TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Arvutitehnika instituut

IAG70LT

Sander Tuulik 111613IASM

RAKENDUSTE FUNKTSIONAALSUSE TAGAMINE PLATVORMI VAHETUSEL

Magistritöö

Vladimir Viies

PhD

Dotsent

Tallinn 2015

Autorideklaratsioon

Olen koostanud antud töö iseseisvalt. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud. Käesolevat tööd ei ole varem esitatud kusagil mujal kaitsmisele.

Autor: Sander Tuulik

[12.01.2015]

Annotatsioon

Antud magistritöö eesmärgiks oli viia Liisingu infosüsteemi rakendus üle teenusorienteeritud arhitektuuril töötavale platormile. Töös uuriti erinevaid tarkvara arhitektuure, detailsemalt teenusorienteeritud arhitektuuri ning valiti välja tehnoloogia, millele rakendus üle viia. Loodi vorm, mis vastab antud arhitektuuri nõuetele ning mis on laialdaselt kasutatav erinevates seadmetes. Magistritöö on kirjutatud eesti keeles ning sisaldab teksti 69 leheküljel, 4 peatükki, 23 joonist, 2 tabelit.

Abstract

Ensuring Application Functionality After Software Platform Change

The aim of this Master's thesis is to find out on what technology meets best financial enterprise leasing system requirements to replace expired Oracle Forms application and to give practical instructions how to migrate application functionality on that technology.

Many enterprises are dealing with various different problems that are caused by expired Forms technology. Applications that use outdated Forms technology do not work with new web browser versions and are complex and most often impossible to interact with new services. These problems are main reason why platform change of leasing Forms application is needed.

To reach objective is necessary to understand service-oriented architecture principles. Because technologies that meet service-oriented principles are flexible and have high compatibility with most of other technologies.

Author investigated the following technologies: Oracle APEX, Oracle ADF, Java Spring framework and latest release of Oracle Forms.

In conclusion an LIS equipment description form and closely related forms where migrated into Oracle ADF. It is reusable for different companies that want to replace out dated Forms platform or even create a new application on ADF.

The thesis is in Estonian and contains 69 pages of text, 4 chapters, 23 figures, 2 tables.

Lühendite ja mõistete sõnastik

SOAP	*Simple Object Access Protocol*, lihtne objektipöördusprotokoll
XML	*Extensible Markup Language*, laiendatav märgistuskeel
LIS	Liisingu infosüsteem
SQL	*Structured Query Language*, struktuurpäringukeel
PL/SQL	*Procedural Language/SQL*
ADF	*Application Development Framework*
APEX	*Application Express*
SOA	*Service Oriented Architecture*, teenustepõhine arhitektuur
JSF	*JavaServer Faces*
HTML 5	*HyperText Markup Language* 5. versioon, millega märgendatakse veebilehti.
MVC	*Module-View-Controller*
CRUD	*Create, Reade, Update, Delete*
DBMS	*Database Management System*, andmebaasihaldur
CRM	*Customer Relationship Management*, kliendisuhete haldus, kliendihaldus
ERP	*Enterprise Resource Planning*, ettevõtte ressursside planeerimine

WSRP	*Web Services for Remote Portlet*
API	*Application Programming Interface*, rakendusliides
IDE	*Integrated Development Environment*, integreeritud programmeerimisekeskkond
iOS	Endise nimega iPhone OS, Apple firma mobiili operatsioonisüsteem.
JAAS	*Java Authentication and Authorization Service*
OPSS	*Oracle Platform Security Service*
Mbeans	*Java Management Beans*
Java EE	*Java Platform, Enterprise Edition*, Oracle ettevõtte Java platvorm.
HTML-DB	Oracle toode, tänapäeval kannab nime APEX
URL	*Uniform Resource Locator,* internetiaadress
J2EE	*Java 2 Platform, Enterprise Edition*, firma Sun Microsystems Java platvorm
HTTP	*HyperText Transfer Protocol*, hüperteksti edastusprotokoll
EPG	*Embedded PL/SQL Gateway*
JNDI	*Java Naming and Directory Interface*
EJB	*Enterprise JavaBean,* ettevõtte JavaBeans on Java rakendusliides firmalt Sun Microsystems
POJO	*Plain Old Java Object*
JDBC	*Java Database Connectivity*, java andmebaasipöördus

ORM	*Object-relational mapping*
AOP	*Aspect Oriented Programming*
ASCII	*American Standard Code for Information Interchange,* Ameerika informatsioonivahetuse standardkood
EBCDIC	*Extended Binary Coded Decimal Interchange Code,* laiendatud kahend- kümnend-infovahetuskood
JAX-RPC	*Java API for XML-based RPC*
JAXM	*Java API for XML Messaging*
PDF	*Portable Document Format*

Sisukord

Jooniste nimekiri

Tabelite nimekiri

1. Sissejuhatus

Tänapäeva tehnoloogia kiire areng ja selle tulemused tõstavad klientide ootusi pakutavale teenusele. Oodatakse, et pakutavad teenused oleksid kvaliteetsed, kiired ja kaasaegsed. Ettevõtted peavad tihedas konkurentsis suutma hoida olemasolevaid kliente ning samas leidma ka võimalusi, kuidas võita juurde uusi. Kõige selle saavutamiseks on vaja head infosüsteemi.

Parim lahendus kirjeldatud eesmärgi täitmiseks on kasutada teenusele orienteeritud arhitektuuri, mille kaasabil on võimalik luua paindlik ja hästi töötav süsteem.

Paljudel ettevõtetel on kasutusel süsteemid, mis töötavad aegunud Oracle Forms tehnoloogial. Aegunud Forms ei võimalda tarkvara kasutada uute veebilehitsejate versioonidega, samuti on keeruline liidestada uusi teenuseid olemasoleva süsteemiga. Kirjeldatud probleemidele lisaks on veel mitmeid teisi põhjuseid, miks on vaja platvormi uuendamist. Platvormi vahetamisel tuleb leida sobilik tehnoloogia, millele rakenduse funktsionaalsus üle viia. Eelistatud peavad tänapäeval olema tehnoloogiad, mis toetavad teenusorienteeritud arhitektuuri põhimõtteid.

Antud magistritöö eesmärgiks oli leida parim tehnoloogia, millele üle viia aegunud tehnoloogial töötav Forms rakendus ja anda praktilisi juhiseid, kuidas realiseerida kõnealust platvormi vahetust finantsasutuse liisingu infosüsteemi näitel. Tehnoloogia valiku võrdluses osalevad Oracle ADF, Oracle APEX, Java Spring ja Oracle Formsi uusim versioon.

Magistritöös püstitatud probleemiasetuse lahendamiseks on sõnastatud järgmised uurimisülesanded:

- Uurida, mis on rakendused, millised on neile esitatavad nõuded ja kuidas on nendega seotud tarkvara arhitektuur. Anda ülevaade teistest levinud tarkvara arhitektuuri tüüpidest ja võrrelda neid teenusorienteeritud arhitektuuriga ning uurida viimast detailsemalt.

- Analüüsida, millised on olulisemad erinevused vananenud Forms tehnoloogial ja teenusele orienteeritud arhitektuuril loodud rakenduste vahel.

- Analüüsida, mida pakub teenusorienteeritud arhitektuuri kasutusele võtmine ettevõtte erinevatele harudele: äri- ja IT poolele.
- Analüüsida erinevaid andmebaasi rakenduse loomise tehnoloogiaid, nende omadusi, arhitektuuri ja nõudeid ning pakuda välja sobivaim.
- Rakendada pakutud tehnoloogiat praktikas, liisingu infosüsteemi peal.

Antud magistritöö koosneb sissejuhatusest, neljast peatükist, kokkuvõttest ja kasutatud kirjanduse loetelust.

1. Rakenduste funktsionaalsuse uurimine

Rakenduse platvormivahetuse teostamisel on vaja teada, mis on põhikomponendid, millest rakendused koosnevad ning millised osad on vaja ümber tõsta.

1.1. Rakendused ja nõuded

Riistvara juhtimiseks vajab iga arvutisüsteem tarkvara. Tarkvara omakorda jaguneb kahte suurde kategooriasse: süsteemi- ja rakendustarkvaraks. Süsteemitarkvara koosneb juhtprogrammidest nagu operatsioonisüsteem ja andmebaasihaldurid (DBMS). Rakendustarkvara hulka kuuluvad kõik programmid, mis töötlevad kasutaja poolt ette nähtud andmeid (tekstitöötlus, tabelarvutus jne).[15]

Seega on rakendustarkvara kogum antud liiki ülesannete lahendamiseks või tegevuste täitmiseks arvutil. Rakendused koosnevad üldjuhul kolmest põhikomponendist, mis on omavahel tihedalt seotud: programmid, andmed ja kasutajaliides.

Programm

Programm on käskude (korralduste, instruktsioonide, lausete) kogum, mis määrab, milliseid tegevusi peab arvuti täitma andmete, objektidega või teenustega tagades tavaliselt ka kasutajaliidese töö.

Programmide koostamiseks kasutatakse spetsiaalseid programmeerimiskeeli ja -süsteeme. Igas keeles on piiratud valik käske ehk lauseid ning nende esitamiseks ja täitmiseks on kindlad reeglid. Programm võib koosneda mitmest üksusest. Erinevates keeltes kasutatakse nende jaoks erinevaid nimetusi: funktsioon, protseduur, skript. [24]

Andmed

Andmed on informatsiooni esitlus kujul, mis võimaldab seda säilitada, töödelda ja edastada programmi juhtimisega süsteemide abil. Andmed võivad olla nii teksti, numbri, faili jne kujul. Andmetetüübist sõltub esitlusviis arvuti seadmetes ja võimalikud operatsioonid ning tegevused nendega. [24]

Kasutajaliides

Kasutajaliides sisaldab vahendeid, mille abil kasutaja saab rakendusega suhelda – näha tulemusi, muuta algandmeid, anda vajalikke korraldusi jms. Erinevad programmeerimissüsteemid ja keskkonnad sisaldavad liideste loomiseks spetsiaalseid vahendeid. Tavaliselt tehakse kasutajaliides rakenduse loomise faasis, kuid programmis võivad olla võimalused liidese muutmiseks täitmise ajal. [24]

Rakendustele esitavad nõuded jagunevad kaheks - funktsionaalsed ja mittefunktsionaalsed nõuded. Mittefunktsionaalsed nõuded on kvaliteedinõuded, mis defineerivad teenusekvaliteedi. Mittefunktsionaalsed nõuded arendavad arhitektuuri ning funktsionaalsed nõudmised disaini. [26]

1.2. Tarkvara arhitektuurid

Tarkvara arhitektuuri mõiste selgitamiseks on mõeldud erinevaid definitsioone. Näiteks, tarkvara arhitektuur on süsteemi illustratsioon, mis aitab aru saada süsteemi käitumisest.[23]

Üks esimestest ja lihtsaim tarkvara arhitektuur oli ühekihiline arhitektuur. See arhitektuur oli tarkvara arenduses tavaline, kui ei eksisteerinud interneti. Rakendused suhtlesid andmebaasiga, mis asus samas masinas. Tavaliselt täitsid andmebaasi osa tarkvarad dBase, Paradox või FoxPro. Ühekihilisel arhitektuuril töötav tarkvara ei suhelnud teiste arvutitega ega jaganud teiste arvutite ja süsteemidega andmeid.

Selle järgnes periood kui tarkvara arenduses oli populaarne struktureerida rakendusi kaheks komponendiks: klient ja server. Sellist lähenemist tuntakse nimetuse järgi “kahekihiline arhitektuur“. Klient on komponent, mis vahendab kasutaja ja serveri vahel infot. Server osutab ühele või enamale kliendile teenuseid. Tüüpiliselt paiknes serveris tsentraalne andmebaas ja üks osa äriloogikast, klientides aga äriloogika ja kasutajaliides.[31] Klienti, kes sisaldab enamust rakenduse loogikast, nimetatakse paksuks kliendiks (*fat client / thick client*). Paks klient teeb ära suurema osa andmetöötlusest, sest vajalikud andmed, programmid või failid paiknevad samas

serveris. Tänu sellele, et andmed töödeldakse kohapeal, võimaldab see luua keerukamaid rakendusi ning vähendada serveri koormust.

Nii kahekihilise - kui ka ühekihilise arhitektuuri puuduseks on, et äriloogika muutudes on vaja tarkvara uuendada. Kahekihilises arhitektuuris on tarkvara vaja uuendada tihti kümnetes ja sadades kliendiarvutites. Samuti nõuab see suurte andmemahtude liigutamist kliendi ja serveri vahel ning suurt arvutusvõimsust kliendi poolel. [31]

Kahekihilises arhitektuuris peab andmebaas võimaldama rakendustele otse suhtlust kõikide klientidega. Kuigi tänapäevased andmebaasid võimaldavad kaitsta andmebaasi erinevate autentifitseerimisetega, on siiski andmebaasi jagamine kohaliku võrgu tasemel või üle interneti turvalisust vähendav.

1.3. Mitmekihiline arhitektuur

Tänapäeval on enamus süsteeme ülesse ehitatud mitmekihilisele arhitektuurile. Mitmekihiline arhitektuur (*n-tier client-server architecture*) on klient-server arhitektuur, kus esituse, töötlemise ja andmete haldamise protsessid on üksteisest loogiliselt eraldatud protsessid. Mitmekihiline arhitektuurimudel aitab luua paindlikku ja korduvalt kasutatavat tarkvara. Muudatuste puhul on vajalik need teha vaid üksikutes kihtides, mitte kogu rakenduses korraga. See lubab hakkama saada vähema töö, lühema aja ja väiksema potentsiaalse vigade hulgaga. [25]

Kolme kihiline arhitektuur

Mitmekihilise arhitektuuri tüüpilisemaks ja enam kaustatavamaks variandiks on kolmekihiline arhitektuur (*three-tier client server architecture*). Kolmekihilise rakenduse puhul paikneb iga kiht arvutivõrgus erinevas kohas ning võib paikneda ka erinevatel platvormidel. [25]

Esitlusloogika kiht on kasutajatele kõige lähemal, täpsemalt tema arvutis olev tööjaama tarkvara. See võib piirneda vaid sisestusvormidega ja platvormile tüüpilise graafilise kasutajaliidesega. Ei ole välistatud selle kihi olemasolu erinevatele platvormidele. Esitusloogika kiht suhtleb rakenduse kihiga (ka äriloogika kiht, keskmine kiht). [25] Esitlusloogika kihiti kannab mitmekihilises arhitektuuris nimetust peenike klient. Peenike või õhuke klient (*thin client*) on klient–server rakendus, kus kliendi eesmärk on

olla eriti väike, nii et suurem osa andmetöötlusest ehk enamik rakenduse loogikast toimub teistes kihtides.

Äri- ja firmaloogikakihi ülessanne on juhtida rakenduse funktsionaalsust, töödeldes selleks alumisest kihist saadud andmeid vastavalt kasutajalt ülemisest kihist tulnud päringutele. See kiht paikneb tavaliselt kohtvõrgu serveris. [25] Kasutajate lisandumisel ja koormuse kasvamisel on võimalik tõsta äriloogikat pakkuvate serverite arvu, vastavalt vajadusele.

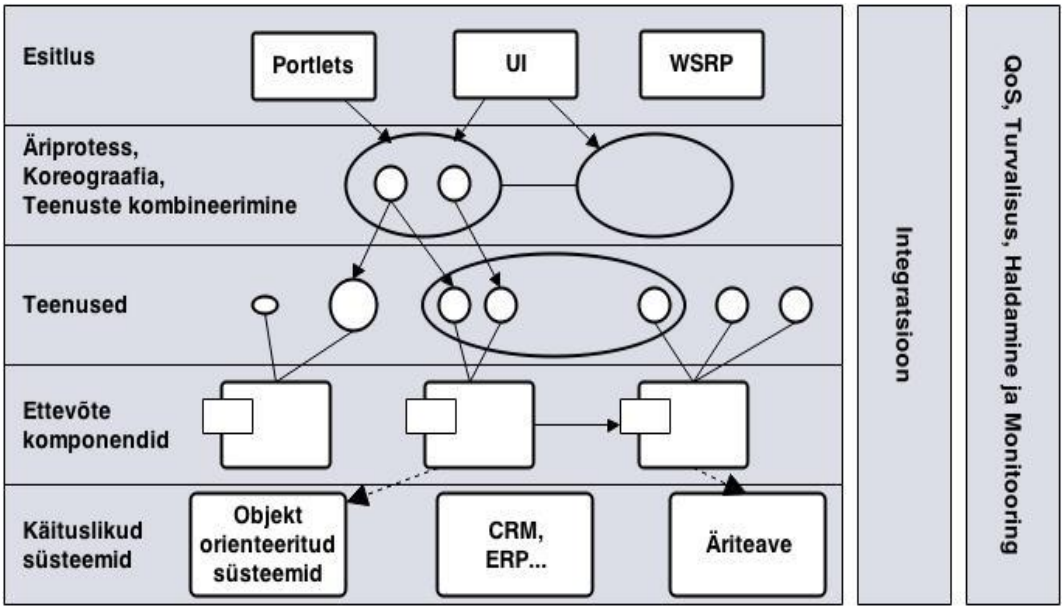
Kolmas kiht on andmekiht. Antud kiht sisaldab andmebaasi ning selle haldamiseks vajaliku tarkvara. Andmeid on vaja hoida selliselt eraldi kihil, et andmed oleks sõltumatud rakendusest ja esitlusloogikast. [25]

Kolmekihilise arhitektuuri kasutamine tõstab oluliselt andmebaasi turvalisust. Sest klient- rakendused ei suhtle otse andmebaasiga ning ei tea seetõttu andmebaasist midagi. Äriloogika kihi kasutamine tagab, et äriloogika asub kõik ühes kohas. Sellise struktuuri kasutamine muudab teenuse äriloogika muutmise lihtsamaks ja odavamaks.

Kolmekihilise arhitektuuri miinuseks on rakenduse loomise keerukuse ja sellest tulenevalt arenduse maksumuse tõus.

1.4. Teenusorienteeritud arhitektuur

Teenusorienteeritud arhitektuur (SOA - *Service Oriented Architecture*) on olemuselt mitmekihiline arhitektuur, mida kasutatakse veebirakenduste loomisel. Arhitektuur moodustub üksteisega suhtlevatest (veebi) teenustest, mis ei sõltu üksteise kontekstist ega olekust ning töötavad hajussüsteemide mudelil. Teenuste vahel puuduvad tugevad seosed ning ükski teenus pole teadlik teise teenuse tehnilistest detailidest nagu näiteks tarkvaraline platvorm, andmestruktuurid, operatsioonisüsteem, kasutatav andmebaasisüsteem jne. Teenuspõhist arhitektuuri võib käsitleda kui rakenduste väljatöötamise kõrgeimat tasandit, kus kasutajate eest on peidetud kogu tehnilise keskkonna keerukus. [27]



Joonis 1. Teenusorienteeritud arhitektuuri kihid.[17]

Igal SOA kihil on arhitektuuris erinev ülesanne:

Käitusliku süsteemi kiht: See kiht koosneb olemasolevatest rakendustest: CRM ja ERP pakendatud rakendused, objekt-orienteeritud süsteemid ja äriteabetarkvara. Nende rakenduste eesmärk on toetada teenuseid nii, et igal rakendusel on oma firmaomane struktuur, andmebaas ja juurdepääs teistele süsteemidele. [17]

Ettevõtte komponentide kiht: On komponendid, mis on spetsialiseerunud teenustele kindlate funktsioonide ja nõuete tagamisele. Antud komponendid moodustavad ärilise vara teenuse osutamiseks.

