1. Peppi - Kokonaisarkkitehtuuri

Peppi-arkkitehtuuri on suunniteltu ratkaisemaan nykyisessä järjestelmäarkkitehtuurissa olevat ongelmakohdat (Esiselvitysvaiheessa tehdyssä dokumentissa "Miksi nykyinen arkkitehtuuri muodostaa haasteita").

Arkkitehtuuurissa on huomioitu valtiovarainministeriön vaatimukset järjestelmäarkkitehtuurille [Liite 2], Raketti-hankkeessa määritellyt opetuksensuunnitteluun liittyvät skeemat ja käsitteet sekä Kuali-yhteisön toteuttamat opetussuunnitteluun liittyvät järjestelmämoduulit. Peppi-arkkitehtuuri pohjautuu avoimen lähdekoodin tuotteisiin. Pääperiaatteena on tehdä arkkitehtuurista niin suoraviivainen, että palvelun kehittäjä voi helposti ymmärtää palvelun toimintaperiatteet sekä palveluiden ja asiakasohjelmien välisen viestinkulkuun liittyvät ratkaisumallit.

1. Arkkitehtuurin periaatteet

Peppi-arkkitehtuuri on suunniteltu palvelupohjaiseksi. Palvelupohjaisuus toteutuu seuraavien periaatteiden mukaan:

Sääntö 1, SOA-metodologia.

- Tietoa käsitellään palveluiden/palvelurajapintojen kautta.
- Palveluiden tulee olla autonomisia, toinen palvelu ei kontrolloi niiden toimintaa. Niitä voidaan ajaa hajautetusti. Ne eivät ole sidottuja toisen palvelun sisäiseen toimintaan.
- Palveluiden tulee olla löyhästi sidottuja toisiinsa, palvelut ovat sidoksissa toisiinsa vain rajapintojen kautta. Tällöin palvelun sisäinen toteutus on vaihdettavissa.
- Palveluita voidaan uudelleenkäyttää.

Sääntö 2, Standardeihin pohjautuvat rajapintaratkaisut

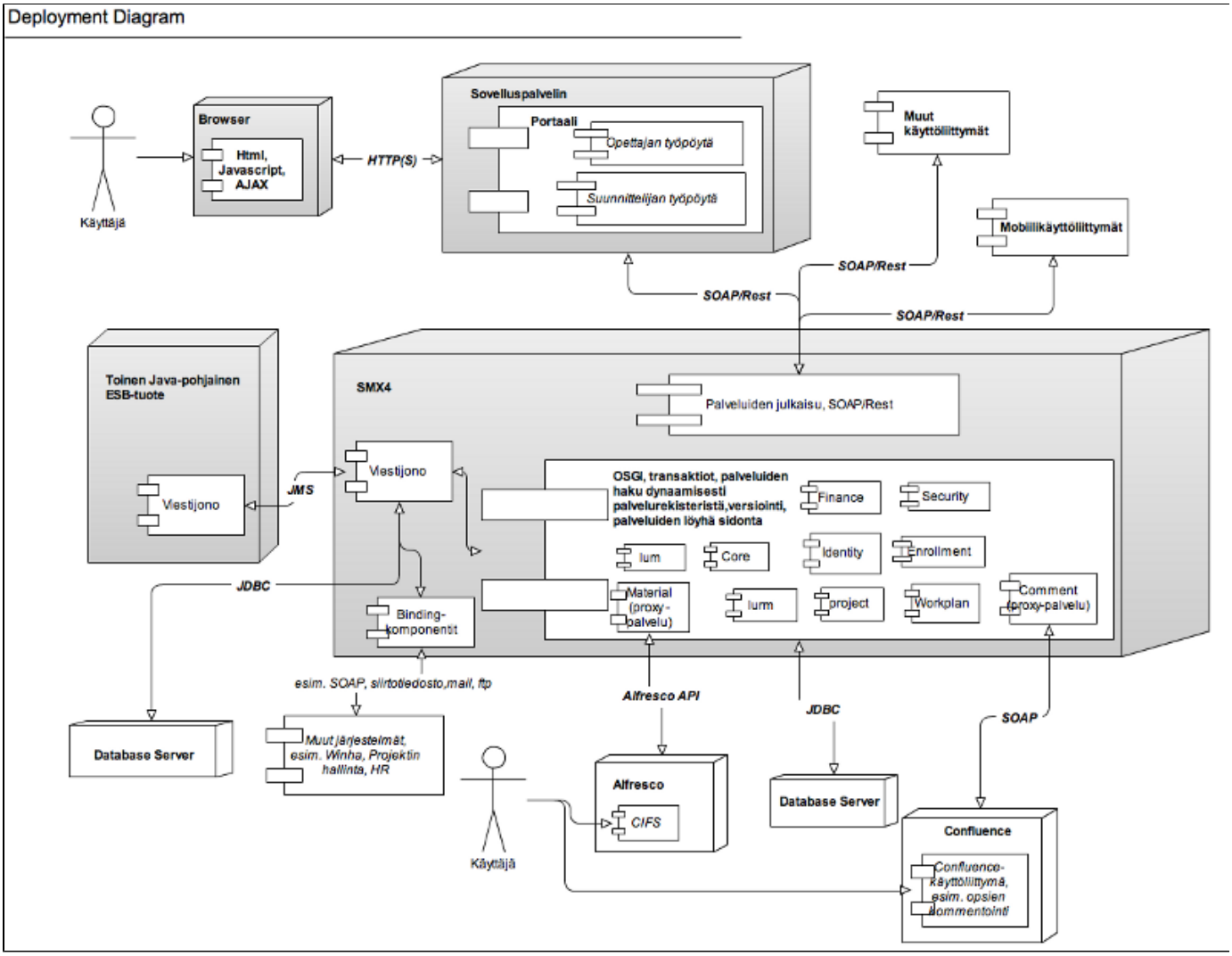
- Rajapinnat julkaistaan SOAP-pohjaisina Webservice-rajapintoina tai Rest tyyppisinä rajapintoina. Ollennaista on, ettei julkaistu rajapinta luo riippuvuutta mihinkään tiettyyn alustaan.

Sääntö 3, Palvelurajapintojen erottaminen käyttöliittymistä

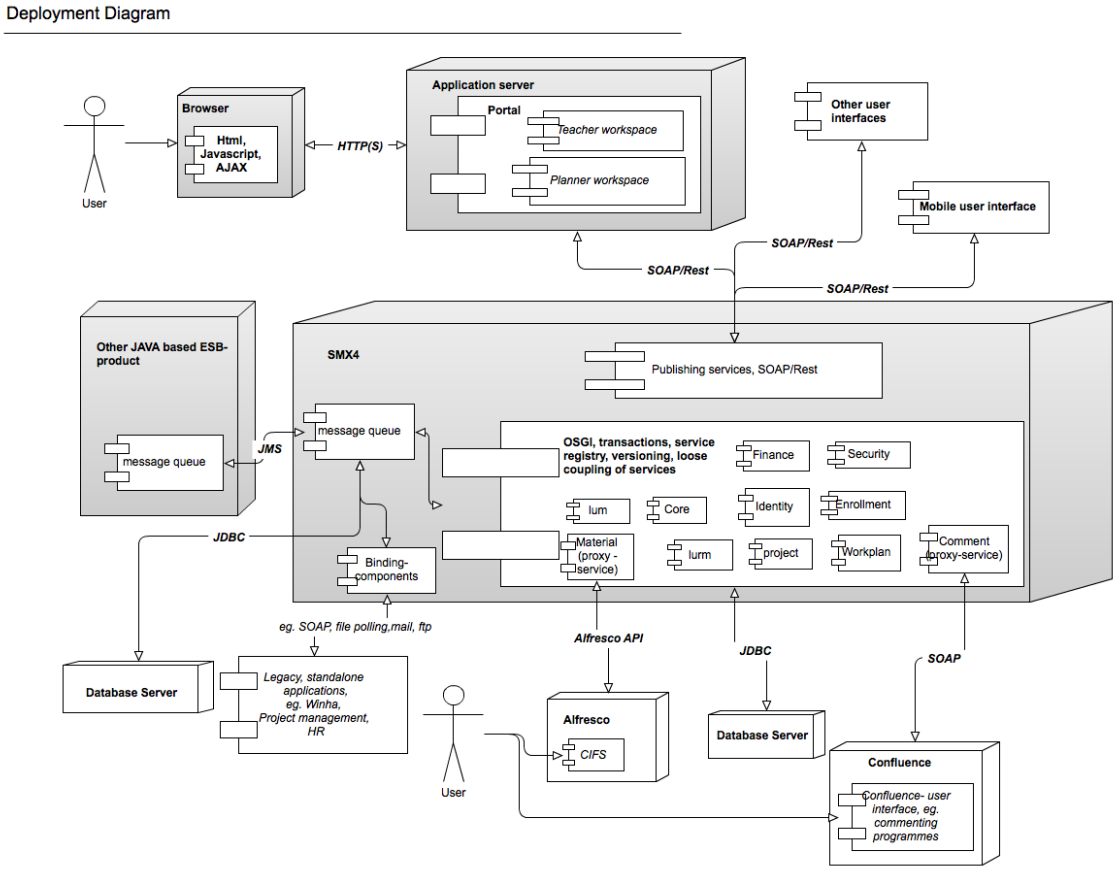
- Käyttöliittymiä ei ole sidota tiukasti palvelun sisäiseen toteutukseen, jolloin palveluita voidaan uudelleenkäyttää ja käyttöliittymiä voidaan uudistaa moduuli kerrallan.

