TAMPEREEN TEKNILLINEN YLIOPISTO

Kyösti Herrala

Rikas internetsovellus luottamukselliseen reaaliaikaviestintään

Diplomityö

Tarkastaja: Tommi Mikkonen
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekunta-
neuvoston kokouksessa 8. syys-
kuuta 2010

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Signaalin käsittelyn ja tietoliikennetekniikan koulutusohjelma
HERRALA, KYÖSTI: Rikas internetsovellus luottamukselliseen reaaliaikaviestintään
Diplomityö, 45 sivua
Maaliskuu 2011
Pääaine: Sulautetut järjestelmät
Tarkastaja: Tommi Mikkonen
Avainsanat: Web-sovellus, RIA, keskustelujärjestelmä, chat

Pelastakaa lapset ry on järjestänyt internetissä tapahtuvaa lapsille ja nuorille tarkoitettua tukiryhmätoimintaa, jonka apuna on käytetty eri palveluntarjoajien online-keskustelujärjestelmiä. Tässä diplomityössä suunnitellaan ja toteutetaan tarkoitusta varten räätälöity keskustelujärjestelmä, jonka ei-toiminnallisin ominaisuuksiin kuten tietoturvaan ja laajennettavuuteen on panostettu. Työn tavoitteena on tutkia miten viestintäsovellus rakennetaan nykyaikaisessa Web-ympäristössä, sekä millä suunnittelupäätöksillä keskustelujärjestelmän ylläpitoa ja jatkokehitystä voidaan tukea.

Työ jakautuu kolmeen osaan: Ensimmäisessä osassa kuvaillaan toteutusympäristöä rikkaiden internet sovellusten näkökulmasta, ja esitellään sovelluksen mahdollistavan selaimeen tarkoitetun push-teknologian kehittymistä. Toisessa osassa kuvataan suunniteltavan järjestelmän vaatimukset, joiden perusteella teknologiaksi valitaan XMPP-protokolla Google Web Toolkit-ympäristö (GWT), joita kuvaillaan yksityiskohtaisemmin. Viimeisessä osassa esitetään suunnitellun järjestelmän arkkitehtuuri ja tärkeitä Web-sovellusta koskevia suunnitteluratkaisuja.

Keskustelujärjestelmä koostuu monipuolisesta mutta kevyestä Web-asiakkaasta, joka integroituu XMPP-verkkoon standardinmukaisella tavalla, joten sitä voidaan jatkokehittää helposti. GWT-ympäristön Java-pohjaisuus helpotti korkean tason suunnittelumallien käyttöä ja paransi Web-ympäristön päälle rakennettuja abstraktioita. Valmiin ja testatun toteutusinfrastruktuurin käyttö oletettavasti parantaa myös järjestelmän laadullisia ominaisuuksia.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Signal Processing and Communications Engineering Technology
HERRALA, KYÖSTI : A Rich Internet Application for confidential real-time messaging
Master of Science Thesis, 45 pages
March 2011
Major: Embedded systems
Examiner: Tommi Mikkonen
Keywords: Web-application, RIA, messaging system, chat

Save the Children Finland has been organizing online support groups for children and young adults using various messaging platforms by different providers. The motivation of this thesis is to produce a tailormade messaging solution, with strong emphasis on non-operational qualities such as security and extendability. The goal is to study how to build a modern messaging application in the Webenvironment, as well as to apply sound patterns of design to simplify further development and maintenance.

The thesis consists of three parts: The first part considers the working environment from the perspective of Rich Internet Applications and explains the enabling role of browser-based push technology. In the second part, the reader is presented with the key requirements that supported the adoption of the XMPP-protocol and the Google Web Toolkit (GWT). Further evaluation on the properties of these technologies ensue. Finally in the concluding part the architecture of the designed system is presented including major design decisions for the Web-client.

The produced messaging solution comprises of a light-weight but fully functional Web-client, which integrates with a standard XMPP-network. Therefore it can be easily further developed. Exploiting the properties of GWT based on Java stimulated the usage of high-level design patterns and enchanced the abstraction level of the underlying WWW-platform. The benefits of using pre-existing system components will likely make a difference from the aspect of system quality.

ALKUSANAT

Tämä diplomityö on tehty Advant Games Oy:n palveluksessa kesän ja syksyn vuonna 2010 aikana. Toteutetun järjestelmän dokumentointi alkoi syksyllä vuonna 2010 ja valmistui keväällä vuonna 2011.

Tampereella, 20. maaliskuuta 2011

Kyösti Herrala

SISÄLLYS

KUVAT

TAULUKOT

LISTAUKSET

1. JOHDANTO

Osana toimintaansa Pelastakaa Lapset ry (myöh. PLRY) on ajoittain järjestänyt varhaisnuorille tarkoitettuja ryhmäkeskusteluja internetissä. Näiden keskusteluiden tarkoituksena on ollut tarjota nuorille tukea sekä keskustelumahdollisuus luotettavien aikuisten kanssa. Keskustelutilaisuuksia on järjestetty sekä säännöllisesti että merkittävien kriisien yhteydessä niin sanottuina kriisichatteinä. Toiminta ei ole siis ympärivuorokautista, vaan ohjattuja keskustelutilaisuuksia on järjestetty vapaaehtoisten valvojina toimineiden aikuisten avulla. [27]

Aiemmin toiminta on järjestetty useiden erillisten palveluntarjoajien, kuten IRC-galleria[1] ja Suomi24[2], avustuksella. Toimittajien erilaisista järjestelmistä ja työkaluista johtuen PLRY:llä on ollut hankaluuksia kouluttaa uusia valvojia tukiryhmätoimintaa varten. Toimintojen keskittämisen ja yhtenäistämisen vuoksi on päädytty rakentamaan irrallinen keskustelujärjestelmä, joka voidaan myös tarvittaessa integroida osaksi aiempia järjestelmiä. Aiempien alustojen toiminnoista haluttiin karsia ylimääräiset pois ja parantaa käytettävyyttä kouluttamisen helpottamiseksi. Sovellukseen haluttiin myös laatia parempia työkaluja moderointia varten. [27]

Tässä diplomityössä esitetään luottamukselliseen internetin välityksellä tapahtuvaan yksityskeskusteluun sekä ohjattuun tukiryhmätoimintaan tarkoitetun järjestelmän suunnittelu ja toteutus. Työn tilaajana toimi Pelastakaa Lapset ry, joka on vuosia toivonut käyttöönsä räätälöityä ja itsenäisesti hallinnoitua viestintäsovellusta. Työ toteutettiin Dimeke-rahoituksella ja toteutusvastuu oli tamperelaisella Advant Games Oy:llä.

Rakennettavalla sovelluksella on monipuolinen käyttäjäkunta ja sen tulee pystyä palvella mahdollisesti suurtakin yhtäaikaista käyttäjämäärää. Täten sovelluksen tulee olla käytettävä, helposti laajennattava ja skaalautuva. Järjestelmän toteutusarkkitehtuuri perustuu XMPP-protokollaperheeseen [9]. Asiakassovellus on rakennettu Google Web Toolkitin (GWT) avulla [4]. Se korostaa ylläpidettävyyttä, komponenttiajattelua sekä helpottaa korkean tason suunnittelumallien käyttöä.

Luvussa 2 käsitellään keskustelujärjestelmän roolia Web-sovelluksena, ja luodaan yleiskuva toteutusympäristön keskeisistä piirteistä. Suunnittelutyössä tehtyjä ratkaisuja pohjustetaan tutustumalla tarkemmin asynkroniseen viestinvälitykseen WWW:ssä, jolla on keskeinen rooli sovelluksen mahdollistavana tekniikkana. Luvus-

[1] `http://irc-galleria.net/`
[2] `http://www.suomi24.fi/`

sa 3 käydään ensin läpi järjestelmälle ja suunniteltavalle Web-sovellukselle asetettuja vaatimuksia, joiden pohjalta luodaan perusteltu joukko teknologisia ratkaisuja. Lisäksi analysoidaan lyhyesti vaihtoehtoisten ratkaisujen ominaisuuksia. Lopuksi perehdytään tarkemmin kahteen keskeisimpään teknologia-ratkaisuun: XMPP-protokollaperheeseen sekä GWT-sovelluskehitysympäristöön. Luvussa 4 kuvataan suunnitellun järjestelmän kokoonpano sekä rajoitetusti joitakin toteutusyksityiskohtia. Esityksen pääpaino on havainnollistaa monimutkaisen Web-asiakasovelluksen toteutuksessa käytettyjä ja hyväksi havaittuja suunnittelumalleja. Luvussa 5 kootaan yhteen projektin suunnittelupäätökset ja arvioidaan sovelluksen vaatimusten toteutumista.

2. WEB TOTEUTUSYMPÄRISTÖNÄ

2.1 Taustaa

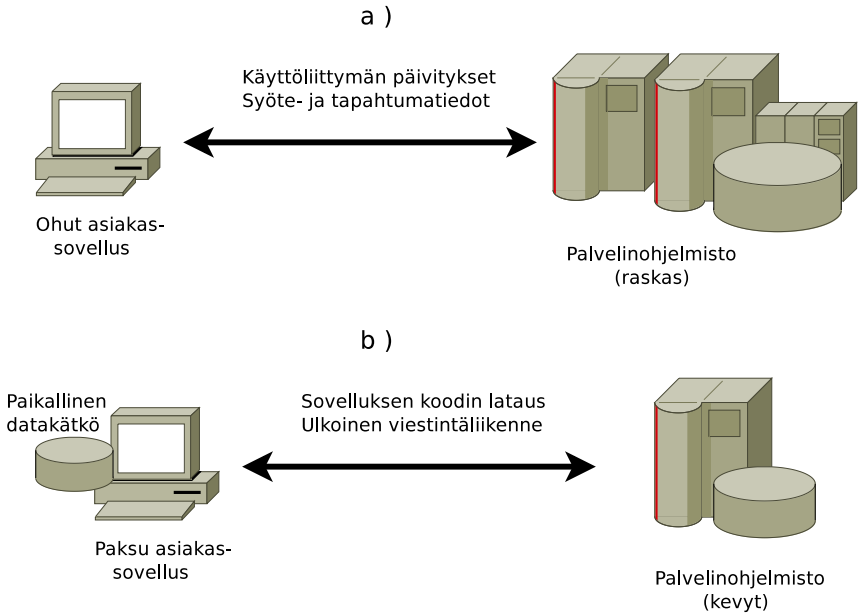
Web-sovellusten kehittäminen ja suunnittelu ovat kokeneet viimeisten vuosien aikana suuria muutoksia. Web-sovellusten teknologia on tuoretta ja ratkaisut monin osin vakiintumattomia. Keskustelutyökalun tyyppisen sovelluksen toteuttamiseen on tarjolla suuri joukko erilaisia vaihtoehtoja ja lähestymistapoja.

Erilaisten lähestymistapojen vertailu vaatii ymmärrystä Web-teknologian viimeaikaisesta kehityksestä. Erityisesti asynkroniseen tiedonkäsittelyyn perustuvat Web-teknologiat ovat edistäneet internetiä hyödyntävien viestintäsovellusten toteuttamista. Myös Web-kehitykseen tarkoitettujen sovelluskehysten ja Web-selaimeen perustuvien alustojen yleistyminen on helpottanut yhä monimutkaisempien asiakassovellusten suunnittelua.

Web-sovellusten monimutkaistuminen on kasvattanut myös erilaisten arkkitehtuurivalintojen merkitystä. Tämän vuoksi seuraavaksi käsitellään tarkemmin Web-sovellusten yleisarkkitehtuurin valintaa. Tämän erottelun esittelemisen jälkeen voidaan myöhemmin keskittyä tarkemmin Web-sovelluksen komponenttien sijoitteluun ja niiden sisäiseen arkkitehtuuriin.

2.2 Sovelluksen yleisen arkkitehtuurin vaihtoehdot

Merkittävin Web-sovellusten arkkitehtuurivalinta koskee sovelluksen komponenttien ja liiketoimintalogiikan sijoittelua asiakkaan ja palvelimen välille. Asiakas-palvelin -sovellusten arkkitehtuurit voidaan jakaa kahteen ryhmään asiakkaan perusteella: ohuisiin (thin-client) ja paksuihin (thick-client) asiakkaisiin. Vaihtoehtoja erottavana tekijänä on asiakkaan käytössä olevan ohjelmiston monimutkaisuus ja sen vaatimat resurssit. Aiemmissa yhteyksissä tällä jaottelulla on voitu tarkoittaa fyysisten laitteistojen erottelua esimerkiksi päätteisiin ja keskuskoneisiin, mutta tässä yhteydessä käsitellään ohjelmistojen eroja. [14]



Kuva 2.1: Ohut ja paksu asiakassovellusarkkitehtuuri [14]

Ohuella asiakkaalla tarkoitetaan komponenttia, jonka asiakaspäässä suorittamien toiminnallisuuksien määrää on karsittu. Tätä havainnollistaa kuva 2.1a. Toimintojen karsimisen syinä voivat olla vähentää sovelluksesta käyttäjälle syntyviä kustannuksia tai tarjota enemmän joustavuutta sovelluksen käytössä. Paksun asiakkaan tapauksessa (kuva 2.1b) toiminnot suoritetaan pääsääntöisesti asiakkaan käytössä olevilla resursseilla ja käsiteltävä data säilytetään paikallisesti. Ainoastaan tietoliikennettä ja viestintää vaativa tiedonkäsittely tapahtuu palvelimen tarjoamilla resursseilla. [14]

Rajoittamalla asiakaspäässä suoritettavien toimintojen määrää kasvatetaan palvelinohjelmiston merkitystä. Tarkkaa jakoa koko sovelluksen toimintojen sijoittelusta eri arkkitehtureissa asiaakkaaseen tai palvelimeen ei ole määritelty. Äärimmäisessä tapauksessa ohut asiakas vastaa ainoastaan esityskerroksen vaatimuksista ja vastaanottaa käyttäjän syötettä. Asiakassovellus toimii niin sanotusti tyhmänä päätteenä. Tällöin esimerkiksi jokainen asiakkaan käyttöliittymätapahtuma, kuten hiiren napsautus, lähetetään palvelimelle käsiteltäväksi. Myös tapahtumaa mahdollisesti seuraava käyttöliittymän päivitys ohjataan yksityiskohtaisesti palvelinpäästä. [14]

Hieman yleisempää on se, että ohut asiakas vastaanottaa palvelimelta käsiteltävää dataa, mutta lataa lisäksi suoritettavakseen ohjelman osia. Tällöin asiakas voi käsitellä jonkin verran informaatiosta paikallisesti riippumatta palvelimesta ja samalla vähentää olennaisesti palvelimen kuormitusta. Ohuen asiakkaan etuna on sovelluksen vaivaton muokkaaminen ja päivittäminen, koska näissä tilanteissa yksittäisiä asiakassovelluksia ei tarvitse ylläpitää. Ohut asiakas kuormittaa myös vähemmän käyttäjän laitteistoa järeämmän palvelinkuorman uhalla, ja järjestelmän ylläpito on

keskittynyt palvelimen ohjelmistoon. [14]

WWW-sovelluksien aiemmin käytettävissä olleen teknologian rajoitteista johtuen on ollut tavallista käyttää ohuita asiakassovelluksia, jotka ovat vahvasti palvelinkeskeisiä. Viimeaikainen uusien selainteknologioiden, joita esitellään jatkossa, aikaansaama muutos on kuitenkin tehnyt paksuista asiakkaista yhä houkuttelevampia. Kehitys on johtanut kohti yhä monipuolisempia niin sanottuja rikkaita internetsovelluksia (engl. RIA = Rich Internet Application).

2.3 Selain sovellusalustana

2.3.1 Johdanto

Internetin kehityksen alkuvaiheessa olemassaolevat WWW-teknologiat rakentuivat ajatukseen staattisesta sisällöstä, jota navigoidaan HTTP-protokollan [10] avulla siirtymällä sivulta toiselle hyperlinkkien avulla. Tällöin käyttäjä lähettää pyynnön selaimen avulla, jotta palvelin esimerkiksi lähettäisi halutusta dokumentista uusimman sisältöversion. Sisällön päivittäminen, uuden sisällön hakeminen, tai muu vuorovaikutus vaatii aina uuden pyynnön tekemisen. Pyyntöjen välillä ei välttämättä ole mitään yhdistävää tekijää, joten ne ovat oletusarvoisesti toisistaan riippumattomia. Toisin sanoen HTTP on pohjimmiltaan tilaton rikkaiden internetsovellusten vaatimuksista poiketen. [28]

Alkuperäisen Webin arkkitehtuurityyli on REST[1] [16]. Sen ajatuksena on yksinkertaistaa tiedon muokkaus- ja kulutusoperaatiot neljään perusoperaatioon, joiden toteutuksena käytetään HTTP-protokollan yhteydessä GET-, PUT-, POST- ja DELETE-metodeja. Pyynnöt kohdistuvat yksilölliset nimettyihin resursseihin (URI), ja ne välittävät resurssin tilatiedon tilattomasti asiakkaan ja palvelimen välillä.

REST-ajatusmaailman yksinkertaistava perusajatus on historiallisesti vaikuttanut WWW-teknologian laajaan leviämiseen. Resursseihin ja niiden vapaaseen esitysmuotoon perustuva arkkitehtuurityyli suo paljon joustavuutta HTTP-teknologian käyttöön ja on siksi vakiintunut osa internetiä. REST:n eduiksi katsotaan hyvä saavutettavuus ja helppo lähestymistapa skaalautuvuuteen, sillä se helpottaa sisällön levitystä välityspalvelimien avulla ja säilömistä välimuisteissa. Säilömisen tulisi olla aina mahdollista GET-metodin yhteydessä, koska sen ei ole luvallista tehdä muutoksia resurssin tilaan, joten pyynnön vastaus voidaan säilöä seuraavaan muutoksen aiheuttavaan pyyntöön asti. [16]

Pyyntö-vastaus -semantiikan käyttäminen ja HTTP-yhteyden luontainen tilattomuus tekivät alkuperäisestä WWW:stä kankean sovellusalustan. WWW:n käyttökokemusta tuli parantaa karsimalla kokonaisten sivulatausten määrää, jotka ka-

[1] Representation State Transfer

dottavat käyttäjän fokuksen ja aiheuttavat tarpeetonta tiedonsiirtoa. Peräkkäisiin sivunlatauksiin perustuvan sovelluksen käyttö on katkonaista. WWW-alustalla toimivan sovelluksen tulisi reagoida käyttäjän aiheuttamiin käyttöliittymätapahtumiin välittömästi ilman kokonaisesta sivunlatauksesta aiheutuvaa ylimääräistä viivettä. Pikaviestimen ja monen muun nykyaisen WWW-sovelluksen käytössä vaaditaan lisäksi keino reagoida taustajärjestelmästä saapuviin tapahtumiin asynkronisesti ilman käyttäjän toimenpiteitä.

