementacija aplikacijsko specifičnega ogrodja**



Slika 7: **Življenjski cikel organizacijskih ogrodij**

infrastrukture. Kot dodatek k infrastrukturi implementiramo določene aplikacijske komponente, ki so na začetku še dokaj grobe in nedodelane, šele v naslednjih iteracijah pa se pravilno izoblikujejo, prečistijo in po potrebi tudi ločijo. Če na sliki 7 pogledamo komponento C_1, vidimo, da ima le-ta svojo interno strukturo in svoje interakcijske mehanizme. V drugi iteraciji se komponenta preoblikuje in tako podkomponenti $C_{1,1}$ in $C_{1,2}$ sedaj komunicirata prek skupne infrastrukture. Na primer, komponenta C_1 bi lahko bil obstoječi sistem, ki imajo svojo "lokalno" arhitekturo. V prvi iteraciji naredimo most (*bridge*) za uporabo tega obstoječega sistema, v naslednjih iteracijah pa preoblikujemo podkomponente obstoječega sistema, pri čemer večamo fleksibilnost infrastrukture, predvsem pa podpiramo izmenljivost komponent.

5 SKLEP

V prispevku je bila predstavljena uporaba ogrodij v objektno orientiranih aplikacijah. Najprej je bilo prikazano, kaj ogrodja sploh so in v čem se razlikujejo od drugih tehnik ponovne uporabe. Čeprav so si omenjene tehnike ponovne uporabe v marsičem podobne, pa med njimi vseeno obstajajo določene razlike.

Prikazane klasifikacije ogrodij so lahko v pomoč razvijalcem, ki želijo razvijati oz. uporabiti ustrezna ogrodja. Klasifikacija se lahko uporablja tudi pri gradnji katalogov ogrodij, ki so na voljo. Pri gradnji takšnih katalogov bi lahko uporabili pristop, podoben klasifikaciji vzorcev [10]. Čimprej bo treba doseči soglasje o jeziku za opisovanje ogrodij. Eden od pristopov, ki naslavljajo ta problem, je jezik UML-F [21].

Organizacijska ogrodja so arhitekturno orientirana ogrodja, ki poleg infrastrukture pokrivajo še del funkcionalnih zahtev aplikacij znotraj neke domene. Zaradi velikosti in kompleksnosti organizacijskih ogrodij je potrebna posebna pazljivost pri njihovem razvoju. Obstaja več pristopov k razvoju ogrodij, vendar žal še nobeden od njih ni splošno sprejet in uveljavljen.

Ogrodja so že in zagotovo bodo tudi v prihodnosti eno od najpomembnejših področij v razvoju informacijskih sistemov. Podjetja se vedno pogosteje odločajo za uporabo ogrodij. Ogrodje je zelo težko zgraditi, vendar lahko na to gledamo kot na strateško investicijo, ki se bo poplačala z uporabo ogrodja pri več aplikacijah.

6 LITERATURA

1. Morisio M., Romano D., Stamelos I., *Quality, Productivity and Learning in Framework-Based Development: an Exploratory Case Study*, IEEE Transactions on Software Engineering, vol 28, n.8, avgust 2002, pp. 340-357,

   http://morserv.polito.it/papers/tse-fwk-200_27.pdf.

2. Moser S., Nierstrasz O., *The effect of object-oriented frameworks on developer productivity*. IEEE Computer Theme Issue on Managing Object-Oriented Software Development, Mohamed E. Fayad and Marshall Cline, editors, september 1996:45-51.

3. Fayad M. E., Schmidt D. C., Johnson R. E., *Building Application Frameworks*, John Wiley & Sons, Inc., 1999.

4. Johnson R. E., Foote B., *Designing Reusable Classes*. Journal of Object-Oriented Programming, 2(1), junij/julij 1988.

5. Eisenecker U. W., *Objektorientierte Software wiederverwendbar entwerfen*, Sonderheft mit den Beiträgen der GI-Fachtagung Softwaretechnik 96, Koblenz, september 1996.

