arvosana päiväys

arvostelija

Avoimet ohjelmistokehykset

Jyri Laukkanen

24.9.2008

HELSINGIN YLIOPISTO

Tietojenkäsittelytieteen laitos

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET

Tiedekunta/Osasto – Fakultet/Sektion – Faculty/Section	Laitos – Institution - Department
Matemaattis-luonnontieteellinen tdk	Tietojenkäsittelytieteen laitos

Tekijä – Författare - Author

Jyri Laukkanen

Työn nimi – Arbetets titel - Title

Avoimet ohjelmistokehykset

Oppiaine – Läroämne - Subject

Tietojenkäsittelytiede

Työn laji – Arbetets art - Level	Aika – Datum – Month and year	Sivumäärä – Sidoantal – Number of pages
Seminaari	24.9.2008	9

Tiivistelmä – Referat - Abstract

Tässä kirjoitelmassa esitellään avoimia ohjelmistokehyksiä yleisellä tasolla, niihin liittyvää käsitteistöä sekä otetaan esimerkkejä muutamista tunnetuista Java-pohjaisista web-ohjelmistokehyksistä kuten Strutsista ja Hibernatesta. Kirjoitelman loppuosassa keskitytään Spring Framework-ohjelmistokehyksen esittelyyn ja sitä tarkastellaan hieman tarkemmin. Lopuksi arvoidaan omakohtaisia kokemuksia kyseisten ohjelmistokehysten käytöstä kaupallisissa projekteissa.

Avainsanat – Nyckelord – Keywords

Ohjelmistokehykset

Säilytyspaikka – Förvaringställe – Where deposited

Muita tietoja – Övriga uppgifter – Additional information

Sisällys

1 Johdanto 1

2 Ohjelmistokehys 1

2.1 Abstraktit kehykset (abstract framework) 2

2.2 Muunneltavat kehykset (white-box framework) 3

2.3 Plugin-kehykset (plug-in framework) 3

2.4 Koottavat kehykset (black-box framework) 3

3 Avoimet ohjemistokehykset 4

3.1 Struts 4

3.2 Hibernate 5

4 Spring Framework 5

5 Yhteenveto 9

Lähteet 10

1 Johdanto

Tässä kirjoitelmassa tarkastelen yleisellä tasolla avoimia ohjemistokehyksiä. Koska ohjelmistokehykset ovat yleensä suunniteltu jollekin tietylle sovellusalueelle, niin keskityn tässä lähinnä Java–pohjaisiin web-sovellusalueen (J2EE) ohjelmistokehyksiin. Aluksi esittelen ohjelmistokehystä käsitteenä, erilaisia kehystyyppejä ja käyttökohteita ohjelmistokehityksessä. Seuraavaksi esittelen muutamia web-sovellusalueelle tarkoitettuja ohjelmistokehyksiä yleisellä tasolla. Lopuksi tarkastelen Spring Framework-ohjelmistokehystä hieman tarkemmin. Esittelen kehyksen perusarkkitehtuuria ja sen tarjoamia palveluita. Lopuksi teen yhteenvedon, jossa tuon esiin omiin kokemuksiini perustuvia näkemyksiä avointen ohjelmistokehysten käytöstä.

2 Ohjelmistokehys

Ohjelmoinnin historiassa erilaiset ohjelmointiparadigmat ovat syntyneet tarpeesta hallita ohjelmistokehitykseen liittyvää monimutkaisuutta. Ohjelmistojen koon kasvu on pakottanut kehittämään erilaisia lähestymistapoja, joissa toteutuksen monimutkaisuutta on pyritty hallitsemaan modularisoimalla ja abstrahoimalla. Sekä proseduraalinen ohjelmointi että olio-ohjelmointi ovat tämän kehityksen tuloksia. Varsinkin olio-ohjelmointi soveltuu hyvin ohjelmistojen rakenteen toteuttamiseen, koska oliomallinnuksella voidaan jäsentää ongelma-alueen toiminnallisuus olioista koostuvaksi loogiseksi kokonaisuudeksi ja olioiden väliseksi yhteistoiminnaksi.