Tavallisten käyttöjärjestelmään sidottujen työpöytäsovellusten siirtyminen WWW-maailmaan vaati alustan mukautumista. Tästä kehityksestä syntyi nykyinen "Web 2.0":ksi kutsuttu ilmiö. Sen tunnettuja suunnannäyttäjiä ovat esimerkiksi Google Mail[2], Facebook[3] ja monet internetin yhteisöpalvelut. Näiden palveluiden navigointi eroaa tavallisista useammasta osoitteella yksilöitävästä sivusta koostuviin siinä, että erilaisia näkymiä varten ei tarvita yksittäisiä sivuresursseja. Vähimmillään riittää vain yksi sovelluksen alustussivu, joka päivittää itseään käytön aikana dynaamisesti.

2.3.2 Web-sovellukset

Web-sovelluksilla tarkoitetaan laajaa joukkoa selainympäristön ja WWW:n päälle rakennettuja sovelluksia, joissa sovelluksen liiketoimintalogiikka, esityskerros sekä tiedon säilyvyys on jaettu *Web-asiakkaan* (engl. web client) ja *Web-palvelimen* (engl. web server) välille. Web-sovellukset ovat yleistyneet monissa julkisissa sekä yritysten kriittisissä liiketoimintajärjestelmissä. Web-alustan suuri levinneisyys ja saavutettavuus ovat saaneet Web-sovellukset korvaamaan aiempia raskaita työpöytäkäyttöön suunniteltuja ohjelmistoja, joten on tullut tarve suunnitella yhä monimutkaisempia Web-sovelluksia. [18]

Gartnerin tuottaman raportin mukaan rikkaat internetsovellukset yhdistävät piirteitä perinteisistä GUI-pohjaisista työpöytäsovelluksista WWW-sivujen hyvään saavutettavuuteen. Työpöytäsovellukset perustuvat paksuun asiakasarkkitehtuuriin ja ovat sidottuja paikallisen alustan ja käyttöjärjestelmän ominaisuuksiin ja rajoituksiin. Niiden arkkitehtuurissa korostuu tapahtumapohjainen käyttöliittymälogiikka. [18]

Ohueen arkkitehtuurin perustuvien Web-sovellusten ongelma on ollut yksittäisten sivujen eli dokumenttien lataukseen perustuva teknologia. Lisäksi Web-sovellusten liiketoimintalogiikka on sijoitettu pääsääntöisesti palvelimen suoritettavaksi ja ainoastaan kevyttä skriptiohjelmointiin perustuvaa käyttöliittymän päivitystä sekä syötteen validointia on suoritettu asiakkaan selaimessa. Päävastuu on ollut tehokkailla sovelluspalvelimilla, jotka tuottavat sovelluksen dynaamiset näkymät HTML-pohjaisina WWW-sivuina asiakkaan selaimen näytettäväksi [15].

[2] `http://mail.google.com/`
[3] `http://www.facebook.com`

Ohuen arkkitehtuurin suosion syytä on perusteltu vaatimuksella toimittaa sovellus mahdollisimmman laajalle käyttäjäkunnalle. Connallenin mukaan on tärkeämpää tarjota laajalle käyttäjäjoukolle rajoitetuin toiminnoin varustettu sovellus kuin toimittaa kehittyneempi sovellus rajoitetulle joukolle [17]. Ohuen Web-sovelluksen käyttämiseen asiakkaalta vaaditaan vähimmillään selain, joka osaa näyttää dynaamisia HTML-sivuja sekä kommunikoida palvelimen kanssa HTTP:n avulla.

2.3.3 Rikkaat internetsovellukset

Kehitys kohti paksumpia sovelluksia lähti liikkeelle siitä, että WWW:n kautta voidaan toimittaa myös selaimen rajoitteet ohittavia ja yhä monipuolisempia sovelluksia. Tällainen monipuolisempi Web-sovellus saattaa vaatia erilaisia selaimen lisäosia, mutta toimii selaimen kaltaisessa rajoitetussa sandbox-ympäristössä. Ulkonäöllisesti näiden sovellusten käyttöliittymäkomponentit ja logiikka muistuttavat usein perinteisten työpöytäsovellusten vastineita. Rikkaat internetsovellukset on edesmenneen Macromedia yhtiön lanseerama termi tällaisille Web-sovelluksille [15]. Lähtökohdistaan johtuen RIA on vahvasti kytköksissä erilaisiin selaimen päälle kehitettyihin RIA-alustoihin, kuten Adobe Flash [1]. Näiden alustojen pyrkimyksenä on korjata ohuiden Web-sovellusten puutteita ja tukea yhä monimutkaisempien Web-sovellusten kehittämistä.

Rikkaiden internetsovellusten tutkimus ja määrittely on vielä keskeneräistä, ja niiden toteutustekniikat vaihtelevat suuresti. Rikkaiden internetsovellusten keskeinen piirre on kasvattaa asiakaskomponentin vastuuta Web-sovelluksen toteuttamisessa. Niissä on ohuesta arkkitehtuurityylistä eroten asiakkaan vastuulla pelkän datan esittämisen lisäksi reagoida myös selaimen tapahtumiin, päivittää näkymää sekä toteuttaa ainakin osa sovelluksen liiketoimintalogiikasta. Tyypillisesti rikas internetsovellus saa heti alustusvaiheessa jonkin verran dataa käyttöönsä, jotta toiminta voisi olla itsenäistä. Sen jälkeen palvelinta käytetään vain tarvittaessa, kun haetaan lisää dataa tai lähetetään viestejä. Toimintojen totetusvastuun siirtäminen pois palvelimelta tekee asiakaskomponentista yhä monimutkaisemman. Tämä kehitys asettaa erityisiä vaatimuksia sovelluksen arkkitehtuurille.

Rikkaita internetsovelluksia voidaan tarkastella myös niille asetettujen vaatimusten kautta. Tällöin tietynlaisen teknologian tai alustan rooli kutistuu. Joidenkin lähteiden mukaan tavallisista dynaamisiin HTML-sivuihin ja niiden evoluutioon perustuvat Web-sovellukset muodostavat RIA:lle rinnakkaisen joukon. Käytännössä rikkaiden internetsovellusten vaatimukset voidaan toteuttaa ilman erillisen selaimen päälle rakennetun alustan tukea. Tämä lähestymistapa on mahdollista niin sanotun AJAX-tekniikan avulla. [15] [38]

AJAX perustuu olemassaolleiden ja vakiintuneiden selainteknologioiden saumattomaan yhdistelyyn. Se koostuu seuraavista komponenteista [23]:

- sisällön kuvauskielistä HTML [40] sekä tyylikielestä CSS [13],
- sivun hierarkisen rakenteen manipulointiin ja käsittelyyn tarkoitetusta Document Object Model -rakenteesta (DOM) [11],
- asynkronisen tiedonvälityksen mahdollistavasta XMLHttpRequest-oliosta [41],
- sekä kokonaisuuden yhdistävästä JavaScript-skriptikielestä [2].

2.4 Push/pull tiedonvälitys internetissä

Tavallisesti internetin käytössä on totuttu niin sanottuun pull-tyyppiseen tiedonvälitysmalliin. Se tarkoittaa sitä, että jokaisen tiedonsiirto-operaation aloittamisesta vastaa asiakaspäässä oleva agentti. Tämä agentti voi olla selainta käyttävä ihminen tai selaimessa toimiva dynaaminen skriptiohjelma, joka lähettää palvelimelle pyyntöjä säännöllisesti.

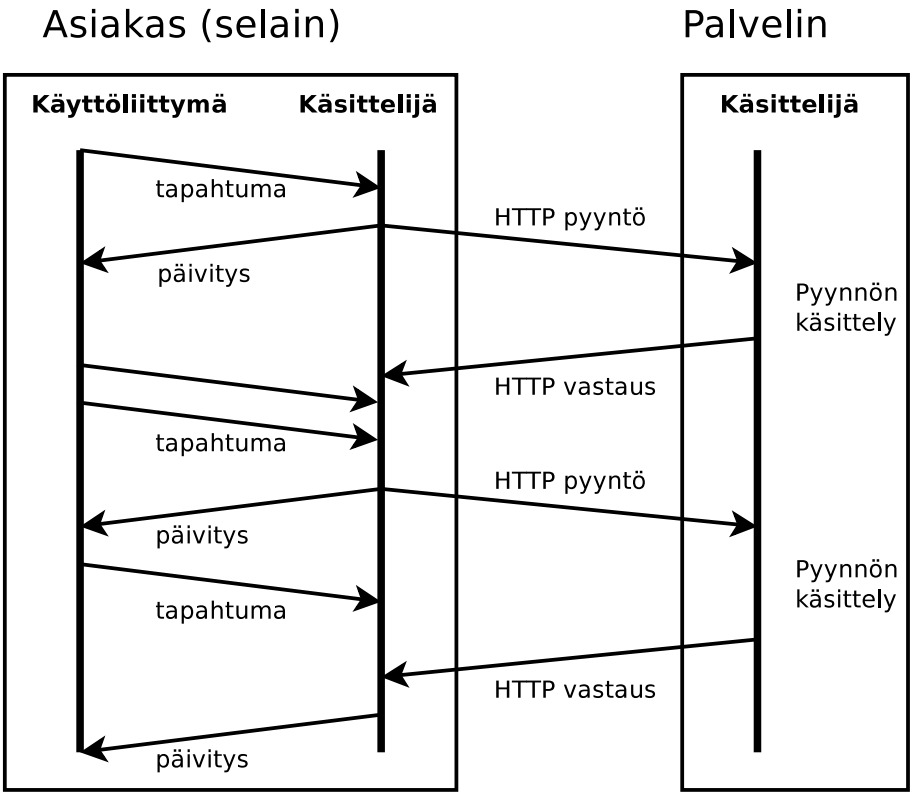
Keskustelualustan toteuttaminen pull-pohjaisella ratkaisulla on mahdollista, mutta ratkaisu kärsii eräistä ongelmista. Merkittävin ongelma on interaktiivisuuden ja lyhyen vasteajan saavuttaminen. Asiakasohjelman täytyisi pystyä synkronoimaan tilansa palvelimen kanssa riittävän usein, jotta käyttäjälle osoitetut viestit saapuvat riittävän pienellä viiveellä. Ainoa tapa tarkkailla viestien saapumista pull-ratkaisussa on suorittaa palvelimen tilakyselyjä toistuvasti säännöllisin väliajoin, tai ääritapauksessa jättää päivitysvastuu käyttäjän tehtäväksi. Kuitenkin siedettävän viiveen aikaansaaminen aiheuttaa valtavan määrän tarpeetonta päivitysliikennettä, jossa hyötykuorman osuus on pieni. Lisäksi jatkuva päivittäminen aiheuttaa mahdollisesti ongelmia palvelun skaalautuvuuteen ja kasvattaa tiedonsiirron kustannuksia. [16]

Pull-tiedonsiirron vaihtoehto on push-tiedonsiirto, jossa päivitysoperaation aloittamisesta vastaa palvelinpuolen viestinjulkaisijakomponentti (engl. publisher). Yleisesti ottaen se reagoi jonkin viestilähteen tuottamaan tapahtumaan, joka kertoo esimerkiksi tietokantamuutoksesta. Tapahtumasta kiinnostuneet vastaanottajat voivat rekisteröityä kuuntelemaan vastaavantyyppistä viestityyppiä, jolloin palvelin lähettää niin sanotuille tilaajille (engl. subscriber) ilmoituksen asynkronisesti ja omaaloitteisesti. Lähettämisestä vastaa viestinvälittäjäkomponentti (engl. dispatcher). [32]

Push-periaatteen käyttötapoihin internetissä kuuluu nykään monenlaisia sovelluksia onlinepeleistä pikaviestintään, koska hajautettujen ja sosiaalista yhteistoimintaa tukevien sovellusten määrä on kasvussa. Työpyötäsovelluksissa push-tiedonsiirron käyttäminen on ollut itsestään selvyys, koska mitään teknologiaestettä ei ole ollut. Sen sijaan Web on vasta hiljattain sopeutunut tämäntyyppiseen käyttöön.

2.4.1 HTTP-pohjainen push-tiedonsiirto

Interaktiivisten Web-sovellusten kehittäminen luo selkeä tarpeen push-tyyppiselle tiedonsiirrolle. Keskustelutyökalun ja muiden samankaltaisten reaaliaikaisten Web-sovellusten toteuttaminen vaatii selkeästi kehittyneen keinon vastaanottaa ulkoisia päivityksiä verkosta. Tarkoituksena on pyrkiä suureen tiedon koherenssiin, eli sopeutua asiakaspäässä ulkoisiin päivityksiin mahdollisimman nopeasti. Ongelmana on kuitenkin se, että kiinnostavien tapahtumien esiintymistahti on tuntematon ja ennalta-arvaamaton.

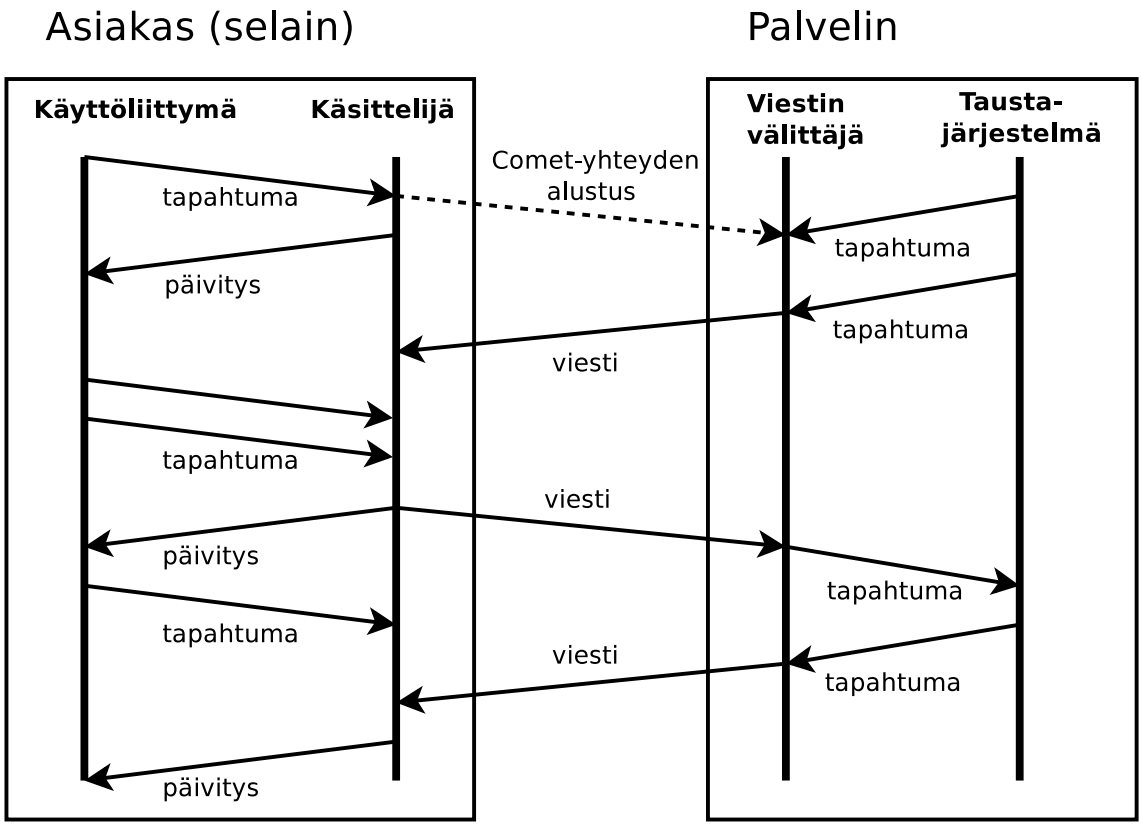


Kuva 2.2: Ajax-tiedonvälitysmalli (pull)

Reaaliaikaisiin sekä kuuntelijatyyppisiin WWW-asiakassovelluksiin perinteisesti käytetty tekniikka on ollut pull-tyyppinen jaksottainen palvelintapahtumien tiedusteleminen [35]. Kuva 2.2 esittää tyypillistä AJAX-taustapyyntöihin perustuvaa pull-tyyppistä sovellusta. Selaimessa toimiva käsittelijäkomponentti kuuntelee käyttöliittymän tapahtumia ja pyytää satunnaisesti tietoja käyttämällä toisistaan erillisiä HTTP-pyyntöjä. Interaktion käynnistäjänä toimii siis aina selain ja palvelin voi välittää taustajärjestelmän tapahtumia vain selaimen pyynnöstä, joka on usein seurausta käyttäjän toiminnasta.

Hienostuneempien push-tyyppisten ratkaisujen käyttö WWW:ssä on lisääntymässä, ja niiden vakiintuminen on alkanut 2000-luvulla. HTTP-pohjaisten push-ratkaisujen kattoterminä käytetään termiä Comet. Termi on sukua AJAX:lle, jotka

kummatkin ovat interaktiivisen “Web 2.0”:n ydinteknologioita (sekä myös pesuaineiden myyntinimiä). AJAX:in tarjoamia työkaluja käytetään Comet:n mahdollistamiseen. [35] [43]



Kuva 2.3: Comet-tiedonvälitysmalli (push)

Kuva 2.3 esittää Comet-tiedonvälitysmallin yleistä toimintaa. Merkittävänä erona Ajax-malliin on se, että asiakas pystyy vastaanottamaan tapahtumia riippumatta käyttäjän tekemisistä. Toiseksi yksittäisiä tapahtumia varten ei vaadita erillistä HTTP-pyyntö-vastausparia. Tieto välitetään olemassaolevan tietoliikennekanavan välityksellä selaimeen välittömästi, kun taustajärjestelmässä julkaistu tapahtuma saapuu palvelimen viestinvälittäjäkomponentille. Tämä lähestymistapa minimoi tiedonvälitykseen kuluvan viiveen tapahtumalähteeltä kuuntelijana toimivalle selaimelle. [35]

Comet-tiedonvälitysmalli on tärkeä lähtökohta keskustelusovelluksen toteuttamiselle. Pieni viive ja tehokas tapahtumien välitys on ensiarvoisen tärkeä ominaisuus reaaliaikaiselle ja monenkäyttäjänvälistä interaktiota vaativalle sovellukselle. Käytännön kannalta Comet-teknologioiden etuna on myös se, että sen käyttämä HTTP-protokolla on yleensä palomuureissa sallittu yhteystapa.

