Representational State Transfer (REST) ja Web-suuntautunut arkkitehtuuri (WOA) arkkitehtuurityylinä

TIIVISTELMÄ

Ohjelmistoarkkitehtuuri voidaan määritellä muun muassa arkkitehtuuristen elementtien, kuten komponenttien (component), liittimien (connector) ja datan (data) konfiguraatioksi, joiden keskinäisiä suhteita on rajoitettu tarkoituksena saavuttaa tiettyjä arkkitehtuurisia ominaisuuksia (architectural properties). Arkkitehtuurityylit tarjoavat joukon rajoitteita, joiden tarkoituksena on mahdollistaa ohjelmistolle esitettyjen vaatimusten toteutuminen. Arkkitehtuureja voidaan suunnitella valmiista komponenteista, tai rajoittamalla arkkitehtuuristen elementtien keskinäisiä suhteita.

Perinteisessä palvelusuuntautuneessa arkkitehtuurissa (Service Oriented Architecture, SOA) palvelut toteutetaan Web-palveluina (Web Services), SOAP (Simple Object Access Protocol)- ja HTTP (Hypertext Transfer Protocol)- protokollaa käyttäen. SOA-mallin mukaisilla Web-palveluilla on yksilöllinen WSDL (Web Service Description Language)-, eli XML-kuvauksen mukainen kutsurajapinta, joka palvelun käyttäjän on tunnettava palvelukutsun suorittaakseen.

REST (Representational State Transfer) koostuu joukosta rajoitteita, jotka on valittu soveltuvin osin muista arkkitehtuurityyleistä, sekä komponenttien kommunikointirajapintaa koskevista yhtenäisen rajapinnan lisärajoitteista. RESTin rajoitteet ovat asiakas-palvelin, tilattomuus, välimuisti, ladattava koodi ja yhtenäinen rajapinta. RESTin ensimmäinen versio kehitettiin 1994. Fielding julkaisi arkkitehtuurityyleihin liittyvän väitöskirjan 2000, jossa REST esiteltiin ja sitä sovellettiin nykyaikaiseen Web-arkkitehtuuriin, kuten HTTP (Hypertext Transfer Protocol)-protokollaan ja URL (Uniform Resource Locator) -osoitteisiin.

Keskeisten RESTin yhtenäisen rajapinnan (Uniform interface) rajoitteiden mukaisesti kaikki palvelut nähdään resursseina (Resource), joilla on yksilöitävissä oleva URL tunniste ja yhtenäinen (Uniform) kutsurajapinta. Tunnetuin esimerkki REST-rajoitteilla toteutetusta hypermediajärjestelmästä on WWW (World Wide Web).

Web-suuntautunut arkkitehtuuri (Web oriented architecture, WOA) on palvelusuuntautuneen arkkitehtuurityylin (Service Oriented Architecture, SOA) alityyli (Substyle). WOA on konsulttiyhtiö Gartnerin nimitys rajapintatason hybridiarkkitehtuurityylille, joka keskeisiltä osin noudattaa Roy Fieldingin väitöskirjassa (v. 2000) julkaisemia RESTin asiakas ja palvelinkomponentin välisiä yhtenäisen rajapinnan (Uniform Interface) rajoitteita. WOA lisää RESTin rajoitteisiin sovellusriippumattomuuden (Application Neutrality). RESTin tarkoituksena on esittää rajoitteet, jotka soveltuvat erityisesti hajautettujen hypermedia (Web) – järjestelmien toteutukseen.

Työ tarkastelee arkkitehtuurityylejä ja Web-palvelujen tuottamista REST- ja WOA-periaatteiden mukaisesti, sekä esittelee lyhyesti RESTin mukaisten järjestelmien toteuttamiseen soveltuvia työkaluja.

SISÄLLYS

1. Johdanto

Arkkitehtuurityylejä käyttämällä voidaan ohjelmiston laatuominaisuuksiin vaikuttaa kehityksen aikaisessa vaiheessa. Arkkitehtuurityylit tarjoavat joukon rajoitteita, joiden tarkoituksena on mahdollistaa ohjelmistolle esitettyjen vaatimusten toteutuminen. Rajoitteet ovat siten suhteellisia, että yhden rajoitteen lisääminen saattaa heikentää toista rajoitetta tai rajoitteen ominaisuutta (Aspect of Property) [FIE00]. Rajoitteet on valittava ohjelmiston nykyiset ja tulevat vaatimukset, sekä toimintaympäristö huomioiden. Tunnetut arkkitehtuurityylit sisältävät joukon rajoitteita ja dokumentoituja suunnittelupäätöksiä.

REST (Representational State Transfer on hybridiarkkitehtuurityyli, jonka rajoitteet on valittu muista arkkitehtuurityyleistä ja lisätty niihin yhtenäisen asiakas-palvelin kommunikointirajapinnan rajoitteet. REST tarjoaa arkkitehtuuristen rajoitteiden joukon, joka sovellettuna kokonaisuutena tuo järjestelmään ominaisuuksina (Properties) komponenttien interaktioiden skaalautuvuutta, rajapintojen yleisyyttä, komponenttien itsenäistä kehitettävyyttä ja mahdollistaa välittäjäkomponenttien käytön (Intermediaries) poistamaan interaktioiden latenssia, varmistamaan tietoturvaa ja kapseloimaan perinnejärjestelmiä (Legacy Systems) [FIE00,FT02].

Web-palveluiden kehittäjät suosivat helppokäyttöisten REST-rajapintojen käyttöä. Amazon julkaisi tiedon, että REST- rajapintoja käytetään 85%:sti, kun vaihtoehtona ovat perinteiset Web-palvelujen SOAP:n (Simple Object Access Protocol)- mukaiset rajapinnat[1]. Liiketoimintamalleja (Business Model) kehitetään, joissa koosteisten ns. Web 2.0 Mashup -sovellusten toteuttajat liittävät omaan sovellukseensa esimerkiksi Amazon:in rajapintoja ja veloittamalla Amazonin avulla tuotetusta kokonaispalvelusta välityspalkkioita loppukäyttäjiltä. ProgrammableWeb:in tilastojen mukaan REST-rajapintoja käytetään 65 %:sti, kun vertailukohtana on SOAP- rajapinnat (21%) [PRG09, 10.05.2009]. Tilastoissa on huomioitava, että monet Web-palvelut markkinoivat tarjoavansa REST- rajapintoja, vaikka RESTin periaatteita ei ole noudatettu kuin osittain. Seurantajakson aikana (10.4- 10.5.2009) REST-palveluiden käyttö lisääntyi 2% [PRG09].