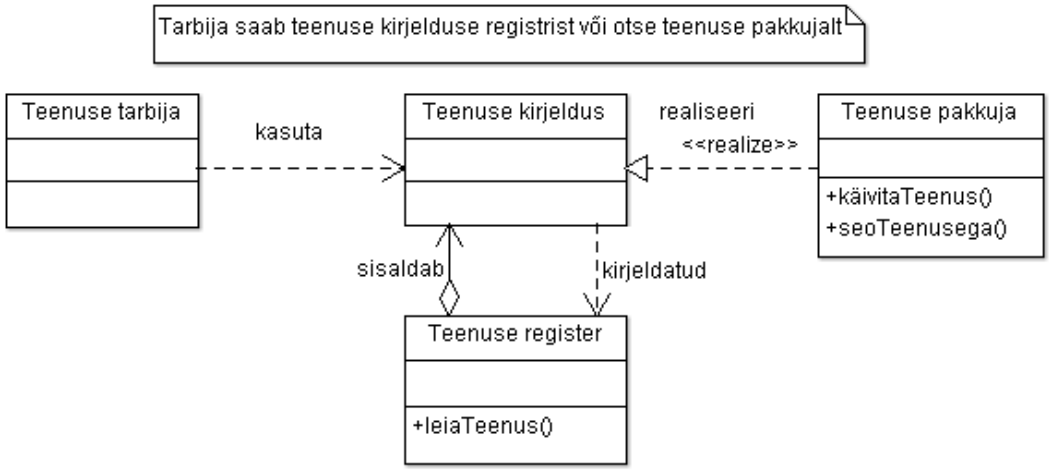
Teenuste kiht: See kiht koosneb teenustest, mis on nähtavad ja kasutatavad teiste rakenduste poolt. Teenused realiseerivad ja levitavad ettevõtte komponendi liideseid teenuse deskriptorina.

Kombineeritud teenuste kiht: Antud kihis saab erinevaid teenuseid kombineerida üheks rakenduseks. Teenused võivad toetada erinevaid kasutusjuhte ja äri protsesse.

Esitluskiht: Pakub kasutajaliidest teenustele ja kombineeritud rakendustele. Esitluskiht on teistest teenusele orienteeritud arhitektuuri kihtidest vähem olulisem, sest teenused saavad pakkuda kasutajaliidest kasutades standardseid tehnoloogiaid nagu näiteks portletid ja WSRP portleteid.

1.4.1. Teenusorienteeritud arhitektuuri kontseptuaalne mudel

SOA on arhitektuuri stiil, mis defineerib interaktsiooni mudelit kolme põhilise funktsionaalse üksuse vahel. Kus teenuse tarbija suhtleb teenuse pakkujaga, et leida läbi otsinguregistri teenus, mis kattub tarbija poolsete nõuetega. Interaktsiooni kirjeldav metamudel on näha alloleval joonisel.



Joonis 2. SOA arhitektuuri stiili kontseptuaalne mudel.[17]

SOA kontseptuaalne mudel koosneb kuuest olemist [11]:

Teenuse tarbija: On SOA olem, mis otsib teenuseid, et käivitada vajalikud funktsioonid. Teenuse tarbija võib olla rakendus, teine teenus või mõni muud tüüpi tarkvara moodul, mis vajab teenust. Teenuse asukoht tehakse kindlaks, kas registrist otsides või kui on teada, siis saab tarbija otse suhelda teenuse pakkujaga.

Teenusepakkuja: On võrgu adresseeritav olem, mis võtab vastu ja käivitab tarbijatelt saabuvaid käsklusi. Teenusepakkuja väljastab täpseid teenuse kirjeldusi ja kontrollib teenuse rakendamist. Teenusepakkuja võib olla komponent või muu tarkvara süsteem, mis täidab teenuse pakkuja ülesandeid.

Teenuse register: On kataloog, millele pääseb ligi üle võrgu ja mis sisaldab kasutatavaid teenuseid. Selle olemi põhieesmärk on ladustada ja jagada teenuste kirjeldusi teenusepakkujatelt teenuse tarbijatele.

Teenuseleping: On kirjeldus, mis ütleb kuidas peab toimima interaktsioon teenuse pakkuja ja teenuse tarbija vahel. Sisaldab informatsiooni: taotlus-vastus (*request-*

response) sõnumi formaadi kohta, tingimusi, mille korral teenus peaks käivituma ja teenuse kvaliteedi aspekte.

Teenuse puhver: On olem, mis toetab interaktsiooni teenusepakkuja ja teenuse tarbija vahel, pakkudes APIt, mis on kirjutatud tarbijale lokaalses keeles. API eesmärk on tõsta jõudlust ja pakkuda puhverdamise võimalusi. Teenuse puhvri kasutamine SOA arhitektuuris on valikuline.

Teenuse rent: Seda olemit juhitakse registri poolt ja selle eesmärk on täpsustada kaua teenuseleping on valiidne. Antud olemi kasutamine toetab lõtva sidusust teenuse pakkuja ja tarbija vahel ning teenuse staatuse informatsiooni säilitamist.

1.4.2. Teenusorienteeritud arhitektuuri omadused

Teenusorienteeritud arhitektuur peegeldab kindlaid põhimõtteid, mida tuleb rakendada kui soovitakse ehitada teenusele orienteeritud rakendust:

Teenused on leitavad ja dünaamiliselt seotud

Teenusele orienteeritud arhitektuur toetab teenuse leitavuse kontseptsiooni, teenused peavad olema käitusfaasis dünaamiliselt leitavad. Teenuse tarbija leiab vajaliku teenuse registrist ning saab kogu teenuse käivitamiseks vajaliku informatsiooni. Tarbijad ei vaja käitusaegset informatsiooni teenuse kohta. Teenuse liidesed leitakse ja sõnumid koostatakse dünaamiliselt. Käitusaegse sõltuvuse eemaldamine muudab paremaks käideldavust, sest tarbija ei vaja iga järgneva liidese muudatuse korral uut liidese sidumist. Teenuse tarbija ei ole enne teenuse käivitamist teadlik, taotluse- ja vastuse sõnumi formaadist ega teenuse asukohast.

Teenused on iseseisvad ja modulaarsed

Teenused on iseseisvad ja modulaarsed. Liidesed on omavahel seotud moodulite kaudu ja sisaldavad piisavalt informatsiooni, et olla eristatavad ilma sõltuvuseta teistest tarkvara moodulitest või rakendustest.

Teenused on koostöövõimelised

Teenusorienteeritud arhitektuuri eesmärk on tõsta koostöövõimelisust. Sellega tagatakse, et teenused oleksid võimelised omavahel suhtlema sõltumata platvormist või

keelest. Iga teenus pakub liidest, mida saab käivitada läbi konnektor tüübi. Koostöövõimeline konnektor koosneb potentsiaalse kliendi protokollist ja andmeformaadist. Koostöövõimelisuse saavutamiseks peab konnektor toetama kõikide olemasolevate ja potentsiaalsete klientide protokolle ja andmeformaate. Standardsete protokollide ja andmeformaatide toetamiseks on vajalik vastendada platvormi omadused ja programmeerimiskeeled vahendaja spetsifikatsiooniga. Vahendaja spetsifikatsioon vastendab omavahel koostöövõimelise andmeformaadid ja platvormi spetsiifilised formaadid. Näiteks mõnikord on vajalik vastendada ASCII märgistik UTF-8 märgistikuga, nagu vastendatakse omavahel erinevaid andmetüüpe. Veebiteenus on vahendaja spetsifikatsiooniks süsteemide vahelises suhtluses, JAX-RPC ja JAXM vastendavad Java andmetüübid SOAPiga.

Iga tarkvara moodul võib olla firmaomase ja tihedalt seotud struktuuriga, kuid teenusepõhine arhitektuur suudab tagada teenuste koostöövõime. Seda põhjusel, et teenusepõhine arhitektuur kasutab koostalitusvõimelisi tehnoloogiaid, mis toetavad olemasolevate ja potentsiaalsete uute klientide protokolle ja andmete formaate.

Teenused ei ole omavahel tihedalt seotud

Sidestus näitab palju on moodulite vahelisi sõltuvusi. Sidestatus jaguneb kaheks: tihe- ja lõdva sidestatus. Sidestus kirjeldab erinevate tarkvara moodulite omavahelist sõltuvuse taset. Lõdvalt seotud moodulid on paindlikud ja omavad hästi määratletud sõltuvusi. Tihedalt seotud tarkvara süsteemid on vastupidiselt raskesti ümber seadistatavad, sest moodulite seosed tarkvara struktuuris on tundmatud. Mida tihedamalt on moodulid sidestatud, seda rohkem muudatusi on vaja teenuse tarbijal teha, kui teenusepakkuja peaks midagi muutma. Sidestatus muutub tihedamaks kui teenuse tarbija vajab teenuse kasutamiseks palju informatsiooni teenuse tarbija kohta. Olukord, kus tarbija teab teenuse asukohta ja muud detailset informatsiooni, on tihe sidestus, aga kui tarbija ei vaja detailset informatsiooni teenuse käivitamiseks, on pakkuja ja tarbija vahel lõdva sidestus.

Teenusorienteeritud arhitektuur mõjutab arendama lõdvalt seotud teenuseid, mida saab kasutada tarkvara loomisel. Lõtvu seosed aitab tagada teenuse registri kasutamine. Teenuse autonoomsel käivitamisel ei ole vaja teenusele ette anda süsteemi spetsiifilist

informatsiooni. Tihedalt seotust tuleb siiski rakendada liidese määramise tasemel või kindla protokolliga sidumisel.

Teenustel on võrgu-adresseeritavad liidesed

Võrk on SOA kontseptsioonis kesksel kohal. Teenus peaks avaldama võrgus oma liideseid, et olla vastavuses teenusorienteeritud arhitektuuri disaini põhimõtetega, võimaldades teenuse tarbijal hajusalt üle võrgu teenust käivitada. Sellisel viisil on võimalik kasutada teenust igal ajal erineva teenuse tarbija poolt ja asukohast sõltumata.

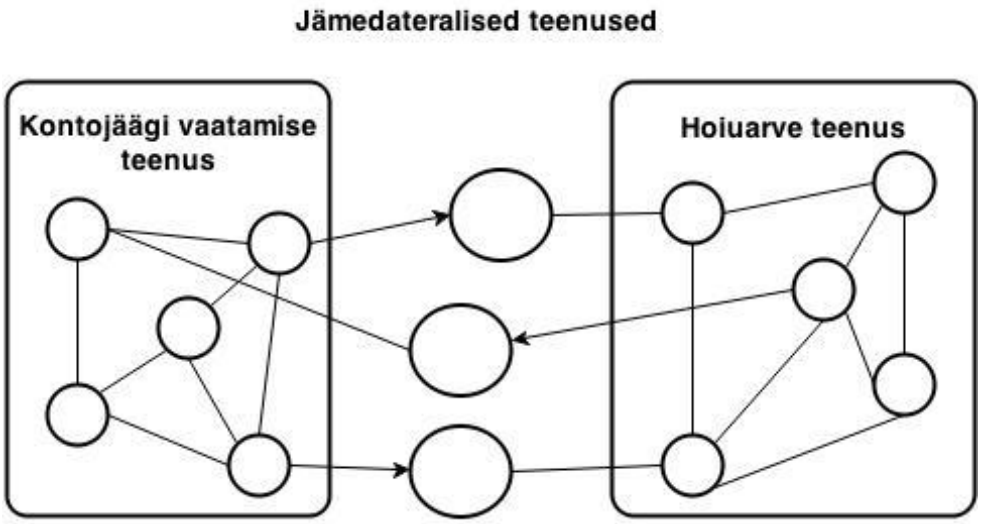
Teenuseid omavahel kombineerides saab luua uusi teenuseid

Teenuste kombineerimine on sõltuvuses modulaarsest struktuurist. Modulaarne struktuur võimaldab teenuseid omavahel kombineerida ja kasutada mitmetes erinevates rakendustes. Olemasolevate testitud teenuste kasutamine tõstab teenuse kvaliteeti ja muudab arendamist odavamaks.

SOA üks peamisi omadusi on võimalus luua uusi rakendusi kombineerides olemasolevaid teenuseid. Sellist kombineerimist iseloomustab efektiivne disain, mille peamine eesmärk on tagada teenuse modulaarsuse ja teenuse funktsionaalsuse taaskasutatavus, omamata eelnevat teadmist, milline rakendus hakkab teenust tulevikus kasutama.

Teenustel on jämedateralised liidesed

Teralisuse mõiste kirjeldab, kuidas on teostatud liidesed tarkvara süsteemis. Tegemist on jämedateralise liidesega, kus liides toetab kõiki funktsionaalsusi, mis on vajalikud, et tagada äriloogika täielikult. Kui liides täidab ainult mingit osa tervik funktsionaalsusest, siis on tegemist peeneteralisusega. Teralisust võib omistada tervele teenuse realisatsioonile ning liidese käivitamise üksikutele meetoditele eraldi. Teenuse orienteeritud arhitektuur toetab jämedateralist liidese üleseehitust, mis võib olla madalamatel tasemetel peeneteraline. See tähendab, et objektid, millest teenus koosneb võivad olla peeneteralised.



Joonis 3. Teenuse teralisus.[29]

Teenused on asukoha läbipaistvad

Asukoha läbipaistvus on oluline teenusorienteeritud arhitektuuri omadus. SOA kasutamine soodustab asukoha läbipaistvust, mis tähendab, et teenuse tarbija ei ole teadlik teenuse asukohast enne selle käitusaegset käivitamist registrit kasutades. Ainus sõltuvus teenuse tarbija ja pakkuja vahel on teenuseleping, mis võib nihkuda ühes asukohast teise, kuid ei mõjuta sellega tarbijat. Süsteemi kasutatavust ja jõudlust saab tõsta, kasutades koormuse stabilisaatorit, mis vahendab päringuid mitmele sama funktsionaalsusega teenusele nii, et klient ei tunneta süsteemi talituses erinevust.

SOA toetab iseparanemist

Hajus rakenduste suuruse ja keerukuse kasvuga muutub olulisemaks võimekus tulla toime vigadega. Iseparanemist kirjeldatakse kui süsteemi omadust taastada end kui peaks käivitusajal ilmnema vigu. Oluline on, et süsteemi töö taastuks inimese sekkumiseta.

Töökindlus mõõdab, kuidas süsteem töötab kui tekivad vead. Teenusorienteeritud arhitektuuris on teenused aega-ajalt töös ja maas. Teenuse mitte töötamist võib esineda rohkem kui teenus on sõltuvuses teiste organisatsioonide või osakondade teenustest. Süsteemi iseparanemise võimekus sõltub mitmetest asjaoludest.

Töökindluse tagamiseks on vaja, et riistvara võimaldaks veast taastuda. Näiteks võrk peab võimaldama käitusaegset dünaamilist ühendamist erinevate süsteemide külge.

Oluline on ka, et süsteem oleks ehitatud arhitektuurile, mis toetab iseparanemist. Iseparanemist toetavad rohkem arhitektuurid, mis toetavad dünaamilist seotust ja komponentide käitusaegset käivitamist.

Teenusorienteeritud süsteemid panevad teistest arhitektuuridest enam rõhku iseparanemisele, sest teenused on kombineeritud käivitama äriloogikat käitusaegselt. Teenuse tarbija saab otsida võrgust erinevaid teenuseid, mis toetaks sama funktsionaalsust eesmärgiga tagada süsteemi probleemideta töö vea ilmnemisel.

2. Oracle Formsi ja teenusorienteeritud arhitektuuri võrdlus

Selles peatükis võrdleb autor teenusorienteeritud arhitektuuri ja organisatsioonis kasutusel oleva Oracle Forms 10g Release 2 versiooni arhitektuuri. Samuti toob välja põhjused, miks on vaja eelistada teenusorienteeritud arhitektuuri ning kuidas teenusele orienteeritud arhitektuur toetab äripoole vajadusi ja eesmärkide saavutamist.

2.1. Teenusorienteeritud arhitektuuri kasulikkus tarkvaraarendaja seisukohast

Alljärgnevalt on välja toodud teenuspõhise arhitektuuri peamised eelised:

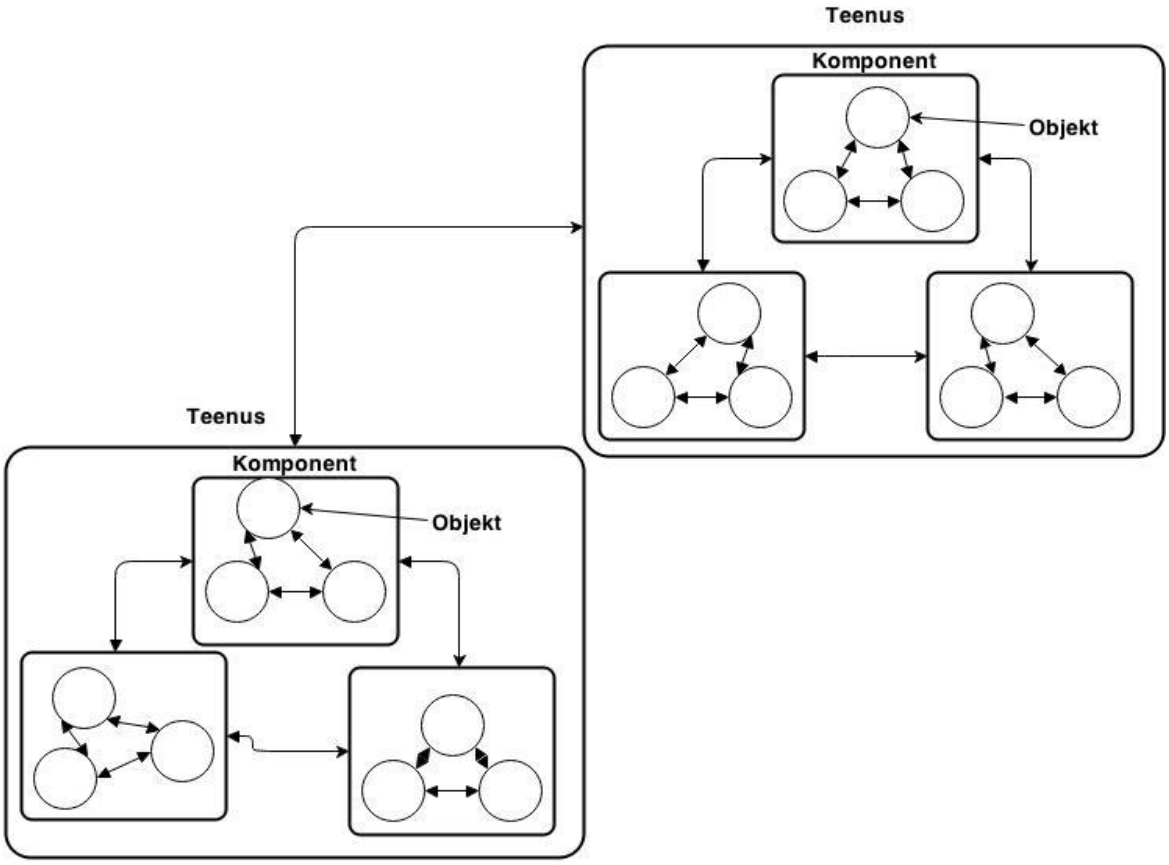
- Teenuspõhise arhitektuuri abil on võimalik integreerida erinevaid infosüsteeme. Arendajad ei pea töötama välja uusi lahendusi erinevate süsteemide kasutamiseks. Oluline on kasutada standardseid protokolle (nt andmevahetusprotokollid), sest standardeid mittejärgivaid teenuseid on raske juurutada.