2.4.2 Erilaiset toteutustavat ja BOSH-protokolla

HTTP-pohjaiseen push-tiedonsiirtoon on olemassa erilaisia lähestysmistapoja. Ne pyrkivät osaltaan tarjoamaan HTTP-liikenteelle kaksisuuntaisen socket-tyyppisen ja pitkäkestoisen viestintäkanavan. Useimmat tavat pyrkivät matkimaan kaksisuuntaisuutta erilaisilla pull-periaatteen johdannaisilla. Tähän syynä on HTTP-protokollan REST-arkkitehtuurityyli, joka perustuu asiakkaan aloittamaan tiedonsiirtoon. Se tekee WWW:stä olemukseltaan yksisuuntaisen järjestelmän. Viimeaikoina W3C:n uusi HTML5-standardi on ottanut kantaa aidon kaksisuuntaisuuden puutteeseen. Standardissa kuvattu WebSocket-laajennos pyrkii helpottamaan socket-ohjelmoinnista tuttujen mallien hyväksikäyttöä WWW:ssä, ja vähentämään muun muassa HTTP protokollan otsaketiedon kuormitusta. Lisäksi W3C:n kehitteillä oleva "Server Sent Events (SSE)" -laajennos tuo mahdollisuuden määritellä push-tyyppisen tapahtuman lähteen osoitteen deklaratiivisesti HTML-koodissa. [16] [24]

Comet-periaatteen mukaisia HTTP-push-toteutuksia on olemassa erilaisia. Käytännössä erilaiset toteutukset voivat ottaa kantaa ainakin siihen, miten:

- yhteys alustetaan,
- asiakas rekisteröityy haluamiinsa viesteihin,
- yhteys pidetään aktiivisena ja
- mitä tiedonsiirtotekniikka käytetään.

Eräs yksinkertainen HTTP-push-toteutus perustuu niin sanottuun loputtomaan sivun lataukseen, jossa palvelin ei katkaise yhteyttä ensimmäisen HTTP-vastauksen päättyessä. Tämän mahdollistaa HTTP-protokollan chunked-tyyppinen sisällönkoodausasetus. Palvelin voi sen avulla pilkkoa sisällön paloihin, joita voidaan lähettää asynkronisesti pitkällä aikavälillä. Chunked-sisällönkoodaus lähettää ensin aina siirrettävän sisällön pituuden tavuissa ennen sisältöä. [43]

Push-tiedonsiirto voidaan toteuttaa loputtomalla sivun latauksen avulla esimerkiksi lisäämällä HTML-sivulle näkymätön `IFRAME`-elementti, johon lisätään ajoittain `SCRIPT`-lohkoja. Koska `IFRAME`-elementtiin ladattavaa sisältöä tulkitaan progressiivisesti, niin uudet `SCRIPT`-lohkot suoritetaan välittömästi niiden latauduttua. Palvelin jää alkuperäisen sivulataukseen jälkeen odotustilaan ja pitää yhteyden auki. Saadessaan asiakasta kiinnostavan tapahtuman, palvelin voi syöttää sivulle uuden `SCRIPT`-lohkon, joka päivittää sivun rakennetta halutulla tavalla. [43]

Sivun loputtomaan lataukseen perustuvalla tekniikalla on omat puutteensa. Sivun lataus voi keskeytyä verkkovirheen takia tai käyttäjän keskeyttäessä siirron vahingossa esimerkiksi escape-näppäimellä. Tämä ongelma on kuitenkin kierrettävissä, koska tiedonsiirto voidaan käynnistää uudelleen käyttäjän huomaamatta taustalla.

Myöhemmin esiteltävää XMPP-pikaviestintästandardia varten on syytä tutustua vielä BOSH-teknologiaan[4], joka on erityisesti HTTP:n kautta tapahtuviin XMPP-yhteyksiin tarkoitettu protokolla. XSF-säätiön[5] määrittelemä BOSH on yksi aikaisimmista vakiintuneista Comet-toteutuksista. Sen mukaisia järjestelmiä on ollut käytössä jo vuodesta 2004, eli ennen Comet-termin vakiintumista. [8]

BOSH perustuu pitkäkestoisiin HTTP-pyyntöihin (engl. long polling), joiden avulla matkitaan yhtäjaksoista sekä kaksisuuntaista socket-yhteyttä palvelimen ja asiakkaan välillä. Palvelin vastaanottaa pyyntöjä, mutta vastaa niihin vasta, kun asiakkaan viestijonossa on odottavia viestejä. Selainten ja välityspalvelimien ominaisuuksista johtuen joudutaan pyyntöihin vastaamaan joskus useammin kuin viestejä on tarjolla. Tällöin palvelin lähettää ennenaikaisesti tyhjän HTTP-vastauksen. Pisin vastausväliaika neuvotellaan BOSH-yhteyden alussa erillisen kättelyvaiheen aikana. Käyttelyn jälkeen samaa istuntoa voidaan jatkaa riippumatta pienistä yhteyden katkoista tai pakettien katoamisista. BOSH-protokolla pyrkii pitämään palvelimen ja asiakkaan välillä auki jatkuvasti yhden pitkäkestoisen HTTP-vastauksen, jonka kautta tietoa voidaan lähettää asiakkaalle lähes viiveettömästi. Tämän vuoksi vastauksen lähetyksen alkamisen huomattuaan asiakas alustaa välittömästi uuden HTTP-pyynnön. [8]

BOSH pyrkii tehokkaaseen tiedon kuljetukseen sekä korkeaan tiedon koherenssiin. Pull-tyyppiseen toistuvaan tiedon tarkkailuun nähden BOSH kuluttaa huomattavasti vähemmän verkon tiedonsiirtoresursseja ja tarjoaa lyhyemmän vasteajan. Socketeista ja puhtaista TCP/IP-yhteyksistä poiketen BOSH sietää myös katkonaisia yhteyksiä paremmin. Ideaali tilanteessa kun käytössä on HTTP/1.1 kaikki BOSH-viestiliikenne kulkee lisäksi vain yhden pitkäkestoisen TCP-yhteyden kautta. [30] [8]

[4]Bidirectional-streams Over Synchronous HTTP
[5]XMPP Standards Foundation `http://xmpp.org/about-xmpp/xsf/`

3. KESKUSTELUJÄRJESTELMÄN SUUNNITTELU JA TEKNOLOGIAVALINNAT

3.1 Sovelluksen toiminnot ja sovellusalueen käsitteet

Suunnittelun lähtökohtana oli toteuttaa asiakkaan tarpeisiin räätälöity selaimen välityksellä käytettävä viestintäalusta. Järjestelmän vaatimukset voidaan jakaa tavallisiin toiminnallisiin sekä epätoiminnallisiin vaatimuksiin. Seuraavaksi määritellään nämä vaatimukset yksityiskohtaisemmin.

Puhuttaessa internetin läpi tapahtuvasta reaaliaikaisesta viestinnästä käytetään usein tiettyjä vakiintuneita käsitteitä. Koska nämä termit esiintyvät jatkossa, esitellään ne hieman tarkemmin. Määritelmien pohjalla on XMPP-standardin mukaiset käsitteet, jotka perustuvat XMPP-protokollan termistöön [31]. Termistö on peräisin monista aiemmista viestintäjärjestelmistä, kuten IRC (*Internet Relay Chat*).

Keskustelutyökalun avulla tapahtuva viestintä jakautuu kahden käyttäjän välisiin yksityiskeskusteluihin, sekä useamman käyttäjän jakamiin ryhmäkeskusteluihin. Yksityiskeskusteluissa voidaan käyttää ominaisuutena niin sanottuja tilapäivityksiä (engl. *chat state notification*), jotka kertovat onko vastapuoli aktiivinen. Tämä tieto viestii esimerkiksi jos keskustelija on juuri kirjoittamassa viestiään tai että hän on ollut inaktiivinen määrätyn ajan.

Ryhmäkeskustelut (engl. *multi-user-chat*) tapahtuvat keskusteluhuoneissa, joihin käyttäjän on liityttävä. Liittyminen voi olla vapaata tai joskus siihen vaaditaan salasana. Käyttäjien tulee voida selata järjestelmässä olevia ryhmäkeskusteluja ja pääkäyttäjällä on oltava mahdollisuus luoda uusia keskusteluhuoneita. Luonnin yhteydessä määritellään huoneiden ominaisuudet, joita voidaan myöhemmin muuttaa.

Ryhmäkeskusteluissa on osaanottajia, joiden identiteetit jaetaan kaikkien huoneen osaanottajien kesken automaattisesti. Keskustelijat tunnistetaan huoneen sisällä yksilöllisen nimimerkin avulla (engl. nickname). Huoneesta poistumiset ja liittymiset ovat asynkronisia tapahtumia, joista välitetään tieto muille osaanottajille. Nämä viestit ovat erityistapaus niin sanotusta paikallaolotiedosta (presence notification).

Järjestelmässä on kolme käyttäjäroolia: anonyymit keskustelijat, keskustelun valvojat (moderaattorit) sekä pääkäyttäjät. Käyttäjätili on järjestelmän salasanalla suojattu identiteetti, johon voidaan sitoa keskusteluhuonekohtaisia oikeuksia. Oi-

keudet näkyvät ryhmäkeskusteluissa osaanottajan roolina, joka on joko moderaattori tai tavallinen keskustelija. Keskusteluihin kirjaudutaan oletusarvoisesti käyttäen anonyymiä käyttäjätiliä, jonka rooli on rajoitetuin.

Valvojat voivat lähettää keskusteluhuoneissa komentoja, joiden vaikutuksesta häiriköivä käyttäjä poistetaan keskustelusta (potkiminen) tai hänen viestien lähettäminen estetään määrätyksi ajaksi (hiljennys). Pääkäyttäjä voi muokata valvojien tietoja muuttamalla salasanoja ja lukitsemalla tilejä. Valvojilla on mahdollisuus tallentaa käyttäjätiliinsä sidottuja pikavastauksia, joita voi lisätä keskusteluviesteihin pikanäppäimillä.

3.2 Keskeiset ei-toiminnalliset vaatimukset

Ennen viestintäsovelluksen toteutusta täytyi tehdä suunnittelupäätöksiä, jotka vaikuttivat sovellukselle asetettujen asiakasvaatimusten täyttymiseen ratkaisevasti. Eniten painoarvoa valinnoissa on tässä tapauksessa annettu ei-toiminnallisille vaatimuksille. Suunnittelupäätöksiä ohjaavat huomattavasti vähemmän edellä kuvatut toiminnalliset vaatimukset, koska toiminnot ovat pääosin yksinkertaisia.

Seuraavaksi käydään läpi ei-toiminnallisiin vaatimuksiin liittyviä näkökohtia. Vaatimusten perusteella teknologioiksi valittiin avoin XMPP sekä rikkaiden internetsovellusten toteutukseen tarkoitettu Google Web Toolkit (GWT) [4], joiden toimintoihin perehdytään kohdissa 3.3 ja 3.4.

Tietoturva. Käyttäjän kannalta suurin tietoturvan uhkatekijä on viestien luottamuksellisuuden rikkoutuminen. Viestien eksyminen väärälle vastaanottajalle tulee estää tehokkaasti. Viestinvälityksen tulee olla suojattua käyttämällä salattua tiedonsiirtoa. Osapuolten tulee voida varmistua vastapuolen identiteetistä ja roolista luotettavasti. Toteutuksessa tulee kiinnittää huomiota syötteiden käsittelyyn ja oikeanlaisiin ohjelmointikäytäntöihin, jotta vältyttäisiin "cross site scripting" -tietoturva-aukoilta (XSS) [12]. Nämä hyökkäykset ovat ongelmallisia piirteitä kaikissa palveluissa, jotka jakavat käyttäjien luomaa sisältöä. Pahimmassa uhkakuvassa viestintäpalvelua käyttävä hyökkääjä voisi sopivasti muotoillun syötteen avulla vakoilla tai kontrolloida kaikkea viestintää osapuolten sitä huomaamatta.

Tietoturvavaatimusten tärkeydestä johtuen on perusteltua käyttää hyväksi havaittuja teollisuustandardeja, kuten XMPP. Uuden ja käyttötapausta varta vasten suunnitellun protokollan käyttäminen on järkevää vain, jos tarkastellaan pelkästään sovelluksen toiminnallisia vaatimuksia. Sen sijaan vuosien käyttökokemukset ja parannuskierrokset tietystä olemassaolevasta protokollasta, kuten XMPP, parantavat tietoturvaa olennaisesti. Lisäksi tällöin voidaan käyttää valmiita jo olemassaolevia protokollakirjastoja sekä palvelintoteutuksia, joiden

tietoturva on hioutunut ajan saatossa. XSS-tyyppisiltä haavoittuvuuksilta ei voida sen sijaan välttyä millään yleispätevällä ratkaisulla, mutta monet esimerkiksi GWT-kirjaston valmiit käyttöliittymäkomponentit edistävät oikeanlaisia ja turvallisia ratkaisumalleja.

Skaalautuvuus ja suorituskyky. Palvelun tulee olla käytettävissä suurella käyttäjäjoukolla ja arkkitehtuurin tulee skaalautua. Tämän vuoksi tulee välttää suorituskyvyn pullonkauloja. Tällaisia ovat ratkaisut joiden skaalautumiseen ei voida vaikuttaa esimerkiksi hajauttamalla palvelinalusta klusteriin.

Web-sovelluksen skaalautumista parantaa asiakaskeskeisen arkkitehtuurin valinta. Tämän vuoksi toteutuksessa päädyttiin käyttämään "thick-client"-mallia. Suurin osa sovelluslogiikasta pyrittiin toteuttamaan suoraan asiakassovelluksen logiikkaan. Palvelimen käyttötarkoitus on ainoastaan tarjota asiakassovelluksen koodi ja viestiliikenteen keskus. Ideaalisesti sovelluksen koodi on staattinen sisältö, joka voidaan jakaa replikoidun HTTP-palvelinklusterin avulla ja tallentaa asiakkaan selaimen välimuistiin mahdollisimman pitkäksi ajaksi.

Käytettävyys. Internet-pikaviestintään on perinteisesti käytetty erillisiä työpöytäsovelluksia, joten sovelluksen käyttöliittymälogiikan on hyvä matkia potentiaalisille käyttäjille tuttuja perinteisiä malleja. Tämän vuoksi tavallisten käyttöliittymän komponenttien, kuten välilehtien ja dialogien käyttäminen on suositeltavaa. Sovelluksen näkymiä tulee voida vaihtaa ilman erillistä sivusiirtymää ja toimintojen tulee olla asynkronisia, eli tapahtua taustalla lukitsematta käyttöliittymää. Koska sovelluksen käyttäjistä suurin osa on satunnaisia vieralijoita, sovelluksen käyttämisen ei tule vaatia erillistä asennusvaihetta tai selaimen lisäosan asentamista. Lisäksi sovelluksen lataamiseen kuluva aika tulee minimoida.

Ylläpidettävyys ja laajennettavuus. Sovelluksen käyttöikä on todennäköisesti useampia vuosia, joten suunniteltua järjestelmää tulee olla helppo ylläpitää ja jatkokehittää. Näitä tarpeita edistää suunnittelun modulaarisuus, laajasti tunnettujen ja hyvin dokumentoitujen standarden käyttö sekä avoimuus.

3.3 GWT-kehitysympäristö

3.3.1 Yleiskuvaus

Google julkaisi Google Web Toolkitin ensimmäisen avoimen yhteisöversion JavaOne-konferenssissa vuonna 2006. GWT:n kantava ajatus on siirtää monimutkaiseksi koettu rikkaiden internetsovellusten kehitys AJAX-tekniikalla yhtenäisen Java-kieleen

perustuvan alustan päälle. Tämä lähestymistapa tuo etuja, jotka näkyvät sekä sovelluksen kehitys-, ylläpito-, että käyttövaiheessa. [26]

Perinteisesti Web-kehitys on vaatinut kehittäjältä kokemusta sekalaisesta joukosta standardeja, kuten HTML, CSS sekä JavaScript. Nykyinen Web-ympäristö on seurausta standardien luontaisesta kehityksestä, jossa näkyy menneisyyden painolasti. Yhteensopivuuden vuoksi selaimet tulkitsevat standardeja vaihtelevasti; selainten DOM-toteutusten pienet erot mutkistavat sovelluskoodia. Selaimessa toimivien natiivien Web-sovellusten toteutuskieleksi on vakiintunut JavaScript. Dynaamisena kielenä JavaScript on joiltain ominaisuuksiltaan epävakaa alusta isompia sovelluksia rakennettaessa. Kielen dynaaminen tyyppijärjestelmä vaikeuttaa ongelmien löytämistä käännöksen aikana, koska ohjelmakoodi voi jopa muutta itseään suorituksen aikana. [29] [38]

GWT:n tarkoituksena on hämärtää web-alustan epäyhteensopivuuksia ja yksityiskohtia korkeamman tason kieliabstraktiolla ja alustaan rakennetulla komponenttiajattelulla. Java-pohjaisuus tehostaa toimiviksi koettujen suunnittelumallien käyttöä sekä tarjoaa vahvasti tyypitetyn kielen edut. Näistä syistä johtuen sovelluksen koodin rakenteesta tulee selkeämpi, ja se helpottaa sovelluksen myöhempää jatkokehittämistä ja ylläpitoa. GWT-alusta koostuu seuraavista komponenteista:

- Java — JavaScript-ristikääntäjä,
- Java SE -luokkakirjaston osittainen JavaScript-toteutus,
- GWT-luokkakirjasto ja standardit käyttöliittymäkomponentit,
- kehitystilan selainlaajennos. [26]

Sovelluskehitys GWT-ympäristössä tapahtuu tavallisessa Java IDE:ssä ja käytössä ovat kaikki ympäristön debug-ominaisuudet kuten jäljityspisteet (engl. breakpoints). Sovelluksen käyttöliittymä voidaan rakentaa Java Swing API:sta[1] tuttua käyttöliittymäohjelmointia muistuttavalla tavalla. Toteutuksessa voidaan käyttää tavallisia Java-standardkirjaston luokkia, kuten `java.util.HashMap`.