Vaikka RESTin periaatteet ovat yksinkertaisia, on niiden soveltaminen ja ymmärtäminen johdonmukaisesti ollut vaikeaa ja tämä on johtanut Web-sovelluksiin, jotka eivät ole hyötyneet RESTin rajoitteiden tuomista ominaisuuksista.[EGS07].

2. Arkkitehtuurityylit

Arkkitehtuurin suunnittelu on tärkeä vaihe ohjelmiston ja yritysarkkitehtuurin (Enterprice Architecture) suunnittelua. Ohjelmistoarkkitehtuurista on kirjallisuudessa useita määritelmiä. Fielding määrittelee ohjelmistoarkkitehtuurin mm. arkkitehtuuristen elementtien, kuten komponenttien (Component), liittimien (Connector) ja datan (Data) konfiguraatioksi, joiden keskinäisiä suhteita on rajoitettu tarkoituksena saavuttaa tiettyjä arkkitehtuurisia ominaisuuksia (Architectural Properties) [FIE00,7].

[1] http://www.oreillynet.com/pub/wlg/3005

Arkkitehtuuristen ominaisuuksien ansiosta järjestelmä voi vastata paremmin nykyisiin ja tulevaisuuden vaatimuksiin. Arkkitehtuuria voidaan suunnitella ympäristöön sopivista, valmiista tunnetuista komponenteista, tai rajoittamalla arkkitehtuurisia ominaisuuksia järjestelmän vaatimusten toteuttamiseksi [FIE00].

Arkkitehtuurityylejä (Architectural Style, Architectural Pattern) käyttämällä voidaan ohjelmiston laatuominaisuuksiin vaikuttaa kehityksen aikaisessa vaiheessa. Arkkitehtuurityyleistä on kirjallisuudessa useita määritelmiä: Fielding määrittelee arkkitehtuurityylin hallituksi kokonaisuudeksi arkkitehtuurisia rajoitteita (Constraint), jotka rajoittavat (Restricts) arkkitehtuuristen elementtien (Elements), roolit (Roles), ominaisuudet (Properties), sekä sallitut suhteet elementtien välillä (Relationships), tietyn arkkitehtuurityylin mukaisessa järjestelmässä. Tyylien avulla arkkitehtuureja voidaan luokitella ja kuvailla näiden keskeiset piirteet [FIE00].

Järjestelmän perusrakenne tai ns. perusluonne voidaan kuvata korkealla abstraktiotasolla arkkitehtuurityylinä tai ns. referenssiarkkitehtuurina, joka määrittelee arkkitehtuurin liittyvät käsitteet ottamatta kantaa yksittäiseen järjestelmään. Tietyn arkkitehtuurityylin mukaisessa järjestelmässä on joukko ohjelmistokomponentteja samassa roolissa ko. tyylin kannalta. Arkkitehtuurityyli määrittää järjestelmän kokonaisrakenteen ja tyyli voidaan käsittää myös järjestelmän toteutuksen rakenteen selityksenä [KM05].

Arkkitehtuurisiin ominaisuuksiin kuuluvat ohjelmiston toiminnalliset (Functional) ja laadulliset (Non-Functional) ominaisuudet. Laadullisilla ominaisuuksilla tarkoitetaan esimerkiksi ylläpidettävyyttä tai muokattavuutta, suorituskykyä ja komponenttien kykyä kehittyä itsenäisesti [FIE00]. Tietovuoarkkitehtuurityyli (Pipe-And-Filters Architecture) tavoittelee komponenttien uudelleenkäytettävyyttä ja konfiguroitavuutta rajoittamalla komponenttien rajapinnat yhtenäiseen ja yleiseen rajapintaan [FIE00]. Järjestelmä voi koostua useista tyyleistä, sillä tyyleillä pyritään vaikuttamaan ohjelmiston erilaisiin ominaisuuksiin. Hybridityyli (Hybrid Style)- termiä voidaan käyttää,yhdestä nimetystä ja koordinoidusta tyylistä joka koostuu muista arkkitehtuurityyleistä. Tyylien vertailu voi olla hankalaa johtuen ohjelmistojen erilaisista vaatimuksista ja toimintaympäristöistä [FIE00, 1.5-1.6].

Arkkitehtuurityylin valinnalla on merkittävä vaikutus suunniteltaessa suorituskykyisiä, korkean saatavuuden (Availibility) ja tietoturvan vaatimuksia toteuttavia järjestelmiä ja tyylin valinnan tulisi olla ensimmäisiä päätöksiä arkkitehtuuria suunniteltaessa [LPR03]. Kokemuksen mukaan arkkitehtuuritason ratkaisut vaikuttavat järjestelmän suorituskykyyn eniten [SW01]. Arkkitehtuurityylejä ovat esimerkiksi asiakas-palvelin (Client- Server), malli- näkymä-ohjain- (Model, View, Controller), palveluperustaiset (Service Oriented Architecture, Representational State Transfer (REST) ja Web-Oriented Architecture (WOA).

Palvelusuuntautuneessa arkkitehtuurityylissä (SOA) järjestelmä voi esimerkiksi noudattaa kolmitaso- (Three-Tier) ja komponenttien kokonaisroolijako malli–näkymä-ohjain (MVC) -arkkitehtuurityyliä. Järjestelmäarkkitehtuurista voidaan myös erottaa

asiakas-palvelin (Client-Server) -, sekä työssä erityisesti käsiteltävät WOA- ja REST-arkkitehtuurityylit.

Ohjelmistoarkkitehtuurien tutkimus keskittyy etsimään menetelmiä, miten parhaiten jakaa systeemi osiin, kuinka komponentit tunnistavat toisensa ja kommunikoivat keskenään, kuinka informaatio kommunikoidaan ja millä tavoin systeemi voi kehittyä itsenäisesti. Keskeistä ohjelmistoarkkitehtuurien tutkimuksessa on, kuinka nämä asiat voidaan kuvata formaaleilla ja epäformaaleilla menetelmillä [FIE00].