Hyvin usein saman sovellusalueen ohjelmistot jakavat yhteisiä arkkitehtuurisia ominaisuuksia, jotka ovat joko suoraan yhteneväiset taikka ainakin hyvin lähellä toisiaan.

Tällöin on jo pelkästään uudelleenkäytettävyyden kannalta järkevää erottaa nämä yhteiset ominaisuudet erilliseksi osakseen. Koskimies ja Mikkonen määrittelevätkin olioperustaisen ohjelmistokehyksen kokoelmaksi luokkia, komponentteja ja rajapintoja, jotka toteuttavat jonkin ohjelmistojoukon yhteisen arkkitehtuurin ja perustoiminnallisuuden. He vertaavat ohjelmistokehystä oliopohjaisesti toteutettuun tuoterunkoon, jolloin ohjelmistokehyksillä on sama perustavoite kuin tuoteruingoilla, eli laajamitainen ja systemaattinen uudelleenkäyttö [KoM05, s.187].

Kuten jo edellä mainittiin ohjelmistokehykset tarjoavat jonkin tietyn sovellusalueen perusarkkitehtuurin- ja toiminnallisuuden. Tästä hyvänä esimerkkinä voidaan ottaa web-pohjaiset ohjelmistokehykset, joista hyvin useat tarjoavat palveluita näkymien (www sivut), business logiikan sekä tietovarastojen hallintaan. Useat näistä kehyksistä perustuvat MVC-malliin (Model-View-Controller), joka puolestaan sisältää joukon muita suunnittelumalleja [GHJ95, s. 4]. Tämän lisäksi ohjelmistokehykset tarjoavat vaihtelevan määrän laajennuspisteitä, joiden avulla kehyksen toimintaa voidaan muuttaa sovelluksen tarpeiden mukaisesti. Ohjelmistokehyksiä voidaankin jakaa ryhmiin niiden tarjoamien laajennusmekanismien mukaan. Koskimies ja Mikkonen jakavat kehykset seuraaviin ryhmiin: abstraktit kehykset, muunneltavat kehykset, plugin-kehykset sekä koottavat kehykset [KoM05, s. 194-202].

2.1 Abstraktit kehykset (abstract framework)

Abstraktit kehykset koostuvat rajapinnoista kuten ryhmän nimityskin kertoo. Ne eivät sisällä suoritettavaa koodia vaan määrittelevät rajapintojen avulla peruspalveluita. Esimerkkinä voidaan ottaa EJB-standardi, joka määrittelee joukon rajapintoja, mutta ei itsessään tarjoa valmista toteutusta [KoM05, s. 195-196].

2.2 Muunneltavat kehykset (white-box framework)

Muunneltavissa kehyksissä sovellus erikoistaa kehyksen tarjoamia laajennusrajapintoja ja luokkia. Tämä tapahtuu pääasiallisesti periyttämällä kehyksen luokkia sovelluksessa. Tämän tyyppiset kehykset käyttävät niin sanottua Hollywood-periaatetta, jossa kehys kutsuu sovelluksen koodia [KoM05, s.196-197]. Kyseisestä periaatteesta voidaan käyttää myös nimitystä IoC (Inversion of Control) [MFo04]. Tästä puhutaan lisää Spring Framework–kehyksen yhteydessä. Muunneltavista kehyksistä voidaan ottaa erimerkkeinä juurikin Spring Framework sekä Struts.

2.3 Plugin-kehykset (plug-in framework)

Plugin-kehyksissä sovellus toteuttaa kehyksen laajennusrajapintoja ja rekisteröi toteutuksen kehykseen. Koskimiehen ja Mikkosen mukaan tämä on selkeämpi tapa liittää sovelluksen laajennuksia kehykseen koska erikoistamisrajapinta on näin helpommin nähtävissä [KoM05, s. 198-199]. Hyvänä esimerkkinä plugin-kehyksestä voidaan ottaa Eclipse-alusta.