Sovelluksen loppukäyttäjän kannalta GWT:n etuja ovat sovelluksen nopeutuminen ja yhtenensopivuuden lisääntyminen. Ensinnäkin sovelluksen latausaika lyhenee, koska GWT pyrkii refaktoroimaan koodista ylimääräisiä rakenteita ja poistamaan koodin, jota ei suoriteta. Toiseksi käyttäjän selaimesta johtuvat ongelmat vähenevät, koska GWT pyrkii kapseloimaan erilaisista selainympäristöistä johtuvat piirteet ja ongelmat omien rajapintojensa taakse. [26]

[1] `http://download.oracle.com/javase/tutorial/uiswing/`

GWT:n käyttämä yhden alustan suunnitteluperiaate tarjoaa useita merkittäviä etuja kehitysvaiheessa, mutta toisaalta myös rajoittaa suunnittelijan vapauksia. Toisaalta yhtenäinen alusta mahdollistaa paremmat abstraktiot sekä matalan oppimiskäyrän, mutta vaatimusten noustessa voi kohdata web-standardien päälle rakennettujen abstraktioiden riittämättömyyden. GWT helpottaa web-suunnittelua lisäämällä suunnittelua ohjaavia rajoitteita, kuten poistamalla JavaScript-kielen monimutkaisen prototyyppipohjaisen perinnän ja duck-typing -ominaisuuden. [26]

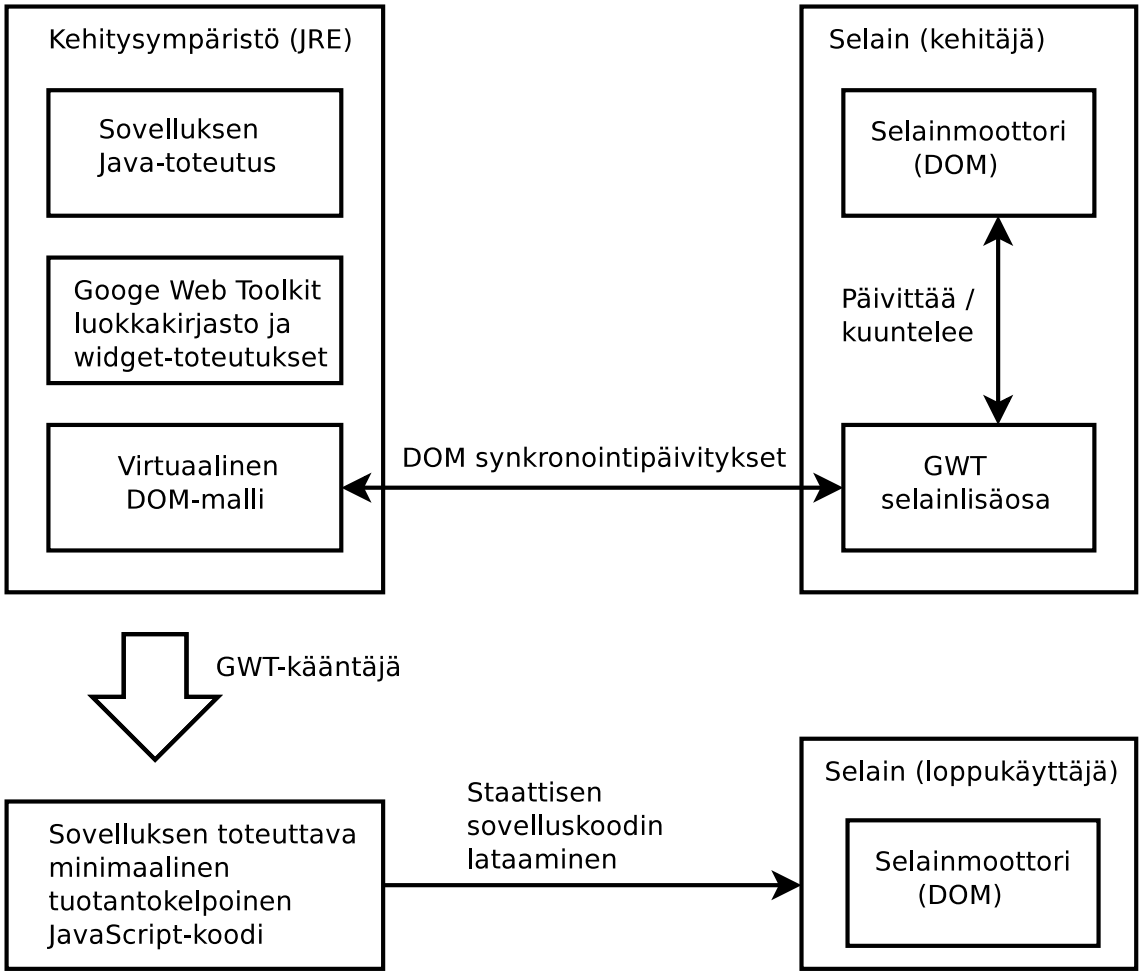
3.3.2 GWT-käännösprosessi ja kehitystila

GWT sovelluksen toteutus kirjoitetaan Java-kielellä, joten se on lopullista tuotantoversiota varten käännettävä selaimessa ajettavaksi JavaScript-koodiksi. Muunnoksen tekevä ristikääntäjä on syy teknologian moniin etuihin. Käännös mahdollistaa monimutkaisten koodin optimointien tekemisen sekä erilaisten selainalustojen ongelmiin ja eroavaisuuksiin varautumisen. GWT-ympäristö pyrkii peittämään web-ympäristön epäjohdonmukaisuudet ja GWT-kääntäjä auttaa kehittäjää keskittymään sovelluksen varsinaisiin toiminnallisiin vaatimuksiin.

Kuva 3.1 esittää sovelluksen kehittämistä ja käännösprosessin tarkoitusta. Tehokkaan ja tuottavan kehitystyön vuoksi valtaosa GWT-sovelluksen toteuttamisesta tapahtuu ilman käännösprosessia. Käännösprosessi on toteutusprosessin viimeinen vaihe, jonka avulla luodaan sovelluksen loppukäyttäjille tarkoitettu tuotantoversio. Sitä edeltävä kehitys tapahtuu kehitysympäristön selaimeen asennettavan GWT-lisäosan avulla.

Kehitystilassa koko sovelluksen logiikka toimii erillisessä Java-virtuaalikoneessa, joka suorittaa DOM-pohjaisia käyttöliittymätapahtumia simuloidun mallin avulla. Selaimen lisäosan tehtävänä on synkronoida konkreettisen selaimen DOM-mallin sisältö vastaamaan simuloidussa ympäristön tilaa sekä lähettää DOM-tapahtumat kehitysympäristöön. Ilman kehitystilaa Java-pohjainen ohjelmointi olisi kyseenalaista, koska GWT-kääntäjän suorittaminen vaatii runsaasti suoritinaikaa ja muistia. Kehitysympäristö eroaa lopullisesta kääntäjän tuottamasta sovelluksesta vain tuottamalla ylimääräistä viivettä toimintoihin.

GWT-kääntäjä suorittaa monivaiheisen käännösprosessin, jossa käsitellään alkuperäisiä Java-lähdekooditiedostoja. Ristikäännösprosessi eroaa tavanomaisesta käännöksesta, jossa ohjelmointikielen rakenteita muokataan alemman tason suoritusalustalle sopiviksi. Erityisesti lähdekielen, siis Javan ominaisuudet ovat monessa mielessä kohdekieltä, eli dynaamista JavaScriptiä suppeammat.



Kuva 3.1: GWT-sovelluksen kehitysprosessi kehittäjän ja loppukäyttäjän kannalta

Käännöksen aluksi kääntäjä kerää tiedot ohjelman GWT-moduuleista, eli ohjelmakoodista joka sisältää asiakaspään toiminnallisuutta. Käännöksen apuna on käytössä JavaScript-emuloitu versio suurimmasta osasta käytetyimpiä Java-standardikirjaston luokkia, jotka löytyvät pakkauksista "java.lang" sekä "java.util". Lisäksi voidaan yhdistellä koodia GWT:n omasta käyttöliittymäkirjastosta sekä ulkoisista GWT kirjastoista, jotka toimitetaan JAR-pakkauksina aina lähdekoodin kanssa.

3.3.3 Modulaarisuus ja luokkakirjastot

Sisäänrakennettu tapa jakaa GWT-ohjelma loogisiin moduuleihin on XML-kielellä luotavat moduuli-kuvaimet. Ne kuvavat moduulin tunnisteen, lähdekoodin ja toiset moduulit jotka ovat sille riippuvuuksia. Tämä mekanismi on karkean tason tapa, jolla voidaan esimerkiksi määritellä otetaanko käännökseen mukaan GWT:n sisäinen käyttöliittymäkirjastoa vai ei. Ohjelman sisäisen moduulijaon tekemiseen voidaan käyttää perinteisempiä "Depedency Injenction"-mallin mukaisia ratkaisuja. Tä-

tä varten on olemassa GWT-kirjastoja kuten Google Gin[2] tai Suco[3].

Tavallisista RIA-ohjelmistokehyksistä eroten GWT ei pakota käyttämään laajaa joukkoa valmiita käyttöliittymäkomponentteja tai suunnitteluratkaisuja. GWT:n oletuskäyttöliittymäkirjasto on laajudeltaan melko suppea, mutta riittää perustarpeisiin. Se tarjoaa Swing API:n kaltaisen komponenttihierarkian, jossa komponenteille voidaan rekisteröidä kuuntelijoita ja niistä voidaan rakentaa monimutkaisempia versioita komposition avulla. GWT:n ympärille on rakentunut useita riippumattomia käyttöliittymäkomponenttikirjastoja, jotka ovat osittain epäyhteensopivia tavallisten GWT:n käyttöliittymäkomponenttien kanssa. [26]

3.3.4 Kehittyneemmät sovelluksen optimointimekanismit

Paksun Web-sovelluksen käyttäminen vaatii sovelluksen koodin lataamisen ensimmäisellä käyttökerralla. Tämän vuoksi sovelluksen koodin koko on merkittävä käytettävyystekijä. Koodin määrään voidaan vaikuttaa käännösvaiheessa kahdella tavalla: käännöspermutaatioilla sekä koodin eliminoinnilla.

Koodin eliminoinnilla tarkoitetaan sovelluksen staattista analysointia, jossa selvitetään mitä luokkakirjaston metodeja oikeasti käytetään. Tavallisessa Java suoritusympäristössä luokan toteutus ladataan aina kokonaisuudessaan, vaikka osaa siitä ei koskaan käytettäisi. Määrittelemällä kuollut koodi voidaan supistaa asiakkaalle ladattavan koodin määrää huomattavasti. Tämäntyyppinen eliminointi on hankalaa tai mahdotonta tavallisilla JavaScript-kirjastoilla, koska se vaatii usein kehittäjältä syventymistä kirjaston sisäiseen toteutukseen. [26]

Permutaatiot koostuvat saman sovelluksen useammasta latausversiosta, jotka on optimoitu käyttäjän selainalustaa varten. Käynnistysvaiheessa version valintaan vaikuttaa esimerkiksi selaimen DOM-toteutus ja kieli. Permutaatioiden ansiosta ranskankielisen käyttäjän ei tarvitse ladata muita kieliversioita, ja Firefox-käyttäjän ei tarvitse ladata koodia, joka testaa ja ohittaa IE6-spesifisiä virheitä. Kehittäjälle erilaiset permutaatiot näkyvät tavalla, jolla esimerkiksi DOM-spesifistä koodia sisältävät luokat luodaan. Tämä tapahtuu eräänlaisella tehdassuunnittelumallin (engl. factory method) kaltaisella kielikonstruktiolla, jota kutsutaan viivästetyksi sitomiseksi (deferred binding): [26]

Listaus 3.1: Käännösaikainen toteutuksen viivästetty sitominen

```
DOM dom = GWT.create(DOMImpl.class);
// Tuottaa muun muassa seuraavat permutaatiot:
DOM dom = new DomImplIE6(); // IE 6,
DOM dom = new DomImplSafari(); // Safari ja
DOM dom = new DomImplGecko(); // Firefox.
```

[2] http://code.google.com/p/google-gin/
[3] http://code.google.com/p/suco/

Käännösvaiheessa sovelluksen kokoon voidaan vaikuttaa lisäksi pilkkomalla ohjelman latauspaketti. Monesti sovelluksessa on toimintoja, joiden käyttäminen on harvinaista tai riippuu käyttäjän oikeuksista. GWT tarjoaa tavan karsia valinnaiset ominaisuudet erillisiin latauspaketteihin, joka lyhentää sovelluksen lataamis- ja käynnistysviivettä. Koodin pilkkominen perustuu irrallisten toimintojen käärintään seuraavanlaiseen rakenteeseen:

Listaus 3.2: Sovelluksen toimintojen pilkkominen

```
GWT.runAsync(new RunAsyncCallback() {
  public void onFailure(Throwable e) {
    /* Ominaisuuden viivästetty lataus ei onnistunut */
    Window.alert("Virhe 206");
  }

  public void onSuccess() {
    /* Irrallisen ominaisuuden "Feature"-koodi on nyt käytettävissä */
    Feature.doWork();
  }
});
```

Web-sovelluksen tehokkuuteen vaikuttaa palvelimelta ladattavien yksittäisten resurssien lukumäärä. Kuvien tapauksessa voidaan muodostaa kuvat yhdistelemällä suuri bittikartta, jonka sisältö puretaan käyttöliittymän tarpeisiin vasta asiakassovelluksessa. GWT osaa tehdä yhdistelyn automaattisesti käännösvaiheessa staattisen analyysin perusteella. Analyysi perustuu Java-annotaatioilla varustettuihin resurssirajapintoihin (client bundle interface). Kehittäjän kannalta mekanismi on yksinkertainen, koska resursseja voidaan käyttää yksinkertaisesti resurssiluokan Java-rajapintakutsuilla. GWT-kääntäjä lisää tarvittavat toteutusyksityiskohdat edellä esitetyn viivästetyn sidonnan avulla. [26]

3.4 XMPP-protokollaperhe

XMPP-standardi (Extensible Messaging and Presence Protocol) on tulosta IETF-standardointiorganisaation työstä, joka tähtää luomaan yleiskäyttöisen viestintäprotokollan. Protokolla on suuntautunut pikaviestinnän kaltaisiin käyttötapauksiin, mutta se on silti hyvin monikäyttöinen reaaliaikaista viestintää tukeva työkalu. XMPP on seurausta vuonna 1998 alkaneesta Jabber-protokollan kehityksestä. Avoimen standardin luonteesta johtuen, protokolla on saavuttanut vankan aseman käytettynä sekä luotettuna teknologiana. Käytössä on kymmeniä tuhansia standardinmukaisia palvelimia, joilla on varovaisesti arvioiden miljoonia käyttäjää. Protokolla on laajasti käytössä monentyyppisissä reaaliaikaviestintää vaativissa sovelluksissa,

joista tunnettuna voidaan mainita "Google Talk"[4]. Myös Facebook on julkaissut ulkoiseen käyttöön XMPP-rajapinnan. [31] [34]

XMPP-protokollaperhe pohjautuu ytimen ympärille [9], joka määrittelee kehyksen XML-kielen avulla tapahtuvalle tiedon välitykselle, jota kutsutaan "XML-streaming" -tekniikaksi. Ydin on määritelty dokumenteissa RFC 3920 ja RFC 3921. Kehyksen suunnittelutarkoituksena on ollut helpottaa pikaviestintään ja läsnäolotiedon välitykseen perustuvien sovellusten toteuttamista. Tätä varten se tarjoaa sovellusten perusarkkitehtuurin sekä joukon protokollan kieliopillisia rakenteita ja niiden karkean semantiikan, jota käsitellään tarkemmin viestirakenteiden yhteydessä. [31]

XML-kielen vapaamuotoisuudesta johtuen, protokolla tarjoaa moninipuolisia laajennusmahdollisuuksia. Räätälöidyn protokolla-laajennoksen tekeminen on hyvin vaivatonta, koska perusarkkitehtuuri on hyvin salliva. Palvelin toteutuksen ei monessakaan tilanteessa tarvitse tukea laajennosta mitenkään erityisesti, jos laajennos voidaan toteuttaa protokolla-pohjaisen liitännäisen avulla.

XMPP Standards Foundation-organisaation työn tarkoituksena on valvoa sekä ohjata XMPP-protokollan laajennosten kehitystä. Monet käytetyimmistä laajennoksista ovat saavuttaneet lopullisen-standardin aseman. IETF seuraa luomiensa standardien levinneisyyttä ja pyrkii vakiinnuttamaan vedos-vaiheessa olevien laajennosten ominaisuudet.

3.4.1 Perusarkkitehtuuri

XMPP verkossa on perinteinen asiakas-palvelinpohjainen arkkitehtuuri, mutta usein myös eri domain-osoitteissa sijaitsevat palvelinkomponentit keskustelevat toisilleen. Asiakkaiden viestinvälitys tapahtuu oletuksena TCP-protokollan avulla portissa 5222. TCP-protokollan ominaisuuksista johtuen XMPP:n etuna esimerkiksi kilpailevaan UDP-pohjaiseen SIP/SIMPLE-protokollaan on tuki kadonneiden viestin huomaamiselle sekä ruuhkautumisenestolle jo verkkokerroksella. [31]

Palvelinkeskeisestä arkkitehtuurista johtuen on mahdollista toteuttaa viestin tarkka auditointi, koska suoraa käyttäjien välistä yhteyttä ei pääse syntymään. Tämä ominaisuus myös suojelee käyttäjän identiteettiä.

Viestin lähettäminen toiselle osapuolelle XMPP-verkossa tapahtuu keskustelemalla paikallisen XMPP-palvelimen kanssa. Käyttäjällä on yleensä rekisteröity käyttäjätili paikallisen palvelimen toimialueessa (domain). Palvelin voi yhdistää toiselle palvelimelle viestin lähettämiseksi, jos vastapuolen toimialue ei ole paikallinen. Verkossa voi olla myös erityisiä siltoja (engl. federation gateways), joiden avulla viestejä voidaan vaihtaa myös yhteensopimattomiin verkkoihin kuten IRC[6] tai Microsoft Live Messenger[5]. Tämän ominaisuuden avulla asiakaskomponettiin ei tarvitse tehdä

[4] `http://www.google.com/talk/`

[5] `http://explore.live.com/windows-live-messenger`

muutoksia, jos myöhemmin halutaan laajentaa sovellusta ulkoisiin verkkoihin.

Osoitteistus

Protokolla määrittelee keskustelunosapuoliksi oliot, joilla on JID-tunniste. Se koostuu kolmesta osasta: käyttäjä, node ja resurssi. Kolmesta osasta koostuva tunniste on aina globaalisti uniikki, mutta usein viestin reititykseen riittää resurssin pois jättäminen, eli niin sanottu paljas JID. Resurssi on vaihtuva istuntoon liittyvä tunniste, joka sidotaan esimerkiksi asiakasohjelman yhdistäessä palvelimeen. Uniikin JID-osoitteistuksen avulla voidaan siis määritellä identiteetti useille saman käyttäjän yhtäaikaisille istunnoille:

- mikko@esimerkki.org/resurssi1
- mikko@esimerkki.org/resurssi2

Palvelin voidaan asettaa sallimaan myös niin sanotut anonyymit käyttäjät, joiden käyttäjänimi arvotaan satunnaiseksi yhdistäessä palvelimeen. Toisaalta voidaan antaa identiteetti keskusteluhuoneelle sekä keskustelun osaanottajille nimimerkkien avulla.

- mikko@keskusteluhuone.esimerkki.org/nimimerkki1
- esko@keskusteluhuone.esimerkki.org/nimimerkki2

XMPP ei tuet nimimerkkien käyttöä anonyymien käyttäjien yksilöivänä tunnisteena, vaikka keskustelupalveluissa tämä on yleistä. Sen sijaan ryhmäkeskustelulaajennos vaatii kanavakohtaiset yksilölliset nimimerkit.