3. REST

REST – on hybridiarkkitehtuurityyli, joka soveltuu erityisesti hajautettujen hypermediajärjestelmien (Web-sovellukset) toteutukseen. REST koostuu joukosta rajoitteita, jotka on valittu soveltuvin osin muista arkkitehtuurityyleistä, sekä komponenttien kommunikointirajapintaa koskevista yhtenäisen rajapinnan lisärajoitteista [FIE00].

RESTin on terminä tarkoitettu herättämään käsitys, miten hyvin suunnitellun Web-sovelluksen tulisi käyttäytyä: Web-sivujen joukko muodostaa virtuaalisen tilakoneen, joka tarjoaa käyttäjälle mahdollisuuden siirtyä sovelluksen tilasta toiseen, tilojen esitystavoissa (Representation) esiintyvien linkkien kautta tai täyttämällä esitystavassa oleva Web-lomake (Form) [FT02].

REST on johdettu arkkitehtuurityyleistä toisinnettu tietovarasto (Replicated Repository), välimuisti (Cache), asiakas- palvelin (Client-Server), kerroksittainen järjestelmä (Layered System), tilaton (Stateless), Virtuaalikone (Virtual machine), ladattava koodi (Code On Demand) ja yhtenäisestä rajapinnasta (Uniform Interface) [FT02]. RESTin keskeinen eroavaisuus sen johdannaisiin arkkitehtuurityyleihin nähden on yhtenäisen rajapinnan rajoitteet [FT02,122]. RESTin asiakas-palvelin kommunikointirajapintaa koskevat rajoitteet ovat resurssien tunnistaminen, resurssien muokkaaminen esitystapojensa kautta, itsekuvaavat viestit ja hypermedia tilasiirtymien mahdollistajana (Hypermedia as the Engine of Application state) [FIE00].

Standardointiorganisaatio W3C jakaa Web-palvelut kahteen pääluokkaan: REST:iä noudattavat (REST- Compilant) Web-palvelut ja REST:iä noudattamattomat (Non-Rest-Compilant) Web-palvelut. W3C määrittelee REST- yhteensopiviksi palveluiksi yhtenäistä rajapintaa noudattavat tilattomat palvelut, joiden tarkoituksena on muokata Web-resurssien XML-esitystapoja ja REST- yhteensopimattomat Web-palvelut palveluiksi, jotka tarjoavat mielivaltaisen määrän rajapinnan operaatioita**.** W3C:n mukaan yhteistä Web-palveluiden arkkitehtuurityyleillä on URI: en, Web-protokollien (HTTP, SOAP 1.2) ja XML-tiedostomuodon käyttö [W3C04]. REST- yhteensopimattomilla palveluilla voidaan käsittää esimerkiksi RPC (Remote Procedure Call)-tyylisiä WS-(Web Services) teknologiapinon avulla toteutettuja palveluita.

RESTin arkkitehtuurisina tavoitteina on latenssin ja verkkoviiveen vähentäminen, komponenttien itsenäisen kehitettävyyden ja toteutuksen skaalautuvuuden parantaminen. REST ei rajoita yksittäisten komponenttien sisäisiä suhteita, toteutustapaa tai protokollien

syntaksia [FIE00, EGS07]. REST mahdollistaa rajoitteidensa ansiosta välimuistien (Web-Cache) käytön, interaktioiden uudelleenkäytön, komponenttien dynaamisen vaihdettavuuden ja toimintojen suorittamisen välittäjäkomponenteissa [FT02]. REST keskittyy komponenttien rooleihin, rajoitteisiin komponenttien välisessä kommunikaatiossa ja data-elementtien tulkintaan. Fieldingin mukaan muut tyylit keskittyvät enemmän komponenttien semantiikkaan, kun REST keskittyy liittimien (Connector) semantiikkaan [FIE00].

Huolimatta RESTin yksinkertaisista periaatteista, ne on havaittu vaikeiksi ymmärtää ja soveltaa johdonmukaisesti. Tästä johtuen useat Web-sovellukset jäävät ilman RESTin lupaamia hyötyjä. Useat palvelut väittävät tarjoavansa REST- rajapinnan, vaikkei kaikkia RESTin periaatteita ole noudatettu. Tilalliset ja välimuistia tukemattomat palvelut ovat yleisiä. Tilallisuus-rajoitteen rikkominen on yksi yleisimpiä REST- rajoitteiden rikkomuksia [EGS07, RR07, FIE00].

3.1 Asiakas-Palvelin

Liittämällä RESTiin asiakas-palvelin (Client-Server) arkkitehtuurityylin rajoitteet, saavutetaan selkeää ohjelmiston osien vastuiden jakamista (separation of concerns) [FIE00]. Asiakkaan ja palvelimen välinen ero on, että asiakas kutsuu palvelinta liitinkomponentin (Connector) kautta ja palvelin kuuntelee yhteyksiä ja muodostaa vastauksen. Molemmilla komponenteilla voi olla molemmat roolit [FT02, FIE00].

Asiakas-palvelin arkkitehtuurissa palvelinkomponentttien skaalautuvuutta voidaan parantaa jakamalla toiminnallisuus sopivasti asiakas- ja palvelinkomponenttien kesken. Yleensä tämä tarkoittaa käyttöliittymän esittämisen vastuun siirtämistä asiakaskomponentille. Asiakas- ja palvelinkomponentit voivat kehittyä itsenäisesti, mikäli niiden väliset rajapinnat säilyvät muuttumattomina [FIE00]. Asiakas-palvelin arkkitehtuurityylin perusmuoto ei rajoita, kuinka sovelluksen tilan hallinta on jaettu asiakas ja palvelinkomponenttien kesken [FIE00].

Esimerkiksi RESTin mukaisessa Web-palvelussa Client- komponentti lähettää käyttäjälle (User Agent) käyttöliittymän, joka voi koostua useista resursseista ja niiden esitystavoista, sekä linkeistä muihin sovelluksen tiloihin (resursseihin).