- Liidesed on taaskasutatavad. Liidest, mida kasutati ühe konkreetse süsteemi arendamisel, saab kasutada ka kõikide teiste analoogsete süsteemide arendamisel.

- Kasvab arendusprojektide tootlikkus. Taaskasutamine aitab sarnaseid lahendusi ehitada kiiremini. Arenduse projektid on odavamad ning kiiremaks muutub ka süsteemide integratsioon.

Teenuspõhine arhitektuur on asendamatu veebiportaalide ehitamisel. Veebiportaal võib hankida uudiseid ja muud informatsiooni kolmanda osapoole veebiteenuste vahendusel. Veebikauplused ja tootjate veebilehed võivad vahendada kaupade andmeid oma andmebaasidest teineteisele edasimüügiks. Seejuures suhtlevad infosüsteemid teenustega automaatselt. Võibki öelda, et SOA levinumaid ärilisi kasutusi on tellijate ja tarnijate infosüsteemide liidestamine. Samamoodi saab panna SOA abil suhtlema sama firma erinevad infosüsteemid, sõltumata kasutatud tehnoloogiatest ja platvormidest - operatsioonisüsteem, andmebaas jne.

Allolev joonis kirjeldab teenuse orienteeritud arhitektuuri komponentide ja objektide kontekstis.



Joonis 4. Teenuse põhine tarkvara arendamine.[29]

Tarkvara on arendatud läbi ajaloo erineval viisil ja arhitektuuri mudeleid kasutades, kuni jõuti platvormi sõltumatute ja taaskasutatavate teenuste arendamiseni. Iga arhitektuuri mudelil on omad omadused, et muuta lihtsamaks ja efektiivsemaks tarkvara arendamine. Objekt orienteeritud arhitektuur pakkus paindliku tarkvara arendust, mis võimaldas kapseldada äriloogika laiateralisemale kujule, funktsioonideks ja klassideks. Veelgi efektiivsemaks osutus tarkvara jagamine komponentideks ja teenusteks.

Allolev tabel kirjeldab erinevate tarkvara arhitektuuri mudelite omadusi ja erinevuseid. Antud tabelist näeb, millised on muidu omavahel väga sarnaste komponendi- ja teenusepõhise arhitektuuri peamised erinevused, ning kuidas erineb neist mõlemast objekt orienteeritud arhitektuur.

Tabel 1. Tarkvara arhitektuuride võrdlus.

Objekt orienteeritud arhitektuur	**Komponendi põhine arhitektuur**	**Teenuse põhine arhitektuur**
Süsteemi struktuur on peeneteraline (*Fine Grained*)	Süsteemi struktuur on keskmiseteraline (*Medium Grained*)	Süsteemi struktuur on jämedateraline (*Coarse Grained*)
Taaskasutatavus on madal	Taaskasutatavus on keskmine	Taaskasutatavus on kõrge
Tihe sidestus	Lõtv sidestus	Lõtv sidestus
Ehitusplokid on iseseisvad klassid	Ehitusplokid koosnevad mitmest klassist	Ehitusplokid koosnevad komponentidest
Omab kompileerimise aegseid sõltuvusi.	Omab kompileerimise aegseid sõltuvusi	Omab ainult käitusaegseid sõltuvusi
Funktsionaalsus on kirjeldatud klassi deklaratsioonides	Funktsionaalsus on kirjeldatud liidese deklaratsioonis	Funktsionaalsus on kirjeldatud võrgus adresseeritava komponendi liidese deklaratsioonis

2.2. Oracle Formsi ja teenusorienteeritud arhitektuuri erinevused

Mõistmaks peamisi erinevusi Oracle Formsi ja SOA vahel, tuleb võrrelda mõlema erinevaid aspekte. Järgnevalt on väljatoodud Forms rakenduse ja teenusorienteeritud arhitketuuri olulisemad erinevused.

2.2.1. Standardiseerimine

Viimasel aastakümnendil on rakenduste loomiseks kasutatud enim Java programmeerimiskeelt. Java kasutamine on niivõrd populaarne, et seda võiks nimetada tööstuse standardiks. Java spetsifikatsioonide ja lähtekoodi avalikuks muutmine tõstsid ühilduvust erinevate Java platvormide vahel.

Teenusorienteeritud arhitektuuri implementeerimisel ei ole oluline, millist programmeerimiskeelt kasutatakse teenuste loomiseks, kuna teenused on nagu mustad kastid, mis peavad avalikustama ainult oma rakenduse liidesed (APId), kasutades standardeid. Niisiis, kuigi Java standardiseerimine on hea, kuna see võimaldab vähendada haldusele tehtavaid kulusid jne, siis arhitektuurilisest seisukohast ei ole vahet, millises keeles on teenused implementeeritud. Arendajad võivad kasutada erinevaid programmeerimise keeli ja erinevaid platvorme (operatsioonisüsteeme, riistvara) niikaua kui rakenduse liidesed kasutavad standardseid lahendusi.

Oluline on teada, et SOA baseerub avatud standarditel, kuid Oracle Forms raamistik põhineb firmaomasel tehnoloogial ja standarditel. SOA muudab arendamise lihtsamaks ja kiiremaks, kuna erinevad rakendused on võimalik lihtsalt panna koos töötama, omamata teadmisi firmaomastest standarditest ja tehnoloogiast.

2.2.2. Probleemide eraldamine

Tarkvaraarenduses ja süsteemide arhitektuuris on levinud praktika probleemide eraldamine. Koodi osad, mis tegelevad sarnaste ülesannete lahendamisega, tuleb eraldada koodist, mis tegelevad teiste ülesannetega. Selline eraldamine on oluline, sest see aitab hoida kooskõlastust kõrgel ja seotust madalal, mis omakorda teeb lihtsamaks muudatuste realiseerimise.[28] Paljud tehnoloogiad ei toeta või ei võimalda probleemide eraldatust, kuigi see on olnud alati programmeerimise parim praktika.

Probleemide eraldamine on oluline SOAs ning seda aitavad tagada: mitmekihiline arhitektuur, objekti ja teenus orienteeritus. Forms rakenduse migreerimise SOA'le vastavaks, muudab keeruliseks tarkvara suur erinevus arhitektuurist. Antud peatükk kirjeldab, kuidas probleemide eraldamine eksisteerib Formsis ja SOA's.

Rakenduse- ja äriloogika võrdlus

Rakenduse- ja äriloogika on omavahel sarnased. Antud töös kasutab autor rakendusloogika mõistet kirjeldamaks tehnilist loogikat, mis tegeleb infotöötlusega. Äriloogika on abstraktne ja kirjeldab äri poliitikat ja reegleid.

Rakenduseloogika näide: kood, mis konverteerib kontojäägi summa ühest valuutast teise. Äriloogika näide: kood, mis kontrollib, millise valuutaga kontojääki kasutajale kuvada. Need kaks loogikat võivad omavahel segamini minna, aga oluline on neid

erinevalt liigitada, sest mõlemad asuvad SOAs erinevas osas. Samas Formsis ei esine sellist eraldi hoidmist.

Rakendusloogika

Teenusele orienteeritud arhitektuuris on rakenduseloogika kasutatav teenusena. Selliste teenuste ülesanne on pakkuda vahekihti andmete ja rakenduste vahel. Formsi arendajad kasutavad talletatud protseduure, mida hoitakse andmebaasis. Talletatud protseduuride eelis on, et neid saab kasutada mugavalt ja korduvalt. Nende protseduuride kasutamise üheks suuremaks puuduseks on see, et kliendid ühenduvad otse andmabaasi külge. Näiteks muudab see süsteemi turvalisust nõrgemaks.

Äriloogika

Forms rakendustes on tavaline, et äriloogika lõimub koodiga, kus on veel lisaks rakenduse- ja esitlusloogika. Teenusele orienteeritud arhitektuuri toimimiseks on vajalik, et äriloogika oleks eraldatud teistes loogilistest osadest.

Esitlusloogika

Forms kasutab firmaomast ja platvormist-sõltuvat tehnoloogiat ning esitlusloogika on tihedalt ülejäänud koodiga koos. Lisaks on tehnoloogia loodud toetama ainult ühte kindlat klienti, näiteks laua- või sülearvutit.

Teenusele orienteeritud arhitektuuris on esitluskiht eraldi. See võimaldab luua mitu kasutajaliidest, mis teenindavad erinevaid kliente.[14] Kuna SOA ei esita piiranguid tehnoloogiale, siis võib kasutada platvormist sõltumatuid tehnoloogiaid, nagu seda on näiteks HTML5. Võib kasutada ka platvormist sõltuvaid ja firmaomaseid tehnoloogiaid ja teha igale kliendile oma kasutajaliidese. Mitme erineva kasutajaliidese loomiseks ei ole vaja dubleerida rakenduse- ja äriloogikat. Nii saab luua eraldi kasutajaliidese töötajale, kellel on kasutada kogu funktsionaalsus ja kliendile, kellel on piiratud funktsionaalsus ja kasutajasõbralikum liides.[14]

2.3. Teenusorieteeritud arhitektuuri eelised võrreldes Oracle Formsiga

Teenusele orineteeritud arhitektuuril töötavad lahendused pakuvad lisaks eelnevas peatükis mainitud plussidele veel mõningat lisa. All järgnevalt kirjeldatud erinevused kuuluvad mitmete põhjuste hulka, miks tasuks võtta kasutusele tarkvara, mis toetab teenusele orienteeritud arhitektuuri.

Integreerimine

Forms ei võimalda otse integreerimist teiste rakendustega. Integreerimine on võimalik ainult otse läbi Oracle andmebaasi. Kirjeldatud otse ühenduse loomine on mitte soovitatav, kuna see vähendab süsteemi turvalisust ja töökindlust. Peatükis 1.4.2. kirjeldasin, et SOA eesmärk on tagada rakendustele probleemideta ja avatud standarditel baseeruv omavaheline integreerimine.

Rakenduse kasutajaliidese kasutatavus

Formsi rakenduste kasutajaliidesed ei ole tänapäevaste tööriistadega võrreldes kasutajasõbralikud. Uued rakenduse kasutajad vajavad koolitusi ja aega harjumiseks enne kui saavad alustada tööd. Varasema Formsi kogemusega kasutaja leidmine tööjõuturult muutub mööduvate aastatega keerulisemaks. Lisaks on rakenduse uued kasutajad harjunud kasutama kaasaegsemaid rakendusi.

SOA võimaldab kasutajaliidese arendamist vabalt valitud tehnoloogiat kasutades. Ehk siis sama äriloogika esitamiseks saab kasutada erinevat tehnoloogiat toetavat kasutajaliidese lahendust.

2.4. Äripoole vajadus teenusorienteeritud arhitektuuri kasutusele võtuks

Äri põhiline ülesanne tänapäeval on olla agiilne, innovatsiooni looja ja soodustaja. Oluline on säilitada ja vajadusel parandada kvaliteeti, hoides seejuures kulutusi madalana. Alljärgnevalt kirjeldab autor, millised on äripoole ootused ja trendid ning kuidas toetab neid teenusorienteeritud arhitektuuri rakendamine.

2.4.1. Äripoole ootused ja trendid

Esmalt on vaja selgitada, mida tähendab äripoole jaoks olla teenusele orienteeritud. Teenus orienteeritud lähenemine ei ole alati mõlemale, nii ITle kui äripoolele, toonud soovitud tulemusi. Näiteks peale pikka ja kulukat tarkvara ümber disainimist ja arendamist on avastatud, et loodud lahendus ei paku äripoolele soovitud tulemusi. Tarkvara disainimine teenusorienteeritud arhitektuurile muudab tarkvara arenduse keerukamaks Seetõttu on oluline selgeks teha, mida äripool soovib lõpptulemusena näha, et teenus orienteeritud lähenemisele üleminek oleks edukas ja kasulik mõlema osapoole jaoks.

Siin on mõned põhilised äriootused:

- Võimalus äril olla maksimaalselt agiilne, et tulla toime pidevalt muutuva turuga, kiirelt ja madalate kuludega.
- Võimekus tulla toime ja olla vastavuses pidevalt muutuvate seaduste ja regulatsioonidega.
- Võimekus saavutada edu ja teenuse kvaliteedi tõus ühildudes mõne sisemise-, välimise- või pilve teenusega.
- Võimekus hinnata täpselt teenuse arenduse mõõdikuid: hinda, arendamiseks kuluvat aega jne.
- Võimekus anda selgemat ülevaadet sisemistest ja välistest teenustest, et neid oleks parem toetada ja juhtida.

2.4.2. Kuidas teenuseorienteeritud arhitektuur toetab äripoolt

Millised on need teenusorienteeritud arhitektuuri omadused, mis teevad sellest tänapäevastele organisatsioonidele vajaliku arhitektuuri ning kuidas see toetab eelmises peatükis kirjeldatud nõudeid.

Paindlik arhitektuur

Rakenduste paindlikus on teine oluline omadus, mis on oluline ka äristrateegias. On oluline, et äri oleks agiilne ning kogu organisatsiooni iseloomustaks omadus kiirelt muutuda, et olla edukas pidevalt muutuvas maailmas. Süsteemid, mis on loodud muutuma, on väärtuslikumad kui süsteemid, mis on loodud kestma. Praktikas töötavad pikemalt süsteemid, mis on loodud muutuma.[8]

SOA tagab paindlikkuse, võimaldades jagada süsteemi iseseisvateks komponentideks (teenusteks). Teenuseid aga saab omavahel lihtsalt ja dünaamiliselt siduda, luues sedasi infosüsteeme, mis toetavad äri protsesside muutusi.[14]

Integreerimine ja standardiseerimine

Rakenduste omavaheline integreerimine on oluline peaaegu kõigis äristrateegiates. Enamik organisatsioone töötavad mitmete erinevate rakenduste ja infosüsteemidega, mis paiknevad kas organisatsiooni sees või väljas. Usaldusväärne ja paindlik rakenduste ning infosüsteemide vaheline integreerimine on ärile vajalik, et luua konkurentsieeliseid, tagada kõrge automatiseeritus, kõrge klienditeeninduse tase ning teised äri edendavad eelised.[14]

Integreerimine rakendusega, mis ei ole loodud SOA arhitektuurile, iseloomustab tihti keeruline ja kulukas arendusprojekt. SOA põhineb mitmetele avatud standarditel ning on loodud olema paindlik. Suured tarkvara tootjad (n. Oracle) on mõistnud, et integreerimine on vanemate toodete suur probleem, ennekõike paindlikkuse puudumise ja standardite mitte kasutamise tõttu. Seepärast on uuemad tarkvara tooted loodud nii, et need kasutaksid võimalikult maksimaalsel määral laialt levinud standardeid.

Kulude kokkuhoid

Infotehnoloogia on äripoolt toetav üksus, mille ülalpidamiseks tehtavad kulutused on organisatsioonides ühed suuremad. Seetõttu on kulude kokkuhoid äripoolt enim motiveerivam punkt, mida kasutatakse SOA tehnoloogia müümiseks. SOA võimaldab muuta lihtsamaks ja odavamaks Forms tarkvara haldamise, mis praegu võtab suure osa organisatsioonis infosüsteemide haldamiseks ettenähtud eelarvest. Seda aitavad tagada: süsteemi efektiivsemaks muutmine, muudetud või kadunud litsentseerimise nõuded. Tarkvara arenduses on võimalik saavutada produktiivsuse kasvu ja kulude kokkuhoidu lihtsustatud integreerimise ja programmeeritud komponentide taaskasutamisega.

3. Kasutatavad tehnoloogiad

Käesolevas peatükis autor analüüsib ja valib välja sobivaima tehnoloogia, millele rakenduse funktsionaalsus platvormi vahetusel üle viia. Analüüsitavad tehnoloogiad on: Oracle Forms, Oracle ADF, Oracle APEX ja Java Spring raamistik. Nimetatud tehnoloogiate kasutamine on määratud organisatsiooni nõuetega, mille peamine eesmärk on hoida ettevõttes kasutusel olevate erinevate tehnoloogiate arv võimalikult väiksena.

3.1. Oracle Forms

Forms on Oracle vanim arendustarkvara, mida endiselt aktiivselt arendatakse. Esimest korda tutvustati seda toodet 80ndate aastate lõpus. Algselt oli tegemist interaktiivse arendus tarkvaraga, kus esmalt tuli vastata küsimustele, mille põhjal koostas tööriist sobiva tarkvara. Hiljem arendati tarkvarale kasutajaliides ja programmikeel PL/SQL.

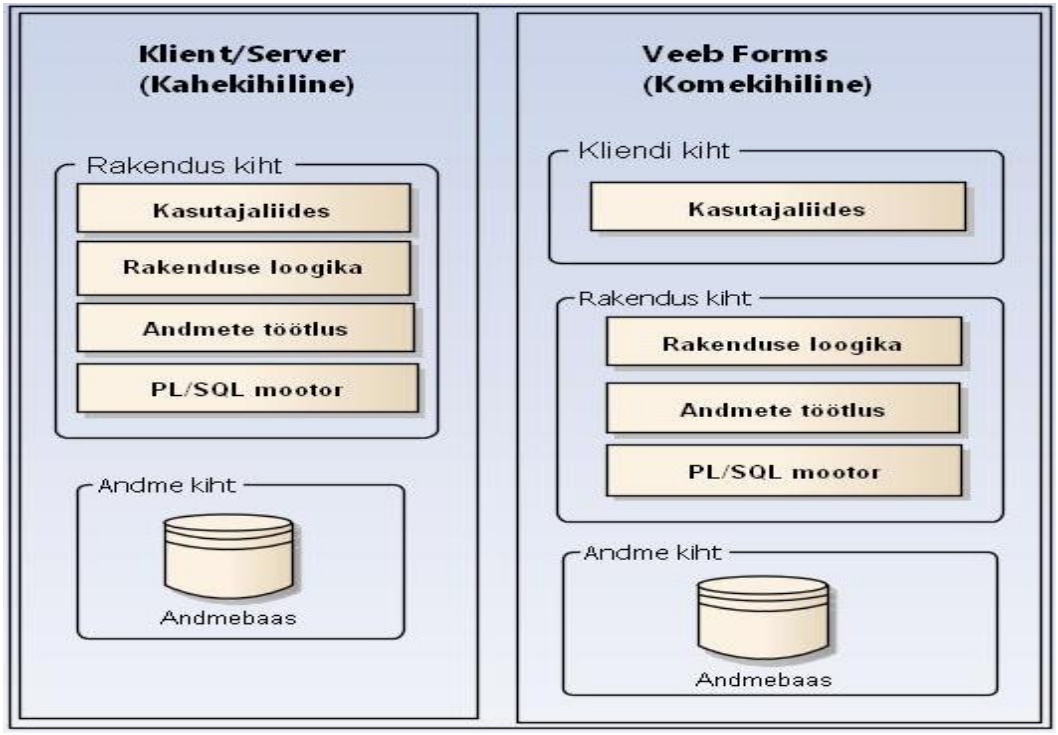
Tänapäeval on Oracle Forms tarkvara, mis võimaldab luua vorme andmete manipulatsiooniks Oracle andmebaasis. Tarkvara sisaldab integreeritud programmeerimiskeskkonda (IDE), programmeerimiseks koodi redaktorit, mis töötleb PL/SQL koodi ning lisaks veel teisi vajalike vahendeid. [21]

Forms loob ühenduse Oracle andmebaasi külge ja genereerib vormid, mis kuvavad andmeid. Formsi vormi fail (*.fmb) kompileeritakse täitmisprogrammis (*.fmx), mida käivitab Formsi käivitus moodul (*runtime module*).