XML viestinvälitysrakenteet

XMPP-protokollan ytimessä ovat XML-tietovirrat. Palvelimen ja asiakkaan välinen viestintä alkaa istunnon neuvottelulla, jonka jälkeen kumpaankin suuntaan lähetetään XML-dokumentin palasia pitkäkestoisen yhteyden aikana. Yhteys eroaa täysin tavanomaisesta HTTP tai sähköpostiliikenteestä, jossa saman yhteyden aikana suoritetaan korkeintaan yksi transaktio, koska viestejä voidaan lähettää rajaton määrä ilman vastauksen odottamista. Päällekkäiset pyynnöt tai vastaukset tulkitaan viestijonojen avulla asynkronisesti. Tämän vuoksi XMPP-käyttävät sovellukset ovat vahvasti viestiohjattuja (engl. event-driven architecture). [31]

XML-tietovirta koostuu kolmenlaisesta perusviestistä (XML stanza) eli XML-elementistä: `<message/>`, `<presence/>` ja `<iq/>`. Viesti sisältää vapaamuotoisia XML-elementtejä, joiden tulkinta riippuu niiden XML-nimiavaruudesta. Lisäksi perusviestillä on tyyppiattribuutti, joka vaikuttaa viestin käsittelyyn. Seuraavaksi esitellään viestien peruskäyttötapaukset. [31]

Message. Viestielementti on yksinkertainen tapa lähettää tietoa kahden osoitteellisen XMPP-olion välillä. Viestityypin käyttämiseen ei liity vuoropuhelua tai kuittausta. Elementtiä käytetään yksityis- ja ryhmäkeskusteluiden yhteydessä, jotka ilmaistaan tyyppiattribuutin arvolla `chat` tai `groupchat`.

Presence. Pikaviestinnässä ollaan usein kiinostuneita vastapuolen läsnäolosta, joka voidaan ilmoittaa toisille tämän viestityypin avulla. Läsnäolotieto on palvelimen tarjoama ominaisuus, jolla kaikkia henkilön tilasta kiinnostuneita tiedotetaan koko XMPP verkon laajudella automaattisesti. Tilatietoon voi liittyä vapaavalintaista tietoa kuten poissaoloviesti. Ryhmäkeskusteluissa viestiä käytetään ilmoittamaan huoneeseen liittymiset ja poistumiset.

IQ (Info/Query). IQ-viestityyppiä käytetään pyyntö-vastaus -tyyppisen transaktion toteuttamiseen, johon liittyy yksilöivä tunniste ja vastausviesti. Viestillä voidaan esimerkiksi pyytää lisätietoa toisesta oliosta ja tallentaa palvelimen asetuksia.

3.4.2 Sovelluksen kannalta oleelliset XMPP-laajennokset

XMPP Standards Foundation (XSF) julkaisee ja ylläpitää laajennosten standardiehdotelmia. Näitä kutsutaan XMPP-laajennosprotokolliksi ja merkitään alkuliitteellä XEP. Laajennosten sisältö on käynyt läpi pitkän suunnittelu ja arviointiprosessin ja niiden sisältämät ongelmakohdat ovat hioutuneet pois käytössä ennen lopulliseen tilaan pääsyä.

XMPP-palvelin toteutukset tukevat yleensä vain pientä joukkoa kaikista olemassa olevista laajennoksista. Laajennokset toteutetaan yleensä palvelimeen liitetyn palvelu-komponentin avulla, jolla on oma JID-osoite. Komponentti voi olla ulkopuolinen verkon avulla yhdistyvä palvelu tai se voidaan asentaa palvelimeen plugin-arkkitehtuurin avulla. Keskustelujärjestelmän toteuttaminen on mahdollista lähes yksinomaan valmiiden laajennosten avulla, joista tärkeimmät esitellään seuraavaksi.

Lomakkeet (XEP-0135 Data-forms)

Data-lomake laajennos (XEP-0004) tarjoaa kevyen mutta monipuolisen tavan siirtää rakenteista tietoa XMPP-välityksellä. Laajennos integroituu osaksi XMPP:n XML-viestirakennetta yksinkertaisen syntaksin ja oman nimiavaruuden `jabber:x:data` avulla. Johtuen lomakkeiden yleiskäyttöisyydestä, niitä käytetään laajasti osana muita laajennoksia. [31]

Listaus 3.3: Lomakelaajennos

```
<message from="mikko@ag.fi/foo" to="huone@muc.mantelichat.fi">
  <x xmlns="jabber:x:data" type="form">
    <title>Lomakkeen otsikko</title>
    <instructions>Muista täyttää tarvittavat kentät</instructions>
    <field type="hidden" var="FORM_TYPE">
      urn:ag:esimerkki
    </field>
    <field label="Teksti kenttä" type="text-single" var="kentta-1">
      <required />
    </field>
    <field label="Valinta kenttä" type="boolean" var="kentta-2" />
    <field label="Monivalinta kenttä" type="list-single"
        var="kentta-3">
      <option label="Valinta A"><value>value1</value></option>
      <option label="Valinta B"><value>value2</value></option>
    </field>
  </x>
</message>
```

Listauksessa 3.3 on esitetty lomakkeiden yksinkertaista rakennetta mielivaltaisen esimerkin avulla. XMPP lomakkeet ovat läheistä sukua HTML-kielen rakenteelle `FORM`, joka käy ilmi tarjolla olevista lomakkeen kenttien tyypeistä. Kaikki kenttätyypin vaihtoehdot eivät ole näkyvissä. Lomakkeiden toteutuksessa on panostettu niiden yleiskäyttöisyyteen ja helppoon niihin perustuvien käyttöliittymänäkymien luontiin. Tämän vuoksi mukana on HTML:stä poiketen ylimääräistä metatietoa, kuten ohjeistukset ja kenttien pakollisuusattribuutit. [31]

Sovelluksen kannalta lomakkeita tarvitaan erityisesti ryhmäkeskustelulaajennoksen yhteydessä. Huoneasetusten muokkaaminen perustuu vakiomuotoiseen konfigurointidataan, jonka siirrossa lukemista ja tallentamista varten käytetään lomakkeita. Yleisimmät käytössä olevat lomakkeiden kenttien nimet on vakioitu, ja lisäksi standardimuotoisen tiedon tyypin kuvaamiseen käytetään piilotettua `FORM_TYPE` kenttää. [31]

Ryhmäkeskusteluhuoneet (XEP-0045 Multi-User Chat)

Ryhmäkeskustelulaajennos (MUC) määrittelee keskusteluhuoneiden käyttöön tarkoitetut viestirakenteet ja nimiavaruuden. Laajennos mahdollista sovelluksen vaatimusmäärittelyyn kuuluvat keskusteluhuoneominaisuudet, sekä monia toivottuja lisäominaisuuksia. [31]

Huoneeseen liittyminen ja poistuminen tapahtuu `<presence/>`-viestityypin avulla, joka välittää läsnäolotiedon automaattisesti muille osaanottajille. Viestin avulla ilmoitetaan haluttu nimimerkki, jonka palvelin vahvistaa yksilölliseksi. Aiempia

huoneen keskusteluita voidaan lähettää liittyneelle viivästettyjen aikaleimojen avulla (engl. delayed delivery). [31]

Listaus 3.4: Ryhmäkeskustelulaajennos

```
<presence from="mikko@example.org" to="huone@muc.example.org/mikko" />

<presence from="huone@muc.example.org/mikko" to="matti@example.org" />
<presence from="huone@muc.example.org/mikko" to="jarno@example.org" />

<message from="huone@muc.example.org/matti" to="mikko@example.org"
    type="groupchat">
  <body>Moimoi</body>
  <delay xmlns="urn:xmpp:delay" stamp="2011-01-01T13:24:03Z"/>
</message>

<message from="mikko@example.org" to="huone@muc.example.org"
    type="groupchat">
  <body>Heippa vaan</body>
</message>
```

Listauksessa 3.4 palvelin saa uudelta keskustelijalta `mikko@example.org` pyynnön liittyä ryhmäkeskusteluun `huone@muc.example.org` nimimerkillä "mikko". Palvelin luo kaksi läsnäoloviestiä, jotka lähetetään vanhoille keskustelijoille "matti" ja "jarno". Lisäksi palvelin lähettää yhden viivästetyn keskusteliviestin, johon uusi keskustelija vastaa. Viestistä luotavia kopioita eri osallistujille ei ole esitetty.

Perustoimintojen lisäksi huoneissa on käyttäjän rooliin ja kytkökseen perustuvia toimintoja. MUC-laajennos määrittelee roolin väliaikaiseksi osaanottajan tilaksi, ja kytköksen keskustelijan pysyväksi ominaisuuksi. Ominaisuuksien arvo voi olla esimerkiksi jäsen, moderaattori tai huoneen omistaja. [31]

Listaus 3.5: Osaanottajan potkiminen ryhmäkeskustelusta

```
<iq from="moderator@example.org" id="tx-123"
    to="huone@muc.example.org" type="set">
  <query xmlns="http://jabber.org/protocol/muc#admin">
    <item nick="Rölli" role="none" />
  </query>
</iq>
```

Listauksessa 3.5 näkyy esimerkki osaanottajan potkimisesta, jonka voi suorittaa vain moderaattori. Potkiminen tapahtuu asettamalla IQ-viestin avulla osallistujan rooli arvoon "none". Palvelin tarkistaa käyttäjän oikeudet ja vastaa IQ-viestiin virheellä, tai tiedottaa muita osallistujia potkitun osallistujen muuttuneesta läsnäolotiedosta presence-viestien avulla.

Huoneen luominen tapahtuu lomakelaajennoksen ja IQ-viestityypin avulla. Luonnin yhteydessä voidaan määritellä ominaisuuksia, kuten suurin yhtäaikainen jäsenten määrä, liittymiseen tarvittava salasana ja huoneen kuvausteksti. Huoneen omistaja pyytää get-tyyppisen pyynnön avulla luotavan huoneen asetuslomakkeen, ja huone avautuu muille osaanottajille, kun asetukset on vahvistettu set-pyynnöllä. Palvelin vahvistaa luonnin result-tyyppisellä viestillä. [31]

Palvelin voi tukea myös ylimääräisiä huoneasetuksia sekä oletusasetuksia, jotka lähetetään vastauksena luontipyyntöön. Oletuksena voi olla esimerkiksi, että huone tuhoutuu kaikkien läsnäolioiden poistuttua, tai että palvelin arkistoi kaikki ryhmäkeskustelut. [31]

Palveluiden selaus (XEP-0030 Service Discovery)

XMPP-verkko koostuu joukosta olioita, joiden olemassaolosta ja ominaisuuksista saadaan tietoa erityisen laajennoksen avulla. Tietoja voidaan pyytää myös siitä, mitä laajennoksia palvelin tai yksittäinen asiakas tukee. Palveluiden selaamista tarvitaan erityisesti, kun halutaan tietoon järjestelmässä olevat ryhmäkeskusteluhuoneet, sekä niiden ominaisuudet.

Listaus 3.6: Palveluiden selaaminen

```
<iq from="mikko@example.org" to="rooms@muc.example.org" id="tx-253"
    type="get">
  <query xmlns="http://jabber.org/protocol/disco#items" />
</iq>
<iq from="rooms@muc.example.org" id="tx-253" type="result">
  <item jid="huone1@muc.example.org" />
  <item jid="huone2@muc.example.org" />
</iq>
```

Selaaminen tapahtuu IQ-viestityypin avulla, johon vastauksena saadan lista tiettyyn olioon kytkeytyneitä olioita. Listauksessa 3.6 pyydetään MUC-palvelulta lista julkisista huoneista. Huoneiden nimi, osaanottajien määrä sekä selitysteksti voidaan hakea kysymällä olion tietoja, joka näkyy listauksessa 3.7. [31]

Listaus 3.7: Palvelun tietojen kysely

```
<iq from="mikko@example.org" to="huone1@muc.example.org" id="tx-254"
    type="get">
  <query xmlns="http://jabber.org/protocol/disco#info" />
</iq>
<iq from="huone1@muc.example.org" id="tx-254" type="result">
  <identity category="muc" type="text" name="Yleinen keskustehuone" />
  <feature var="http://jabber.org/protocol/muc" />
  <feature var="muc_public" />
  <x xmlns="jabber:x:data" type="result">
```

```
    <field type="hidden" var="FORM_TYPE">
      <value>http://jabber.org/protocol/muc#roominfo</value>
    </field>
    <field label="Number of occupants" var="muc#roominfo_occupants">
      <value>14</value>
    </field>
  </x>
</iq>
```

Käyttäjän henkilökohtainen säilyvyys (XEP-0049 Private XML Storage)

Paikkariippumattomissa Web-sovelluksissa on usein tarve säilöä keskistetysti käyttäjätiliin sidottua tietoa. Keskustelujärjestelmässä käyttäjillä on esimerkiksi henkilökohtaisia pikavastauksia, jotka säilytetään palvelimella.

Säilyvyyslaajennos mahdollistaa yksinkertaisen XML-dokumentin tallentamisen palvelimelle. Dokumentti voi sisältää mielivaltaista tietoa, ja sen sisältöä voidaan muokata IQ-viestityypin avula. Pyynnöt kohdistetaan oikeaan dokumenttiin XML-nimiavaruuden avulla. [31]

Listaus 3.8: Tiedon säilyttäminen XMPP-palvelimella

```
<iq type="get" id="tx-423">
  <query xmlns="jabber:iq:private">
    <replies xmlns="http://mantelichat.fi/replies" />
  </query>
</iq>
<iq type="result" id="tx-423" from="mikko@example.org"
    to="mikko@example.org">
  <query xmlns="jabber:iq:private">
    <replies xmlns="http://mantelichat.fi/replies" />
      <reply ndx="0" value="MOI" />
      <reply ndx="1" value="moikka" />
    </replies>
  </query>
</iq>
```

Listauksessa 3.8 käyttäjä pyytää listaa palvelimelle tallennettuja vastauksia lähettämällä tyhjän nimiavaruudella varustetun XML-elementin. Palvelin vastaa lähettämällä nimiavaruuteen kytketyn dokumentin sisällön.

4. VIESTINTÄSOVELLUKSEN TOTEUTUS

4.1 Järjestelmän rakenne

4.1.1 Yleiskuvaus ja käyttöliittymä

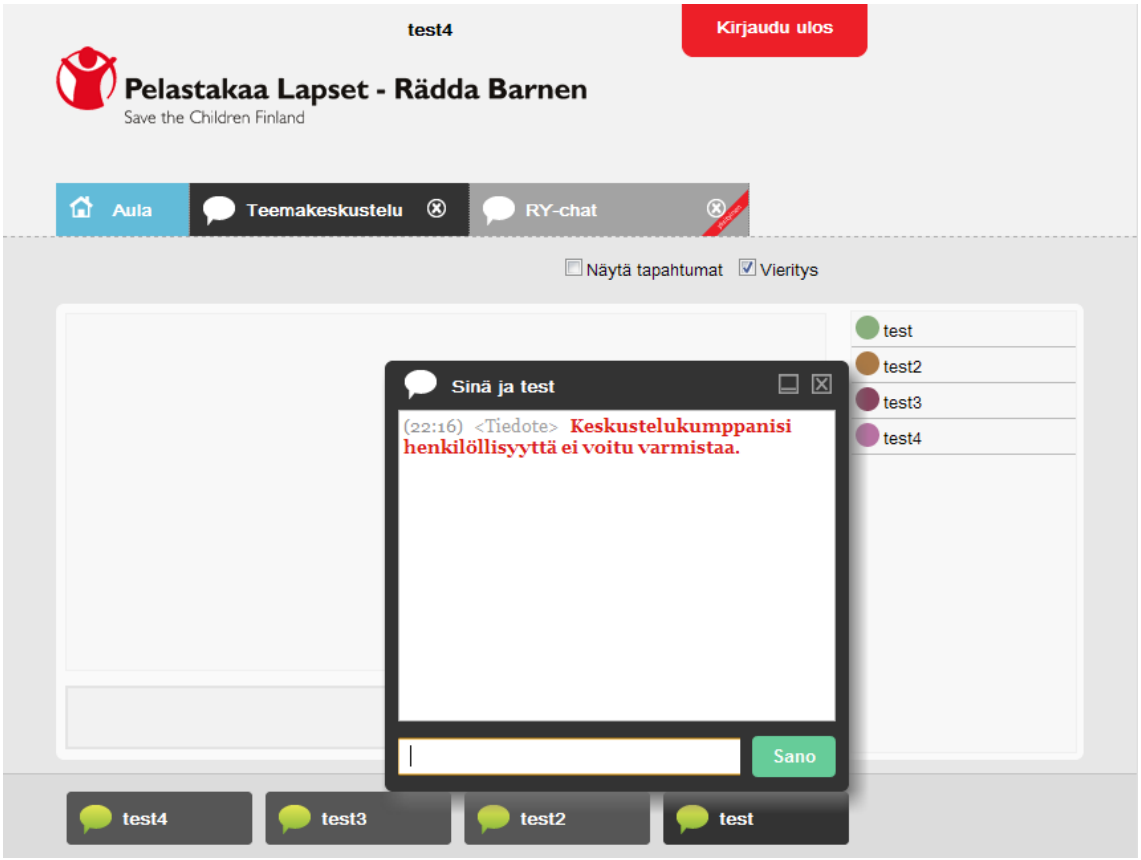
Valmis viestintäsovellus koostuu viestipalvelimesta ja asiakassovelluksesta. Suunnittelu ja toteutus on painottunut asiakassovelluksen kehittämiseen. Asiakassovelluksen pääasiallinen tehtävä on kommunikoida valmiin palvelintoteutuksen kanssa XMPP-protokollan avulla. Monipuolisesta ja itsenäisestä asiakassovelluksesta johtuen järjestelmä perustuu paksuun arkkitehtuuriin. Sovelluksen esityskerros ja logiikka toimivat olennaisilta osiltaan tavallisessa selaimessa ilman liitännäisiä. Asiakassovelluksen arkkitehtuuriin sekä nykyaikaiseen oliopohjaiseen toteutukseen GWT:n avulla syvennytään kohdassa 4.2.

Kuvassa 4.1 on esillä sovelluksen anonyymille keskustelijalle tyypillinen käyttötilanne. Keskustelijalla on meneillään useita yksityiskeskusteluja, jotka näkyvät sovelluksen alapalkissa. Taustalla näkyy aktiivisena oleva ryhmäkeskustelu ja sen osanottajat on listattu oikeaan sivupalkkiin. Nimimerkkiä napsauttamalla saa esiin lisää toimintoja, mikäli käyttäjän rooli on valvoja. Aula-välilehden kautta voi liittyä keskusteluihin ja pääkäyttäjän tapauksessa luoda uusia.