2. Arkkitehtuuri

Peppi-arkkitehtuurissa palvelut on erotettu käyttöliittymistä erillisiksi moduuleiksi. Palvelurajapinnat julkaistaan SOAP/Webservice ja Rest -teknologioita käyttäen. Palvelumoduulit asennetaan OSGI-säiliön tarjoavaan Servicemix 4 -alustaan, jonka avulla voidaan tehdä integraatioita ulkopuolisiin järjestelmiin. Alfresco ja Confluence ovat kuvassa esimerkin vuoksi. Palvelun sisäinen tallennusratkaisu/toteutus piilotetaan muilta palveluilta.



Kuva 1. Peppi - Kokonaisarkkitehtuuri



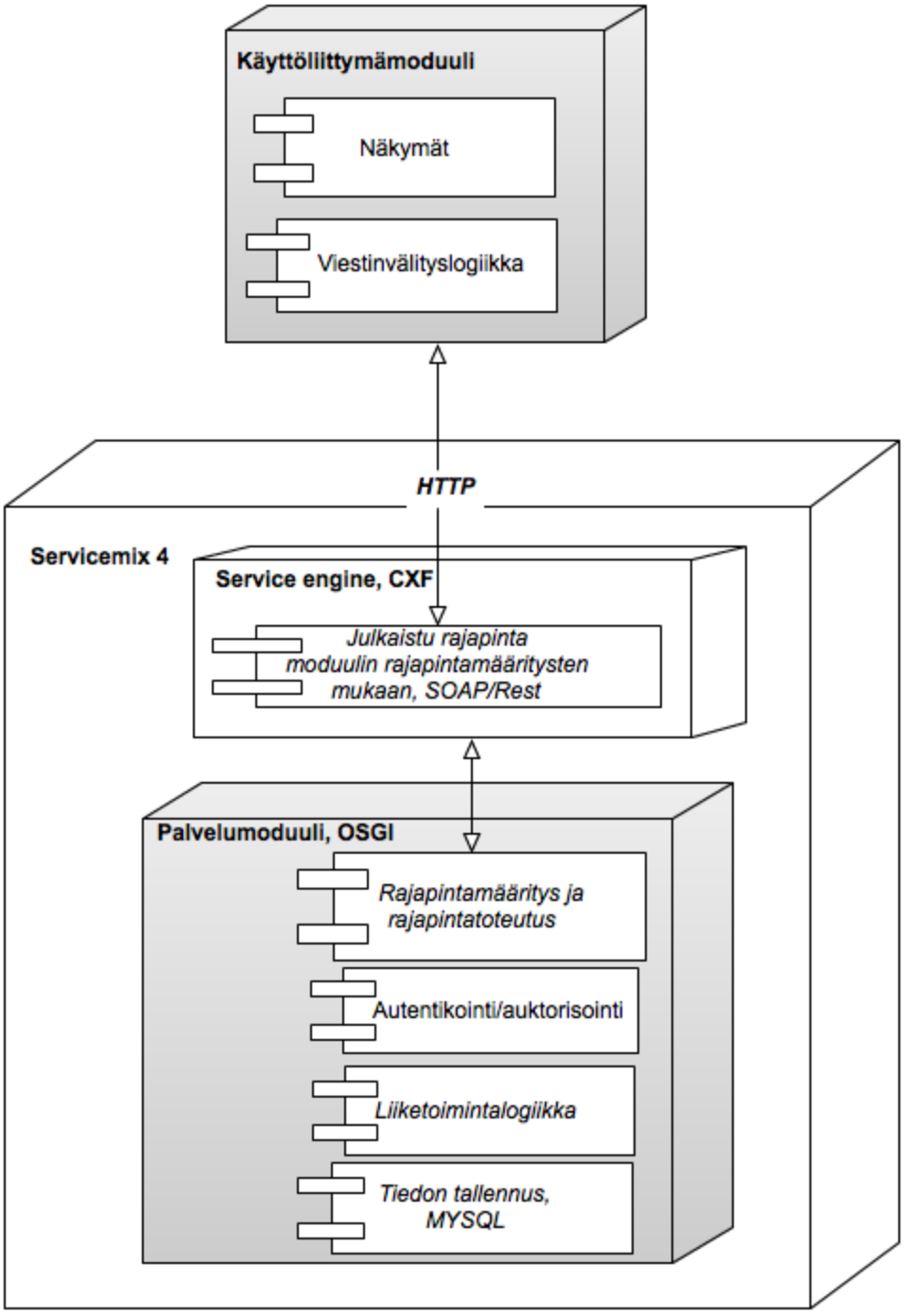
2.1 Palvelu- ja käyttöliittymämoduulit

Peppi-arkkitehtuurissa jokainen liiketoiminnan osio jakaantuu kahdeksi erikseen ajettavaksi moduuliksi:

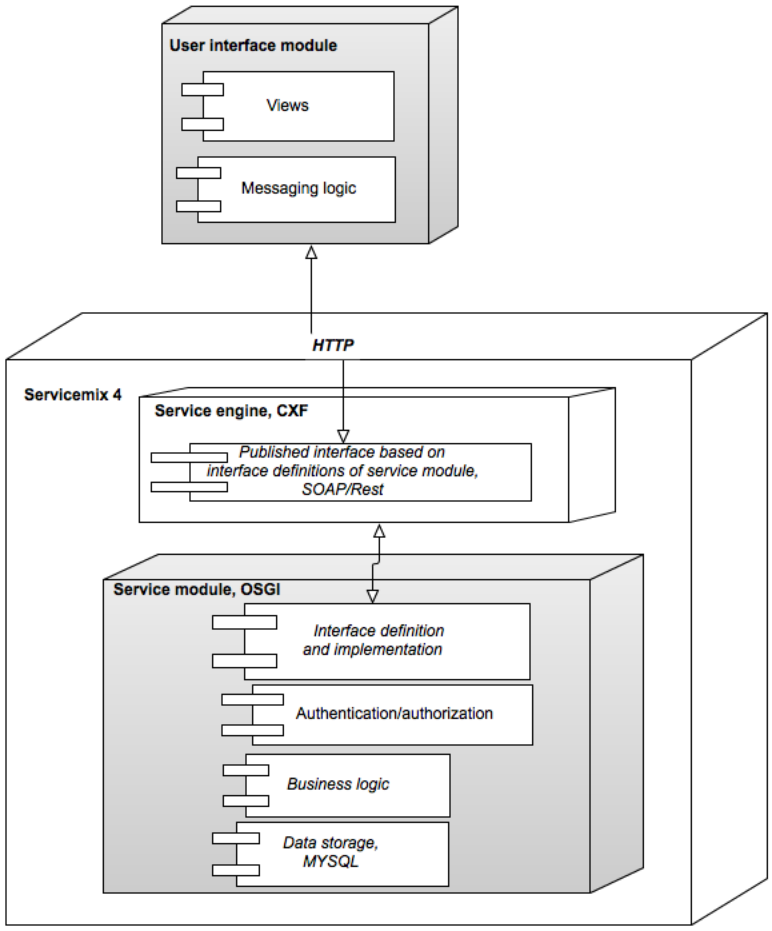
- palvelumoduuli
- käyttöliitymämoduuli.