Formsi arhitektuur

Oracle Forms on algselt loodud toetama kahekihilist arhitektuuri, mis koosneb ühelt pool andmebaasist ja teiselt poolt rakendusest. Andmebaas on andmekiht ja paks klient rakendus moodustab rakenduse kihi.

Uuemad Formsi versioonid toetavad alternatiivset kolmekihilist arhitektuuri, mida nimetatakse Veeb Forms’iks (Web Forms). (Joonis 5)



Joonis 5. Formsi arhitektuurid.[33]

Talletatud protseduurid ja trigerid

Talletatud protseduurid on PL/SQL koodi kogumid, mida rakendused saavad vastavalt vajadusele käivitada. Seetõttu on võimalik taaskasutada koodi nagu funktsioone teistes programmeerimiskeeltes. Talletatud protseduurid paiknevad Formsis nii rakenduses kui andmebaasis. Talletatud protseduuri nimetatakse trigeriks kui see on seotud andmebaasi tabeli või sündmusega (näiteks andmebaasi uue korteeži lisamine).

Rakenduskiht

Rakendusekihi moodustavad üks või enam Oracle Formsi klient rakendust. Need rakendused jagunevad mooduliteks, millest enamasti igaüks on seotud kindla objektiga, näiteks kliendid, tellimused, töötajad jne. Iga moodul on sisuliselt kasutajaliides, mida saab kasutada kindla objektiga töötamiseks, seetõttu on moodul tihedalt seotud objekti andmeid sisaldavate andmebaasi tabelitega.

Iga moodul sisaldab liidest, liidese kontrollimise, andmete töötlemise ja teisi loogikaid. Kõik see loogika on kirjutatud PL/SQL koodis.

Ka rakendus jaguneb mooduliteks, millest igaüks sisaldab firmaomast liidese kirjeldust, mida interpreteerib Formsi käitusmootor, et visualiseerida kasutajaliides, liidese loogika

(implementeeritud PL/SQL koodis) ja rakenduse loogika (ja äriloogika), mis töötleb andmeid ja saadab ad-hoc SQL päringuid andmebaasi või kutsub välja talletatud protseduure.

Rakenduses navigeerimiseks on kasutajal võimalus kasutada menüüd, mis näitab nimekirja moodulitest, mida on võimalik avada või teise võimalusena lasta liidese loogikal avada teisi mooduleid.

Oracle soovitab hoida "klient" võimalikult õhukesena. See tähendab, et võimalikult suur hulk loogikat tuleks kirjutada andmebaasi talletatud protseduuridesse. Samas on Formsi arendajal võimalik käituda vastupidiselt, seetõttu leidub arendusi, kus enamus rakenduse loogikast on kliendi moodulis.

Kliendi kiht (ainult Veebi Forms)

Alates Oracle Application Server 9i tarkvara versioonist on võimalus luua veebipõhiseid Formsi rakendusi, mis interpreteerivad mooduli koodi ja saadavad kasutajaliidese koodi kliendi brauserile, mis omakorda visualiseerib koodi kasutades Java apletti.[33] See tehnoloogia muudab võimalikuks veebipõhise juurdepääsu rakendusele ning seda ilma vajaduseta paigaldada spetsiaalset klienttarkvara. Samas ei sisalda see tehnoloogia eraldi esituskihti, kuna liidese- ja rakenduse loogika on ikkagi integreeritult samas koodis ja tihedalt seotud.

Ühildumine

Mõne teise tarkvara ühildumiseks Oracle Forms rakendusega on vaja, et see tarkvara saaks luua käitusajal ühenduse andmebaasiga ja edastada päringuid otse andmebaasi kasutades SQL päringuid. Selline lahendus põhjustab mitmeid probleeme. Esiteks peab kirjeldatud lahenduse jaoks olema andmebaas avatud ühendustele, mis luuakse väljas poolt, tihti ka väljas poolt ettevõtte sisevõrku. Lahenduse taoline rakendamine võib põhjustada probleeme turvalisuses ja teenuse tagamisel.

Teine probleem on selles, et teise tarkvara arendajad peavad tundma Oracle SQL dialekti, et kirjutada korrektseid SQL päringuid.

Otse andmebaasi külge ühendamist tuleks vältida, et oleks lihtsam ja odavam teenust hallata. Vastasel juhul tuleb välist programmi tihti muuta kui tehakse muutusi

andmebaasis ning kuna üldiselt arendavad väliseid programme inimesed teisest osakonnast või ettevõttest, siis võib tarkvara kohandamine andmebaasiga osutuda rahaliselt kalliks.

Forms versiooni uuendamine

Oracle Formsi müüakse Oracle andmebaasist eraldi, kuigi reegliks on kujunenud, et iga kord, kui antakse välja uus suurem reliis Oracle andmebaasist, siis hakatakse pakkuma ka uut Oracle Formsi versiooni, et toetada andmebaasi uusi funktsionaalsusi.

Iga Formsi versioon ühildub enamuste Oracle andmebaasi väljalasetega, kuigi neid müüakse lahus ja on välja antud erinevatel aegadel. See tähendab, et Formsi nimega tooted on enamuses edasi- ja tagasiühilduvad Oracle andmebaasidega. Näiteks Forms 9't on võimalik kasutada Oracle andmebaasi versioonidega 8, 9, 10 ja 11. [22]

Üks võimalik lahendus rakenduse funktsionaalsuse tagamiseks, on viia rakendus üle uuele Formsi tehnoloogiale. Oracle Forms 11gR2 on hetkel kõige uuem versioon Oracle Forms tarkvarast. Turule toodi see versioon 2011. aasta lõpus ning Oracle on kinnitanud, et tootel on täielik tugi aastani 2017. Tänaseks on kahekihilisel arhitektuuril töötavate versioonide tarkvara tugi lõppenud, aga versioonid alates 10gR1'st omavad õigust igavesti kestvale säilitus toele (*sustaining support*). [34]

Kas Oracle jätkab Formsi arendamist ja toetamist on küsimus, mis on tarkvara kasutajaid vaevanud juba paar viimast aastat. Aastal 2007 teatati Gartneri raportis, et ei ole lõplikult võimalik vältida Formsi platvormilt üleminekut mõnele teisele tehnoloogiale. Organisatsioonidele soovitati alustada migratsiooniga modernsemale tehnoloogiale nagu näiteks Java või .NET suunas, kuna Forms ei ole võimeline lahendama järgmise põlvkonna tarkvara arenduses eesootavaid väljakutseid. Kokkuvõtvalt tõdeti, et tuhanded rakendused, mis on loodud Formsis, tuleb migreerida teisele tehnoloogiale varem või hiljem. [2]

Viimase Oracle Formsi puudutava Gartneri raporti seisukoht on sisuliselt sama. 2011 aasta lõpus avaldatud raportis leitakse, et migratsioon on vajalik, kuigi selle aja jooksul on Formsil välja tulnud kaks uut versiooni (11gR1 ja 11gR2). [4]

Gartneri 2011. aasta aruandes ollakse selgel arvamusel, et Oracle jätkab Formsile toe pakkumist ka tulevikus, kuid samas usutakse, et Oracle ei sea Formsi edaspidi eelistatud

tehnoloogiaks. Tulevased uuendused saavad olema eelmiste versioonidega kergesti ühilduvad, kuid tehnoloogiliselt mitte väga uuenduslikud, seetõttu on need kliendile ka odavamad. [4]

Tulevased Formsi arendused hakkavad pakkuma enam lahendusi ja tuge, et Formsi rakendused saaksid olla osa hübriid rakenduste kasutuselevõtust, et oleks võimalik kasutada sissetöötanud Formsi lahendusi kombineeritult Oracle uuemate tehnoloogiatega, nagu Oracle Application Development Framework (ADF). Arenduste üldine eesmärk on viia Forms sellisele tasemele, et migreerumine Oracle enda järgmise generatsiooni Java tehnoloogiale (Oracle JDeveloper ja Oracle ADF raamistikule) oleks võimalikult kerge barjääri ületamine.

Järgnevalt loetlen Forms tehnoloogia olulisemad eelised ja puudused.

Oracle Forms tehnoloogia eelised:

- **Kogemustega Oracle andmebaasi ja Formsi arendajad on seda tehnoloogiat kasutades efektiivsed:** Kogenud Formsi arendajad suudavad kiiresti valmis teha keerulisi rakendusi, milleks nad saavad kasutada erinevad vahendid, näiteks automaatne andmebaasi rea lukustus jne. Oracle andmebaasi arendajad ei pea õppima uusi tarkvaraarendus keeli, kuna Forms kasutab PL/SQL koodi. Töötajate kogemused Forms rakenduste arendamisel on oluline argument jätkamaks Formsi platvormil.

- **Oracle tugi tasulistele toodetele:** Oracle pakub head tugiteenust tasulistele toodetele. Probleemide korral saab tasu maksmata edastada küsimusi litsentsi lepingu alusel vastavatele kontaktidele.

Oracle Forms tehnoloogia puudused:

- **Aegunud tehnoloogia:** Paljudes hinnangutes kirjeldatakse Oracle Formsi kui aegunud tehnoloogiat, mistõttu on seda tarkvara kasutavatel ettevõtetel järjest raskem leida Oracle Formsi arendajaid.

- **Aegunud kasutajaliidese disain:** Lisaks on kasutajaliidese disain aegunud. Aegunud kasutajaliides ei ole küll probleemiks töötajatele, kes on eelnevalt töötanud Forms tehnoloogiale loodud tarkvaraga, kuid teistel tuleb pingutada, et

tarkvara kasutamisega harjuda. Formsi kasutajaliidese moodsamaks muutmine nõuab küllaltki suurt lisaarendust.

- **Kõrged litsentsitasud:** Forms tehnoloogia kasutamiseks on vaja litsentsi ja selle eest tuleb maksta raha, versiooni uuendamine on samuti tasuline. Forms tehnoloogia kasutamise muudab kulukaks ka tehnoloogia jäik sõltuvus Oracle andmebaasist ja Oracle omandis olevast WebLogic serveri tehnoloogiast, mida saab kasutada küllaltki suure litsentsitasu eest.

- **Toetab vähe teenusele orienteerituse põhimõtteid:** Forms on tihedalt seotud ja loodud töötama Oracle andmebaasiga. Lisaks on antud tehnoloogiat võimalik kasutada ainult WebLogic serveriga.

3.2. Oracle Application Development Framework

Aastal 1999 tutvustas Oracle avalikkusele tehnoloogiat ADF (Application Development Framework).[5] Tänaseks päevaks ligemale 15 aastat arendatud tehnoloogia on oma koha leidnud teiste Oracle tehnoloogiate kõrval ning tegeletakse ka selle edasi arendamisega.

Oracle ADF baseerub Java EE arendusraamistikul, mis on loodud, et kiirendada ja muuta lihtsamaks ettevõte rakenduste arendamist.

Oracle ADF on raamistik, mis tähendab, et enamuse funktsionaalsusest paikneb teekide kogumikus. Vastavaid tarkvarasid kasutades on võimalik muuta või lisada funktsionaalsusi teekide klassidele, luues seeläbi spetsiifilisi äri komponente.

Oracle ADFi arhitektuur

Oracle ADF baseerub disainilt MVC (Model-View-Controller) disaini mustril. MVC rakendus jaguneb neljaks osaks:

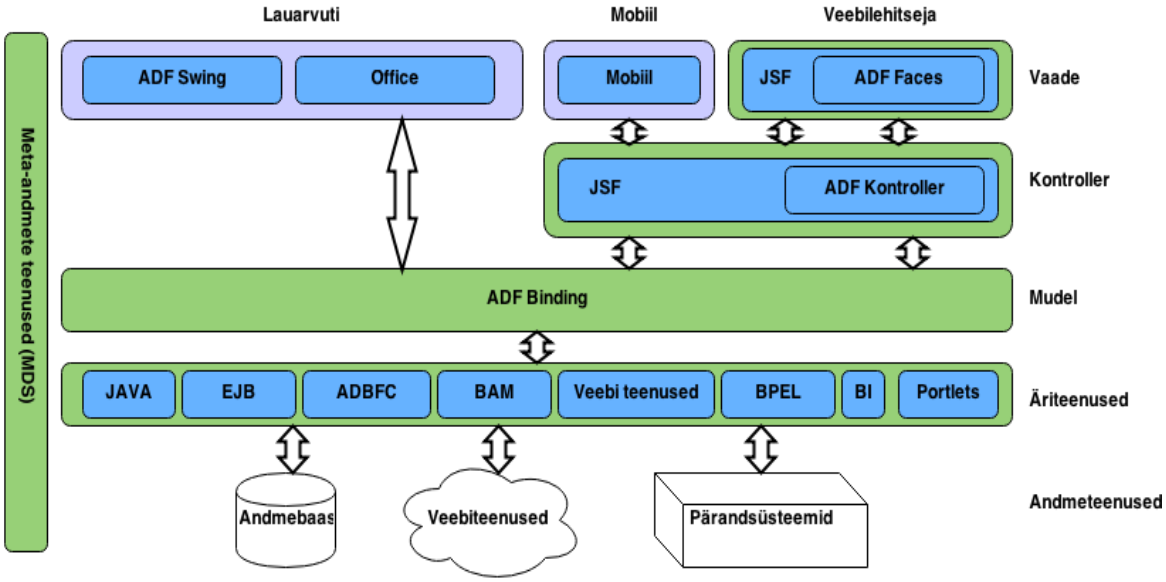
- Äriteenuse kiht (Business Service layer), selles kihis pakutakse ligipääsu andmetele erinevastest andmeallikatest ning rakendatakse äriloogikat.

- Mudeli kiht (Model layer), mille eesmärk võimaldada interaktsiooni andmeallikaga ja rakendada äriloogikat.

- Vaatekiht (View layer), on kiht mille eesmärk on pakkuda rakenduse graafilist liidest.

- Kontroller (Controller layer) töötleb rakenduse voogu ja on liideseks mudeli ja vaatekihi vahel.

Sellisena rakenduse kolmeks kihiks jagamine muudab lihtsamaks halduse ja komponentide taaskasutamise erinevate rakenduste vahel. Kihtide omavahelise tiheda sõltuvuse puudumise tõttu on Oracle ADF teenuspõhise arhitektuuriga.



Joonis 6. ADF arhitektuur.[18]

ADFi litsentseerimine

Kõiki võimalusi toetava ADFi täis versiooni saab kaasa Oracle Application Serveri litsentsiga. Kasutajad, kes ei soovi tarkvara kasutada Oracle tootega, saavad osta ka eraldi ADFi litsentsi. Aastal 2008 ostis Oracle ettevõtte BEA Systems ja sai selle tehninguga WebLogic tarkvara omanikuks, seetõttu on ADFi litsents ka Weblogic tarkvaraga kaasas.[6]

Oracle ADF Essentials on litsentsipakett Oracle ADF tehnoloogiatest, mida võib kasutada tarkvara arendamiseks ja juurutamiseks ilma litsentsikuludeta.

Oracle ADF Essentials pakett sisaldab järgmisi tehnoloogiaid (rohelised moodulid joonisel 6) :

- **Oracle ADF Faces** paksu kliendi komponendid – Koosneb üle 150'st JSF komponendist, mis lihtsustavad paksu veebipõhise kasutajaliidese loomist. Lisaks sisaldab ADFi andmete visualiseerimise tarkvaralisi vahendeid.

- **Oracle ADF Controller** – On kiht, kus integreeritakse JSF ja ADF mudel. ADF kontroller on lisa standardsele JSF kontrollerile, pakkudes lisa funktsionaalsust, nagu taaskasutatavaid ülesannete voolu (*task flow*), mis annab üle kontrolli JSF lehtede vahel. Lisaks on ADF kontrolleri eesmärgiks vahendada ülesannete voolu teiste komponentide vahel nagu näiteks meetodi kõned (*method calls*) või teised ülesannete voolud.

- **Oracle ADF Binding** – On kiht, mille eesmärk on tagada lihtsam moodus ühendamaks kasutajaliides äri teenustega, kasutades metaandmete abstraktsiooni kihti.

- **Oracle ADF Business Components** – On kiht, mille eesmärk on muuta lihtsamaks äri teenuste arendamine relatsioonilisele andmebaasile. See kiht pakub deklaratiivselt konfigureeritud ja taaskasutatavaid komponente, mis implementeerivad üldlevinud disaini mustreid.

Kasutada saab ka tasuta Oracle integreeritud arenduskeskkondi: Oracle JDeveloper ja Eclipse tarkvaras olevat Oracle Enterprise Pack. Mõlemad pakuvad spetsiifilisi funktsionaalsusi, mis muudavad lihtsamaks tarkvara arendamise Oracle ADFi tehnoloogial.

ADFi täislitsentsi omades on võimalus kasutada järgmisi peamisi funktsionaalsusi:

- Oracle ADF Mobile – on Javal ja HTML5'l töötav mobiilirakenduste arendamise raamistik, mis annab võimaluse arendada või laiendada firmarakendust Apple iOS ja Android platvormile. [1]

- Oracle ADF Desktop Integration – ühendab ADFi rakenduse töölaua tarkvaradega nagu näiteks Microsofti Excel. Arendajad saavad kiiresti luua integreeritud Excel tabeleid, mille kaudu saavad kasutajad andmeallikas (näiteks andmebaasis) sisestada ja muuta kriitilisi äriandmeid.

- Oracle ADF Security – on autentimise ja autoriseerimise raamistik. ADF Security vajab töötamiseks Weblogic serverit, millega on tihedalt seotud läbi OPSS (Oracle Platform Security Services) teenuse. Essentials litsentsi omanikel on see võimalik asendada JAAS (Java Authentication and Authorization Service) turvalisuse mudeliga.[3]

- Enterprise Manager and Mbeans (Java Management Beans) – on vahendid rakenduse jõudluse seireks ja rakenduse seadistuse konfigureerimiseks WebLogic serveris.

Järgnevalt loetlen ADF tehnoloogia olulisemad eelised ja puudused.

Oracle ADF tehnoloogia eelised:

- **Tehnoloogia kasutamine ei kohusta kasutama kindlat andmebaasi või serveri toodet:** Paljud teiste raamistikud seovad arendajad kindlate tarkvara tootjate ja toodetega, kuid ADF ei tööta ainult Oracle toodetega. Seda on võimalik panna tööle kõikidele Java EEd toetavatele rakendus serveritele ning tarkvara andmeid saab kasutada kõikidest SQL-92 standardil töötavate andmebaasidega[16].