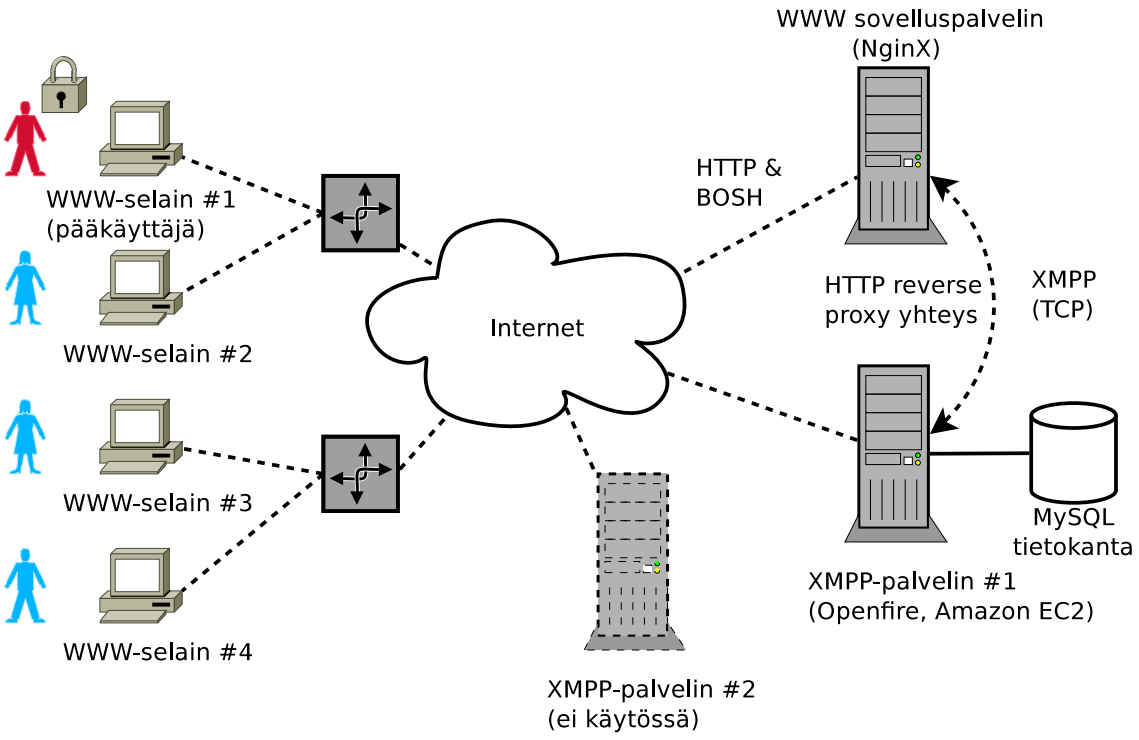
4.1.2 Arkkitehtuuri ja käytetyt valmisohjelmistot

Kuvassa 4.2 on esitetty viestintäjärjestelmän komponentit ja niiden kytkökset. Asiakkaina toimivat eri valmistajien selainohjelmat, joissa tässä työssä suunniteltu ja toteutettu asiakassovellus toimii. Selaimet viestivät Internetin kautta kaiken liikenteensä edustapalvelimelle. Edustapalvelimia voi olla useampia, ja ne jakavat selainohjelmiston koodin sekä toimivat viestinvälittäjinä. Edustapalvelinten replikointi on mahdollista täysin symmetrisesti, eli niiden ei tarvitse olla tietoisia toisistaan.

Edustapalvelimet toimivat XMPP-viestiliikenteen yhdyskäytävinä, eli ne reitittävät BOSH-tyyppiset XMPP-yhteydet viestipalvelimelle. Viestipalvelimia on mahdollista olla useampia, mutta toistaiseksi useammille viestipalvelimille ei ole käyttöä. Palvelimet voivat edustaa joko erillisten palveluntarjoajien ylläpitämiä palveluita tai yksittäinen palvelin voidaan kuormantasauksen tai korkean saatavuuden vuoksi klusteroida. Nykyisessä toteutuksessa viestipalvelimen ohjelmistona toimii Openfi-



Kuva 4.1: Sovelluksen perusnäkymä, jossa yksityiskeskustelu valittuna.



Kuva 4.2: Asennuskaavio

re[1], joka tukee natiivisti BOSH-tyyppisiä asiakkaita. Palvelintoteutuksen valinta ei ole merkityksellinen, mutta Openfire tarjoaa sovelluksen muulle toteutukselle yhtenäisen tuen Java-pohjaisille liitännäisille. Tarvittaessa palvelinohjelmisto voidaan vaihtaa useampaan ei-kaupalliseen standardienmukaiseen tuotteeseen.

Taustajärjestelmä koostuu: NginX[2] WWW-sisältöpalvelimesta ja MySQL[3]-tietokantapalvelimesta. MySQL-tietokanta säilöö XMPP-palvelimen konfiguroinnin ja tilatiedon lisäksi logidataa ja viestintätunnisteita auditointia varten. WWW-palvelimen valinnan ratkaisi NginX-palvelimen erinomaisuus staattisen sisällön jakamisessa [5] ja sen tarjoama mahdollisuus lähettää asiakkaan viestiliikenne lähetetään edelleen Openfire-palvelimen BOSH-liityntäporttiin.

Edustapalvelimen proxyn käyttäminen on välttämätöntä selainten käyttämästä "same origin policy" -tietoturvaominaisuudesta johtuen [7]. Se estää eri toimialueiden välillä tapahtuvan liikenteen osittain sivun JavaScript-koodissa. Etäkutsujen tekeminen on käytännössä mahdollista, mutta niiden vastauksen lukeminen ei ole mahdollista ilman toisessa toimialueessa sijaitsevan palvelimen apua. Tästä rajoituksesta johtuen kaikki sovelluksen pyynnöt täytyy lähettää samalle kohdepalvelimelle, josta myös Web-asiakas on ladattu, koska käytetty XMPP-kirjasto tukee vain tavallisia AJAX-kutsuja.

On mahdollista että myöhemmin halutaan käytännön syistä sijoittaa sisältöpalvelin erilleen toiseen toimialueseen. Tätä tarkoitusta varten on varauduttu mahdollisuuteen kehittää Adobe Flash-teknologiaan perustuva yhdyskäytävä, jolla tavalliset selaimen AJAX-pyynnöt korvataan. Flash sallii domainien välisen viestinnän pääsylistoihin pohjautuvan konfiguraation avulla. Tulevaisuudessa W3C:n kehittämän CORS-standardin valmistuminen ja yleistyminen mahdollistaa myös samankaltaisen mutta natiivin HTML-pohjaisen toteutuksen [39].

4.1.3 Moderointiominaisuuden toteuttaminen

Julkisessa käytössä olevien palveluiden ikuisena ongelmana ovat satunnaiset häiriköt. Häirikkö voi haitata palvelun käyttöä monella tavalla. Yleisin tapa on lähettää asiatonta sisältöä ryhmäkeskusteluun tai tukkia keskustelu viestien tulvalla. Osaanottajien anonyymiys aiheuttaa häiriköiden poistamiselle ongelmia. Yksinkertainen keskustelijan potkiminen ryhmäkeskustelusta ei estä häirikköä liittymästä uudelleen. Lisäksi porttikiellon asettaminen on vaikeata, koska keskustelijan identiteettiä ei voida lähtökohtaisesti varmentemaan luotettavasti.

Keskustelupalvelun moderointimenetelmäksi valittiin yksinkertainen keskustelijan hiljentäminen tai keskustelusta potkiminen. Häirikön identiteetin tunnistami-

[1] `http://www.igniterealtime.org/projects/openfire/`
[2] `http://nginx.org/en/`
[3] `http://www.mysql.com/`

seen käytetään selaimen kekseihin (engl. cookie) tallentuvaa tunnistetta. Sen sisältämästä tiedosta voidaan päätellä, onko keskustelijan hiljennys voimassa tietyssä keskusteluryhmässä ja koska se vanhenee.

Selaimen kekseihin perustuva moderointi on kompromissi vahvemman käyttäjän IP-osoitteeseen perustuvan rajauksen ja helposti kierrettävän keskustelusta potkimisen välillä. IP-osoitepohjaista estämistä haluttiin välttää, koska se vaikuttaa kokonaisiin aliverkkoihin, jotka on voitu piilottaa yleisesti käytetään NAT-tekniikan[37] avulla yhden julkisen IP-osoitteen taakse. Toiseksi IP-osoitepohjainen moderointi vaatisi enemmän toteutusta palvelinpuolella. Keksipohjaisen moderoinnin suurin ongelma on sen helpohko kiertäminen tyhjentämällä selaimen keksit.

Pelkästään asiakkaspäässä toimivaa moderointia on myös mahdollista jatkokehittää, jotta sen kiertäminen olisi vaikeampaa. Selaimeen kätkettävän tiedon menetelmät eivät rajoitu vain yleisesti tunnettuihin HTTP-kekseihin. Muita menetelmiä ovat esimerkiksi Flash-liitännäisen tarjoama säilyvyysrajapinta [36] sekä uusi HTML5-pohjainen säilyvyys [42]. Tiedon paikallista tallentamista varten on kehitetty JavaScript-kirjasto[4], joka huolehtii että tietystä datasta on säilössä kopio käyttäen saatavilla olevia tekniikoita. Tällöin tavallisten keksien tuhoaminen voidaan havaita ja alkuperäinen data palauttaa muusta säilöstä.

Toteutuksen kannalta moderointia varten tarvitaan lisäksi viestintäprotokolla, jonka avulla keskustelua moderoiva henkilö voi aktivoida hiljennystoiminnon. Viestin välitykseen ei tarvitse käyttää XMPP-protokollassa tavallista in-band -yksityisviestin sisältökenttää, koska viestirakenne on laajennettavissa XML-nimiavaruuksien avulla. Käytännössä siis voidaan luoda uusi rakenne-elementti, joka on sidottu palvelun käyttämään yksityiseen nimiavaruuteen. Nimiavaruuden tunnistenimenä käytetään yleensä URI-osoitetta, kuten "`http://mantelichat.fi/moderation`". Rakenne lisätään moderointiviestiin ja XMPP-palvelin välittää sen normaalisti protokollan osoittestuksen mukaisesti. Häirikön saadessa moderointiviestin tarkistetaan viestin laillisuus asiakaspäässä, eli selvitetään onko se peräisin keskustelun valvojalta. Mikäli näin on, ohjelma tallentaa hiljennyksen tiedot keksiin ja estää häirikön ohjelmaa lähettämästä uusia viestejä tietyn aikavälin sisällä.

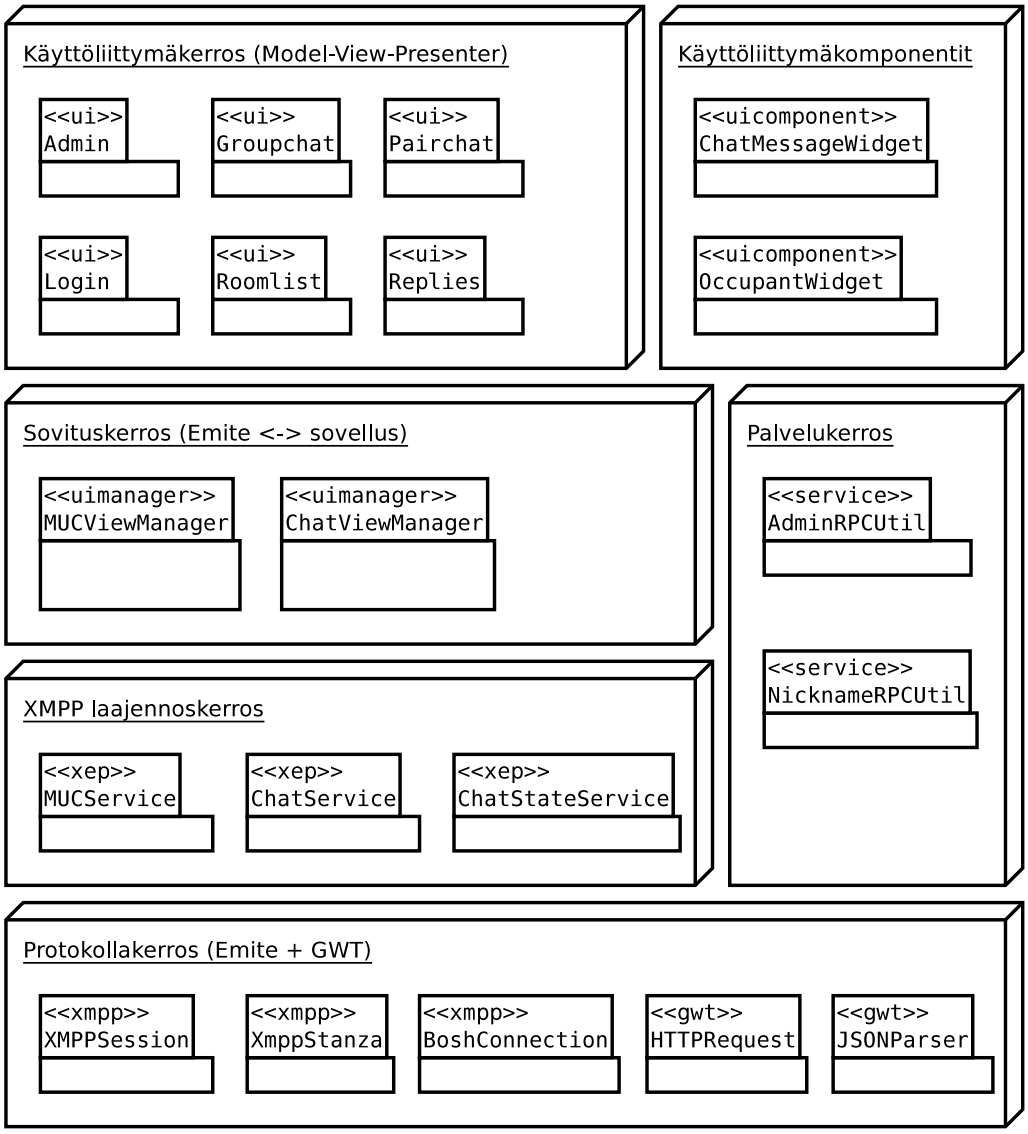
4.2 Asiakassovelluksen arkkitehtuuri

4.2.1 Yleiskuvaus

Selaimeen tarkoitetun asiakassovelluksen toteutuksen avuksi otettiin käyttöön avoimen lähdekoodin Emite GWT-kirjasto [3], joka toteuttaa XMPP-protokollatason liikenteen ja palvelinyhteyden yksityiskohdat. Kirjaston ominaisuudet voidaan jakaa kolmeen kerrokseen: alimmalla tasolla on AJAX-pyyntöjen käsittely ja BOSH-

[4]evercookie - `http://samy.pl/evercookie/`

protokollan mukainen yhteysmekanismi, toisella tasolla on XMPP-protokollan ytimen mekanismien toteutus, kuten XMPP-viestirakenteet, kirjautuminen ja istunto-abstraktio, kolmannella tasolla on joitakin XMPP-protokollan laajennosten toteutuksia, kuten yksityisviestiliikenne. Emite-kirjaston kerrokset on toteutettu löyhän sidonnan periaatteen mukaisesti, joten kerrosten välisten riippuvuuksien määrä on minimaalinen. Tämä helpottaa sovellusohjelmoijan kannalta kirjaston ominaisuuksien laajentamista ja parantelua. Toisen kerroksen istunto-abstraktion tarjoaa kaikista monipuolisimman laajentamismekanismin.



Kuva 4.3: Arkkitehtuurikaavio

Asiakassovelluksen arkkitehtuuri jakautuu useampaan abstraktiokerrokseen, joita on hahmoteltu kaaviossa 4.3. Kerrokset limittyvät osittain Emite-kirjaston omien kerrosten kanssa alemmilla tasoilla. Alin protokollakerros koostuu lähinnä valmiista Emite-kirjaston XMPP-protokollan ytimen toteutuksesta. Tälle tasolle kuuluu myös

GWT:n matalan tason toimintoja kuten JSON[5] ja XML-jäsentäjät sekä HTTP-pyyntötyökalut.

Laajennoskerros koostuu sovellusta varten räätälöidystä protokollalaajennoksista, jossa on täydennetty joitakin Emite-kirjaston puutteita. Kerroksen rinnalla on palvelukerros, jota käytetään sellaisten etäkutsujen tekemiseen, jotka eivät ole mahdollista pelkän XMPP-protokollan avulla. Näitä toimintoja ovat esimerkiksi palvelimen käyttäjätilien käsittely. Palvelut on toteutettu laajentamalla XMPP-palvelinohjelmistoa plugin-lisäosan avulla, joka vastaa JSON-muotoisiin HTTP-etäkutsuihin.

Sovituskerroksen tehtävänä on sitoa XMPP-protokollatason tapahtumat ja käyttöliittymä toisiinsa. Tämä taso huolehtii esimerkiksi yksityiskeskustelun viestien reitityksestä vastaavalle näkymäkomponentille. Kerroksen tehtävänä on muuntaa protokollatapahtumat käyttöliittymätapahtumiksi.

Käyttöliittymäkerros on rakennettu käyttäen GWT:n käyttöliittymäkirjaston komponentteja ja UIBinder-työkalua. UIBinder on XML-pohjainen käyttöliittymien deklaratiivinen kuvauskieli. Sovelluksen käyttöliittymä on rakennettu yhdistelemällä valmiskomponenteista yleiskäyttöisiä elementtejä kuten ChatMessageWidget, joka formatoi yksittäisen keskusteluviestin ulkoasun tapauskohtaisesti. Kerroksen sisällä suunnitteluun on vaikuttanut Model-View-Presenter suunnittelumalli, jonka käyttöä esitellään alakohdassa 4.2.4.

Osa sovelluksen näkymistä voi ilmentyä vain yhtenä kopiona sovelluksen elinkaaren aikana, mutta toisia voidaan luoda lennossa useampia. Esimerkiksi käyttäjän liittyessä keskusteluhuoneeseen luodaan uusi ryhmäkeskustelunäkymä, mutta kirjautusmisruutuja tarvitaan aina vain yksi. Samanlaiset näkymät niputetaan yhteen niiden tyypin perusteella. Tämä ominaisuus toistuu sovelluksessa kahdelle loogisella tasolla. Ensinnäkin käyttöliittymäkomponentit on käytettävyyden puolesta hyvä liittää yhden kokoavan käyttöliittymäelementin alle. Tämä tarkoittaa keskusteluhuoneiden selausta, joka tapahtuu `TabLayoutContainer`-tyyppisen elementin välilehtien avulla. Toisaalta komponentit tulee niputtaa hallinnollisesti yhteen, jotta viestin saapuessa protokollatasolla voidaan se helposti näyttää vastaavassa käyttöliittymäelementissä.

Koska luotuja näkymiä on mahdollisesti suuri määrä, tulee näkymiä voida hallita keskitetysti. Lisäksi on kiinnitettävä huomiota näkymien tiedonvälityksiin ja riippuvuuksien hallintaan. Näiden asioiden vuoksi tarvitaan asianmukaisia suunnitteluratkaisuja. Ratkaisuja kuvaillaan seuraavaksi.

[5] `http://www.json.org/`

4.2.2 Komponenttien riippuvuuksien hallinta

Hyvin ositettu sovellus rakentuu järkevästä määrästä heikosti kytkettyjä komponentteja, jotka kytkeytyvät toisiinsa selkeiden palvelurajapintojen välityksellä. Komponenteilla on sekä tarjottuja rajapintoja, että toisten komponenttien palvelurajapintoja, joiden toteutus täytyy tarjota alustusvaiheessa.