Palvelumoduuli tarjoaa tiukasti toisiinsa liittyvät rajapinnat palvelun käyttämiseksi. Rajapinnat tehdään käyttäen yleisesti käytössä olevia standardeja, SOAP/WebService ja Rest.

Käyttöliittymät ovat erotettu palvelumoduuleista, jolloin niitä voidaan uudistaa ilman muutoksia liiketoimintalogiikkaan. Samoin voidaan tehdä kokonaan uusia käyttöliittymiä, jotka voivat helposti hyödyntää olemassaolevia palveluita/palveluiden rajapintoja.



Kuva 2. Peppi - Moduuulit



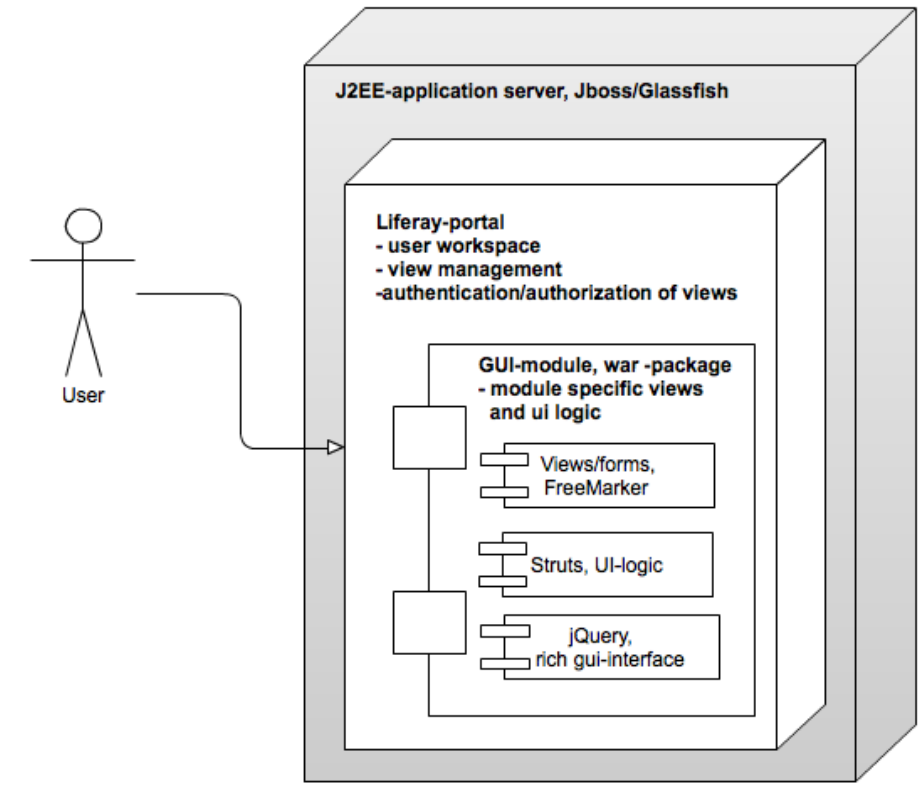
2.2 Näkymäkerros

2.2.1 Valitut teknologiat

Teknologia	Versio	Kuvaus	Peruste	
Java J2EE		Java-pohjainen alusta	Metropolialla ja Tamkilla tällä hetkellä käytössä. Javalle löytyy useita avoimen lähdekoodin tuotteita, jotka ovat hyödynnettävissä projektissa.	
Struts	2	Javalle tehty sovelluskehys, jota käytetään käyttöliittymän logiikan rakentamiseen	Pitkään koeteltu avoimen lähdekoodin sovelluskehys, joka integroituu Liferay-portaalin hyvin, myös Liferay-portaalin omat portletit on tehty käyttäen Struts-sovelluskehystä.	Struts
FreeMarker	2	Javalle tehty sovelluskehys, jonka avulla tehdään lomakkeet/näkymät.	Helpottaa ja nopeuttaa käyttöliittymän tekijän ja ohjelmoijan välistä yhteistyötä, koska ratkaisu ei vaadi html-mallipohjan muuttamista alustaan sopivaksi, ts. html-mallipohjaa voidaan hyödyntää portleteissa sellaisenaan.	FreeMarker
Liferay	6	java-pohjainen portaali, jonka avulla voidaan koostaa erilaisia työpöytiä/koostenäkymiä sekä hallita niihin liittyviä käyttöoikeuksia.	Helpottaa erilaisten näkymien organisointia. Ei vaadi uuden tuoteen käyttöönottoa, koska Liferay on sekä Tamkilla että Metropolialla käytössä.	Liferay
jqQuery		Javascript-kirjasto	Yleisesti käytössä oleva javascript-kirjasto, jonka avulla näkymistä ja niihin liittyvistä toiminnoista saadaan helppokäyttöisempiä.	jQuery

2.2.2 Käyttöliittymämoduulin paketointi ja suorituksenaikainen ympäristö

Liferay-portaalia varten täytyy olla J2EE-yhteensopiva sovelluspalvelin. Käyttöliittymät, konfiguraatiotiedostot ja käyttöliittymälogiikka paketoidaan war-pakkaukseen ja asennetaan Liferay-portaaliin.