3.2 Välimuisti

Välimuisti (Cache) voidaan määritellä asiakkaan ja palvelimen väliseksi välittäjäkomponentiksi (Mediator), sekä erilliseksi arkkitehtuurityyliksi [FIE00]. Välimuistikomponentteja voidaan käyttää verkkoviestien välttämiseen. Asiakas voi käyttää välimuistikomponenttia välttääkseen verkkoliikennöinnin ja palvelin esimerkiksi välttääkseen hakuja tietokannasta. Välimuistikomponentti toteutetaan yleensä samassa muistiosoitteessa, kuin komponentti joka sitä käyttää [FT02]. Välimuisteja voidaan jakaa yhden (private cache) tai useamman käyttäjän (Public Cache, Shared Cache) käyttöön. Selainohjelmistoissa on yksityinen välimuisti ja julkinen välimuisti voi toimia esimerkiksi välittäjäkomponentissa[GT02].

HTTP-protokollan mukaisesti palvelin voi liittää viestiin vanhenemispäivän otsikon (Header)-elementteinä. Asiakkaan suorittaessa palvelukutsun, voi Cache-välittäjäkomponentti palauttaa sopivan vastauksen, tai välittää pyynnön palvelimelle, joka muodostaa vastauksen uudelleen.

3.3 Kerroksittainen

Kerroksittaisessa järjestelmässä ylemmän kerroksen komponentit käyttävät alemman kerroksen komponenttien palveluita ja kullakin kerroksella on joukko rajapintoja, jotka kerros tarjoaa ja joukko rajapintoja, jotka se vaatii. Kerrokset ovat vaihdettavia, sillä ylemmän kerroksen toteutus ei riipu alemman kerroksen toteutuksesta.

Rajoittamalla järjestelmän kykyä kommunikoida kerrosarkkitehtuurissa lähimmän kerroksen kanssa saavutetaan yksinkertaistusta ja tuetaan komponenttien itsenäisyyttä [FIE00]. Kerroksia voidaan yleisesti käyttää kapseloimaan perinnejärjestelmiä, suojaamaan uusia palveluja perinnejärjestelmiltä ja yksinkertaistamaan järjestelmää siirtämällä toiminnallisuutta välittäjäkomponenteille [FIE00].

3.4 Ladattava koodi

REST:iin kuuluu valinnaisena rajoitteena ladattava koodi (Code on Demand). REST sallii asiakaskomponentin toiminnan laajentamisen ja konfiguroinnin lataamalla palvelimelta sovelmia (Applet) tai skriptejä (Scripts) [FIE00]. Ladattava koodi voisi olla esimerkiksi paikallisesti suoritettavan rajapinnan uusi toteutus, jonka asiakaskomponentti rekisteröi käyttöönsä. Java Web Start -ympäristö voidaan käsittää myös ladattavan koodin ilmentymänä. Java sovelman avulla voidaan muodostaa RMI (Remote Method Invocation)-kutsuja suoraan esimerkiksi Java EE (Java Enterprice Edition)-palvelimelle.

Rajoitteen käytöllä saavutetaan suorituskykyetuja etäkutsuihin verrattuna ja palvelinkomponenttien skaalautuvuutta. Valinnainen rajoite heikentää kuitenkin näkyvyyttä (Visibility). Organisaation asiakkaat voivat esimerkiksi tukea Java-sovelmien käyttöä, mutta palomuurikäytännöt saattavat estää sovelmien lataamisen ulkoisista lähteistä. Monitorointi hankaloituu, koska palvelin lähettää asiakkaalle koodia datan sijasta. Valinnaisessa rajoitteessa on huomioitava, että arkkitehtuuri hyötyy sen eduista, mutta myös kärsii haitoista kokonaisjärjestelmän kannalta. Käytettäessä valinnaista rajoitetta on tiedostettava, missä mittakaavassa se vaikuttaa kokonaisjärjestelmään [FIE00].

3.5 Tilattomuus

REST- asettaa tilattomuusrajoitteen komponenttien kommunikaatiolle. Tilattomuusrajoitteen mukaisesti jokaisen asiakkaan palvelupyynnön tulee sisältää kaikki tarvittava tieto palvelupyynnön suorittamiseksi palvelimella.

Käyttäjä voi RESTin mukaisessa järjestelmässä siirtyä sovelluksen tilasta toiseen esitystavoissa olevien hyperlinkkien tai lomake (Form)- komponenttien avulla. Mikäli asiakas haluaa palvelimen huomioivan tilan, on se lähetettävä jokaisen pyynnön yhteydessä. Esimerkiksi autentikointitiedot on lähetettävä jokaisen pyynnön yhteydessä [RR07]. Tilattomuus- rajoitteesta seuraa arkkitehtuurisina ominaisuuksina näkyvyyttä,

luotettavuutta ja skaalautuvutuutta. Näkyvyys paranee, koska mahdollinen monitorointijärjestelmä voi päätellä yhdestä pyynnöstä, mikä koko pyynnön varsinainen tarkoitus oli. Luotettavuus paranee, sillä osittaisista häiriöistä (Partial Failures) voidaan toipua paremmin [FIE00].

Tilattomuuden ansiosta asiakkaat eivät joudu palvelimen häiriötilanteessa epäjohdonmukaiseen tilaan, sillä ne eivät luota palvelimella säilöttyyn tilaan, toisin kuin tilan palvelimelle säilövissä sovelluksissa. Tilalliselle palvelulle on helpommin toteutettavissa kuormantasaus (Load Balancing), välimuistikomponenttien käyttö ja ryvästys (Clustering) [PAU09]. Asiakkaan ei täydy olla tiettyyn palvelinohjelmiston instanssiin yhteydessä yhden kokonaiskeskustelun aikana, eikä tilaa tarvitse toisintaa muille käytössä oleville palvelimille.

Palvelimen skaalautuvuus parantuu, sillä tilanhallinta on asiakaskomponentin vastuulla. Tämä parantaa palvelimen kykyä (kapasiteettia) vapauttaa resursseja muuhun käyttöön ja yksinkertaistaa palvelinkomponenttien toteutusta. Tilattomuus- rajoite on kuitenkin valintakysymys (Design-Tradeoff). Haittapuolena on suorituskyvyn laskua, sillä asiakkaan tilatieto lähetetään palvelimelle jokaisen palvelupyynnön yhteydessä [FIE00].