Riippuvuuksien hallintaan on tarjolla kaksi lähestymistapaa, joita erottaa toisistaan se miten komponentit on kytketty toisiinsa. Ensinnäkin riippuvuudet voidaan kytkeä komponentin toteutukseen sisäisesti eksplisiittisillä käskyillä, jotka luovat tarvittavat toteutusluokat tai etsivät toteutuksen. Tämän vahvasti kytketyn lähestymistavan ongelmana on tarve tietää palvelun tarjoavan komponentin alustukseen tarvittavat parametrit. Toinen lähestymistapa on eriyttää komponenttien alustaminen ja riippuvuuksien hallinta omaksi kokonaisuudekseen. Tätä lähestymistapaa kutsutaan "Dependency Injection"-suunnittelumalliksi (DI), joka on erikoistapaus "Inversion on Control"-periaatteesta. DI pyrkii muokkaamaan suorituspolkua niin että ohjelman komponentit olisivat heikommin kytkettyjä. [21]

Toteutetussa sovelluksessa käytetään Suco-nimistä DI-kirjastoa, koska Emite-kirjasto vaatii sen. Riippuvuuksien hallinta tapahtuu Suco:ssa luomalla moduuleja, jotka voivat rekisteröidä palveluiden toteutuksia niiden rajapinnan avulla.

Listaus 4.1: Palvelurajapinnan toteutuksen rekisteröinti ohjelman moduulissa

```
public class RepliesModule extends AbstractModule implements
    EntryPoint {
  public void onModuleLoad() {
    /* GWT suorittaa palvelun rekisteröinnin Suco:on */
    Suco.install(this);
  }

  protected void onInstall() {
    /* Suco rekisteröi palvelun rajapinnan avulla */
    register(Singleton.class, new
        Factory<StoredRepliesManager>(StoredRepliesManager.class) {
         public StoredRepliesManager create() {
           return new StoredRepliesManagerImpl(
           $(Session.class), $(UserManager.class),
               $(PrivateStorageManager.class));
         }
    });
  }
}
```

Listauksessa 4.1 on esimerkki moduulista, joka tarjoaa palvelun toteutuksen rajapinnalle `StoredRepliesManager`. Palvelun rekisteröinti tapahtuu määrittelemällä tehdasluokka, joka palauttaa aina saman toteutusluokan singleton-näkyvyysalueen

avulla [22]. Rekisteröinnin jälkeen palvelu on ohjelman muiden moduulien käytettävissä. Muut moduulit viittavaat siihen rajapinnan avulla.

Moduulilla on riippuvuuksia ulkoisiin rajapintoihin, joita tarvitaan palvelun luomiseen. DI-suunnittelumallin mukaisesti riippuvuudet kytketään toteutuksen ulkopuolella `create()`-tehdasmetodissa, joka pyytää muiden tarvittavien palveluiden toteutukset Suco:lta. Tarvittavat palvelut on rekisteröity ohjelman toisissa moduuleissa.

4.2.3 Asynkroniset tapahtumat ja viestiväylä -suunnittelumalli

Keskusteluohjelmaa suunniteltaessa oli selvää, että suurin osa tiedonkäsittelystä tulee tapahtumaan viestiliikenteestä johtuen asynkronisella periaatteella. Asynkroniset viestiohjatut ohjelmat käsittelevät tapahtumia, joiden esiintyminen on sattumanvaraista. Tällaisia tapahtumia ovat esimerkiksi saapuvat XMPP-viestit ja käyttöliittymätapahtumat. Käyttäjän hiiren painallus luo tapahtuman, jonka siirtäminen käsiteltäväksi tapahtuu selaimen tapahtumienkäsittelijäsäikeessä.

Toinen viestintäsovelluksen asynkronisten tapahtumien muoto ovat asynkroniset verkkotapahtumat. AJAX-kutsujen lähettäminen palauttaa kontrollin välittömästi kutsujalle, mutta pyynnön tila päivitetään kutsujalle asynkronisesti takaisinkutsun avulla. Takaisinkutsu aktivoidaan satunnaisella ajanhetkellä, joka riippuu niin verkkoviiveestä kuin palvelimen prosessointiajastakin. Hieman korkeammalla abstraktiotasolla myös XMPP-protokollan viestit ja pyynnöt toimivat asynkronisesti. BOSH-yhteyden ominaisuuksien vuoksi esimerkiksi IQ-viestin vastaus voi saapua vasta useamman HTTP pyynnön jälkeen.

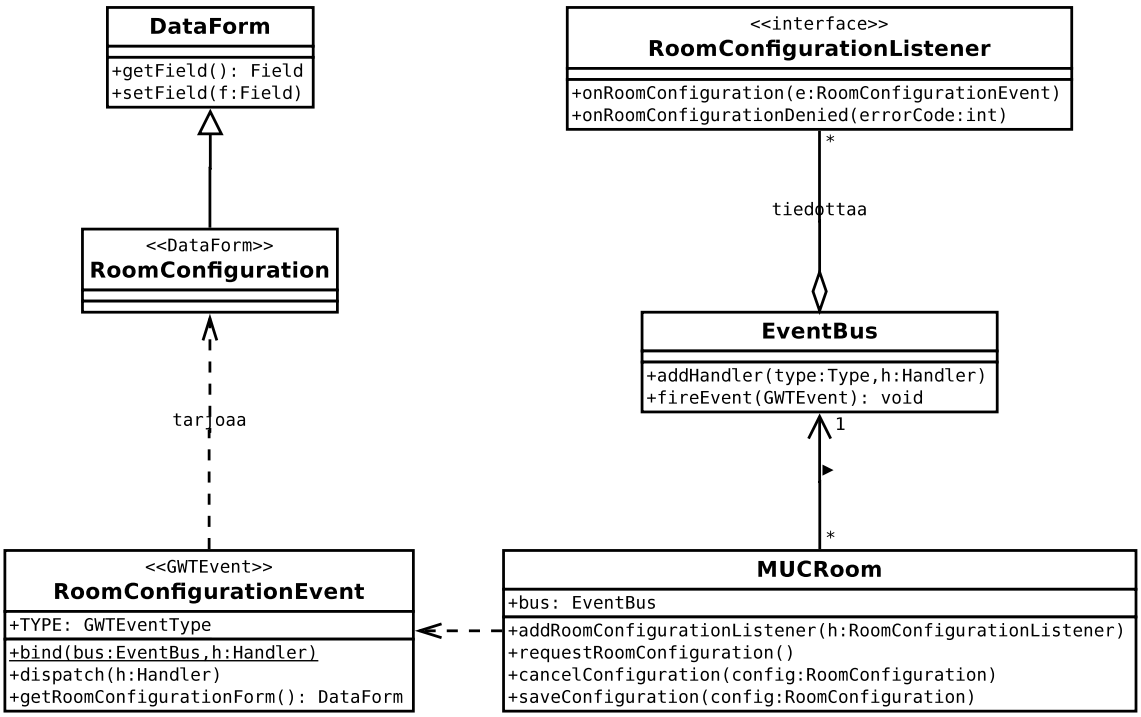
Edellä mainittujen asynkronisten piirteiden vuoksi monet sovelluksen toiminnot toteutettiin tarkkailija-suunnittelumallin [22] mukaisesti. Suunnittelumallissa on kaksi roolia: tarkkailija (engl. observer) ja tarkkailtava tapahtumalähde (engl. observable). Tarkkailija on komponentti, joka on kiinnostunut tarkkailtavan tilan muutoksista. Esimerkiksi keskusteluhuoneen tapauksessa käyttöliittymäkomponentti on kiinnostunut reagoimaan kun keskustelijaluettelo muuttuu, jotta muutos voidaan esittää käyttäjälle. Tällöin sen tulee ohjelman alustusvaiheessa rekisteröityä keskusteluhuoneesta vastaavan komponentin tarkkailijaksi. Tarkkailtavan komponentin vastuulla on kutsua jokaisen tarkkailevan komponentin rajapintafunktiota, jolla tapahtumasta voidaan tiedottaa olennaiset tiedot.

Tavanomaisen tarkkailijasuunnittelumallin heikkoutena on, että se edellyttää vahvaa sidontaa komponenttien välillä. Lisäksi sen laajentaminen on hankalaa, jos tulevaisuudessa halutaan välittää ja tarkkailla uudenlaisia kiinnostavaan komponenttiin liittyviä tapahtumia. Tällöin tarvitaan muutoksia komponentin rajapintaan, jolla voidaan rekisteröidä tarkkailijoita uudenlaisia tapahtumia varten. Näiden ongelmien ratkaisuna voidaan hyödyntää viestiväylä -suunnittelumallia [25].

Ohjelmistotekniikassa viestiväylän tehtävänä on vähentää komponenttien keskinäisiä riippuvuuksia ja kapseloida niiden toteutukset paremmin. Viestiväylän kautta voidaan lähettää ja vastaanottaa tapahtumia, joiden tyyppi voidaan valita vapaasti. Tapahtuman lähettäjä voi siis laajentaa tapahtumatyyppien joukkoa halutessaan. Tarkkailijan täytyy tuntea vain juuri sitä kiinnostuvan tapahtuman tyyppi, joten riippuvuutta sen lähettävän komponentin tyyppiin ei synny.

Viestiväylää päätettiin käyttää sovelluksen perusrakenteena edellä mainittujen syiden vuoksi. Toisaalta sen käyttö johtaa modulaarisempaan suunnitteluun ja siksi parempaan ylläpidettävyyteen.

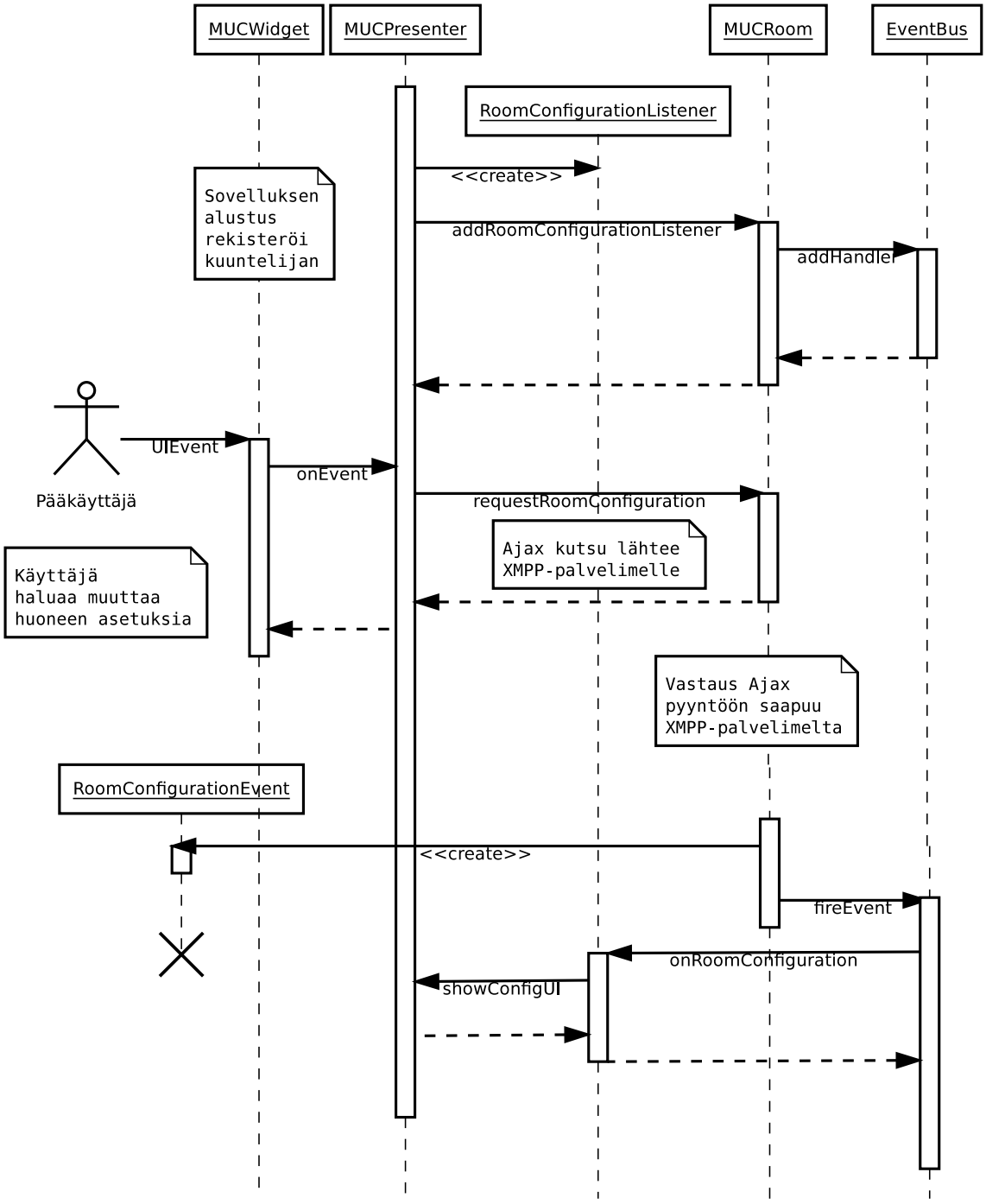
Seuraavaksi asynkronista toimintaperiaatetta ja viestiväylää esitellään käytännön tasolla esimerkin avulla. Tarkasteltavaksi on valittu käyttötapaus, jossa viestintäpalvelun pääkäyttäjä on luonut uuden ryhmäkeskusteluhuoneen ja haluaa muokata sen asetuksia. Tilanteen kulkua esittää tapahtumasekvenssikaavio 4.5. Asetusten muokkaaminen tapahtuu XMPP-viestinnän avulla, ja toimintoon osallistuu useita sovelluksen komponentteja.



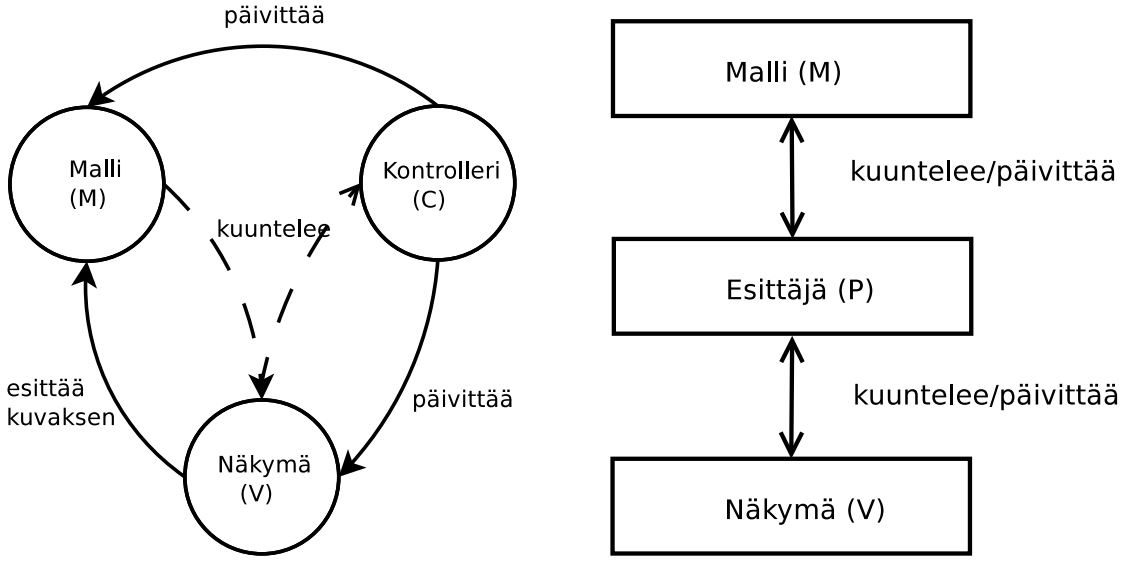
Kuva 4.4: Huoneen asetusten asynkroniseen muokkaamiseen liittyvät oliot

Kaaviossa 4.4 on esitelty toiminnon kannalta kiinnostavimmat komponentit ja niiden viitteelliset rajapinnat olennaisilta osin. Keskiössä on keskusteluhuoneen toteuttava kontrolliolio MUCRoom, jolla on toimintoja huoneasetusten käsittelyyn. Asetusten muokkaaminen tapahtuu pyynnöllä XMPP-palvelimelle, joka palauttaa standardimuotoisen lomakkeen. Tämä pyyntö lähetetään MUCRoom luokan

`requestRoomConfiguration`-metodilla. Pyyntö on lähtöisin keskusteluhuonenäkymän käyttöliittymäkomponentista. Pyynnön vastauksen käsittelyä varten sovelluksessa on toinen käyttöliittymä moduuli, joka esittää muokattavat huoneasetukset dialogissa. Sen toteutus on erillään huonenäkymän toteutuksesta, joten niiden välille ei haluta riippuvuutta.



Kuva 4.5: Käyttöliittymäkomponenttien viestinvälitys, kun käyttäjän haluaa muokata ryhmäkeskusteluhuoneen asetuksia.



(a) Model–View–Controller -suunnittelumalli (b) Model–View–Presenter -suunnittelumalli

Kuva 4.6: Käyttöliittymäohjelmoinnin suunnittelumallit

Kahden käyttöliittymän näkymän välissä on viestiväyläkomponentti EventBus. Sen toteutusinstanssi on yhteinen kaikille huoneeseen liittyville tapahtumille, joten se voidaan pyytää huoneoliolta MUCRoom tarvittaessa. Huoneoliossa on luokkakohtainen apumetodi, jonka avulla voidaan rekisteröidä viestiväylään helposti huoneasetusten kuuntelija `RoomConfigurationListener`. Tämä kuuntelija vastaanottaa tapahtuman, joka on lähtöisin XMPP-vastausviestistä. Jos kysyjän oikeudet riittävät ja virhettä ei tapahtunut, kuuntelijalle luodaan `RoomConfiguration`-sovittaja, jonka kautta vastausviestin sisältöä voidaan käsitellä. Jos pyyntö epäonnistui niin kuuntelijan toista `onRoomConfigurationDenied`-metodia kutsutaan. Olennaista tapahtumasekvenssissä on, että suoraa rajapinta riippuvuutta kahden käyttöliittymä komponentin välille ei synny.