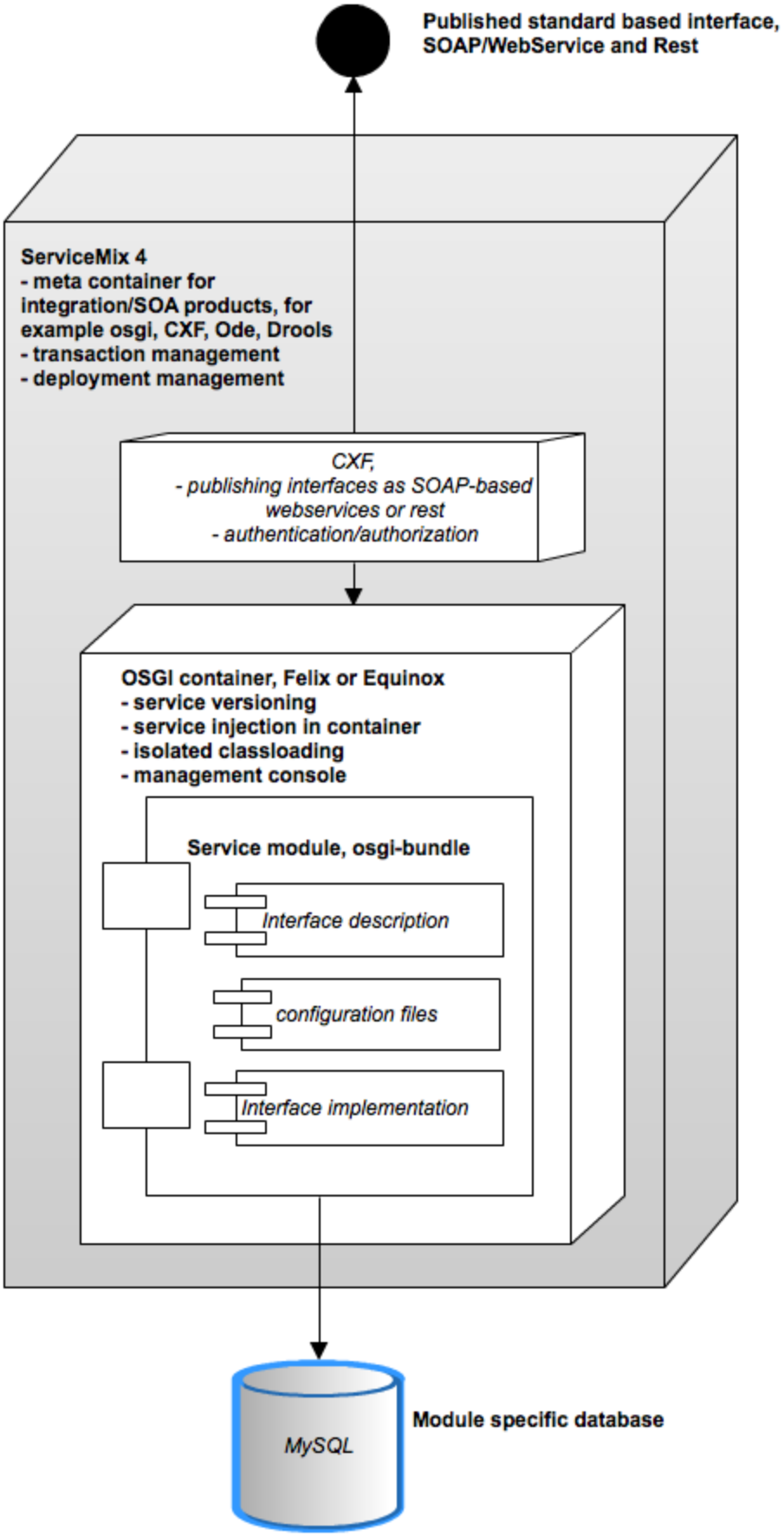
Kuva 3. Peppi - Näkymäkerros

2.3 Palvelukerros

2.3.1 Valitut teknologiat

Teknologia	Versio	Kuvaus	Peruste	
Java	6	Ohjelmointikieli, jolla palvelukerroksen komponentit on ohjelmoitu		
OSGI		Java-pohjainen plugin-teknologia	OSGI-mahdollistaa palveluiden versioinnin sekä ohjelmoinnin rajapintoja vasten OSGI-säiliön sisällä. ServiceMix alusta hyödyntää OSGI-teknologiaa.	OSGI
ServiceMIx	4	Integraatiotuote tai eri integraatiotuotteiden muodostama kokonaisuus, joka sisältää keskeiset toiminnot palvelupohjaiselle järjestelmälle	Peppi projektin aikana evaluoitiin eri integraatiotuotteita evaluointia varten tehtyjen skenaarioiden avulla. Servicemix 4 vastasi parhaiten palvelualustalle asetettuja tavoitteita.	ServiceMix
CXF		Kirjasto, jonka avulla rajapinnat voidaan julkaista SOAP Rest-tyyppisinä käyttöliittymille ja muiden järjestelmien käytettäväksi.	Suosittu avoimen lähdekoodin tuote, jolla voidaan julkaista helposti standardeihin perustuvia rajapintoja	CXF
Spring		inversion of control (IoC) -sovelluskehys, joka helpottaa eri komponenttien sitomista toisiinsa sekä ohjelmoitia tietokantaoperaatioissa ja transaktioiden käsittelyssä.	Suosittu avoimen lähdekoodin tuote, joka soveltuu hyvin käytettäväksi ServiceMix-alustassa	Spring
Camel		Komponentti viestin välitykseen ja reititykseen	Servicemix pitää sisällään tämän komponentin, Servicemixin jbi-pohjainen nmr on poistumassa ja Camelia tullaan suosimaan tulevissa versioissa	http://camel.apache.org/

2.3.2 Palvelumoduulin paketointi ja ajonaikainen ympäristö


Published standard based interface,
SOAP/WebService and Rest
ServiceMix 4
- meta container for
integration/SOA products, for
example osgi, CXF, Ode, Drools
- transaction management
- deployment management
CXF,
- publishing interfaces as SOAP-based
webservices or rest
- authentication/authorization
OSGI container, Felix or Equinox
- service versioning
- service injection in container
- isolated classloading
- management console
Service module, osgi-bundle
Interface description
configuration files
Interface implementation
MySQL
Module specific database

Kuva 4. Peppi - Palvelumoduulien paketointi

2.4 Rajapintojen toteutus

SOA-arkkitehtuurissa palvelut ovat tilattomia ja proseduraalisia, joukko operaatioita ja niihin liittyviä rakenteitä. Lähestymistapa on siten vastakohta Domain driven -tyyppiselle ohjelmoinnille, jossa oliot ovat tilallisia ja huolehtivat itse operaatioista.

2.4.1 Rajapintoihin liittyvät ongelmat

Sisäisten rakenteiden näkyminen rajapinnassa

Palvelurajapinnat ovat julkisia ja niihin liittyy useampia käyttöliittymiä tai integraatioita. SOA-arkkitehtuurissa täytyy huolehtia siitä, etteivät tietokantaan tehtävät muutokset heijastu rajapintaan ja riko siihen yhteydessä olevia käyttöliittymiä. Jos esimerkiksi tietojen tallennukseen on käytetty JPA (Hibernate) tekniikkaa, on huolehdittava siitä, ettei tietokantatauluja kuvaavia entity-olioita ole käytetty julkaistussa rajapinnassa parametreinä. Jos näin on tehty, rajapinta ei voi olla muuttumaton (tai ainakin erittäin vaikea pitää sellaisena), eikä ratkaisu tuo oikeastaan mitään lisäarvoa siihen että oltaisiin suoraan yhteydessä tietokantaan JDBC:n avulla. Tämä ongelma korostuu SOA-ratkaisussa, jossa lähtökohtaisesti rajapinnat ovat kenen tahansa muun palvelun käytettävissä (ei välttämättä ole edes saman toimittajan tekemiä).

Taaksepäin yhteensopivuus

Jossain palvelun kehityskaaren vaiheessa voi tulla tarvetta muuttaa julkaistua rajapintaa. Esimerkiksi halutaan tehdä uusi toiminto, jossa toimintalogiikkaa tulee muuttaa tai jokin tietokantakenttä poistetaan käytöstä. Tällöin on huolehdittava siitä, että rajapintaan vasten olevat käyttöliittymät ja integraatiot eivät rikkoudu ja että niiden päivittäminen voidaan tehdä hallitusti. J2EE:ssä ei ole mahdollisuutta versioida pakkauksia siten, että niitä voitaisiin ajaa samanaikaisesti muuten kuin pakkausten nimeä muuttamalla.

2.4.2 Peppi-arkkitehtuurissa käytettävä ratkaisu rajapintojen tekemiseen

Sisäisten rakenteiden näkyminen rajapinnassa

Peppi-projektin yhteydessä on tehty Kuali-moduulien selvitystyötä sekä havaittu tiettyjen Kualissa käytettävien ratkaisumallien olevan hyödyllisiä Peppi-arkkitehtuurissa. Seuraavaksi on käyty esimerkinomaisesti läpi Kuali Student -moduulin rajapintojen toteutus sekä arvioitu ratkaisumalleja käytettäväksi Peppi-arkkitehtuurissa.

Kuali Student -projektin rajapintojen toteutus

Rajapinta

LuService-rajapinta sijaitsee **ks-lum-api/src/main/java** kansiossa **org.kuali.student.lum.lu.service** pakkauksessa. Rajapinta huolehtii learning unit -olioihin, esimerkiksi opintojakso, liittyvistä toiminnoista. Rajapinta sisältää esimerkiksi seuraavan opintojakson perustiedot palauttavan metodin.

```
/**
     * Retrieves core information about a CLU
     * @param cluId identifier of the CLU
     * @return information about a CLU
     * @throws DoesNotExistException cluId not found
     * @throws InvalidParameterException invalid cluId
     * @throws MissingParameterException missing cluId
     * @throws OperationFailedException unable to complete request
     */
    public CluInfo getClu(@WebParam(name="cluId")String cluId) throws
DoesNotExistException, InvalidParameterException,
MissingParameterException, OperationFailedException;
```

Metodi palauttaa DTO-olion(**org.kuali.student.lum.lu.dto.CluInfo**), joka sijaitsee **ks-lum-api/src/main/java** kansiossa **org.kuali.student.lum.lu.dto** pakkauksessa.