- **Tehnoloogia baseerumine Java keelel:** ADF raamistiku kasutamisel on oluline, et kaasatud töötajad omaksid kogemust Java arenduses. Google otsingut kasutades on võimalik leida mitmeid graafikuid populaarsematest programmeerimise keeltest. Paljude aruannete andmetel on Java hetkel populaarsuselt teine programmeerimiskeel (näiteks ettevõte TIOBE aruandes [30]), sellest võib järeldada, et Java arendaja leidmine ei ole probleem praegu ega ka lähitulevikus.

- **Toetab teenusele orienteerituse põhimõtteid:** ADF tehnoloogiat saab kasutada erinevate andmebaasi ja serveri toodetega. ADF ei ole tihedalt seotud Oracle toodetega. Antud tehnoloogia töötab Java programmeerimiskeelel, see universaalne keel ühildub paljude teenuste ja tehnoloogiatega ning on kasutatav erinevatel paltvormidel.

Oracle ADF tehnoloogia puudused:

- **Pikk tehnoloogia õppimise periood:** Lihtsaid rakendusi saab ADF raamistikuga kiiresti ja vähese pingutusega luua. Keerulisemate rakenduste loomiseks kulub tehnoloogiat mitte tundval arendajatel küllaltki palju aega. Keskmiselt kulub töötajal 3-6 kuud, et saavutada piisav produktiivsus ADF rakenduse arendamises, eeldusel, et töötaja omab kogemusi Javas veebirakenduste arendamises. See omakorda muudab arenduse kallimaks ja ajaliselt pikemaks.

- **Vähene populaarsus:** Tingituna nõutavatest litsentsitasudest ja suhteliselt hiljutisest tasuta versiooni välja andmisest ei ole ADF tehnoloogia saavutanud suurt populaarsust arendajate seas. Seetõttu on raske leida tööjõuturult ADF tehnoloogiaga töötamise kogemusega inimesi.

3.3. Oracle Application Express

APEX on tarkvara veebirakenduste juurutamiseks ja arendamiseks Oracle andmebaasile. Antud tarkvara on arendatud 1999. aastast alates. Esmalt loodi WebDB baasil Oracle Middleware tootegrupi kuuluv tarkvara Portal. Hiljem arendas Oracle WebDB tarkvarast toote HTML-DB, mis oli sisuliselt Portal tarkvara "õhem" versioon. Aastal 2006. andis Oracle tootele HTML-DB uue nime APEX. [12]

APEX on tarkvara, mida saab kasutada ilma litsentsitasu maksmata. Tarkvara on võimalik kasutada tasuta andmebaasi tootega Oracle Database Express Edition (Oracle XE) või tasustatud Oracle andmebaasiga.

APEX tarkvara kasutades ei ole arendajatel vajadust eraldi installeerida arendustarkvara, sest tarkvara arendus toimub kasutades veebilehitsejat.

APEXi arhitektuur:

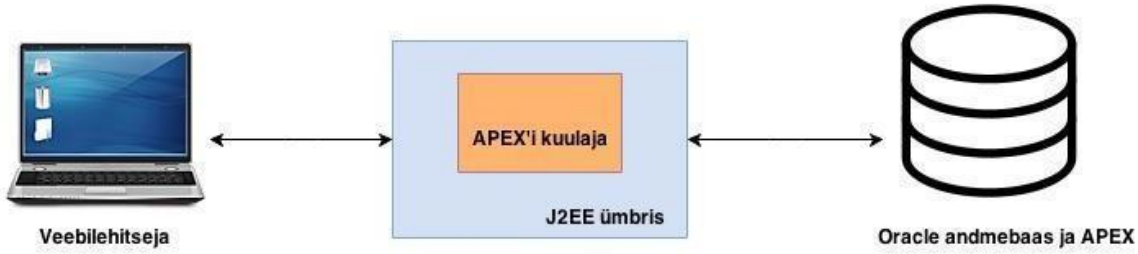
Antud tarkvara paikneb täielikult Oracle andmebaasis. Tarkvara koosneb ainult andmetest tabelites ja suurest hulgast PL/SQL koodist. APEX koosneb ligikaudu 215-st tabelist ja 200-st PL/SQL objektist, mis sisaldavad kolmsada tuhat rida koodi.[7]

Olenemata sellest, kas kasutatakse APEX arendus keskkonda või veebirakendust, mis on ehitatud APEX tarkvaraga, on protsess sama. Veebilehitseja edastab päringu URL aadressi, mis transleeritakse vastavaks APEXi PL/SQL kutseks. Järgmisena töötleb

andmebaas PL/SQL koodi ning saadab veebilehitsejale tagasi HTML koodi formaadis vastuse. Kirjeldatud tsükkel kordub igakord kui toimub veebilehekülje pärimine või saatmine.

Rakenduse sessiooni staatust juhitakse andmebaasi tabelite kaudu. APEX ei kasuta andmebaasiga suhtlemiseks püsiühendust, vaid loob iga järgneva päringu saatmiseks uue andmebaasi sessiooni, tagades sedasi minimaalse protsessori jõudluse kasutamise.

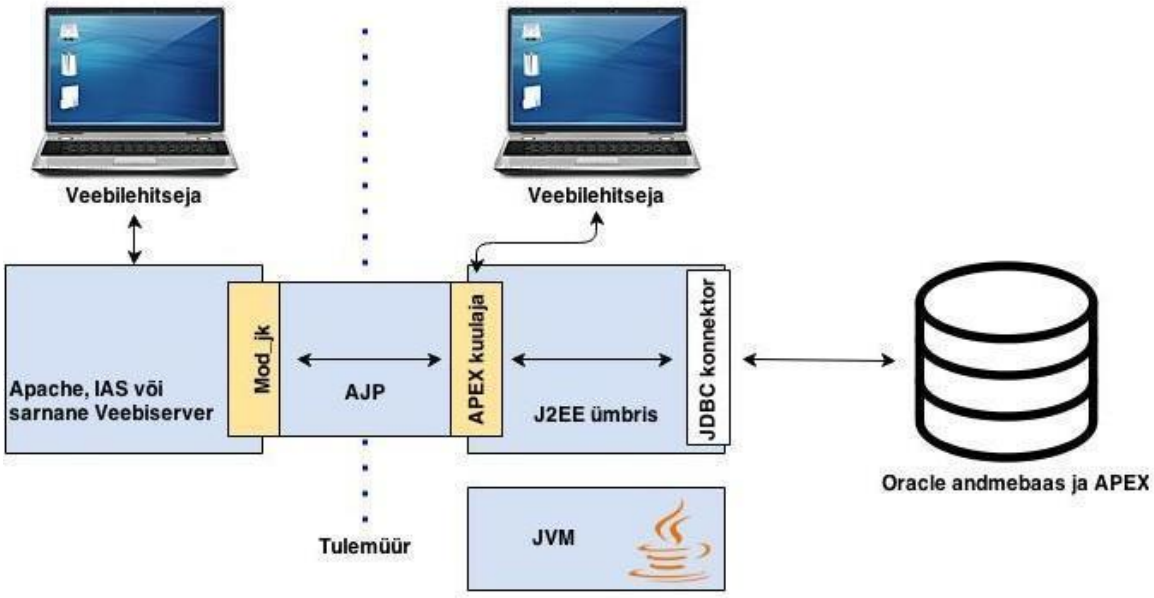
APEXit on võimalik kasutada kolme erinevat tüüpi server-klient lahendusena:



Joonis 7. APEX standardne arhitektuur.[19]

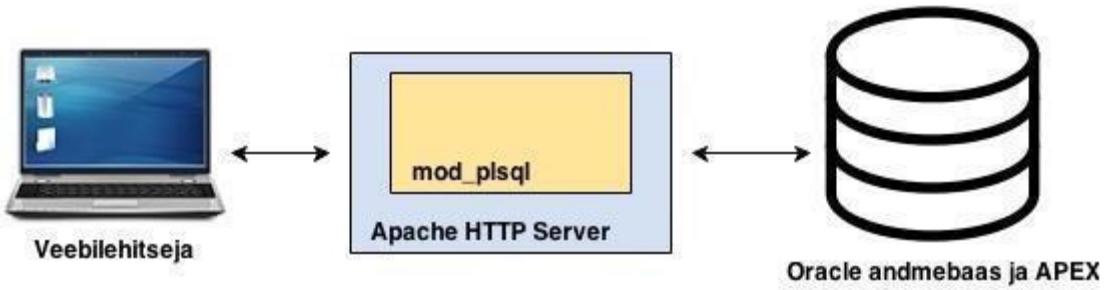
1.) Joonisel 7 on kujutatud APEXi kasutamise võimalus standardse lahendusena. Selline lahendus on mõeldud lokaalses võrgus kasutamiseks. Oracle APEX kuulaja on paigaldatud J2EE toetavale veebiserverile.

Joonisel 8 on kujutatud standardse lahenduse täiustamise võimalus. Antud täiustatud lahendus võimaldab teenust kasutada läbi välisvõrgu. Standardsele mudelile on lisatud komponendid, mis võimaldavad J2EE ümbrisel suhelda veebiserveriga läbi tulemüüri. Lokaalsed kasutajad võivad teenust kasutada tulemüüri tagant. HTTP kuulaja on paigaldatud tulemüüri ette. Välisvõrgust teenust kasutavad kliendid saadavad päringuid välisesse veebiserverisse, mis omakorda suhtleb läbi tulemüüri APEXi kuulajaga.



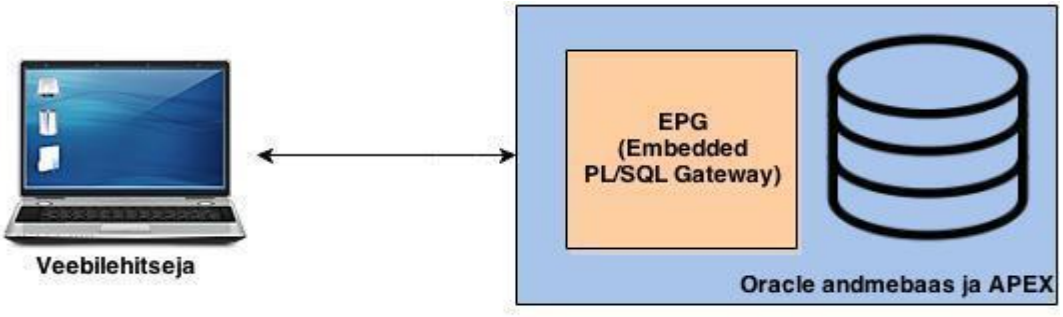
Joonis 8. APEX täiustatud standardne arhitektuur.[19]

2.) Joonisel 9 kujutatud lahenduses kasutatakse Oracle HTTP veebiserverit, millele on lisatud Apache mod_plsql moodul. Andud moodul võimaldab serveril otse andmebaasi ja APEXiga suhelda. Oracle HTTP server on veebiserveri komponent populaarsest Oracle Fusion Middleware tarkvara paketist.



Joonis 9. APEX arhitektuur Oracle HTTP veebiserveri ja Apache mod_sql mooduliga.[19]

3.) Joonisel 10 on kujutatud lahendus, kus APEX teenuse kuvamiseks klientidele kasutatakse EPG Oracle andmebaasi manus komponenti. EPG sisaldab endas mõningaid mod_plsql tuumik omadusi. EPG paigaldatakse tööjaama koos andmebaasi installeerimisega. Antud komponenti on võimalik kasutada veebiserverina ja vajaliku infrastruktuuri osana, et luua dünaamilisi rakendusi.



Joonis 10. APEX arhitektuur Oracle EPG andmebaasi manus komponendiga.[19]

Järgnevalt loetlen APEX tehnoloogia olulisemad eelised ja puudused.

Oracle APEX tehnoloogia eelised:

- **Lihtne juurutada:** APEX'it on lihtne juurutada, kuna tarkvara on täielikult veebipõhine, sealhulgas ka tarkvaraarenduse keskkond. APEX tehnoloogial loodud tarkvara kasutajad ei pea eraldi midagi installeerima. Tarkvara kasutamiseks on vaja vaid avada teenus veebilehitsejas.

  APEX'ile tarkvara arendajal on võimalik kasutada palju valmiskomponente, mis muudavad arendamise kiiremaks.

- **Madalad litsentsitasud:** APEX tarkvara saab kasutada tasuta. Kulutustega tuleb arvestada, kui tahetakse kasutada tasulist Oracle andmebaasi versiooni.

Oracle APEX tehnoloogia puudused:

- **Liigne sõltuvus tarkvara omanikust:** APEX rakendusi saab luua ainult kasutades Oracle tarkvara vahendeid ning teenust on võimalik hoida töös ainult Oracle andmebaasi kasutades. Seetõttu on APEX tarkvara kasutaja tugevas sõltuvuses tarkvara omanikust.

- **Keeruline versioneerida arendatavat tarkvara:** APEX rakendusi saab eksportida skripti kujule ja neid skripte versioneerida, kuid eksporditud PL/SQL skriptid on pikad, raskesti loetavad ja mõistetavad. Seetõttu on raske teha muudatusi rakenduses skripti koodi kirjutades ning võrrelda ja kontrollida erinevaid lähtekoodide versioone.

- **Ei toeta täielikult teenusele orienteerituse põhimõtteid:** APEX tehnoloogia on tihedalt seotud Oracle andmebaasiga. APEX töötab ainult Oracle andmebaasis, aga andmete töötlust on võimalik teha mõndadest teistest andmebaasidest. Antud tehnoloogiale arendatakse rakendusi veebilehitseja APEX kasutajaliidese kaudu. Samas on APEX tehnoloogiat võimalik kasutada paljudes J2EE veebiserverites.

3.4. Java

Tarkvara arendamine kasutades Java keelt on olnud viimastel aastatel väga populaarne. Samas näitavad ka erinevad uurimused, et Java püsib enim kasutatavate keelte hulgas[30].

Aastal 2009. ostis Oracle ära ettevõtte Sun Microsystems ja sai selle tehinguga Java omanikuks. Sellest ajast alates on Oracle tarkvara arendustes olulisel kohal Javaga ühildumine. Ka Java keele arendamisel on olulisel kohal koostöö Oracle toodetega, see on ka üks põhjus, miks valida Oracle toodetega töötamiseks teiste programmeerimise keelte hulgast Java.

Eelnevates peatükkides kirjeldatud Oracle tarkvaraarendus keskkondade ja tehnoloogiate asemel on võimalik kasutada ka teisi Java raamistike ja arenduskeskkondi. Eclipse ja Netbeans on enim kasutatavad vabavaralised Java arenduskeskkonnad. Samas on Oracle arendanud ka oma Java arenduskeskkonna JDeveloper. JDeveloper omab mõningaid vahendeid Oracle teenustele rakenduste arendamiseks, mida teistel arenduskeskkondadel ei ole pakkuda. Kasutama peaks tarkvara, mis sobib kõige enam tarkvaraarendajale või mille kasutamist nõuab organisatsioon.

Oracle ADF Java raamistiku asemel võib kasutada mõnda teist raamistiku. Java arendajal on võimalus valida paljude erinevate raamistike vahel. Tänapäeval üks enim kasutatavaid Java raamistike on Spring. Kõnealune raamistik on valitud võrdlusesse, kuna antud raamistiku kogemus on organisatsioonis kõige suurem ning organisatsioon nõuab selle kasutamist teiste raamistike asemel.

3.4.1. Spring

Spring raamistiku esimene versioon ilmus juba 2003. aastal. Tänaseks päevaks üle 10 aasta vanune tehnoloogia on ennast edukalt tõestanud ja saavutanud suure populaarsuse.

Enne Springi arendati firmarakendusi JEE standardite järgi. Antud standardite järgi programeeritud rakendusi oli võimalus kasutada erinevates JEE rakendus serverites, operatsiooni süsteemist sõltumata. Lisaks pakkusid rakendus serverid tuge rakendustele veel mitmel erineval viisil. Rakendus serverid pakkusid teenuseid nagu: transaktsioonide juhtimist, sõnumite lähetamist, posti teenust, kataloogi liidest (JNDI) jne. Ainult JEEga ühilduv kood oli võimeline töötama nimetatud teenustega, eeldusel, et kood on kirjutatud JEE spetsifikatsioonis defineeritud liidesele [10]. Antud JEE standardite programmeerimine muutus keeruliseks. Komponendi (EJB) programmeerimine vajab juurutamise deskriptoreid (*deployment descriptors*) (konfigureeritud XML failides), kodu liideseid (*home interfaces*), kaug/kohalike liideseid, jne. Keerukust lisas erinevate tarkvaratootjate poolt nõutud erinevad juurutamise deskriptorid, mis muutsid keeruliseks rakenduse migreerimise erinevate tootjate vahel [10].

Teine probleem oli seotud JNDI otsinguga. Kui komponent vajas töötamiseks teist komponenti, siis tuli sellel komponendil otsida komponente, millega tal sõltuvus on. Sõltuvused oli kirjas nimes, seega tuli seoste nimed koodi või juurutamise deskriptori sisse kirjutada [10]. Lisaks oli alati olnud keeruline komponentide omavaheline suhtlemine erinevate tootjate JEE rakendus serverite vahel.

Viimane probleem, mis vajas lahendamist, oli JEE komponentide suurus ja kohmakus. Komponendid ei vaja alati tingimata kõiki teenuseid, mida võimaldab rakenduse server, kuid kuni ei olnud sellist sõltumatud APIt komponentide ehitamiseks tuli kasutada JEE komponente. Programmeerijad olid sunnitud kirjutama mitmeid liideseid, juurutamise deskriptoreid, mis tingisid omakorda selle, et rakendused olid suured ja kohmakad. Seetõttu eelistasid paljud programmeerijad kasutada tavalisi Java objekte XML konfiguratsioonide kirjutamisele. [10]

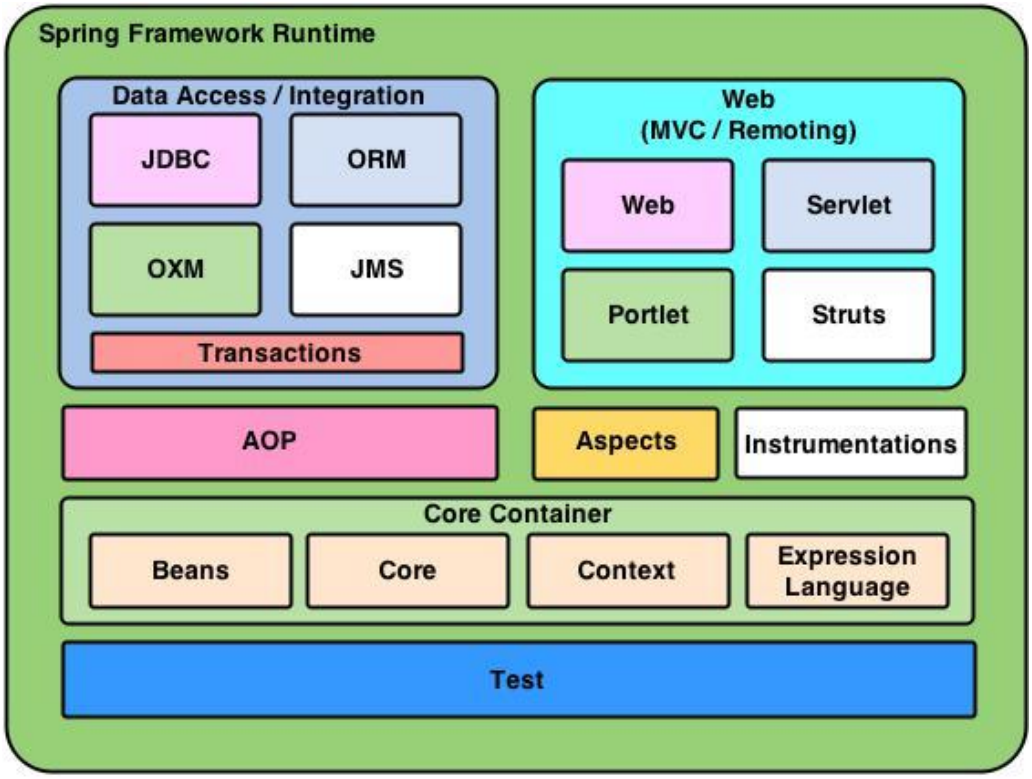
Kirjeldatud probleemide lahendamiseks loodi Spring raamistik. Spring raamistik tõi kerge ja dünaamilise konteineri JEE raske ja kohmaka konteineri asemele. Kerges Spring konteineris talletatakse kogu POJO elutsüklit. Komponendid on selles konteineris võimalikult kerged. Spring korraldab kõik komponentide ühendamised,

säästes sellest tööst programmeerijat. Erinevalt JEE komponentidest, kasutab POJO ainult vajalike komponente mudelisse kiindumata.