6. Taligent Inc., *Building Object-Oriented Frameworks*, A Taligent White Paper, 1994.

7. Fayad M. E., Schmidt D. C., *Object-Oriented Application Frameworks*, Communications of the ACM, Vol. 40, No. 10, oktober 1997.

8. Mattsson M., *Object-oriented Frameworks - A survey of methodological issues*, Department of Computer Science, Lund University, CODEN: LUTEDX/(TECS-3066)/ 1-130/(1996), http://www.ipd.hk-r.se/michaelm/thesis/index.html.
9. Sun, *The Java Community Process Program*, http://java.sun.com/aboutJava/communityprocess/.
10. Gamma E., Helm R., Johnson R., Vlissades J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995.
11. Krajnc Andrej, *Komponentni razvoj informacijskih sistemov*, FERI Maribor, marec 1999.
12. Johnson R. E., *Frameworks = Components + Patterns*, Communications of the ACM Special Issue on OO Application Frameworks, ACM, Vol. 40, No. 10, oktober 1997.
13. Landin N., Niklasson A., *Development of Object-Oriented Frameworks*, CODEN:LUTEDX(TETS-5231)/1-146, Department of Communication Systems, Lund University, 1995, http://www.tts.lth.se/Personal/bjornr/Papers/OOFW.ps.
14. Schmidt D. C., *Applying Design Patterns and Frameworks to Develop Object-Oriented Communication Software*, 1997, http://www.cs.wustl.edu/čschmidt/PDF/HPL.pdf.
15. Bieber G., Carpenter J., *Introduction to Service-Oriented Programming (Rev 2.1)*, http://www.openwings.org/download/specs/ ServiceOrientedIntroduction.pdf.
16. Kiczales G., Lamping J., Mendhekar A., Maeda C., Lopes C., Loingtier J-M., Irwin J., *Aspect-Oriented Programming*. In Proceedings of ECOOP'97, Volume 1241 of LNCS. Pages 220—242. Springer Verlag, junij 1997.
17. *CBD Reference Model Part 2, What are Components? - Concepts and Classification, Version 0.95*, Butler Group, marec 1998.
18. Mili H., Fayad M., Brugali D., Hamu D. S., Dori D., *Enterprise frameworks: issues and research directions*. Software - Practice and Experience Volume 32, Number 8: 801-831 julij 2002.
19. Mili H., Mili A., *Lifecycles for Enterprise Frameworks*, Enterprise Frameworks: "Adequacies and Inadequacies", OOPSLA 2000 Workshop Submission.
20. Kobryn C., *Modeling components and frameworks with UML*, Communications of the ACM, Volume 43, Issue 10 (October 2000), 31—38.
21. Fontoura M., Pree W., Rumpe B., *The UML Profile of Framework Architectures*, Addison-Wesley, 2000.

▪

Andrej Krajnc je študent magistrskega študija na Fakulteti za elektrotehniko, računalništvo in informatiko v Mariboru. Njegovo raziskovalno delo obsega objektno tehnologijo, komponentni razvoj, ponovno uporabo, ogrodja, vzorce, standarde družine XML in tehnologije medorganizacijskega povezovanja. Diplomiral je na Fakulteti za elektrotehniko, računalništvo in informatiko v Mariboru.

▪

Marjan Heričko je izredni profesor na Inštitutu za informatiko na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru. Njegovo raziskovalno-razvojno delo obsega vse vidike objektne tehnologije in komponentnega razvoja, s poudarkom na metodologijah razvoja, metrikah in razvojnih okoljih. Svoje izkušnje je predstavil v številnih prispevkih na domačih in tujih konferencah ter v revijah. Je tehnični koordinator aktivnosti Centra za objektno tehnologijo ter predsednik konference OTS Objektna tehnologija v Sloveniji. Diplomiral, magistriral in doktoriral je na Fakulteti za elektrotehniko, računalništvo in informatiko v Mariboru.

▪