Toteutus

Clu-luokka (canonical learning unit, joka kuvaa esimerkiksi opintojaksoa) on JPA-tekniikan avulla tehty rakenneluokka, jonka avulla tiedot voidaan tallentaa tietokantaan. Se sijaitsee **ks-lum-impl/src/main/java** kansiossa **org.kuali.student.lum.lu.entity** pakkauksessa.

LuServiceImpl toteuttaa LuService-rajapinnan ja sijaitsee **org.kuali.student.lum.lu.service.impl pakkauksessa.** LuServiceImpl luokka sisältää myös JAX-WS annotaatiot, jotta samaa luokkaa käytetään webservice-endpoint luokkana.

```
@WebService(endpointInterface = "org.kuali.student.lum.lu.service.LuService",
serviceName = "LuService",
portName = "LuService", targetNamespace = "http://student.kuali.org/wsdl/lu")
@Transactional(noRollbackFor={DoesNotExistException.class},rollbackFor={Throwable.clas
s})
public class LuServiceImpl implements LuService {
```

Rajapinnassa käytettävät rakenteet on eristetty DTO-olion avulla persistoitavasta Clu-oliosta siten että käytetään LuServiceAssembler luokkaa muuttamaan moduulin sisäinen Clu-olio rajapinnassa käytettäväksi CluInfo-olioksi.

getClu metodin toteutus:

```
 @Override
 public CluInfo getClu(String cluId) throws DoesNotExistException,
   InvalidParameterException, MissingParameterException,
   OperationFailedException {

  checkForMissingParameter(cluId, "cluId");

  Clu clu = luDao.fetch(Clu.class, cluId);
  return LuServiceAssembler.toCluInfo(clu);
 }
```

Assembler muuttaa olion toiseksi toCluInfo- metodin avulla

```
public static CluInfo toCluInfo(Clu entity) {
  if (entity == null) {
   return null;
  }
  CluInfo dto = new CluInfo();

  // copy all simple fields - exclude complex data types
  BeanUtils.copyProperties(entity, dto, new String[] {
    "officialIdentifier", "alternateIdentifiers", "descr",
    "participatingOrgs", "primaryInstructor", "instructors",
    "stdDuration", "luCodes", "credit", "offeredAtpTypes", "fee",
    "accounting", "intensity",
    "campusLocationList", "accreditationList",
    "adminOrgs", "attributes", "metaInfo", "versionInfo" });
  dto.setOfficialIdentifier(toCluIdentifierInfo(entity
    .getOfficialIdentifier()));
  dto.setAlternateIdentifiers(toCluIdentifierInfos(entity
    .getAlternateIdentifiers()));
  dto.setDescr(toRichTextInfo(entity.getDescr()));

  // accreditingOrg Deprecated in v 1.0-rc2 Replaced by Primary and
  // Alternate admin orgs
```

```
dto.setAccreditations(toAccreditationInfos(entity.getAccreditations()));

dto.setAdminOrgs(toCluAdminOrgInfos(entity
  .getAdminOrgs()));

dto.setPrimaryInstructor(toCluInstructorInfo(entity
  .getPrimaryInstructor()));
dto.setInstructors(toCluInstructorInfos(entity.getInstructors()));
dto.setStdDuration(toTimeAmountInfo(entity.getStdDuration()));
dto.setLuCodes(toLuCodeInfos(entity.getLuCodes()));

if (entity.getOfferedAtpTypes() != null) {
 List<String> offeredAtpTypes = new ArrayList<String>(entity
   .getOfferedAtpTypes().size());
 for (CluAtpTypeKey key : entity.getOfferedAtpTypes()) {
  offeredAtpTypes.add(key.getAtpTypeKey());
 }
 dto.setOfferedAtpTypes(offeredAtpTypes);
}

dto.setFeeInfo(toCluFeeInfo(entity.getFee()));
dto.setAccountingInfo(toCluAccountingInfo(entity.getAccounting()));

dto.setAttributes(toAttributeMap(entity.getAttributes()));
dto.setMetaInfo(toMetaInfo(entity.getMeta(), entity.getVersionNumber()));

dto.setType(entity.getLuType().getId());

if (entity.getCampusLocations() != null) {
 List<String> campusLocations = new ArrayList<String>(entity
   .getCampusLocations().size());
 for (CluCampusLocation cluCamp : entity.getCampusLocations()) {
  campusLocations.add(cluCamp.getCampusLocation());
 }
 dto.setCampusLocations(campusLocations);
}

dto.setIntensity(toAmountInfo(entity.getIntensity()));

dto.setVersionInfo(toVersionInfo(entity.getVersion()));

return dto;
```

```
 }
```

Kuten tästä metodista voi nähdä entity-olio ja rajapinnassa julkaistu Dta-olio sisältävät samat attribuutit ja operaatio tuntuu tarpeettoman monimutkaiselta. Tämä on kuitenkin keino SOA-arkkitehtuurissa huolehtia siitä, että julkaistu rajapinta ei muuttuisi. Toteutus käyttää Apachen BeanUtils-kirjastoa helpottamaan tietojen kopiointia dto-olioon.

Yhteenveto Kuali Student -projetissa käytettävälle ratkaisulle

Hyötyjä DTO-suunnittelumallissa ovat:

- Julkaistu rajapinta saadaan eristettyä moduulin sisäisitä rakenteista ja palvelun kehittämiselle jää pelivaraa
- Voidaan optimoida julkaistua rajapintaa, kaikki tallennettuja tietoja ei välttämättä tarvitse siirtää
- Voidaan tehdä toinen "näkymä", toinen julkinen rajapinta, joka käyttää eri dto-olioita

Heikkouksia:

- Lisää palvelun monimutkaisuutta ja tietoja täytyy muuttaa useampaan paikkaan.

Peppi-arkkitehtuurissa käytettävä ratkaisu

Ratkaisumallina noudatetaan Kuali Student-projektissa esitettyä tapaa, **rajapinnat eristetään moduulin sisäisistä rakenteista DTO-suunnittelumallin avulla**.

Rajapinnat tehdään omiin pakkauksiin moduulikohtaisesti.

Dto-olioiden rakentamisessa hyödynnetään BeanUtils-kirjastoa kuten Kuali student -projektissa. Toinen tapa rakentaa Dto-oliot on käyttää Builder-mallia. Tässä mallissa Dto-luokka sisältää staattisen Builder-luokan joka palauttaa Dto-olion.

```
public class CourseunitDto implements Serializable {

private String name; //setters and getters

private CourseunitDto() {}

public static class Builder {

   private CourseunitDto courseunitDto;

  Builder() {
     this.courseunitDto = new CourseunitDto();
  }
  public Builder name(String name) {

   if (name == null) {
      throw new IllegalArgumentException("Course should have a name");
   }

   this.courseunitDto.setName(name);
   return this;
 }

 public CourseunitDto build() {
   return this.courseunitDto; }
}
}
```

Tällöin dto-olio voidaan rakentaa seuraavasti.

```
new CourseunitDto.Builder.name("test").build();
```

Taaksepäin yhteensopivuus

Käytetään OSGI-teknologiaa tehdään palvelut OSGI-bundleina, joita ajetaan Servicemix4-alustassa. OSGI mahdollistaa jar-pakkauksen(OSGI-bundle) eri versioiden ajamisen rinnakkain.