Tilattomuus rajoitteen mukaan palvelimen eri tilat ovat myös resursseja, joille tulee antaa oma URL- osoite. Tekniikoita tilansiirtämisen toteuttamiseen (Exchange State) on URI: n uudelleenkirjoitus (URI- Rewriting), evästeet (Cookies) ja piilotetut Web-lomakemuuttujat (Form). Tila voidaan upottaa palvelimen vastausviesteihin osoittamaan asiakkaan seuraavia mahdollisia tilasiirtymiä sovelluksessa [PAU09]. Evästeiden käytössä on ongelmia, eivätkä ne kuulu HTTP-standardiin [FIE00, RR07]

4. Yhtenäinen Rajapinta

RESTin keskeinen rajoite muihin arkkitehtuurityyleihin verrattuna on komponenttien välinen yhtenäinen rajapinta. Useita rajoitteita on sovellettava, jotta yhtenäiseen rajapintaan päästäisiin. REST määrittää neljä (4) rajapintarajoitetta.

1) Resurssien tunnistaminen (Identification of resources)
2) Resurssien muokkaaminen esitystapojensa kautta (Manipulation of resources through representations)
3) Itsekuvaavat viestit (Self-descriptive messages)
4) Hypermedia sovelluksen tilakoneena (hypermedia as the engine of application state)

Yhtenäisellä rajapinnan käytöstä on etuina konfiguroitavuutta, uudelleenkäytettävyyttä, itsenäistä kehitettävyyttä, kokonaisarkkitehtuuri yksinkertaistuu ja interaktioiden näkyvyys paranee. Yhtenäisen rajapinnan haittana on, että se kuluttaa verkon suorituskykyä, jos tietoa joudutaan muuntamaan natiivin ja ohjelmointikielikohtaisen rajapinnan välillä [FIE00]. REST ei määritä mitä yhtenäistä rajapintaa on käytettävä, REST määrittää että rajapinnan on oltava yhtenäinen ja jokaisen rajapinnan käyttäjän on käytettävä rajapintaa samalla tavalla [FIE00,RR07].

REST-toteutuksissa voidaan käyttää HTTP:n sallimia operaatioita (esim. GET, PUT, POST, DELETE, HEAD, OPTIONS), useissa lähteissä (EI Fielding) näitä on liitetty ns. CRUD- operaatioihin. GET vastaa lukuoperaatiota (READ), PUT vastaa päivitysoperaatiota (UPDATE), POST vastaa lisäysoperaatiota (INSERT) ja DELETE vastaa poisto-operaatiota (DELETE). Algoritminen resurssi, joka ei muokkaa dataa voidaan pyytää GET- pyynnöllä.
GET ja HEAD- pyynnöt ovat HTTP-protokollan mukaisesti turvallisia (Safe). Asiakas ei ole vastuussa, mikäli turvallisessa operaatioissa muokataan dataa.

GET,- HEAD-, PUT- ja DELETE- operaatiot ovat idempodenttejä. Idempodentin operaation suorittaminen useamman kerran, vastaa vaikutuksiltaan palvelimella operaation suorittamista kerran. POST-pyyntö ei ole turvallinen, eikä idempodentti. Idempodenttien operaatioiden puuttuminen SOAP-viesteistä (POST), vaikeuttaa Web-Cache komponenttien kykyä säilöä viesti välimuistiin, sillä koko viesti on käsiteltävä, eikä vain otsikko-elementtejä, kuten HTTP-viestissä.
[RR07, HTTP1.1]

4.1 Yhtenäisen rajapinnan rajoitteet ja resurssitunnisteet

Informaation abstraktiota kutsutaan REST:ssä resurssiksi (Resource) [FIE00].
Kaikki, mikä voi olla käyttäjän hypertekstiviitteen kiinnostuksen kohteena tulee nimetä resurssiksi. Resurssi on käsitteellinen liitäntä joukolle käsitteitä, eikä käsitteelle joka noudattaa liitäntää tietyssä ajanhetkessä. Esimerkiksi jonakin ajanhetkenä kaksi eri resurssia voi osoittaa samaan dataan. Ohjelmistoversion 1.03 ja viimeisimmän version URL (Uniform Resource Locator) voivat osoittaa hetken samaan dataan[FIE00,RR07]. Resurssijoukon alkiot voivat olla resurssien esitystapoja tai resurssien tunnisteita. Resurssi voi myös viitata tyhjään joukkoon (empty set), jonka ansiosta viittauksia voidaan tehdä, ennen kuin resurssin toteutus on olemassa. Resursseilla voi olla myös suhteita toisiin resursseihin.