Spring on avatud lähtekoodiga raamistik, mis integreerub suuremate probleemideta kõigi teiste Java tehnoloogiatega ehk API'dega, mis oskavad kasutada lihtsat POJOt. IoC konteinerit kasutades saab integreeruda mitmete eessüsteemide arendamiseks mõeldud tarkvaradega nagu: Struts, JSF jne. Lisaks saab integreerida mitmete ORM raamistikega nagu: Hibernate, iBatis, Open JPA või JDBC abstraktsiooni kihiga [9]. Spring raamistik toetab deklaratiivset transaktsioonide haldust ja mitmeid võimalusi andmete esitlemiseks ning pakub täisfunktsionaalset MVC raamistiku [9].

Spring arhitektuur

Alloleval joonisel (Joonis 11) on näha Spring raamistiku mooduleid, mida on kokku ligikaudu 20. Need moodulid on grupeeritud tuumik konteineritesse: data access/integration, web, AOP, aspects, instrumentation, core container ja test [9].



Joonis 11. Spring raamistiku ülevaade.[20]

Spring raamistikule tarkvara arendamiseks on võimalik kasutada kõiki kolme antud peatükis mainitud arenduskeskkonda: JDeveloper, Eclipse, Netbeans või mõnda arenduskeskkonda, mida siin töös ei ole mainitud.

Alljärgnev tabel kirjeldab, millised on peamised erinevused ADFi ja Springi kasutades.

Tabel 2. ADF ja Spring võrdlus.

Kategooria	**ADF raamistik**	**Spring raamistik**
Lähtekood	Saab kasutada juba valmis olevaid kasutajaliidese komponente.	Tarkvaraarendaja peab ise kirjutama lähtekoodi kõikide kasutajaliidese komponentide jaoks.
SQL päringud	Deklaratiivset lähenemist kasutades genereeritakse vajalikud SQL päringud.	Tarkvaraarendaja peab kirjutama SQL päringud ja neid käivitama.
Arendamiseks kuluv aeg	Keskmiselt kulub tavalise CRUD moodulil põhineva süsteemi arendamiseks tarkvaraarendajal paarkümmend minutit.	Keskmiselt kulub tavalise CRUD moodulil põhineva süsteemi arendamiseks tarkvaraarendajal 3 tööpäeva.
Transaktsioonide juhtimine	Transaktsioonide juhtimist korraldab raamistik.	Transaktsioonide juhtimise ülesanne on tarkvaraarendajal.
Viisard	Palju viisardeid, tarkvaraarendaja töö lihtsamaks/kiiremaks muutmiseks.	Väga vähe viisardeid, mis aitaks tarkvaraarendajaid.
Valideerimine	ADFi on lisatud keerukas valideerimise süsteem, mis töötab mitmel erineval tasemel.	Valideerimise loogika tuleb kirjeldada spetsiaalsetes meetodites ning iga viga tuleb eraldi töödelda.

Turvalisus	Deklaratiivne lähenemine võimaldab tõsta turvalisust mitmel erineval tasemel.	Peab ise arendama kasutajaliidese kaudu sisselogimise.
Litsents	Avatud lähtekoodiga ja tasuta kasutamiseks.	ADF Essentials ja täisversioon ei ole kumbgi avatud lähtekoodiga. Kuigi täisversiooni kasutavad kliendid saavad soovi korral ADF lähtekoodi Oracle tugiüksusest küsida.[32]

Järgnevalt loetlen Spring tehnoloogia olulisemad eelised ja puudused.

Spring tehnoloogia eelised:

- **Madalad litsentsitasud:** Spring tehnoloogia kasutamine on tasuta ning selle tehnoloogiaga arendatud tarkvara on võimalik seadistada töötama erinevate komponentidega. Näiteks on võimalik kasutada veebiservereid, mille eest ei tule maksta litsentsitasusid.

- **Populaarsus Java arendajate seas:** Nagu eelpool mainitud on Spring üks populaarseimaid Java raamistike. Sellest tulenevalt ei ole keeruline seda tehnoloogiat valdavate Java arendajate leidmine. Probleemide korral leiab internetist palju erinevat informatsiooni, abimaterjale ja probleemi lahendusi.

- **Toetab teenusele orienteeritust:** Spring tehnoloogia toetab teenus orienteeritud süsteemide loomist, seda saab kasutada paljude erinevate rakendus serveritega, andmebaasidega ja teiste tarkvarade ja tehnoloogiatega.

Spring tehnoloogia puudused:

- **Keerulisem ühilduvus Oracle toodete ja keskkonnaga:** Teised võrdluses olevad tehnoloogiad on tehtud nii, et need oleksid Oracle taustaga arendajatele lihtsad kasutatavad. Võrreldes teistega, on Springi kasutades keerukam taasluua

Oracle Forms tehnoloogiale loodud rakendust, mis koostöötaks Oracle andmebaasi ja teiste toodetega.

- **Ei toeta kasutajaliidese arendamist:** Springi kasutades on keeruline arendada kasutajaliidest, seetõttu realiseeritakse see kasutades mõnda teist tehnoloogiat. Tehnoloogia lisamine tõstab süsteemi keerukust. Võrdluses olevad teised tehnoloogiad toetavad kõik kasutajaliidese arendamist.

- **Palju koodi kirjutamist:** Teisi võrdluses olevaid tehnoloogiaid kasutades on võimalik arenduses kasutada pukseerimise (*drag and drop*) lahendusi, mis muudavad arenduse kiiremaks ja mugavamaks. Samas on arendajaid, kellele meeldib kõik ise programmeerida.

3.5. Tehnoloogia valiku kriteeriumid

Rakenduse platvormi elutsükli loomulik osa on platvormi uuendamine. Platvormi uuendamisel tuleb analüüsida erinevaid võimalusi, kuna olemas oleva platvormi versiooni uuendamine ei pruugi olla parim lahendus.

Olulised tehnoloogia valiku kriteeriumid on:

Valitud tehnoloogia ei tohi tõsta teenuse halduskulusid. Teenuse halduskulud koosnevad peamiselt litsentsitasudest, tööjõukuludest ja muudest kuludest. Kui võrrelda kirjeldatud tehnoloogiaid litsentsitasude osas, siis ilma litsentsitasuta on võimalik kasutada Springi, ADFi ja APEXit. Samas tuleb ADFi eest maksta litsentsitasu kui on vajadus kasutusele võtta lisateenuseid. ADF ja tasuline tarkvara Forms on seega võrreldavatest kõige suuremate litsentsitasudega. Litsentsitasude madalal hoidmise soovikorral tuleks eelistada Springi või APEXit, kuid tehnoloogia valikul tuleb arvestada ka teisi kriteeriumeid.

Valitud tehnoloogia peab toetama asutuses rakendatavaid tarkvara arenduse reegleid. Oluline on kõik tarkvara arenduse komponentide versioonid oleksid hallatud oleksid. Selleks kasutatakse peamiselt SVN ja CVS versiooni halduse tehnoloogiaid. Versioonide haldamist on vaja, et oleks võimalik aru saada missuguseid muudatusi toodangusse paigaldatakse. Erinevad veaparandused (*bug fixes*) ja projekti arendused peavad olema arendatud erinevates harudes (*branch*) ja harude vahelised sõltuvused

peavad olema korrektselt paigas. Kirjeldatud versioonihalduse reegleid järgides saab olla kindel, et arendus keskkonnas tehtud arendused jõudsid õigel kujul toodangusse. Võrreldavatest tehnoloogiatest toetavad versioonihaldust kõik peale APEXi. Nagu APEXi peatükis sai miinuste all välja toodud, ei toeta antud tehnoloogia versioonipõhist tarkvara arendust.

Valitud tehnoloogia peab olema jätkusuutlik. Tehnoloogia jätkusuutlikust on keeruline hinnata, kuid samas on see äärmiselt oluline. Tarkvara realiseerimine väljasurevale tehnoloogiale on majanduslikult kallis ja seda tuleb vältida. On palju räägitud, et Oracle Forms tehnoloogia arendamise kinni paneb, kuid siiani on Oracle kinnitanud, et neil sellist plaani ei ole. Oluline ohumärk kinni panemise kohta on see, et Oracle andmebaasi arendus ja Formsi arendus ei käi enam viimased aastad ühte sammu. APEX ja ADF tehnoloogiad on tihedalt seotud Oracle põhitoote, milleks on andmebaas, uuendustega.

Valitud tehnoloogia peab võimaldama süsteemi realiseerimise madalate kuludega. Süsteemi arendamisel tuleb vältida tarkvara, mille juurutamine on mõnest teisest võrreldavast tehnoloogiast kallim. Eeldusel, et võrreldavad tehnoloogiad pakuvad võrdväärseid tehnilisi lahendusi. Java Spring raamistikule süsteemi rajamine nõuab kõige suuremaid kulutusi. Kõige odavam on aga uuendada Oracle Formsi versiooni, kuna versioonide vahelised erinevused on palju väiksemad kui erinevused erinevate tehnoloogiate vahel. APEX ja ADF tehnoloogiatele süsteemi realiseerimine nõuab rohkem aega, kuid kuna tegemist on Oracle toodetega, siis on nende ühildumine sama tootja andmebaasi ja süsteemiga lihtsam.

Valitud tehnoloogia peab toetama võimalikult palju teenusepõhist arhitektuuri. Eelnevates peatükkides kirjeldasin teenuspõhise arhitektuuri omadusi ning miks on tänapäeva tarkvaraarenduses oluline seda kasutada. Erinevad tehnoloogiad on ülesse ehitatud erinevatele arhitektuuridele, mis toetavad erineval määral SOA arhitektuuri. Kõige enam toetavad SOA arhitektuuri Java Spring raamistik ja ADF raamistik. Forms tehnoloogiat on uuemas versioonis arendatud SOA arhitektuurile lähedasemaks, kuid sellel on endiselt mitmeid puudusi. APEX tarkvara töötab ainult Oracle andmebaasis, tegemist on olulise miinusega arvestades teenuspõhise arhitektuuri põhimõtteid.

Valitud tehnoloogia peab olema kooskõlas asutuse tarkvara arhitektuuri plaaniga. Tehnoloogia peaks olema juba asutuse kasutusel või on selle kasutamine kooskõlas üldise arhitektuuri plaaniga. Uue tarkvara asutusse toomine on võimalik, kuid keeruline, selleks peab olema möödapääsmatu vajadus. Eespool välja toodud tehnoloogiad vastavad kõik antud kriteeriumile. Autor ei näinud vajadust lisada võrdlussesse tehnoloogiat, mille kasutamine ei oleks võimalik.

Eelnevat analüüsi arvestades on parim valik rakendus üle viia Oracle ADF tehnoloogiale.

4. Liisingu infosüsteemi platvormi vahetuse realisatsioon

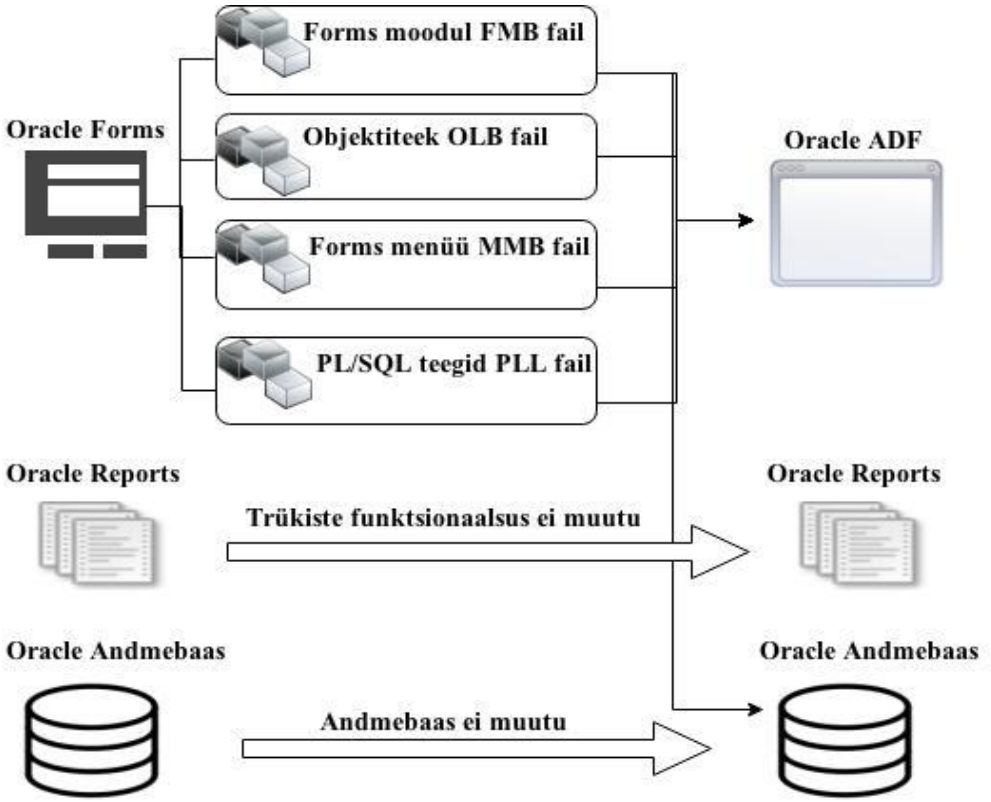
Selles peatükis esitab autor finantsettevõtte liisingu infosüsteemi ülevaadet, kus näitab LIS struktuuri ja tööd. Viimasena esitab autor võrdluse tulemusena välja valitud tehnoloogia realisatsiooni liisingu infosüsteemis, kasutades hetkel uusimat ADF 12c versiooni.

4.1. Liisingu infosüsteem

Liisingu infosüsteem on mõeldud finantsettevõttes liisingu lepingute haldamiseks. Tarkvara kasutab umbkaudu 3000 kasutajat, kõik kasutajad on firma töötajad. Kasutusel on LIS Baltimaade ja Skandinaavia turul, kõikides riikides, kus antud finants- ettevõte tegutseb. Tarkvara kasutajaliides on inglise keelne, et seda oleks võimalik kasutada võimalikult väikseid muudatusi tehes olemasolevatel ja tulevikus lisanduda võivatel turgudel. Mõningad turu spetsiifilised erinevused on lisatud kasutajaliidese vormidele. Kui kasutaja on ettevõtte Eesti haru töötaja, siis näeb ta Eesti töötaja kasutajaliidest.

Liisingu infosüsteemi kasutajaliidese kõrval on teiseks oluliseks osaks lepingute vormid. Lepingute vorm ehk trükis on vajalikus keeles lepingute muutumatutest tekstidest ja muutuvatest tekstidest automaatselt genereeritav lepingu vorm. LIS rakendus töötab Forms tehnoloogial, trükised genereeritakse Oracle Reportsi kasutades. Infosüsteemi andmebaas on rajatud Oracle 10gR2 andmebaasile.

Joonisel 12 on näha, kuidas toimub LIS rakenduse üle viimine ADF platvormile. Rakenduse üleviimisel ei muutu andmebaasi struktuur ja andmebaasi tarkvara, küll aga võib tekkida vajadus muuta või luua uusi protseduure ja funktsioone. Jätkub ka Oracle Reports tehnoloogia kasutamine, sest selle asendamine millegi moodsama ja mugavamaga nagu näiteks Oracle BI Publisher tehnoloogiaga on plaanis tulevikus.



Joonis 12. Forms rakenduse üle viimine ADF platvormile.

LIS infosüsteemi turvaklass (ISKE: Turvaklass) on T2S2R0K2; Turbeaste M (keskmine turbeaste)

T2 - info allikas, selle hävitamise ja muutmise fakt peavad olema tuvastatavad.

S2 – informatsioon on salajane, selle kasutamine on lubatud ainult teatud kindlatele kasutaja gruppidele, juurdepääs teabele on aktsepteeritav ainult õigustatud huvi korral.

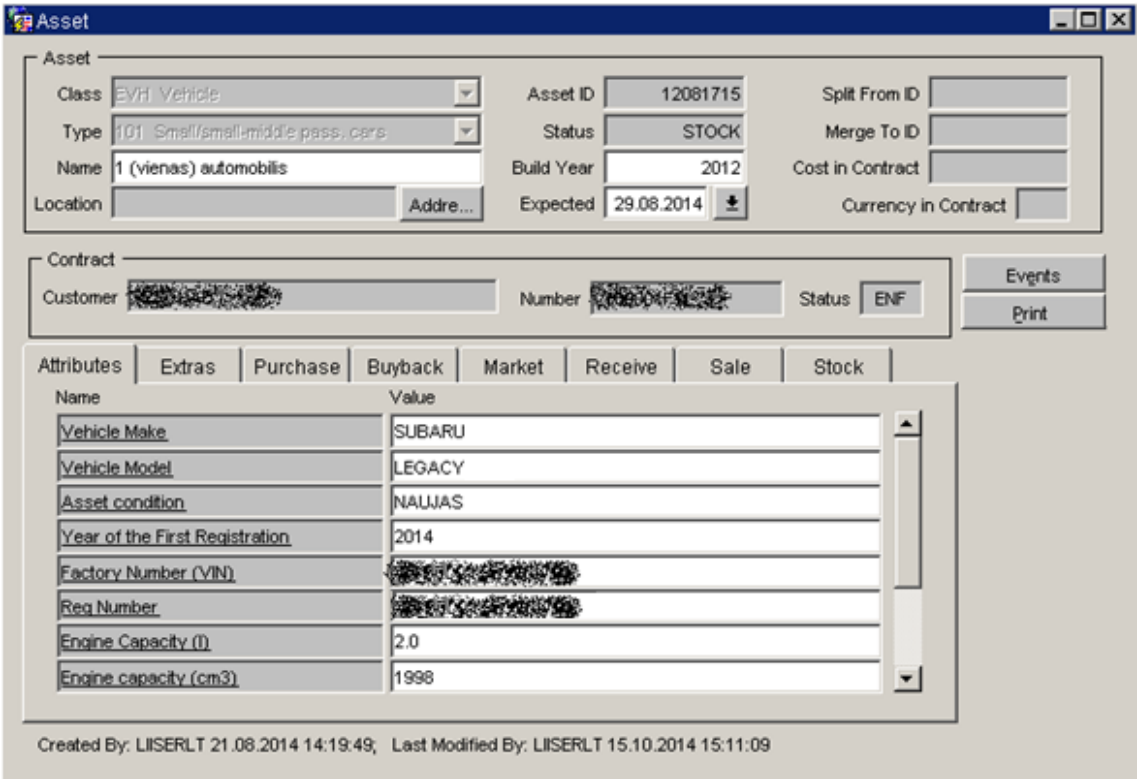
K2 – Oluline on teabe saamine tundide jooksul. Töökindlus peab olema üle 90% ning lubatud summaarne seisak nädalas ~ 2 tundi.