2.4 Koottavat kehykset (black-box framework)

Koottavat kehykset ovat tietyssä mielessä kirjastojen ja kehysten välimuoto. Näissä tarvittava muunneltavuus saadaan aikaan konfiguroimalla eikä laajentamalla kuten esimerkiksi muunneltavissa kehyksissä. Koskimaan ja Mikkosen mukaan koottavia kehyksiä voidaankin pitää muunneltavan kehyksen evoluution lopputuloksena [KoM05, s. 199-200].

3 Avoimet ohjemistokehykset

Tässä kappaleessa esitellään kaksi avointa Java-pohjaista ohjelmistokehystä, jotka ovat suosittuja web-sovellusten alustoina ja osaltaan olleet vaikuttamassa avoimien ohjelmistokehysten yleistymiseen.

3.1 Struts

Avoimien ohjelmistokehysten yleistyminen alkoi 2000-luvun alussa web-sovellusten tarpeista. Kehittäjät huomasivat niin sanotun JSP (Java Server Pages) Model-1 -suunnittelumallin heikkoudet. Tässä mallissa JSP-sivut käsittelivät sekä sovelluksen kontrollia, business logiikkaa että näkymien luontia [Joh05, s. 108]. Luonnollisestikin tämä johti usein erittäin sekaviin ja vaikeasti ylläpidettäviin sovelluksiin. Ratkaisuna Model-1 –suunnittelumallin heikkouksiin Apache julkaisi vuonna 2001 Struts-ohjelmistokehyksen, joka pohjautui arkkitehtuuriltaan jo aikaisemmin mainittuun MVC-malliin [Str08]. Struts kehyksen tarkoituksena oli erottaa malli eli business logiikka (model), näkymä eli JSP-sivut (view) sekä kontrollointi (controller) toisistaan.

Struts vakiinnutti nopeasti asemansa web-sovellusten pääasiallisena ohjelmistokehyksenä ja kehittäjät alkoivat suosia sitä J2EE-projekteissa. Struts kehyksen saavuttama suosio ja juurikin sen tunnettavuus kehittäjien keskuudessa helpotti henkilöstön rekrytoimista myös kaupallisiinkin projekteihin. Johnsonin mukaan tämä oli hyvä esimerkki avoimien ohjelmistokehysten hyödyistä [Joh05, s. 108]. Struts-kehystä on kehitetty aktiivisesti sen julkaisusta lähtien ja projekti kulkee tällä hetkellä nimellä Struts 2. Itseasiassa Struts 2 pohjautuu WebWork [Web08] nimiseen projektiin, joka irtautui alkuperäisestä Struts-kehityksestä. Projektit yhdistettiin uudestaan vuonna 2005 ja näin syntyi nykyinen Struts 2.

3.2 Hibernate

Edellä esitelty Struts-ohjelmistokehys tarjoaa hyvän alustan web-sovelluksen näkymien, business- sekä kontrollilogiikan hallintaan, koska Strutsin käyttämä MVC-malliin perustuva arkkitehtuuri soveltuu tähän erittäin hyvin. Struts ei kuitenkaan tarjoa riittävän tehokkaita työkaluja tiedon tallennukseen (persistence). Myöskään Sun Microsystemsin ajama EJB (Enterprise Java Beans) –määrityksen aikaisemmat versiot eivät ole tarjonneet tiedon tallennukseen kovinkaan joustavia menetelmiä. Osittain johtuen EJB-määrityksen kankeudesta alettiin etsimään muita ratkaisuja ja eräänä vaihtoehtona nousi esiin ORM (Object-Relational Mapping). Ensimmäinen suuren suosion saanut avoimeen lähdekoodiin perustuva ORM-ohjelmistokehys oli vuonna 2001 aloitettu Hibernate-projekti [Joh05, s. 108-109]. Hibernate tarjoaa tehokkaan tietokantariippumattoman ohjelmistokehyksen, jonka avulla sovellukset voivat tallentaa tavallisia Java-olioita (POJO, Plain Old Java Object) relaatiotietokantoihin [Hib08].