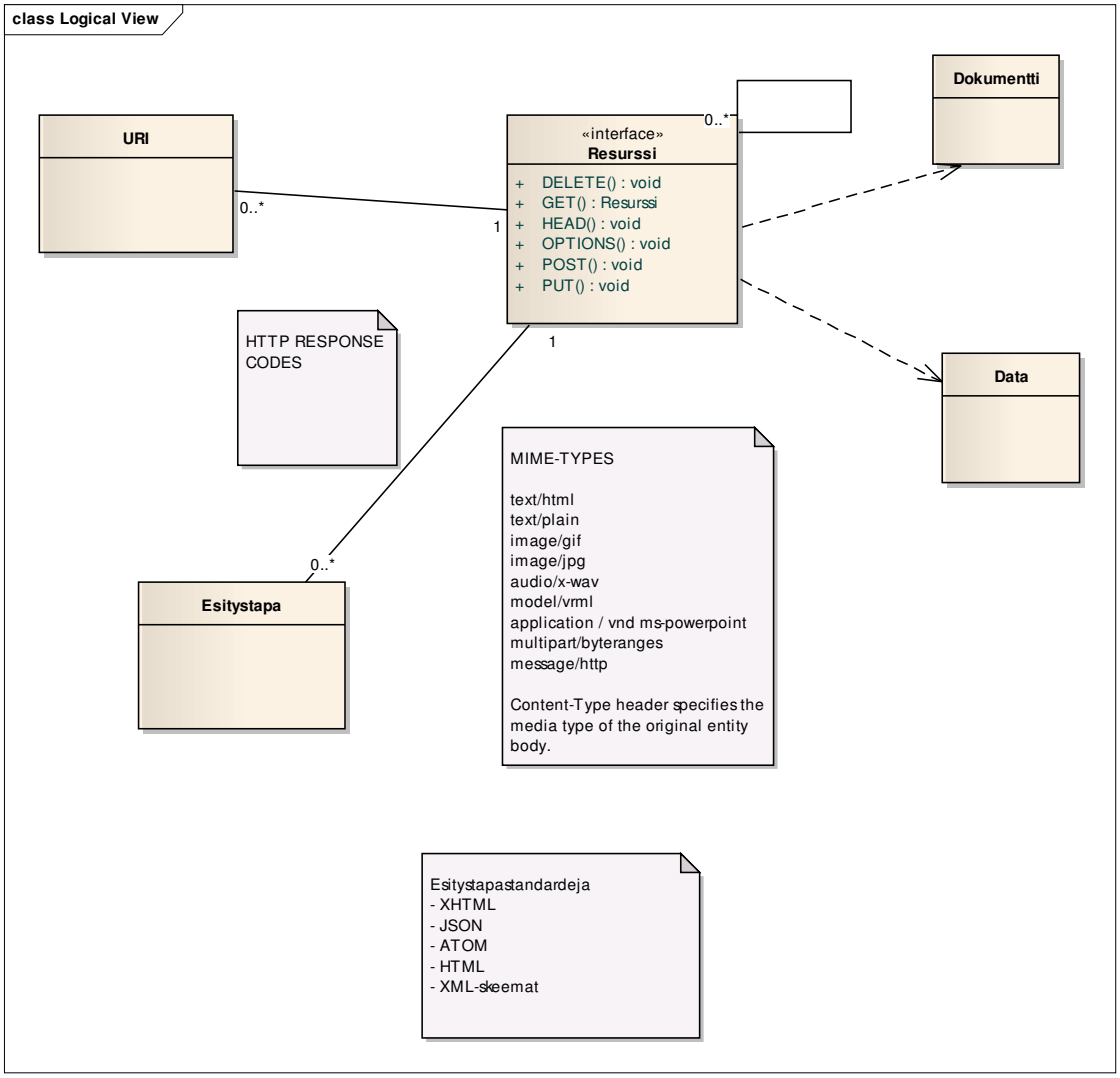
Mikäli resurssilla on useita URL-osoitteita, on asiakkaiden helpompi viitata resurssiin. Haittapuolena on kuitenkin se, että jokainen URL-osoite kumoaa toisten merkityksen, sillä jotkut asiakkaat käyttävät yhtä ja toiset toista osoitetta, eikä voida automatisoidusti verifioida, että URI-osoitteet osoittavat samaan resurssiin [RR07].

Käytettävyystutkija Jakob Nielsenin mukaan jokaisen resurssin tulisi olla osoitettavissa ns. mukavan (”nice”) URI:n avulla. URI:en ominaisuuksiksi listataan mm. pysyvyys (Durability), ennustettavuus (Predictability), suppeus (Conciseness), luettavuus (Readability), yhdenmukaisuus ja abstraktointi toteutuksen yksityiskohdista. [NIE99]

Sisällön neuvottelemisella (Content Negotiation) tarkoitetaan palvelimen kykyä erotella asiakkaan ympäristöstä esimerkiksi käytettävä kieli. Ryby suosittelee HTTP:n Accept Language otsikon (Header) käyttöä ja samaa URL - osoitetta resurssin eri kieliversioille.

Ruby suosittelee, että jokaiselle resurssin erilliselle esitystavalle määritetään oma URI, sillä URI:ja käytetään usein syötetietona toiselle palvelulle, jolloin oletetaan tietty esitystapamuoto (esim: XML tai JSON). [RR07] Esimerkiksi W3C ylläpitää palvelua,

johon syötetietona voidaan antaa URL- osoite ja palvelu palauttaa raportin, onko sivun esitystapamuoto valitun standardin mukainen.



Kuva1. Esim: URI, Esitystapa ja Resurssi.

Kuvan 1 mukaisesti, resurssilla tulee olla vähintään yksi URL [RR07]. Jokainen URL osoittaa yhteen resurssiin. Resurssiin voi liittyä myös muita resursseja. Resurssilla tulee olla vähintään yksi nimi ja nimiä tulee olla niin vähän, että jokainen nimi on merkitsevä [RR07]. RESTin mukainen Web-palvelu tarjoaa käyttäjälleen resurssin esitystapoja. Resurssilla voi olla 0 tai useampi esitystapa. Esitystavalla tarkoitetaan tietomuotoa, joka sisältää informaatiota resurssista [RR07].

Esimerkkejä mahdollisista resursseista Web-sovelluksessa

- Ohjelmiston versio 1.03
- Ohjelmiston viimeisin versio
- Algoritmin tulos

[RR07]

- Henkilötietojen esitys

4.2 Resurssien muokkaaminen esitystavan kautta

RESTin mukaisessa palvelussa resursseja ei voida muokata kuin niiden esitystapojensa kautta [RR07] yhtenäistä rajapintaa käyttäen. Esitystapaformaatteja on useita mm. XML, XHTML, JSON, ATOM, SVG,RDF ja erilaiset XML-sanastot.

4.3 Itsekuvaavat viestit

REST:ssä välittäjäkomponentit voivat aktiivisesti muuntaa ja käsitellä viestien sisältöä, sillä viestit ovat itsekuvaavia (SelfDescriptive) ja niiden semantiikka on näkyvissä välittäjäkomponenteille. XML-viestit ovat esimerkiksi itsekuvaavia.

Viestien metatietoja voidaan käyttää välimuistinhallinnassa (Cache Control), tietonsiirtovirheiden havaitsemisessa, esitystavan valinnassa, autentikoinnissa ja pääsynhallinnassa (Access Control) [FIE00]

4.4 Hypermedia sovelluksen tilakoneena

RESTin mukaisessa palvelussa palvelin johdattaa asiakasta palvelun tiloista toisiin liittämällä resurssien esitystapoihin linkkejä toisiin resursseihin ja niiden esitystapoihin.[FIE00]

4.5 REST- rajapintojen kuvaaminen

RESTin mukaisten palveluiden rajapintojen kuvaamiseen voidaan käyttää erityisen XML-skeeman mukaista XML-kieltä [WADL].