2.5 Kehityksessä käytettävät työkalut

Työkalu	Kuvaus	Peruste	Linkki
Eclipse	Java-koodaamiseen tarkoitettu kehitystyökalu	Suosituin ja ilmainen työkalu Java-kehitykseen	http://www.eclipse.org/
Maven	Työkalu Java-koodin kääntämiseen ja paketointiin	Suosittu käännöstyökalu, jolla voidaan hoitaa paremmin riippuvuuksia projektien välillä ja kolmannen osapuolen kirjastoihin. Integroituu lisäksi saumattomasti mm. Servicemixiin, eli palvelut voidaan myös asentaa palvelukerrokseen maven työkalun avulla	http://maven.apache.org/
Flyway	Tietokantapäivitysten hallintaan kehitysvaiheen aikana	Työkalun avulla kehittäjät voivat hallitusti tehdä päivityksiä kehityskoneella olevaan tietokantaan	http://code.google.com/p/flyway/
Subversion	Versionhallinta	Lähdekoodin muutokset voidaan tehdä hallitusti	http://subversion.tigris.org/
Bamboo	Työkalu jatkuvaan integrointiin	Maksullinen työkalu, joka hakee versionhallinasta muutokset ja kääntää lähdekoodin sekä ajaa testit. Työkalun avulla päivitetään myös Eduixin testiympäristö. Työkalun avulla pidetään huolta että muutokset eivät aiheuta riko muuta osaa kehitettävästä koodista. Mikäli versionhallintaan lähetetty muutos rikkoo käännöksen, lähtee siitä ilmoitus kehittäjille automaattisesti.	http://www.atlassian.com/software/bamboo/overview
Junit	Työkalu automaattiseen testaukseen	Peppi projektin agoritmit sekä eri komponenttien välinen viestin välitys testataan automatisoideusti Junit-työkalun avulla.	http://www.junit.org/
Jmeter	Työkalu suorituskykytestien tekemistä varten	Testejä ajetaan testiympäristöön asennettua versiota vasten. Jmeter integroidaan bamboo-järjestelmään, jolloin testit voidaan myös ajaa automatisoidusti. Tämän avulla voidaan mitata myös sitä miten koodin muutokset ovat vaikuttaneet suorituskykyyn.	http://jmeter.apache.org/
Soapui	Rajapintojen testaamista varten oleva työkalu	Käytetään soap- ja rest -rajapintojen testaukseen	http://www.soapui.org/

Liite 1 - Arkkitehtuurin perustaan ja toteutukseen vaikuttavat ratkaisumallit

Taulukko 1. Peppi - arkkitehtuurin perustaan ja toteutukseen vaikuttavat ratkaisumallit

Ratkaisu	Lähde	Selite
Sääntömoottori	Kuali	Kuali Student-moduulissa on ominaisuutena sääntöjen asettaminen erilaisille tietueille. Tämä on koettu hyväksi ominaisuudeksi ja otettava huomioon toteutuksen suunnittelussa.
Rajapintojen toteutus siten että toteutuksen muutokset eivät heijastu rajapintaa käyttäviin asiakasohjelmiin	Kuali	Selvitetty Kuali toteutuksen käyttävän DTO-suunnittelumallia, jonka avulla rajapinnat saadaan eristettyä moduulien sisäisestä toiminnasta. Tämän avulla saadaan tehtyä rajapinnan ja sitä käyttävän asiakasohjelman välistä sidosta kevyemmäksi.
Opetussuunnitelman dynaaminen rakenne	Kuali	Kuali Student -moduulissa on tehty abstrakti tietue, Learning unit, joka kuvaa opetussuunnitelmassa olevaa tietuetta, esimerkiksi opintojakso. Learning unit -käsite otetaan lähtökohdaksi OPS-moduulin suunnittelussa.
Moduulien tietokantarakenne	Kuali, OPS, Toisu	Tietokantarakenteessa huomioidaan Kuali Student -modulin, OPS-editorin ja Toisun tietokantarakenne.
Nimeämiskäytäntö	JHS-170	Rajapintojen nimeäminen ja versiointi tehdään kuten JHS-170 määrityksessä on määritelty.
Skeemoihin vaikuttavat käsitteet	Raketti	Raketti-hankkeessa on kerrottu opetussuunnitteluun liittyviä käsitteitä ja niiden välisiä liitoksia. Toteutuksen suunnittelu aloitetaan näiden käsitteiden ja skeemojen pohjalta.

Liite 2 - Vaatimukset arkkitehtuuriratkaisulle ja niiden toteutuminen Peppi - arkkitehtuurissa

Valtionvarainministeriön julkishallinnon perustietovarantojen rajapinnat (PERA) -työryhmä on tehnyt muistion, jossa asetetaan vaatimuksia arkkitehtuuriratkaisulle (vaatimukset). Näitä vaatimuksia kuvataan pääpiirteittäin Peppi-arkkitehtuurin näkökulmasta alla olevassa taulukossa. Vaatimuksista puuttuvat tunnisteet, joten vaatimukset on kopioitu taulukkoon. Huomioitavaa on, että vaatimukset ja muut työryhmän tulokset ovat luonnosvaiheessa (4.5.2011).

Taulukko 2. PERA -työryhmän vaatimuksia arkkitehtuuriratkaisulle Peppi-arkkitehtuurin näkökulmasta