Hibernate on saavuttanut huomattavan suosion ja sitä käytetään hyvin paljon sekä Struts, että Spring Framework-ohjelmistokehysten yhteydessä. Hibernate-projektin kehittäjät ovat olleet mukana määrittelemässä EJB 3.0-standardia ja näin tuomassa ORM:ää myös standardin kautta tunnetummaksi [Hib08]. Mielestäni tämä on hyvä esimerkki siitä kuinka laadukkailla avoimen ohjelmistokehityksen tuotoksilla on vaikutusta myös kaupallisiin toimijoihin.

4 Spring Framework

Kuten edellä mainittiin Struts ja Hibernate-ohjelmistokehyksiä käytetään paljon yhdessä koska ne tarjoavat web-sovellukselle sekä MVC-mallin mukaisen rakenteen että helpon

tavan tiedon tallentamiseen. Kaikesta huolimatta ne eivät kuitenkaan tarjoa riittävästi toimintoja varsinaisen business logiikan tueksi. Myöskään EJB-standardi ei vastaa tähän tarpeeseen riittävän tehokkaasti [Joh05, s. 110]. Rod Johnsonin alunperin kehittämä, ja kirjansa Expert One-on-One J2EE Design and Development yhteydessä vuonna 2002 julkaisema Spring Framework on pyrkinyt täyttämään nämä tarpeet kuitenkaan sitomatta sovellusta kehyksestä aiheutuviin rajoituksiin [Spr08].