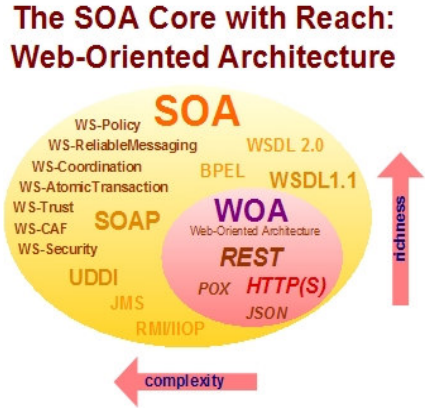
Pautasso ym. toteavat, että rajapintojen kuvaaminen on yleisesti hyödyllistä, jotta muutokset rajapinnoissa ja niiden aiheuttamat kehityksenaikaiset virheet saadaan kiinni aikaisessa vaiheessa. RESTin mukaisesti rajapinnan operaatiot eivät muutu, mutta URL-osoitteet ja resurssien esitystavat voivat muuttua kehityksen aikana [PAU09].

Rubyn mukaan myös useat REST rajapintojen tarjoajat eivät julkaise WADL-kuvauksia rajapinnoista, mutta esimerkiksi Yahoo NewsSearch palvelu julkaisee [RR07]. WADL-kielellä voidaan kuvata mm. resurssit, suhteet resurssien välillä, HTTP-metodit joihin resurssit vastaavat ja rajapinnan palauttamat tiedostoformaatit [HAN06]. Liitteessä 1 on esimerkki Yahoo News Search- WADL-tiedostosta ja Java-työkaluilla tuotetuista tynkäluokista.

5. Web- suuntautunut arkkitehtuurityyli

Web-suuntautunut arkkitehtuuri (Web Oriented Architecture, WOA) on teknologiakonsulttiyhtiö Gartnerin (Nick Gall), vuonna 2005 konferenssissä käyttämä termi, josta myöhemmin kehitettiin nimi arkkitehtuurityylille. Gall kuvaa Krishanin haastattelussa, että WOA on palvelusuuntautuneen arkkitehtuurityylin (SOA) alityyli (Substyle). Gall määrittelee myös, että Web-palvelun tulee noudattaa, mutta ei tarvitse täyttää kaikkia RESTin rajoitteita ollakseen WOA: n mukainen. Gall määrittelee yhtälön, joka kuvaa mitä WOA on.

WOA = SOA + REST + WWW
[LAW08]



Kuva2 SOA, WOA ja REST

Toinen määritelmä WOA:lle voisi olla arkkitehtuurinen SOA:n (Service Oriented Architecture) alityyli, joka yhdistää järjestelmiä ja käyttäjiä linkittyneen hypermedian avulla, perustuen WWW:n arkkitehtuuriin. Gartnerin määritelmän mukaan WOA on arkkitehtuurityyli, joka keskittyy käyttäjä- (UI) ja ohjelmointirajapintojen (API) yleisyyteen asettamalla viisi yleistä rajapintarajoitetta. Neljä rajapintarajoitteista on kuvattu Fieldingin REST:ssä (ks. Luku 4) ja Gall lisää yhden rajapintarajoitteen sovellusriippumattomuus (Application Neutrality). Sovellusriippumattomuus tarkoittaa yleisesti hyväksyttyjen mediatyyppien (MIME- types) käyttöä. WOA suosittelee rajapintojen tunnisteiden, protokollien ja tietomuotojen (*Generic Identifiers, Protocols and Formats, IPAFS*) olevan niin yleisiä kuin mahdollista ja pitää perinteisten Web-palvelujen toteutusriippumattomuutta (Implementation Neutrality) toisarvoisena tavoitteena rajapintojen suunnittelussa [NIC08]

6. Työkalutuki

Java-ympäristössä palvelinkomponentti voidaan toteuttaa Web-palvelimella HttpServlet-komponenttina ja useimmille kielille löytyy HttpClient- kirjastoja kutsujen suorittamiseksi ohjelmallisesti. Web-palvelimelle asennettuja REST-komponentteja voidaan kutsua ja testata WWW-selaimella. Java-Sun on tehnyt JAX-RS (Java API For

RESTful WebServices)-määrittelyn[2] ja toteuttanut määrittelyjä tukevan kehyksen Jerseyn. Jerseytä ennen kehitettiin RestLet - kehys[3]. JBoss - tarjoaa palvelinriippumattoman kehyksen RestEasy[4]. Java EE-palvelinohjelmistoa voidaan käyttää myös palvelinkomponenttien ajoympäristönä, mikäli se on tarpeellista. XML-viestien parsimiseen löytyy työkaluja (esim. Java: SAX, DOM, JDOM) useimmilta ohjelmointikieliltä. XML-tiedostoja voidaan muokata XHTML/XML-muotoon XSLT / XSL- tyylitiedostojen avulla. REST -rajapintojen kuvaukseen suositellaan XML:n mukaista WADL -kieltä. Java-ympäristössä on ainakin Sun:in WADL-tuki. Useimmista tietokannoista löytyy XML-tukea. Avoimen lähdekoodin Exists- tietokannassa on REST-tuki[5]. XML-viestejä voidaan salata standardoidusti[6]. Työssä ei tilanpuutteen vuoksi käsitellä työkaluja enempää.

5. YHTEENVETO

Arkkitehtuureja voidaan suunnitella valmiista komponenteista tai määräämällä rajoitteita, jotka tuovat järjestelmään ominaisuuksia. REST- palveluja voidaan kehittää vähillä työkaluilla ja asiakasohjelmien kehittäminen on helppoa. REST- rajapintoja voidaan testata kehityksenaikana suoraan Web-selaimella.[PAU09] REST on erityisen käytännöllinen myös mobiililaitteissa verrattuna SOAP-viesteihin ja niiden mahdollisesti tarpeettomaan yleisrasitteeseen (Overhead) [TY06]

WWW koostuu resursseista, mutta perinteiset Web-palvelut eivät tarjoa resursseja. WWW koostuu URL- osoitteista ja linkeistä, mutta tyypillisesti perinteiset Web-palvelut paljastavat yhden URL- osoitteen, eivätkä linkkejä toisiin palveluihin. WWW- perustuu HTTP:hen, mutta perinteiset Web-palvelut eivät käytä HTTP:n ominaisuuksia juuri lainkaan [RR07].