Vaatimus (PERA)	Toteutuminen Peppi-arkkitehtuurissa
Avoimuus	
Arkkitehtuuriratkaisu ei saa edellyttää asiakkailta viimeisimpien teknologioiden soveltamista eli integraatiot eivät saa edellyttää asiakkailta isoja järjestelmäuudistuksia.	Peppi-arkkitehtuurissa hyödynnetään uusia teknologioita (esim. OSGI), mutta sen käyttöönotto ei edellytä ulkopuolisten järjestelmien uudistamista.
Arkkitehtuuriratkaisu ei saa sisältää ohjelmistotoimittajasidonnaisia ratkaisuja Rajapinnan käytön tulee olla ohjelmistoriippumatonta Ratkaisussa ei saa sitoutua tiettyyn tuotteeseen/laitteistoon, joka aiheuttaa rajoituksia tiedon tarjoajalle/vastaanottajalle.	Ratkaisu perustuu avoimen lähdekoodin tuotteisiin, eikä luo riippuvuutta tiettyyn toimittajaan. Järjestelmän palvelupohjaisuus mahdollistaa paremmin palveluiden laajentamisen ja uudelleenkäytön. Rajapinta toteutetaan Webservice/SOAP ja Rest - tekniikoilla, joten rajapintojen käyttö ei rajoitu tiettyyn ohjelmistoon. Mikäli tarvitaan käyttöön transaktioita eri palveluiden kutsujen välillä (koosteisia palveluita), vaatii tämä käytännössä, että palvelu ohjelmoidaan samoin kuin Peppi - projektin aikana tehdyt palvelut on toteutettu. Huomioitavaa on kuitenkin se, että mikäli tiedon omistajuus on selvillä ja tiedot sijaitsevat vain yhdessä paikassa, poistaa se osaltaan tarvetta hajautetuille transaktioille. Tietoja ei siis tarvitse synkronoida useaan paikkaan transaktionaalisesti. Palvelupohjaisessa ratkaisussa palveluiden sisäinen toteutus piilotetaan muilta palveluilta, joka vähentää sidoksia tiettyyn tuotteeseen/laitteistoon.
Ratkaisun tulee mahdollistaa palvelujen koostaminen toisista, useam- man eri viranomaisen eri teknikoilla tehtyjen järjestelmien palveluista. Palvelujen yhdistämisessä tulee huomioida tietosuojan asettamat ra- joitteet.	Ratkaisussa hyödynnetään servicemixin integraatio-ominaisuuksia, jotka mahdollistavat integraatiot eri tekniikoilla tehtyihin rajapintoihin (esim. Webservice, Rest, siirtotiedosto)
Arkkitehtuurin määrittelyn tulee ohjata yhteisiin yleisiin ratkaisuihin, mutta sen pitää sallia toimialakohtaiset erityistarpeet. Erityisesti on huomioitava toimialakohtaiset kansainväliset standardit.	Arkkitehtuurissa on huomioitu nykyarkkitehtuurin muodostamat ongelmat, kansalliset määritykset (M-määritykset) sekä Kuali-yhteisön arkkitehtuuriratkaisut ja teknologiavalinnat.
Arkkitehtuurin tulee tukea yleisiä tietopoliittisia linjauksia. Tietosuojaan ja tietojen saatavuuteen liittyvät linjaukset on huomioitava.	Arkkitehtuuri on selkeämpi kuin nykyinen arkkitehtuuri ja rajapintojen tietosuoja määritellään palvelukohtaisesti.
Arkkitehtuurin tulee mahdollistaa joustava siirtyminen uusien teknologioiden ja standardien käyttöön pitkällä aikavälilla	Arkkitehtuuri on mietitty modulaariseksi ja rajapintojen versiointi on huomioitu. Tämä mahdollistaa teknologioiden asteittaisen siirtymisen uusien teknologioiden käyttöön pitkällä aikavälillä.
Arkkitehtuuriratkaisun tulee käyttää standardoituja rajapintoja aina silloin, kun sellaisia on käytettävissä usean eri toimittajan tukemina.	Rajapinnat tukevat Webservice/SOAP ja Rest teknologioita.
Arkkitehtuuriratkaisu ei saa estää asiakkaan oman arkkitehtuuriratkaisun soveltamista integraatiossa	Rajapinnat tukevat Webservice/SOAP ja Rest teknologioit.
Arkkitehtuuriratkaisun tulee edistää organisaatioiden välistä yhteistyötä tietovarantojen hyödyntämisessa	Ratkaisussa rajapinnat on julkaistavissa eri organisaatioille.
Ratkaisussa pitää käyttää tekniikoita ja ratkaisuja, joihin tiedontuottajien on mahdollista integroitua helposti.	Rajapinnat tukevat Webservice/SOAP ja Rest teknologioit.
Ratkaisussa pitää käyttää tekniikoita ja ratkaisuja, joille löytyy referens- sejä.	Rajapinnat tukevat Webservice/SOAP ja Rest teknologioita, joille löytyy paljon referenssejä.
Skeemojen suunnittelussa tulisi noudattaa yhteisesti sovittuja käy- täntöjä (JHS 170 linjaa XML- skeemojen suunnittelutyyliksi ”Gar- den of Eden”)	Rajapintojen suunnittelussa hyödynnetään Raketti-hankkeen skeemoja (M-määritykset).
Nykyisten olemassa olevien rajapintamääritysten tulee olla sovitettavis- sa tulevaan rakenteeseen sellaisenaan.	Servicemix mahdollistaa vanhojen rajapintojen sovittamisen järjestelmään.
Eheys	
Synkronisen palvelupyyntö-palvelu- rakenteen hallinnan pitää toimia. Rajapinnassa kulkeville sanomille pitää määritellä vakiotiedot, jotka takaavat synkronoinnin. Esimerkiksi autentikointitavat pitää linjata.	Synkroninen viestintä on mahdollista rajapintojen avulla.
Tarvitaan Master-tiedon ja kopioidun tiedon sijoitussuunnitelma.	Peppi-projektin tavoitteena on selkiyttää tiedon omistajuutta eri palveluiden välillä ja poistaa turhat tiedon synkronoinnit.

Turvallisuus	
Arkkitehtuuriratkaisun tulee täyttää yleiset tietosuojan ja tietoturvan vaatimukset	Rajapinnat on mahdollista suojata tietoliikenneteknisesti, autentikaation ja auktorisoinnin avulla.
Arkkitehtuuriratkaisun on taattava tiedon tietoturvallinen siirto asiakas- järjestelmän ja kohdejärjestelmän rajapinnan välilla	Siirrettävä tieto voidaan suojata kryptauksen avulla.
Järjestelmän tulee huomioida tietosuojavaatimukset.	Rajapinnat on mahdollista suojata tietoliikenneteknisesti, autentikaation ja auktorisoinnin avulla.
Tiedon käyttöä tulee pystyä valvomaan (kuka käyttää)	Palveluissa tiedot versioidaan sekä tallentaan tiedon tallentajan tunnisteet. Tämän avulla voidaan seurata palveluiden käyttöä sekä poistaa osaltaan auktorisoinnin tarvetta.
Ratkaisussa pitää huomioida käyttäjän tunnistus	Palveluissa tiedot versioidaan sekä tallenetaan tiedon tallentajan tunnisteet.
Arkkitehtuuriratkaisu ei saa edellyttää tietojen käyttöä suoraan operatii- visesta kannasta, vaan se pitää pystyä eristämään palvelukerroksella.	Palvelut toteutetaan rajapintoja vasten, jolloin palvelut eivät ole liitoksia muiden palveluiden sisäiseen toteutukseen.
Siirrettävien tietojen aikaleimat pitää määritellä ja toteuttaa yhtenäisesti.	-
Ratkaisussa pitää huomioida valtuutus.	-
Ratkaisussa pitää huomioida pääsynhallinta.	-
Ratkaisussa pitää huomioida lokitus.	Lokitus huomioidaan palvelu- ja käyttöliittymämoduulissa.
Ratkaisussa pitää huomioida poikkeustilanteiden hallinta.	Poikkeustilanteet jaetaan eri luokkiin ja poikkeustilanteiden hallinta määritellään jokaiselle luokalle erikseen.
Rajapintaratkaisussa on huomioitava häiriöseuranta ja jäljitettävyys	Häiriönseuranta toteutetaan lokituksen avulla sekä niin, että virheen sattuessa virheen aiheuttanut viesti kirjoitetaan lokiin, jolloin virhe on toistettavissa.
Rajapintojen autentikoinnin ja autorisoinnin tulee olla mahdollista pal- velutasolla.	Palvelurajapinnat liitetään autentikoinnin/autorisoinnin piiriin.
Saavutettavuus	
Ratkaisun tulee tukea sekä suora- että eräkäyttöä. Ratkaisun pitää sisältää useita eri tiedonsiirtotapoja. Ratkaisun on sisällettävä sekä reaaliaikaiset että massasiirtoa tukevat rajapinnat.	Ratkaisu tukee eri protokollilla tehtäviä integraatioita ja palveluista voidaan koostaa uusia palveluita, joihin mahdollista tehdä tuki massasiirroille.
Arkkitehtuuriin tulee olla sovitettavissa legacy-rajapintojen hyödyntä- minen niiden luonnollisen elinkaaren mukaisesti	Ratkaisu tukee eri protokollilla tehtäviä integraatioit.
Integraatiopalvelun on oltava keskeisiltä osiltaan valmis ennen laajaa käyttöä.	Rajapinnat toteutetaan palveluittain eikä käyttötapaus kerrallaan.
Pitää olla palvelu, jonka avulla paikkatiedot yritystilastoista saadaan käyttäjien ulottuville.	-
Yksityisen sektorin toimijoiden on kyettävä käyttämään yhteistä raja- pintaratkaisua hakiessaan ja ylläpitäessään tietovarantojen tietoja.	Rajapinnat tai osa niistä voidaan julkaista täysin avoimiksi.
Muutos tietovarantoon ei saa aiheuttaa pakollista muutostarvetta raja- pintaan ja päinvastoin.	Rajapinnan tarkoitus on suojata muutoksilta. Rajapinnat toteutetaan tämän käsityksen mukaisesti.
Rajapintaratkaisulle pitää määritellä yhteinen poikkeus-/virheskeema se- kä yleiset linjaukset käyttäytymiselle poikkeustilanteissa	Poikkeustilanteet jaetaan eri luokkiin ja poikkeustilanteiden hallinta määritellään jokaiselle luokalle erikseen.
On toivottavaa, että SAPin vakio-IDOC-määritykset tulee ottaa mu- kaan rakenteeseen.	-
Käytettävyys	
Ratkaisun on taattava eri palveluille niiden edellyttämät suorituskyvyt, tarvittaessa 24/7.	Järjestelmän palvelut ovat tilattomia, joka mahdollistaa helpon skaalautuvuuden kuorman kasvaessa.
Ratkaisun tulee tarjota valvontavälineet (sanoman sisältö, sääntöpohjai- suus)	-
Tiedon käyttöä tulee pystyä seuraamaan (tiedonsiirron onnistuminen, lokitus)	Tiedon käyttöä seurataan versiohistorian ja lokituksen avulla.
Ratkaisuun pitää kuulua asianmukainen virhekäsittely.	Virheenkäsittely rakennetaan siten, että se on yhdenmukainen koko Peppi-järjestelmässä.
Ratkaisun tulee sisältää palvelun tuottamisen toimintamallit	-
Vastuiden on oltava selkeästi määritelty (ilman harmaita alueita) -> pal- velun tuottamisen roolit	-