4.2. Realiseerimine

Liisingu infosüsteem koosneb 161’st erinevast Forms vormist ning ADF tehnoloogiale üle minnes tuleb need kõik ümber migreerida. Forms rakenduse ADF tehnoloogiale migreerimiseks on võimalik kasutada erinevaid tarkvaralisi tööriistu, näiteks JHeadstart, OraFormsFaces, PITS.CONN. Migreerimise tarkvarasid kasutades ei ole võimalik automaatselt üle viia kogu funktsionaalsust, vaja on küllaltki palju manuaalset

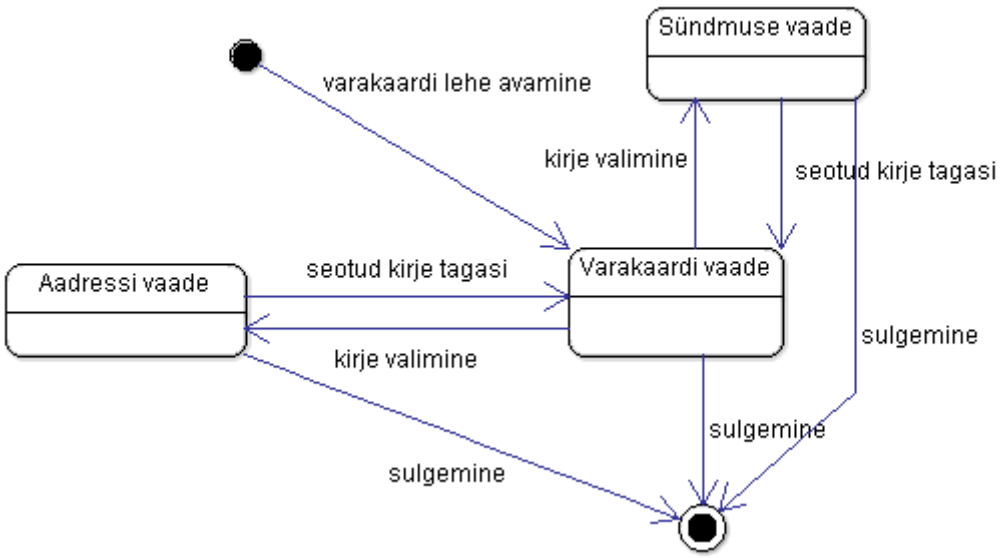
sekkumist migratisooni protsessi. Parema lõpptulemuse ja ADF tehnoloogiast arusaamise tagab manuaalne üleviimine.

Antud peatükis kirjeldab autor, kuidas üle viia varakaardi vorm, mis on LIS infosüsteemis üks olulisemaid vorme. Sellel vormil kuvatakse kasutajatele kõik liisinguvara puudutav informatsioon. Vastavalt lepingu olekule saab kasutaja sisestada või muuta andmeid. Alljärgnevalt on pilt üle viidavast vormist, kriitilisemad kliendi andmed on varjatud musta hägusjoonega.



Joonis 13. LIS varakaardi vorm.

Alljärgnevalt on lisatud kasutusjuhud varakaardi vormi kohta. Need on olekuskeem ja tegevusskeem. Antud skeemid aitavad visualiseerida ning paremini mõista kliendipoolseid tegevusi ning komponentide vahelist suhtlust.



Joonis 14. Olekuskeem.

Olekuskeem näitab erinevaid olekuid, mida läbivad süsteemis olevad objektid oma elutsükli jooksul. LIS varakaardi osas on meil 3 vaadet, milleks on varakaardi vaade, aadressi vaade ning sündmuse vaade. Varakaardi vaade on esialgne vaade, mis avatakse LIS lepingu vormilt. See pakub erinevaid valikuid, mille abil minnakse edasi aadressi vaatesse, sündmuse vaatesse või otsingu vaatesse. Aadressi vaates kirje valimisel liigutakse tagasi ning soovitud kirjetele. Eelnevalt välja toodud olekuskeem (Joonis 14) näitabki neid vaateid ning nendevaheliste liikumiste võimalusi.

Joonis 15 kujutab varakaardi vormi tegevusskeemi, mis näitab antud vormil toimuvaid sündmusi enne ADF tehnoloogiale migreerimist. Antud skeemil on kirjeldatud kolmekihilise arhitektuuriga Forms rakenduse tegevusskeem kus on kasutusel järgmised rollid:

Kasutaja – antud vormi kasutava isik.

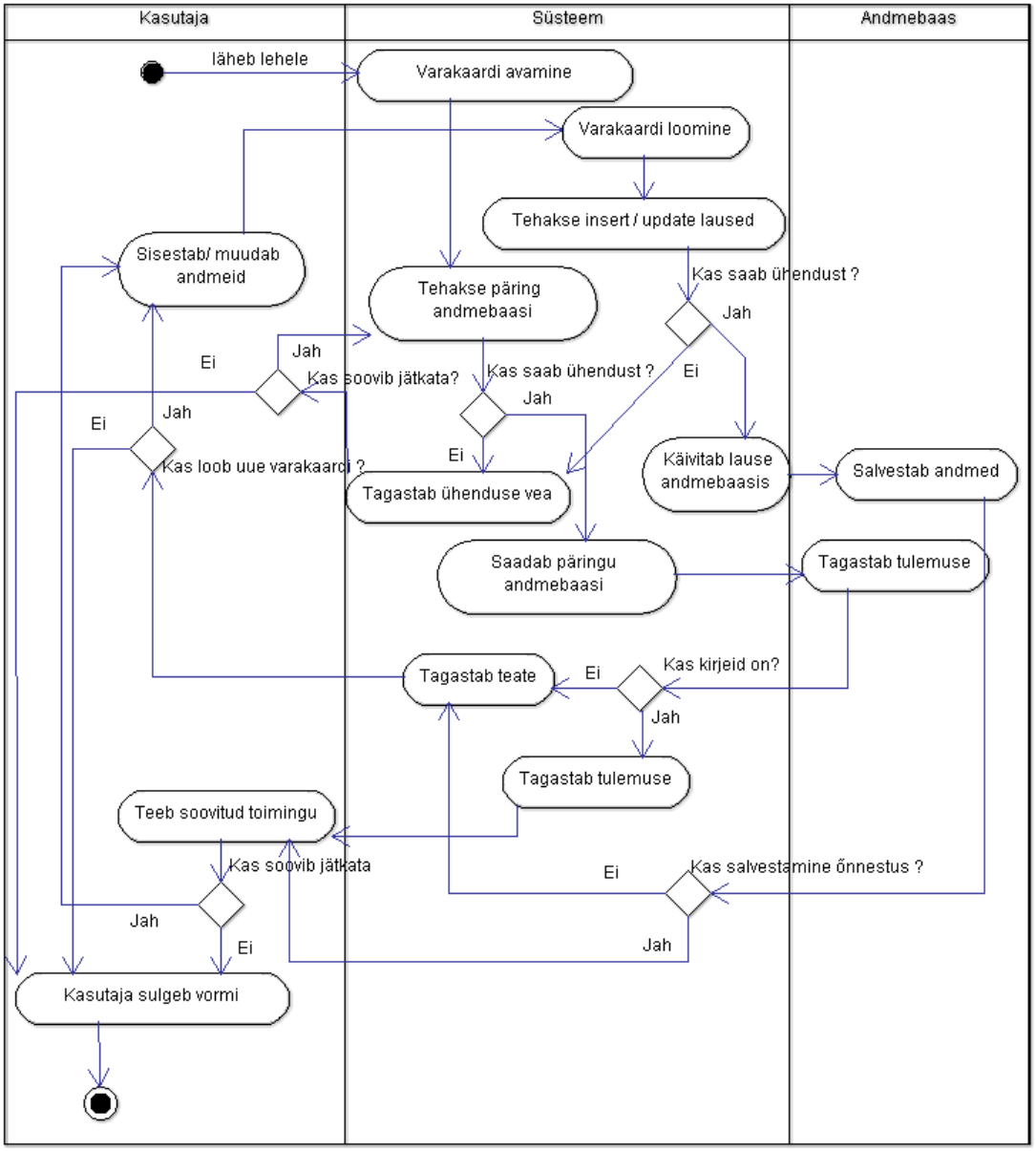
Süsteem – Oracle Formsi vorm ehk klient.

Andmebaas – Oracle andmebaas.

Rollidega tegevus skeemilt saab lugeda välja, milliseid konkreetseid tegevusi kes või mis teeb. Antud skeem kujutab süsteemi kasutamise protsessi, mis algab vormi avamisega. Kasutaja otsustab, millise tegevuse teeb ja vastavalt sellele reageerib

süsteem. Kui valik on tehtud, tehakse süsteemi ja andmebaasi poolt vajalikud toimingud ja kuvatakse kasutajale tulemus. Peale informatsiooni kuvamist on kasutajal võimalik teha soovitud toiminguid (näiteks printida varakaardi info). Tööprotsess lõppeb varakaardi vormi sulgemisega.

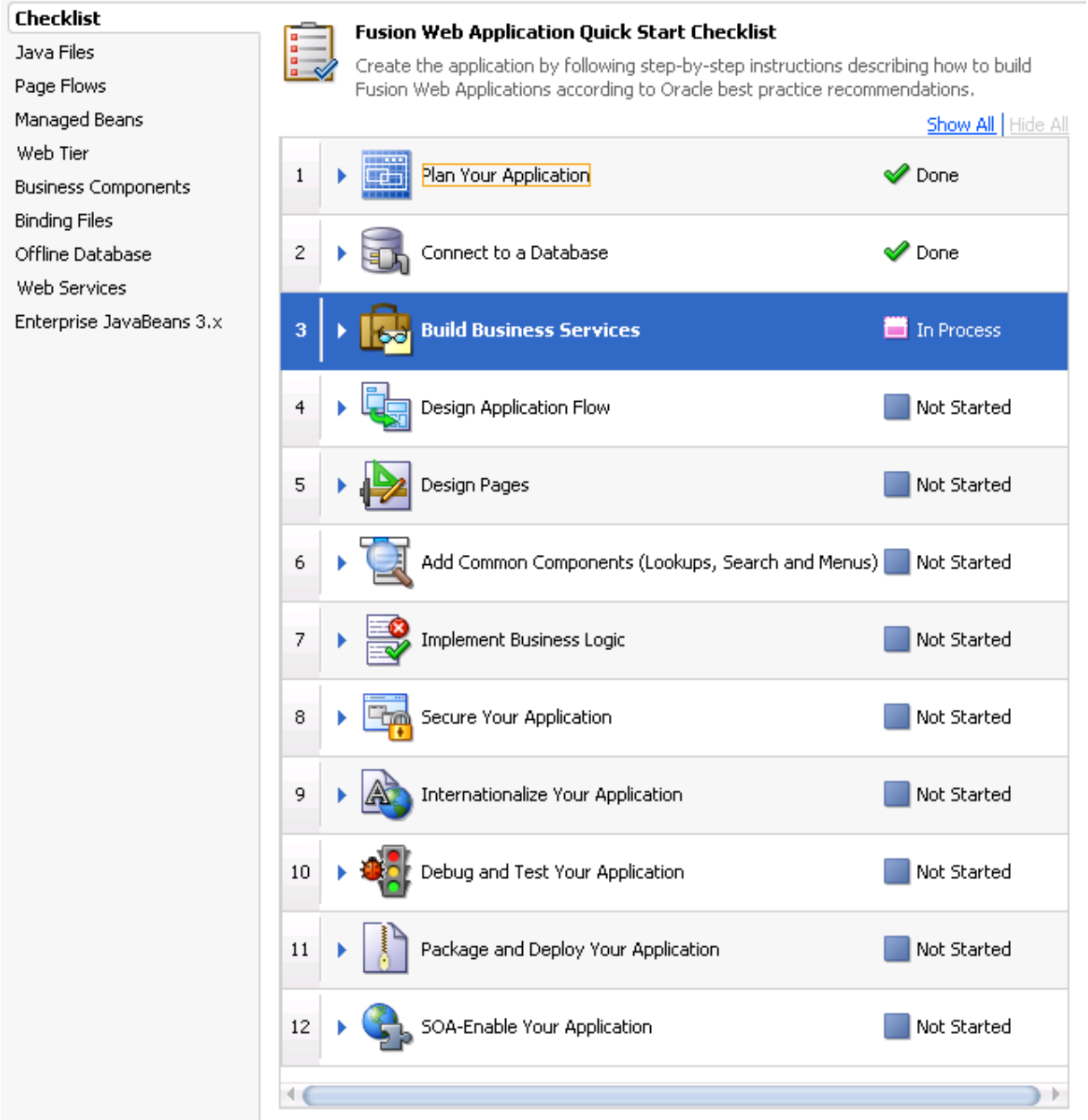
ADF tehnoloogiale üleminnes realiseeritakse olulised muutused süsteemi tegevustega ("Tehakse insert/update laused", "Tehakse päring andmebaasi", "Käivitab lause andmebaasis" jne.), mis asendatakse teenustega. Tegevusskeemil kirjeldatud tegevused jagunevad peale muudatusi kliendi ja veebiteenuse tegevusteks. Selle tulemusena muutub rakendus teenusele orienteeteritud arhitektuurile vastavaks.



Joonis 15. Tegevusskeem.

JDeveloperi kasutamine ADF rakenduse arendamiseks

Nagu peatükis 3.2. sai väljatoodud on ADF arendamiseks erinevaid vahendeid, antud töö raames kasutab auto JDeveloper 12c tarkvara. Alloleval pildil on näha JDeveloper nimekiri (*Checklist*), mis aitab arendada ADF rakendust. Nimekirjas olevate põhipunktide all on vastavad alampunktid. Nende punktide eesmärk on tagada, et arendatav rakendus valmiks kooskõlas parima praktikaga ja oleks SOA arhitektuuri toetav. Kuigi kõik punktid suunavad kasutama SOA arhitektuuri, siis on nimekirja viimases punktis veel eraldi välja toodud seda puudutavad tegevused.



Joonis 16. JDeveloper nimekiri (Checklist).

PL/SQL Protseduurid

Üks olulisemaid ülesandeid Oracle Forms rakenduse migreerimisel ADF tehnoloogiale, on otsustada, mida teha olemasoleva PL/SQL koodiga. Nagu eelpoolmainitud on ADF JEE baseeruv tehnoloogia ning seetõttu ei saa seda kasutada kõrvuti PL/SQL koodiga, nii nagu seda on võimalik Formsi rakendustes. Formsi rakendusse kirjutatud PL/SQL koodi kasutamiseks ADF rakenduses on kaks võimalust: viia kõnealune kood üle andmebaasi või realiseerida äriloogika vahevara (*middleware*) tasemel. Üks oluline erinevus nende kahe meetodi vahel on, et äriloogika realiseerimine andmebaasis seob rakenduse tihedamalt Oracle andmebaasiga. LIS andmebaasis on aga talletatud suures mahus äriloogikat PL/SQL protseduurides ja funktsioonides ning infosüsteemi Oracle andmebaasist sõltumatuks muutmiseks ei ole ettevõtel veel vajadust. Seetõttu viiakse kõik andmebaasiga seotud äriloogika üle andmebaasi.

Äriloogika andmebaasi viimine ei takista teenusorienteeritud arhitektuuri rakendamist. ADF tehnoloogia võimalusi kasutades luuakse äri komponendid mis kutsuvad välja andmebaasi protseduurid. Äri komponendi kasutamiseks tehakse SOAP teenus mida saab SOA komposiit rakenduse komponent käivitada.

Allpool on väljatoodud üks olemasolev PL/SQL protseduur, mis oli kirjutatud Forms rakendusse. Vormi ADFi migreerimisel viime antud protseduuri muutmata kujul üle andmebaasi.

```
PROCEDURE find_eqp_name(
  c_type  IN      VARCHAR
, c_lang  IN      VARCHAR
, c_name  OUT     VARCHAR) IS
  n_lang_txt NUMBER;
BEGIN
  SELECT lang_txt_id
    INTO n_lang_txt
    FROM hcl.eqp_equipment_type
   WHERE equipment_type_code = c_type;
  c_name := hcl.get_lang_txt(p_id     => n_lang_txt, p_lang => c_lang,
p_type => 'GEN');
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    c_name := '';
END;
```

Joonis 17. Protseduuri näide.

Joonisel 18 on välja toodud Java meetod, mille eesmärk on kirjeldatud talletatud protseduuri käivitada. Meetod edastab protseduurile vajalikud sisendparameetrid *c_type* ja *c_lang* ning PL/SQL protseduur tagastab *c_name* väärtuse, mida saab omakorda kasutada äri komponent. Kirjeldatud lahendus on ainuke võimalus kuidas soovitatakse kasutada talletatud PL/SQL protseduure ADF rakendustes.

```
public String callFindEqpName(String c_type, String c_lang) {
  CallableStatement st = null;
  try  {
    String stmt = "begin hcl.equipment.find_eqp_name(?,?,?); end;";
    st = getDBTransaction().createCallableStatement(stmt,0);
    st.registerOutParameter(3,Types.VARCHAR);
    st.setObject(1,c_type);
    st.setObject(2,c_lang);
    st.executeUpdate();
    StringBean result = new StringBean();
    result.setStringVal(st.getString(3));
    return result;
  } catch (SQLException e)  {
    throw new JboException(e);
  } finally  {
    if (st != null) {
      try {
        st.close();
      }
      catch (SQLException e) {}
    }
  }
}
```

Joonis 18. Java meetod talletatud protseduuri käivitamiseks.