Generoidut SOAP/WSDL rajapinnat ovat osittain yhteensopimattomia. Erilaiset Web Services -työkalupinot tulkitsevat standardeja erilailla ja generoivat toisistaan poikkeavia WSDL-kuvauksia, eivätkä ymmärrä välttämättä toistensa viestejä ajonaikana. Web-palvelujen asiakkaat ovat näin vahvasti kytkettyjä palvelijoihin, jotka käyttävät samoja WS* teknologiapinoja ja niiden versioita [RR07]. Kuvassa 3 on Web-palveluiden standardeja listattu kategorioittain.

Yhteentoimivuus (Interoperability)	10
Tietoturva (Security)	10
Meta-tiedot (Metadata)	9
Viestinvälitys (Messaging)	9
Liiketoimintaprosessi (Business Process)	7
Resurssi (Resource)	7
Tietomuunnokset (Translation)	7

[2] http://jcp.org/aboutJava/communityprocess/final/jsr311/index.html
[3] http://www.restlet.org/
[4] http://www.jboss.org/file-access/default/members/resteasy/freezone/docs/1.0.1.GA/userguide/html_single/index.html
[5] http://exist.sourceforge.net/devguide_rest.html
[6] http://www.w3.org/TR/xmlenc-core/; Michael Rosen, Boris Lublinsky, Kevin T. Smith, Marc Balser: Applied SOA, Service Oriented Architecture and Design Strategies

XML (Extensible Markup Language)	7
Management (Hallinta)	4
SOAP (Simple Object Access Protocoll)	3
Esitystapa (Presentation)	1

Kuva 3 WS*- standardeja kategorioittain [KAL09]

Tulevaisuudessa RESTin mukaiset järjestelmät ja palvelut tulevat lisääntymään. RESTin mukaisten järjestelmien suunnitteleminen ja toteuttaminen on osoittautunut vielä toistaiseksi haastelliseksi. Toteutettuja palveluita on useita eivätkä ne ole noudattaneet RESTin rajoitteita kuin osittain. Palvelut myös kärsivät rajoitteiden rikkomisesta.

RESTin mukaisissa Web-palveluissa joudutaan kehittämään enemmän omia ratkaisuja ja mahdollisesti organisaatiokohtaisia standardeja, kuin toteutettaessa ja käytettäessä perinteisiä WS*- teknologiapinon mukaisia Web-palveluita ja standardeja. Järjestelmässä voidaan käyttää sekä RESTin, että WS* -teknologiapinon mukaisia Web-palveluita. RESTin resurssien ja niiden linkittämisen avulla voidaan esimerkiksi helposti tehdä käytettäväksi liiketoiminnallisesti merkittävää raportointia yli järjestelmä- ja organisaatiorajojen. Koosteiset ns. MashUp –sovellukset lisääntyvät jatkuvasti ja toimittajat tarjoavat omia RESTin mukaisia rajapintoja käytettäväksi järjestelmäintegraatiossa.

Kannattaa harkita tarkkaan, mitkä osat yritysarkkitehtuurista ja sen tarjoamista palveluista toteutetaan RESTin mukaisilla Web-palveluilla ja missä käytetään WS*-teknologiapinoa. Kehittäjät suosivat RESTiä helppokäyttöisyyden vuoksi ja WS* -teknologiapinon työkaluihin kehitetään jatkuvasti tukea RESTin mukaisten Web-palveluiden toteuttamiseen.

LÄHTEET

GAL08 WOA keskustelulista: http://tech.groups.yahoo.com/group/rest-discuss/messages/11535?threaded=1&m=e&var=1&tidx=1

GT02 HTTP, The definitive Guide. O'Reilly.

RLS08 Michael Rosen, Boris Lublinsky, Kevin T. Smith, Marc Balser: Applied SOA, Service Oriented Architecture and Design Strategies, ISBN: 978-0-470-22365-9

NIC08 Nick Gall, WOA: Putting the Web Back in Web Services http://blogs.gartner.com/nick_gall/2008/11/19/woa-putting-the-web-back-in-web-services/

LOR08 Loraine Lawson Why WOA vs. SOA Doesn't Matter

http://www.itbusinessedge.com/cm/community/features/interviews/blog/why-woa-vs-soa-doesnt-matter/?cs=23092

HIN06 Dion Hinchcliffe, Verkkojulkaisu, The SOA with reach: Web-Oriented Architecture http://blogs.zdnet.com/Hinchcliffe/?p=27

EGS07 Justin R. Erenkrantz, Michael M. Gorlick, Girish Suryanarayana, Richard N. Taylor : From Representations to Computations: The Evolution of Web Architectures

FIE00 Fielding, Roy Thomas. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, 2000

FT02 Roy T. Fielding, Richard, N. Taylor: Principled design of the Modern Web Architecture University of California, Irvine. ACM Transactions on Internet Technology, Vol. 2, No. 2, May 2002.

HAN06 Marc J. Hadley, Sun Microsystems Inc. Web Application Description Language (WADL)

KM05 Kai Koskimies, Tommi Mikkonen. Ohjelmistoarkkitehtuurit

LPR03 Len Bass, Paul Clements, Rick Kazman: Software Architecture in Practice, Second Edition, Addison Wesley ISBN: 0-321-15495-9

TY06 Sameer Tyagi, Verkkojulkaisu. RESTful Web Services
http://java.sun.com/developer/technicalArticles/WebServices/restful/

PAU08 Cesare Pautasso, Olaf Zimmermann, RestFul Web Services vs. “Big” Web services: Making the right architectural decision

W3C04 David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion Web Services Architecture, W3C Working Group Note 11 February 2004. Saatavilla osoitteesta (http://www.w3.org/TR/ws-arch/)

RR07 Leonard Richardson, Sam Ruby. RESTful Web Services, O’Reilly.

HINCH06 Dion Hinchcliffe, Verkkojulkaisu, The SOA with reach: Web-Oriented Architecture http://blogs.zdnet.com/Hinchcliffe/?p=27

PRG09 http://www.programmableweb.com/apis

WADL https://wadl.dev.java.net/wadl20061109.xsd

NIE99 Jakob Nielsen, URL as UI: http://www.useit.com/alertbox/990321.html