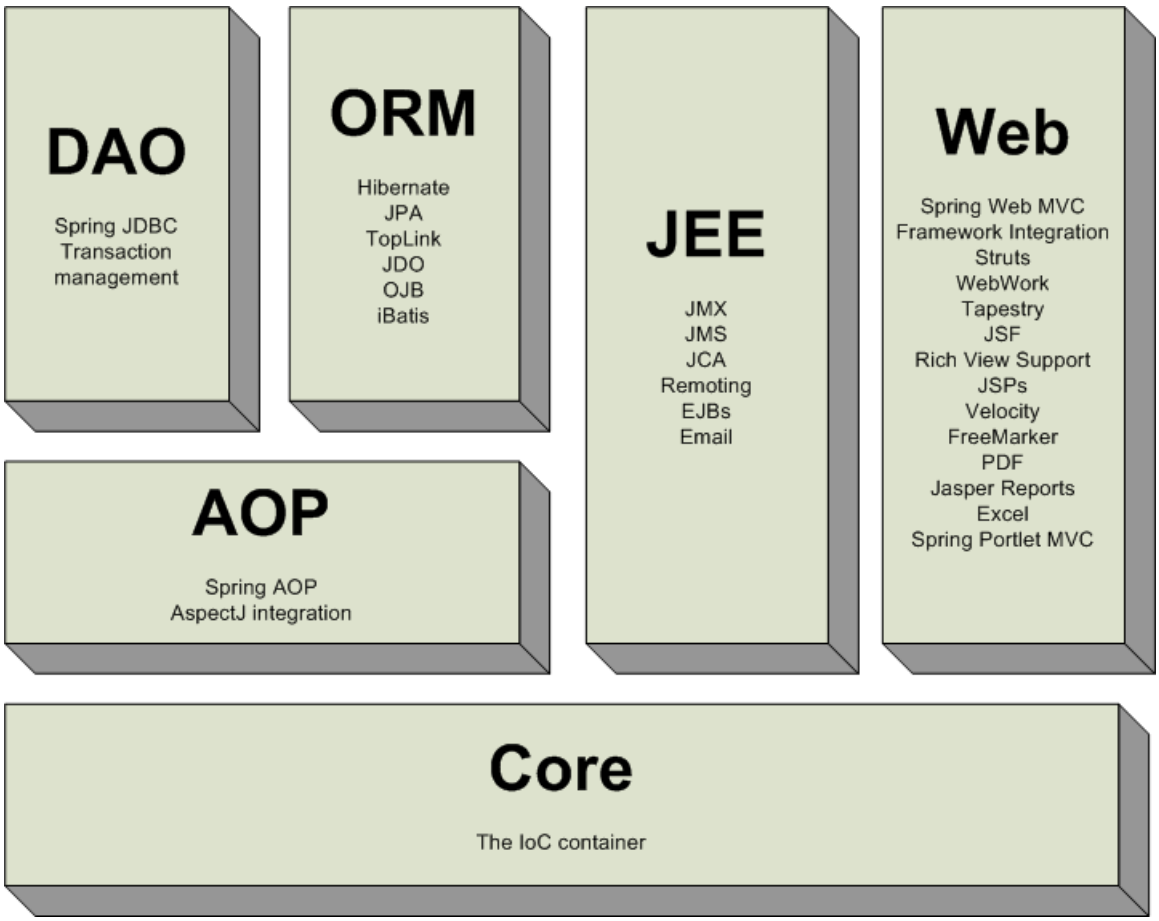
Spring Framework-ohjelmistokehyksen perusarkkitehtuuri pohjautuu niin sanottuun IoC (Inversion of Control) -säiliöön ja aspektiohjelmoinnin (AOP, Aspect-Oriented Programming) hyödyntämiseen. IoC-säiliö noudattaa toimintalogiikaltaan aikaisemmin mainittua Hollywood-periaatetta (Don’t call us, we’ll call you). Tällä tarkoitetaan sitä, että ohjelmistokehys kutsuu sovelluksen koodia eikä päinvastoin. Spring Framework-kehyksessä tähän liittyy olennaisesti myös niin sanottu riippuvuuksien syöttäminen (Dependency Injection) [MFow04], jolla tarkoitetaan eri luokkien ja komponenttien välisten riippuvuussuhteiden hallintaa. IoC-säiliölle määritellään komponenttien väliset riippuvuudet ja säiliö huolehtii sovelluksen käynnistyessä riippuvuuksien täyttämisestä. Tämä vapauttaa komponentit suorittamaan vain niille olennaisinta toimintalogiikkaa ja jättää muun ohjelmistokehyksen hoidettavaksi.

Toinen Spring Framework-ohjelmistokehyksen perusarkkitehtuurin kulmakivistä on AOP-toteutus. Usein perinteinen olio-ohjelmointi ei ole riittävän voimakas tarjoamaan työkaluja niin sanottujen läpileikkaavien ominaisuuksien hallintaan [Kic97, s. 1]. Tällöin on järkevää soveltaa aspektiohjelmointia, johon Spring Framework-kehys tarjoaa hyvät palvelut. Kehys tarjoaa täysin valmiin AOP-toteutuksen sekä liitynnän tunnetuimpaan AOP-toteutukseen AspectJ:hin [Spr08].

Spring Framework tarjoaa erittäin kattavan joukon abstraktioita jo standardeiksi muodostuneiden rajapintojen ja teknologioiden päälle. Näitä on esitelty kuvassa 1.

Hyvinä esimerkkeinä voidaan ottaa esimerkiksi JDBC- ja transaktioabstraktiot, jotka tekevät tietokantakäsittelystä huomattavan helppoa. Tämän lisäksi Spring Framework voidaan integroida jo edellä mainittuihin Struts- ja Hibernate-kehyksiin sekä lukuisiin muihin. Huomattavaa näissä abstraktioissa ja integroinneissa on kuitenkin se, että kehys ei itsessään pakota sovellusta mihinkään tiettyyn malliin tai arkkitehtuuriin.