Formsi trigerid

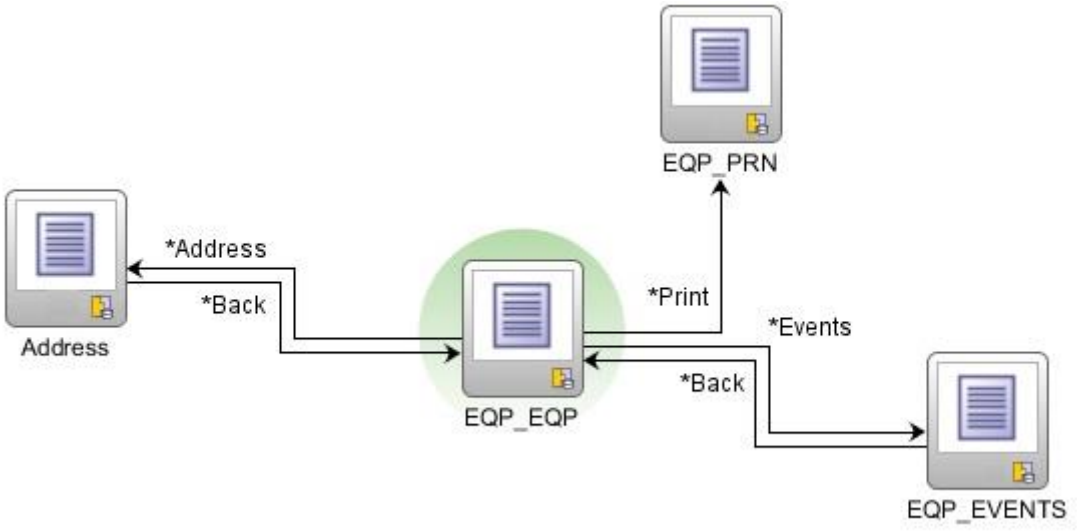
Forms rakendustes on kasutusel mitmed erinevad trigerid: vormi-, andmebloki- ja objekti trigerid. Trigereid võib jagada ka selle järgi, millist eesmärki need täidavad. Enim kasutatavate trigerite vastandamine ADF tehnoloogia vastavate lahendustega on kirjeldatud Oracle dokumentides [13]. Varakaardi vormil on trigereid, millele ei pakuta antud dokumendis lahendust ADFis. Paljud trigerid asendatakse automaatselt, kuid on ka selliseid, mis tuleb ise üle tõsta, varavormil on triger *When-Button-Pressed.*

Alljärgnevalt on näide, kuidas Formsi rakenduses on realiseeritud triger *When-Button-Pressed* nupule „Print“. Nupule vajutamise järel avab antud triger teise vormi. Enne vormi väljakutsumist täidetakse ära vajalikud parameetrid (*svc_id, cnt_id, EQP_NAME, EQP_TYPE, EQP_ID*).

```
DECLARE
    pl_id   ParamList;
BEGIN
  pl_id := Get_Parameter_List('P');
  IF Id_Null(pl_id) = FALSE THEN
     Destroy_Parameter_List(pl_id);
  END IF;
  pl_id := Create_Parameter_List('P');


  Add_Parameter(pl_id, 'svc_id', TEXT_PARAMETER, :GLOB.SRV_ID);
  Add_Parameter(pl_id, 'cnt_id', TEXT_PARAMETER, :GLOB.CNT_ID);
  Add_Parameter(pl_id, 'EQP_NAME', TEXT_PARAMETER, :POHI.C_EQP_NAME);
  Add_Parameter(pl_id, 'EQP_TYPE', TEXT_PARAMETER,
:POHI.C_EQUIPMENT_TYPE_CODE);
  Add_Parameter(pl_id, 'EQP_ID', TEXT_PARAMETER, :GLOB.EQP_ID);

  Call_Form('Eqp_Prn',NO_HIDE,NO_REPLACE,NO_QUERY_ONLY,pl_id);


END;
```

Joonis 19. Formsi When-Button-Pressed trigeri näide.

Joonisel 20 on näha, millist visuaalset tööriista pakub JDeveloper ADF Task Flow arendamiseks. Seda vahendit kasutades on võimalik lihtsalt realiseerida rakenduse erinevate vaadete vahelist navigeerimist. Üleval välja toodud Formsi triger on realiseeritud Control Flow noolega, mille nimi joonisel 20 on *Print. Kõnealune Control Flow avab EQP_PRN akna, mille abil saab kasutaja välja trükkida varaga seotud andmed ja lepingud. ADF tarkvaras ei ole vaja Print akna avamisel ette anda parameetreid, kuna andmeid on võimalik jagada erinevate Task Flowde vahel.



Joonis 20. JDeveloper Task Flow realisatsioon.

Kasutaja autentifitseerimine

Oluline on piirata juurdepääsu rakendusele, et täita eelpool väljatoodud ISKE nõudeid. Lisaks peab olema kasutaja ka tuvastatav, et kogutav kasutaja tegevuse ja andmete logi informatsioon oleks nõuetele vastav.

Oracle Forms 10g toetab ainult Single Sign On (SSO) kasutaja autentifitseerimist ehk kasutajapõhist tuvastamist. ADF toetab erinevaid kasutajapõhiseid tuvastamise meetodeid nii SSOga kui ka ilma.

JAAS ehk Java kasutajapõhine tuvastamine ja juurdepääsukontroll on rakendusliides, mis defineerib, kuidas tagatakse turvalisus. Kasutaja tuvastamiseks kasutab autor JAAS HTTPS kasutajapõhist tuvastamist. ADF rakendusele lisati sisselogimiseks HTML lehekülg, oluline on, et allkirjeldatud tingimused oleksid täidetud. Siis on võimalik siduda antud HTML lehekülg ADF kasutajatuvastusega.

form action = j_security_check

sisend element nimega j_username

sisend element nimega j_password

Andmete muutmise logimine

Forms rakenduses kasutatakse andmete muutmise logimiseks allpool olevat lihtsat protseduuri. Antud protseduur väärtustab POHI andmeploki kaks parameetrit *n_user_activity_session_id* ja *c_user_activity_module_name*. Kõnealust protseduuri käivitatakse iga kord kui POHI andmeblokil käivitub järgmistest trigeritest: PRE-DELETE, PRE-UPDATE, PRE-INSERT. Logimise eesmärk on salvestada kasutaja sessiooni tunnus ja vormi nimi kui muudetakse andmeid vormil. Salvestatud andmeid kasutades saab siduda vormilt sisestatud andmed kindla kasutajaga.

```
PROCEDURE init_log_values IS
BEGIN

  :POHI.n_user_activity_session_id := :global.user_session_id;
  :POHI.c_user_activity_module_name := get_form_filename;

END;
```

Joonis 21. Formsi logimise protseduuri näide.

Java programmide arendamisel kasutatakse kõige enam Log4j Apache raamistiku. Seda on võimalik kasutada ka ADF rakendusega, kuid on olemas ka teine vahend, milleks on ADF Logger. ADF rakenduse loomisel on soovitatav kasutada viimast, kuna see pakub rohkem võimalusi ning JDeveloper viisardeid kasutades lihtsat logimise seadistamist.

Reports serveri kasutamine

Oracle Reports serveriga ühendamiseks tuleb arendada Java meetod. Antud töö tarvis loodud Java meetod on näha joonisel 22. Osade muutujate (*reportUrl*, *port*) väärtused võetakse konfiguratsioonist, teised väärtustab rakendus siis kui printimine vormilt käivitatakse. Rakendus on realiseeritud nii, et kasutaja saab trükkida ainult PDF dokumente. Reports Server koostab automaatselt trükise vastavalt ette antud parameetritele.

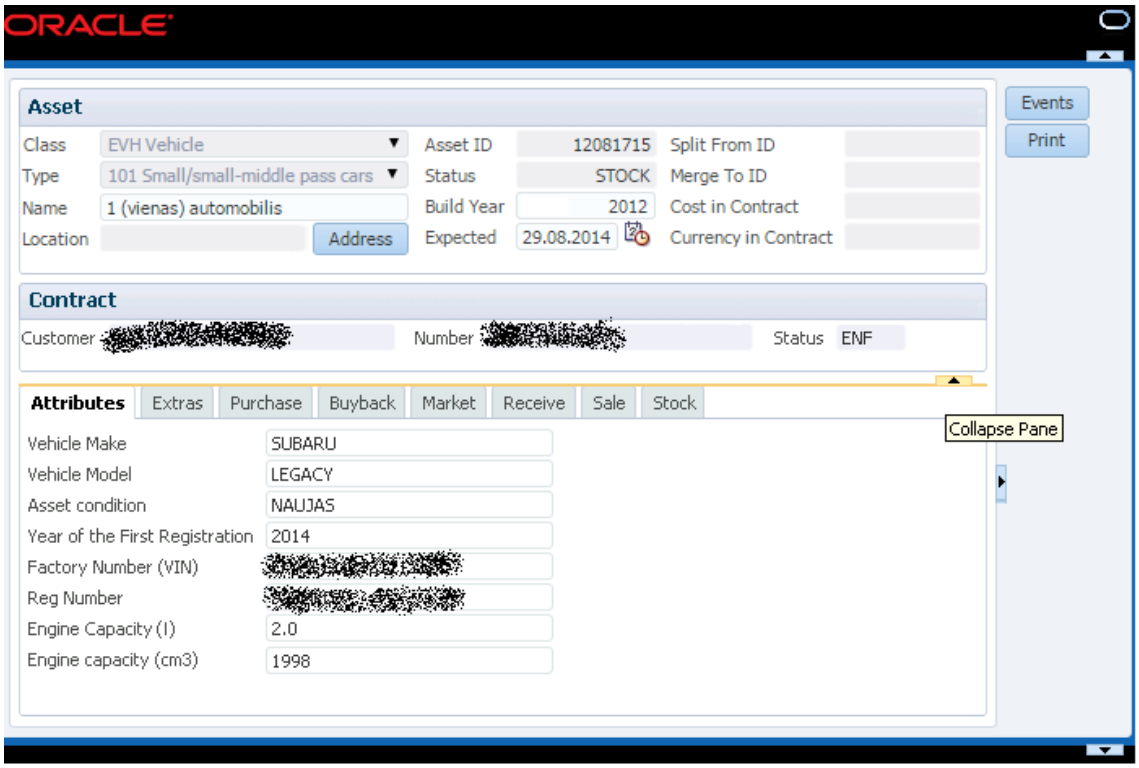
```
public String callReport(){
    OracleReportBean reportBean =
      new OracleReportBean(reportUrl, port, null);
    reportBean.setReportServerParam(OracleReportBean.RS_PARAM_DESTYPE,
      "file");
    reportBean.setReportServerParam(OracleReportBean.RS_PARAM_DESFORMAT,
      "PDF");
    reportBean.setReportID(report_id);
    reportBean.setReportType(report_type);
    reportBean.setReportCount(count);
    reportBean.setReportName(report_name);
    String url = reportBean.getReportServerURL();
    reportBean.openUrlInNewWindow(url);
    return null;
}
```

Joonis 22. Trükise printimise Java meetod.

Kasutajaliidese loomine

Kasutajaliidese komponentide paigutamine on Forms ja ADF tarkvarades erinev. Forms tarkvaras on kasutusel absoluutne positsioneerimine, seetõttu on komponentide paigutamine lihtne, kuna komponentide asukohad ei muutu. Puuduseks on see, et rakendus ei ole mugavalt kasutatav erineva ekraanisuurusega rakendustes. Näiteks väiksema ekraaniga mobiilses seadmes on seetõttu Forms rakendust keeruline ja ebamugav kasutada.

ADF tarkvara on komponentide paigutamisel dünaamilisem. ADF Faces RC pakub erinevaid komponente, et ehitada moodsaid ja dünaamilisi kasutajaliideseid. JDeveloper rakendusega saab kasutaja liidese erinevaid objekte lohistada ja paigutada sobivatele positsioonidele. Alljärgnevalt on näha varakaardi vormi ADF platvormile üleviiduna.



Joonis 23. Varakaardi vorm ADF platvormile üleviiduna.

Kokkuvõte

Antud magistritöö eesmärgiks oli leida parim tehnoloogia, millele üle viia aegunud tehnoloogial töötav Forms rakendus ja anda juhiseid, kuidas realiseerida kõnealust platvormi vahetust finantsasutuse liisingu infosüsteemi näitel.

Magistritöös püstitatud eesmärgi täitmiseks olid sõnastatud ja täidetud järgmised uurimisülesanded:

- Uurida, mis on rakendused, millised on neile esitatavad nõuded ja kuidas on nendega seotud tarkvara arhitektuur. Anda ülevaade teistest levinud tarkvara arhitektuuri tüüpidest ja võrrelda neid teenusorienteeritud arhitektuuriga ning uurida viimast detailsemalt.

- Analüüsida, millised on olulisemad erinevused vananenud Forms tehnoloogial ja teenusele orienteeritud arhitektuuril loodud rakenduste vahel.

- Analüüsida, mida pakub teenusorienteeritud arhitektuuri kasutusele võtmine ettevõtte erinevatele harudele: äri- ja IT poolele.

- Analüüsida erinevaid andmebaasi rakenduse loomise tehnoloogiaid, nende omadusi, arhitektuuri ja nõudeid ning pakkuda välja sobivaim.

- Rakendada pakutud tehnoloogiat praktikas, liisingu infosüsteemi peal.

Töö esimeses peatükis uurib autor, mis on rakendused, millest need koosnevad, millised on neile esitavad nõuded ning kuidas on rakenduse nõuetega seotud tarkvara arhitektuur. Veel uurib autor antud peatükis, millised on erinevad tarkvara arhitektuurid ja detailsemalt teenusorienteeritud arhitektuuri.

Magistritöö teises peatükis võrdles autor aegunud Oracle Formsi arhitektuuri teenusorienteeritud arhitektuuriga. Eesmärk oli leida peamised erinevused ning tuua välja teenusorienteeritud arhitektuuri eelised. Antud peatükis selgus, millised on peamised positiivsed tulemused teenusorienteeritud arhitektuuri kasutusele võtust äri- ja tarkvaraarenduse poolele.

Töö kolmandas osas analüüsis autor nelja erinevat tehnoloogiat, et leida kõige sobilikum, millele rakenduse funktsionaalsus üle viia. Analüüsi kaasati: Oracle ADF, Oracle APEX, Java Spring ja Oracle Formsi uusim versioon. Antud peatüki viimases osas kirjeldati peamised valiku kriteeriumid ja valiti välja tehnoloogia, millele olemasoleva tarkvara funktsionaalsus üle viia.

Neljandas osa rakendab autor pakutud tehnoloogiat praktikas finantsasutuse Liisingu infosüsteemi peal ehk viib varakaardi vormi ja sellega lähedalt seotud vormid üle Oracle ADF platvormile ning esitab kokkuvõtvalt tulemusi tehtud tööst.

Kokkuvõtteks võib öelda, et antud töö raames on leitud kriteeriumitele vastav platvorm, millele lähitulevikus kogu rakenduse funktsionaalsus üle viia. Antud töö annab hea ülevaate, kuidas realiseerida ülejäänud vormide platvormi vahetust ning on samas taaskasutatav ja laiendatav erinevatel ettevõtetel, kes soovivad rakendust Oracle ADF tehnoloogiale üle viia.

Kasutatud kirjandus

[1] Oracle ADF mobile data sheet. [WWW] http://www.oracle.com/technetwork/developer-tools/adf/overview/new-adf-mobile-9-29-clean-513078.pdf (03.08.2014)

[2] Gartner. Mark Driver. How to Maneuver Oracle Forms Into an Ideal Position for Next-Generation Challenges. (03.06.2007)

[3] Developing Secure Applications. [WWW] https://docs.oracle.com/cd/E16162_01/user.1112/e17455/dev_secure_apps.htm#OJDUG3733 (11.08.2014)

[4] Gartner. Mark Driver. Modernization and Migration Strategies for Oracle Forms (2011)

[5] Oracle. Shay Shmeltzer, Duncan Mills. From Apache Beehive to Oracle's Application Development Framework (Oracle ADF). (2008)

[6] Gartner. Peter Wesche, Jane B. Disbrow. BEA Customers Should Seek Contractual Protections Before Acquisition by Oracle (2008)

[7] What is APEX ? [WWW] http://www.oracle.com/technetwork/testcontent/what-is-apex-099128.html (14.11.2014)

[8] Gartner. Charles Abrams, Roy W. Schulte. Service-Oriented Architecture Overview and Guide to SOA Research. (2008)

[9] Spring Source. Introduction to Spring Framework [WWW] http://docs.spring. io/spring/docs/3.1.x/spring-framework-reference/html/ overview.html (05.07.2014)

[10] Nico Mommaerts and Pieter Degreauwe. The Spring Framework. [WWW] http://www.methodsandtools.com/archive/archive.php?id=93 (17.12.2014)

[11] James McGovern, Michael E. Stevens, Sameer Tyagi, Sunil Mathew. Java Web Services Architecture. (2003)

[12] Doug Gault, Karen Cannell, Patrick Cimolini, Martin Giffy D'Souza, Timothy St. Hilaire. Beginning Oracle Application Express 4.2 (2013)

[13] ADF Equivalents of Common Oracle Forms Triggers. [WWW] https://docs.oracle.com /cd/E14571_01/web.1111/b31974/appendix_formstriggers.htm#ADFFD1446 (23.11.2014)

[14] Jeroen Versteeg. A method to leverage legacy Oracle Forms applications in an SOA. (2008)

[15] Tarkvara. [WWW] http://www.vallaste.ee/index.htm?Type=UserId&otsing=1201 (23.11.2014)

[16] Oracle JDeveloper and ADF 12c (12.1.2) Supported Systems. [WWW] http://www.oracle.com/technetwork/developer-tools/jdev/documentation/1212-cert-1964670.html (29.11.2014)

[17] Conceptual model of a SOA architectural style. [WWW] http://www.ibm.com/ developerworks/library/ws-soa-design1/ (12.08.2014)

[18] Oracle ADF Key Components. [WWW] https://docs.oracle.com/middleware /1212/adf/ ADFCG/intro.htm#ADFCG116 (25.11.2014)

[19] Application Express Arcitecture [WWW] http://www.oracle.com/technetwork/ developer-tools/apex/apex-arch-086399.html (10.07.2014)

[20] Overview of Spring Framework. [WWW] http://docs.spring.io/spring/docs/3.0.x/ spring-framework-reference/html/overview.html (07.07.2014)

[21] Malcolm, Coxall. (2013). Oracle Quick Guides - Part 1 - Oracle Basics: Database & Tools.

[22] Oracle Form. [WWW] http://panzra.com/oracle-form-2/ (20.11.2014)

[23] Software Engineering Institute. What Is Software Architecture? [WWW] http:// www.sei. cmu.edu/architecture/ (25.11.2014)

[24] Sissejuhatus VBAsse ja programeerimisse. [WWW] http://www.tud.ttu.ee/~vilip/VBA _raamat /Programmeerimine/Prog_P.html (05.12.2014)

[25] Mitmekihiline arhitektuur. [WWW] http://www.e-uni.ee/e-kursused/eucip/arendus /1631_mitmekihiline_arhitektuur.html (05.12.2014)

[26] Riigi Infosüsteemi Amet. (2012) Mittefunktsionaalsete nõuete kirjeldamise juhend. [WWW] https://www.ria.ee/public/publikatsioonid/Mittefunk_nouded.doc (06.12.2014)

[27] Teenusorienteeritud arhitektuur. [WWW] http://www.e-uni.ee/e-kursused/eucip/ arendus/1632_teenusorienteeritud_arhitektuur.html (06.12.2014)

[28] The Art of Seperation of Concerns. (2008) [WWW] http://aspiringcraftsman.com/2008 /01 /03/art-of-separation-of-concerns/ (06.12.2014)

[29] James McGovern, Sameer Tyagi, Micheal Stevens, Sunil Mathew. (2003) Java Web Services Architecture.

[30] TIOBE. Index for December 2014 [WWW] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html (17.12.2014)

[31] IT Arhitektuur. [WWW] http://www.e-uni.ee/e-kursused/eucip/arendus/163_it_ arhitektuur.html (23.11.2014)

[32] Oracle. Oracle ADF Essentials and Overview and Frequently Asked Questions. (2013)

[33] Oracle. (2002) Oracle 9i Forms Migrating Client/Server To the Web.

[34] Oracle. (2014) Lifetime Support Policy: Oracle Fusion Middleware.