Tiedon on oltava saatavilla reaaliaikaisesti	Tiedot on saatavilla palvelurajapintojen kautta.
Rajapintaratkaisun tulee olla stabiili	Rajapinnat toteutetaan pohjautuen SOAP/Webservice tai Rest teknologioihin , jotka ovat tällä hetkellä yleisesti käytössä ja koeteltuja.
Rajapinnan tulee olla suorituskykyinen	Rajapinnat toteutetaan pohjautuen SOAP/Webservice tai Rest teknologioihin , jotka ovat tällä hetkellä yleisesti käytössä ja koeteltuja. Rajapintojen toteutuksissa käytetään cacheja jos se on mahdollista.
Palveluiden hallintaratkaisut, (esim. palveluhakemisto, versionhallinta, SLA-sopimukset) pitää kehittää ja ottaa käyttöön.	Palveluluiden hallintaratkaisut toteutetaan OSGI-teknologian ja dokumentaation avulla.
Rajapintojen tulisi olla tilattomia, ts. rajapintakutsun tuloksen tulee olla riippumaton sitä edeltäneistä kutsuista	Palvelurajapinnat tehdään tilattomiksi. Tila pidetään yllä käyttöliittymämoduulissa, käyttöliittymän logiikan tila, tai tallentamalla tilatieto tietueeseen. Mikäli hyödynnetään tulevaisuudessa prosessimoottoria, tila on tallentuneena prosessi-instanssiin.
Säädökset, ohjeet ja standardit	
Uusien rajapintojen tulee olla sanomapohjaisia (kysymys-vastaus-sanomapareja)	Rajapinnat toteutetaan pohjautuen SOAP/Webservice tai Rest teknologioihin.
Uusien rajapintojen määrittelyssä tulee huomioida modernit internet- ja web-teknologiat (esim. HTTPS, WS-I ja SOAP)	Rajapinnat toteutetaan pohjautuen SOAP/Webservice tai Rest teknologioihin.
Rajapintaratkaisun pitää huomioida julkishallinnon tietoarkkitehtuurissa tehdyt linjaukset ja täyttää osaltaan sen vaatimukset.	-
Yhtenäisten rajapintarakenteiden (skeemat, sanastot) tulee perustua JHS 170 ja JHS 175 -suosituksiin.	Rajapintarakenteissa huomioidaan JHS-suositukset, mm. versiointi sekä skeeman rakenne.
Rajapintapalvelujen teknologia ja tietotuotemäärittelyt eivät saa olla ristiriidassa kansainvälisten säännös- ten kanssa (esim. INSPIRE direktii- viin liittyvät komission asetukset koskien paikkatietoja)	-
Arkkitehtuuriratkaisun tulee perustua yleisiin standardeihin, joiden yk- sityiskohtainen soveltaminen ei saa johtaa järjestelmätoimittajariippu- vuuksiin.	Arkkitehtuuriratkaisussa on suosittu standardeja palvelurajapinnoissa.
Ratkaisussa pitää käyttää vakiintuneita ja toimivaksi todettuja menetelmiä ja standardeja. Rajapintapalveluiden standardien tulee olla myös käytännössä testattuja.	
Arkkitehtuurin tulee huomioida toimialakohtaiset standardit	On huomioitu JHS-suositukset sekä raketti-hankkeen käsitteet sekä skeemat.
Ratkaisussa on kytkeydyttävä kansainvälisesti sovittuihin toimintata- poihin ja standardeihin (osa perustuu EU-lakeihin).	-
Ratkaisussa pitää käyttää tekniikoita ja ratkaisuja, joiden elinkaari ei ole päättymässä ja joille löytyy tuki.	
Ratkaisun tulee tukeutua Valtion IT-palvelukeskuksen tarjoaman yhtei- sen integraatiopalvelun käyttöön.	-
Arkkitehtuuriratkaisun tulee tukea hajautettua palveluarkkitehtuuria	Peppi - arkkitehtuuri on suunniteltu hajautettaviksi. Tämä toteutuu mm. tekemällä palveluista tilattomia, tekemällä kutsut palveluiden välillä rajapintoja vasten.
Rajapintojen sanomaformaatin tulisi olla "document/literal"-tyyppinen	SOAP-sanomista tehdään document/literal tai document/literal wrapped tyyppisiä sanomia.
Hallittavuus	
Rajapinnasta tulee olla tekninen kuvaus.	-
Tiedon kuvauksen tulee olla saatavilla ja hyödynnettävissä.	-
Tiedot tulee olla kuvattu.	-
Rajapinnasta tulee olla käyttöohje.	-
Rajapintaratkaisulla tulee olla toimiva versionhallinta.	Palvelurajapinnoista voidaan ajaa useampaa eri versiota rinnakkain OSGI-teknologian avulla.
Rajapintamääritykset on versioitava ja ylläpidettävä versiohistoria määrittelydokumentissa.	-
Uusi versio ei saa häiritä vanhan version käyttäjiä.	Palvelurajapinnoista voidaan ajaa useampaa eri versiota rinnakkain OSGI-teknologian avulla.
Vanhat versiomääritykset on säilytettävä	Versiomääritykset säilyvät versionhallinnassa. Jokainen julkaistu versio tagataan versionhallintaan.

Rajapintapalveluissa tulee mahdollistaa tuoteversiointi ja vanhojen versioiden alasajo.	Palvelurajapinnoista voidaan ajaa useampaa eri versiota rinnakkain. Käytäntönä voidaan pitää, että ylläpidetään tarvittaessa palvelusta kolmea versiota. Tällöin päivitykset voidaan tehdä hallitusti, koska päivityksillä on siirtymäaika.
Palvelun käyttäjälle on oltava saatavissa muuttunut tieto.	-
Ratkaisun pitää sisältää päivitysrajapinnat. Päivitysten edellyttämä transaktioiden hallinta pitää huomioida.	-
Rajapintaskeemat tulisi kuvata omissa, erikseen ylläpidettävissä do- kumenteissaan.	-
Rajapintamääritysdokumenttien on löydyttävä keskitetystä paikasta (esim. Yhteentoimivuusportaali). Perustietovarantojen sisällön ja tie- topalvelurajapintojen dokumentaatio pitää tuoda helposti ja julkisesti saa- taville.	-
Kansainvälisyys, kieliversiot ja lokalisointi	
Rajapintapalveluiden tulee mahdollistaa sekä kansainvälisten että kansallisten palveluiden ja tietotuotemäärittelyjen rinnakkaisuuden	Järjestelmässä on mahdollista muuntaa tietyn palvelun palauttama viesti toisen skeeman mukaiseksi.
Muut vaatimukset (mm. taloudellisuus)	
Ratkaisun tulee kestää pitkälle tulevaisuuteen ja muutosten tullessa muutostöiden kustannukset/tehtävä tulee olla ratkaisua jo käyttäville minimaalinen.	Palvelupohjaisessa arkkitehtuurissa lähtökohtana on moduulien ja palveluiden uudelleenkäytettävyys. Tämä vähentää tietojärjestelmien tekemiseen liittyviä kustannuksia tulevaisuudessa.