Toisin kuten esimerkiksi Struts, Spring Framework ei ole pelkästään web-sovelluksille tarkoitettu ohjelmistokehys. Vaikka kehyksestä löytyy paljon J2EE-sovelluksille tarkoitettuja palveluja, kehystä voidaan käyttää hyvin erilaisten Java-sovellusten runkona. Itseasiassa web-sovellusten käyttämä Spring MVC on täysin irrallinen moduuli kuten kuvasta 1 huomataan. Spring Framework-kehyksestä voidaan siis koota kutakin sovellusta parhaiten palveleva ohjelmistokehys. Tämä lisää selvästi kehyksen käyttökohteita. Osittain juuri tästä syystä kehys onkin saavuttanut huomattavan suuren suosion Java-kehittäjien piirissä.



Kuva 1. Spring Framework-ohjelmistokehyksen korkean tason arkkitehtuuri [Spr08].

5 Yhteenveto

Avoimet ohjelmistokehykset ovat saavuttaneet suurta suosiota myös kaupallisissa projekteissa. Tämä ei ole yllättävää, koska kaupallisissa projekteissa halutaan minimoida time-to-market, jotta tuotto saadaan maksimoitua. Käyttämällä valmista avointa ohjelmistokehystä sovellukset voidaan toteuttaa huomattavasti nopeammassa tahdissa kuin muuten olisi mahdollista. Tämä on luonnollisesti myös laadun kannalta järkevää, koska monia palveluita ei tarvitse tehdä itse vaan voidaan hyödyntää valmiin ohjelmistokehyksen ominaisuuksia. Usein avoimet ohjelmistokehykset ovat myös hyvin testattuja, koska käyttäjäkunta on laajempi kuin yritysten sisäisesti kehittämillä suljetuilla kehyksillä ja sovelluksilla.

Itse olen tehnyt projekteja sekä Struts-, että Spring Framework-ohjelmistokehyksillä ja olen huomannut niiden tarjoaman selkeän hyödyn. Kun kehysten toiminnan ymmärtää tarpeeksi hyvin, on sovellusten kehittämien niiden päälle vaivatonta. Alussa oppimiskäyrä on tietenkin melko jyrkkä, mutta varsinkin yleisimmät kehykset, kuten Struts ja Spring Framework sekä Hibernate, ovat hyvin dokumentoituja ja niistä löytyy paljon kirjallisuutta. Tästä syystä itseopiskelu onnistuu helposti eikä vaadi välttämättä järjestettyä koulutusta.

Uskon, että avoimien sovelluskehysten käyttö tulee vain lisääntymään, koska projektien avoimuus itsessään ruokkii niiden menestystä. Laaja ja aktiivinen käyttäjäkunta takaa projektien laadun ja osaltaan pitää huolen siitä, että ne pysvät elinvoimaisina ja kehittyvät jatkuvasti.

Lähteet

GHJ95 Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Design Patterns - Elements of Reusable Object-Oriented Software. 1995.

Hib08 Hibernate-projektin kotisivu, http://www.hibernate.org/. 2008.

Joh05 Rod Johnson, J2EE development frameworks. *Computer*, January 2005. Volume 38, Issue 1, sivut 107-110.

Kic97 G. Kiczales et al., Aspect-Oriented Programming. *ECOOP'97, 220-242*. *Springer Lecture Notes in CS*, 1997

KoM05 Kai Koskimies, Tommi Mikkonen, Ohjelmistoarkkitehtuurit. Tampere 2005.

MFo04 Martin Fowler, Inversion of Control Containers and the Dependency Injection pattern. http://martinfowler.com/articles/injection.html. 2004.

Spr08 Spring Framework–projektin kotisivu, http://www.springframework.org/. 2008.

Str08 Struts-projektin kotisivu, http://struts.apache.org/index.html. 2008.

Web08 WebWork-projektin kotisivu, http://www.opensymphony.com/webwork/. 2008.