4.2.4 Model-View-Presenter -suunnittelumallin soveltaminen

Perinteisessä käyttöliittymäohjelmoinnissa on muodostunut tavaksi käyttää hyviksi havaittuja ja luotettuja suunnittelumalleja kuten "*Model-View-Controller*" tai "*Command*". GWT-ympäristön Javapohjaisuus tarjoaa helpomman lähestymistavan soveltaa tuttuja ratkaisuja myös ohjelmoijien kannalta vieraaksi koettuun HTML/CSS-maailmaan. [33]

Sovelluksen käyttöliittymän kulmakiveksi valittiin "*Model-View-Presenter*" (MVP) rakenne. MVP (kuvassa 4.6(b)) on sukua "*Model-View-Controller*" suunnittelumallille (MVC) (kuvassa 4.6(a)). Yhteistä näille suunnittelumalleille on jakaa käyttöliittymän toteutus kolmeen komponenttiin. Malleista on olemassa useita variaatioita, mutta pääpiirteissään komponenttien vastuut ovat seuraavat: sunnittelumallissa

mallikomponentin (M) vastuuna on huolehtia näkymässä näytettävän ja muokattavan tiedon säilyvyydestä. Näkymäkomponentin (V) vastuuna on tarjota rajapinta näkyvälle käyttöliittymän toteutukselle. Kontrolleri- tai esittäjäkomponentin tehtävänä on koordinoida edellämainittujen avulla käyttäjän kanssa tapahtuvaa interaktiota. [20]

MVP:n käyttöä on perusteltu paremmalla tuella testaukselle sekä yksinkertaisemmalla riippuvuusmallilla. Tärkeimpänä erona on suoran riippuvuuden poistaminen näkymäkomponentin ja mallikomponentin väliltä MVP-mallissa. Tämä nostaa esittäjäkomponentin (P) keskeisempään rooliin kuin kontrollerin (C) MVC-mallissa. [19]

Esittäjäkomponentin tehtävänä on kunnella käyttöliittymätapahtumia ja reagoida niihin. Esittäjä rekisteröityy kuuntelemaan käyttöliittymän tapahtumia ja vastaa muutosten välittämisestä käyttöliittymä- ja mallikomponetin välillä. [19]

Näkymäkomponentti on vastuussa käyttöliittymän näkyvästä osuudesta. Se siis kokoaa visuaalisista käyttöliittymäkomponenteista hierarkian ja asettelee nämä näkymään sekä tyylittelee niiden ulkoasun. Lisäksi se tarjoaa rajapinnassaan rekisteröintimetodit kuuntelijoille. Rekisteröintimetodien avulla esittäjäkomponentti voi seurata käyttöliittymän tapahtumia. [19]

Listaus 4.2: MVP-suunnittelumallin esittäjäkomponentin rajapinta

```
public interface Presenter<T extends Display> {
  String getID();
  String getFocusGroupId();
  T getDisplay();
  Visibility getVisibility();
  HeaderDisplay getHeaderDisplay();
}
```

Listaus 4.3: MVP-suunnittelumallin näkymäkomponentin rajapinta

```
public interface Display {
  public enum DisplayType { groupchat, pairchat, roomlist, roomconfig,
      occupants, admin, chatmessage, tabheader, main }
  DisplayType getDisplayType();
  Widget asWidget();
}
```

Toimivaa MVP:n mukaista vastuunjakoa varten sovellukseen luotiin rajapinnat esittäjä- (listaus 4.2) ja näkymäkomponenttia (listaus 4.3) varten. Mallikomponentti haluttiin luoda vapaasti, joten sitä varten ei tehty kiinteätä rajapintaa. Rajapintojen yksityiskohtia tarkastellaan seuraavaksi esimerkin avulla.

Käytännössä esittäjäkomponenteilla on paljon yhteisiä toteutuspiirteitä. Yksi tällainen yhdistävä tekijä on tarve saada tieto siitä kun näkymä saa fokuksen eli tulee

näkyviin tai kun näkymä katoaa näkyvistä. Yksinkertaisimmillaan voidaan näkymän saatua fokuksen keskittää kursorin sijainti automaattisesti tekstikenttään samalla kun yksityiskeskustelynäkymä valitaan. Toiseksi voidaan päivittää näkymän välilehden otsikon sisältöä, jossa yleensä lukee kuinka monta viestiä näkymään on saapunut sen ollessa poissa fokuksesta.

Listaus 4.4: Näkymiä säilövän komponentin toteuttama säiliörajapinta

```
public interface PresenterContainer {
  void addPresenter(Presenter<?> presenter);
  void removePresenter(Presenter<?> presenter);
  void blinkPresenter(Presenter<?> presenter);
}
```

Käyttöliittymän näkymät sijoitetaan kahteen säiliöön. Keskusteluhuonenäkymät kasataan yhteen välilehdillä toimivaan säiliöön ja yksityiskeskustelut toiseen. Näkymien valinta tapahtuu välilehtien otsikoita painamalla, jotka näkyvät kuvan 4.1 ylä- ja alapalkissa. Kummatkin säiliöt ovat näkyvissä samanaikaisesti ja niissä on aktiivisena jokin sovelluksen näkymäkomponenteista. Säiliöiden toimintoja kuvaavat rajapintaluokan metodit listauksessa 4.4. Säiliöön voidaan lisätä ja poistaa näkymiä, sekä niiden aktiivisuudesta voidaan ilmoittaa vilkuttamalla. Lisääminen tapahtuu antamalla säiliölle näkymän esittäjä komponentti, josta on aina myös linkki näkymän toteutukseen.

Listaus 4.5: Abstrakti kantaluokka esittäjäkomponenteille, joiden ohjaama näkymäkomponentti on kiinnostunut fokustapahtumista.

```
public abstract class AbstractFocusablePresenter<T extends Display>
    implements FocusEventHandler, Presenter<T> {
  public AbstractPresenter(final HandlerManager eventBus, final T
      display) {
    eventBus.addHandler(FocusEvent.TYPE, this);
  }

  public void onFocus(FocusEvent event) {
      if (this.getID().equals(event.getFocusedID())) {
      setVisibility(Visibility.focused);
      } else if
          (this.getFocusGroupId().equals(event.getFocusGroupId())) {
        setVisibility(Visibility.hidden);
    }
  }

    private void setVisibility(Visibility value) {
      if ((this.visibility = value) == Visibility.focused) {
        onAfterFocus();
      } else {
```

```
            onAfterDefocus();
        }
    }

    abstract protected void onAfterFocus();
    abstract protected void onAfterDefocus();
}
```

Käyttöliittymän säiliössä olevien komponenttien toteutus yhtenäistettiin luomalla abstrakti kantaluokka *AbstractFocusablePresenter* (listaus 4.5). Kaikki säiliössä olevien näkymien esittäjäkomponentit perivät tämän luokan. Luokka toteuttaa rajapinnan *FocusEventHandler*, joka mahdollistaa rekisteröitymisen viestiväylän tapahtumiin, jotka tiedottavat fokuksen siirtymisestä. Luokka toteuttaa myös esittäjäluokkien yhteisen rajapinnan *Presenter<T extends Display>*.

Abstrakti kantaluokka jättää toteuttamatta kaksi suojattua metodia, joihin toteuttava luokka voi kytkeä fokustapahtumiin liittyvän toiminnon. Näiden metodien kutsuminen perustuu tarkasteluun, joka suoritetaan kuuntelijarajapinnan toteutuksessa metodissa onFocus(FocusEvent event). Algoritmin tehtävänä on päätellä onko näkymä vielä näkyvissä. Jos valitun esittäjäkomponentin tunniste on sama kuin nykyisen komponentin, on selvää että näkymä on näkyvissä. Muussa tapauksessa täytyy tarkastella fokusryhmän tunnistetta.

Fokusryhmä tarkoittaa joukkoa näkymiä, joista vain yksi voi kerrallaan olla näkyvissä. Kummatkin välilehtikomponentit muodostavat siis oman fokusryhmänsä. Tämän vuoksi esittäjä voi päätellä kadottaneensa fokuksen, jos valittu komponentti kuuluu samaan ryhmään.

4.3 Järjestelmän asennuksen yksityiskohdista

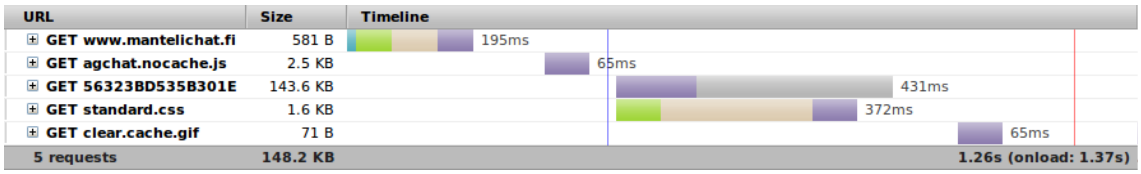
Joidenkin ei-toiminnallisten vaatimusten toteuttaminen vaati kiinnittämään erityistä huomiota palvelimen hankinnan ja asennuksen yksityiskohtiin. Palvelinasennuksessa alustana käytettiin Amazonin tarjoamaa EC2-pilvipalvelua[6], sen joustavuuden ja hyvän ylläpidettävyyden vuoksi. Amazon takaa palvelulle korkean 99,95 % saatavuuden, ja valittua asennustyyppiä voidaan myöhemmin päivittää tehokkaammaksi tarvittaessa. Päivitys on mahdollista tehdä ilman katkoja ja hyvin pienellä vaivalla. Lisäksi pilvialustan tarjoama tilannevedoksiin perustuvan koko asennuksen kattava varmuuskopiointi katsottiin eduksi. Amazon lupaa varmuuskopioille korkean luotettavuuden, koska ne voidaan hajauttaa laajalti maantieteellisesti suuresta ympäristöstä johtuen.

Tietoturvamielessä oli selvää, että luottamuksellisia keskusteluja sisältävän liikenteen tulisi olla salattua. Toiseksi palvelun identiteetin todentamista varten hankit-

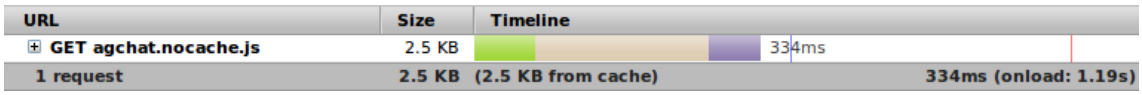
[6] `http://aws.amazon.com/ec2/`

tiin asianmukainen HTTPS/SSL -palvelinsertifikaatti. Siirrossa olevan tiedon luottamuksellisuuden varmistaminen oli helppoa, koska oli riittävää salata vain yhden HTTP-edustapalvelimen kautta kulkeva liikenne sertifikaatin varmistaman julkisen avaimen kryptografian avulla. Palvelimesta kytkettiin käytöstä pois vanhempi ja haavoittuvaisempi SSLv2-tuki sekä heikot alle 128-bitin avainta käyttävät salausmekanismit jotta saavutettaisiin FIPS-tietoturvastandardin mukainen taso.

Asiakassovelluksen käyttäminen koostuu suunnilleen 500:n kilotavun staattisen sisällön lataamisesta. Latauksen koko on melko pieni laajakaistayhteyksien aikana. Ylimääräistä palvelinkuormaa ja liikennettä haluttiin kuitenkin rajoittaa, koska pilvilpalvelun laskutusmallissa siirtokulujen osuus voi olla merkittävä. GWT:n tuottama ohjelma koostuu käytännössä vain muutamasta resurssista, joilla voidaan asettaa hyvin erilainen välimuistipolitiikka. Sovelluksen ydin eli skripti-, kuva- ja tyyli-tiedostot ovat tiivistettynä yhteen tiedostoon, jonka yksilöi tiedostonimen sisältöä kuvaava MD5-tarkistussumma. Sisällölle uniikista tiedostonimestä johtuen voidaan sen säilyminen selaimen välimuistissa asettaa varsin pitkäksi (kymmenen vuotta nykyhetkestä). Vaaraa esimerkiksi välityspalvelimella sijaitsevasta vanhasta kopiosta ei ole. Sen sijaan pääsivun ja sillä sijaitsevan bootstrap-skriptin tuoreus tulee varmistaa estämällä välimuistin käyttö lähes kokonaan. Palvelin konfiguraatiossa tiedostojen välimuistiasetukset voidaan asettaa tiedostopäätteen ".nocache" tai ".cache" avulla.



Kuva 4.7: Sivun lataaminen ilman välimuistia



Kuva 4.8: Sivun lataaminen välimuistista

Lopputuloksen kokonainen sivunlataus sisältää pakkauksen jälkeen noin 150 kilotavua dataa. Tämän ensimmäisen latauspyynnön ajoitus ja tietomäärät on esitetty kuvassa 4.7. Tätä seuraavat lataukset ilman selaimen välimuistin tyhjennystä ovat kooltaan marginaalisia, kuten kuvasta 4.8 nähdään.

4.4 Toteutuksen koodipohjan analysointi

Ohjelman ei-toiminnallisia vaatimuksia määritellessä oli selvää, että totetuskoodin koko on tärkeä Web-sovelluksen käytettävyystekijä. GWT-ympäristön Java-pohjaisuus antaa toteuttajalle mahdollisuuden käyttää Java SE -luokkakirjaston palveluita, joiden rajapinta eroaa monesti natiivin JavaScript:n vastaavista. Tämän vuoksi Java-rajapintojen käyttäminen lisää ohjelmaan ylimääräistä koodia, vaikka tarvittu ominaisuus olisi voitu toteuttaa natiivisti. GWT-ympäristö sisältää myös ylimääräistä arkkitehtuurikoodia, sekä käytetyt suunnittelumallit ja kirjastoriippuvuudet tuovat oman osansa käännetyn ohjelman kokoon.

GWT-kääntäjään on sisäänrakennettu ominaisuus, joka kerää staattisen analyysin tuloksista tilastoja. Tilastojen avulla saadaan likimääräinen kuva ohjelman eri rakenteiden tuottamasta koodista. Taulukossa 4.1 on esitetty sovelluksen pakkaamattoman koodin yleinen jakautuminen. Arkkitehtuurikoodin (GWT+JRE) määrä on suhteellisen pieni eli noin 18 % koko sovelluksesta. Tarkemmassa analyysissä ylimääräiseksi koodiksi arvioitiin esimerkiksi Java `java.util.Date`-luokan toteutukseen liittyvä päivämäärien käsittely, sekä monimutkaiset tietorakenteet, kuten `java.util.HashMap`.

Kokonainen sovelluskoodi	GWT arkkitehtuurikoodi	JRE kirjastot	Merkkijono–vakiot	Muu sovellus
379.0 (100 %)	19.4 (7,1 %)	28.3 (10,4 %)	75.1 (19,8 %)	237.9 (62.8 %)

Taulukko 4.1: Sovelluksen koodin jakautuminen

Kääntäjä tuottaa lisäksi raportin, josta voidaan tutkia ohjelman lähdekoodipakkausten kokoa. Taulukkoon 4.2 on kerätty ohjelman suurimmat toteutusyksiköt. Osuuksiin ei sisälly ohjelman koodin sisältämä merkkijonodata, jota on noin 20 %. Luvuista nähdään, että uudelleenkäytetyn toteutuksen osuus on ohjelmassa suuri eli noin 48 %. GWT:n osuus siitä on suuri.

Pakkauksen nimi	Osuus (%)
com.gogle.gwt.user.ui	15.23
com.calclab.emite	13.74
fi.ag.chat.muc	8.09
fi.ag.chat.pairchat	6.67
java.util	6.63
com.google.gwt.lang	6.33
fi.ag.chat.core	5.80
com.google.gwt.dom	4.06
com.google.gwt.i8n	3.82
java.lang	3.73
fi.ag.chat.roomlist	2.31
fi.ag.chat.storedmessage	2.27
fi.ag.chat.admin	2.03
com.google.gwt.core	1.85
com.google.gwt.layout	1.82
com.google.gwt.xml	1.63
loput yhteensä	19.29
sovelluskoodi	27.17
kirjastokoodi (emite)	13.74
kirjastokoodi (gwt)	34.74

Taulukko 4.2: Sovelluksen koodin jakautuminen pakkauksiin

5. YHTEENVETO

Tämän diplomityön tavoitteena oli suunnittella ja toteuttaa asiakkaan tarpeisiin räätälöity keskustelujärjestelmä. Palvelun vaatimuksissa korostui ei-toiminnallisten vaatimusten tärkeys, kuten ohjelmiston hyvä käytettävyys, korkea tietoturva, sekä tarve kevyelle yksinkertaisesti skaalautuvalle järjestelmälle.

Toteutettu järjestelmä on moderni paksuun arkkitehtuuriin perustuva rikas internetsovellus (RIA). Järjestelmän arkkitehtuuri perustuu XMPP-protokollaan, joka on vakiintunut ja laajasti käytössä oleva pikaviestintästandardi. Sovelluksen toiminnot voitiin toteuttaa käyttäen protokollan valmiita laajennoksia, kuten MUC-ryhmäkeskustelut ja XML-tiedonsäilyvyys. Olemassa oleva protokolla paransi sovelluksen tietoturvaa sekä mahdollisti laajasti testattujen palvelintoteutusten ja Emite-asiakaskirjaston käyttämisen. Yhteyden salaaminen onnistui yksinkertaisesti HTTPS-yhteyden avulla, joka on perustana järjestelmän BOSH-tyyppisessä push-ratkaisussa.

Sovelluksen Web-asiakas on toteutettu käyttäen hyväksi GWT-toteutusympäristön tarjoamaa abstraktiota. Abstraktioiden käyttäminen yhtenäisti selaimen käyttöä toteutusalustana sekä helpotti keskittymistä korkean tason suunnitteluun ja sovelluksen käytettävyyteen. Web-asiakkaan suunnittelussa on sovellettu hyväksi havaittuja RIA-suunnittelumalleja, kuten komponenttien heikkoa sidontaa viestiväyläsuunnittelumallin avulla sekä riippuvuuksien eriyttämistä “Inversion of Control”-säiliön avulla. Koodipohjan analysoinnissa havaittiin, että arkkitehtuuri ja valmiskirjastojen koodi ei kasvata sovelluksen latauskokoa merkittävästi.

XMPP-standardin mukainen järjestelmä tarjoaa tulevaisuudessa mahdollisuuden laajentaa sovellusta ottamalla käyttöön uusia laajennoksia, sekä integroida ulkoisia keskustelujärjestelmiä. GWT:n käyttäminen lisäsi ohjelman modulaarisuutta ja ymmärrettävyyttä, jotka ovat tärkeitä ominaisuuksia järjestelmän ylläpidon sekä jatkokehityksen kannalta.

LÄHTEET

[1] Adobe Flash. http://www.adobe.com/products/flashplayer/. [Online; viitattu 15.3.2011].

[2] ECMAScript Language Specification. http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf.

[3] Emite XMPP kirjasto. http://code.google.com/p/emite/. [Online; viitattu 23.9.2010].

[4] Google Web Toolkit - Productivity for developers, performance for users. http://code.google.com/intl/fi-FI/webtoolkit/. [Online; viitattu 10.1.2011].

[5] A little holiday present: 10,000 reqs/sec with Nginx! http://blog.webfaction.com/a-little-holiday-present. [Online; viitattu 7.12.2010].

[6] RFC 1459: Internet Relay Chat Protocol. http://www.irchelp.org/irchelp/rfc/rfc.html. [Online; viitattu 20.3.2011].

[7] Same origin policy for JavaScript. https://developer.mozilla.org/En/Same_origin_policy_for_JavaScript. [Online; viitattu 7.12.2010].

[8] XEP-0124: Bidirectional-streams over synchronous http (bosh). http://xmpp.org/extensions/xep-0124.html. Viitattu 23.9.2010.

[9] XMPP-protokollapinon ydin. http://xmpp.org/xmpp-protocols/xmpp-core/#base. [Online; viitattu 23.9.2010].

[10] Hypertext transfer protocol - http/1.1. http://www.ietf.org/rfc/rfc2616.txt, 1999.

[11] Document Object Model (DOM) Level 3 Core Specification. http://www.w3.org/TR/2004/REC-DOM-Level-3-Core20040407, 2004.

[12] Are AJAX Applications Vulnerable to Hack Attacks? The importance of Securing AJAX Web Applications. http://www.acunetix.com/websitesecurity/ajax_applications.pdf, 2006. [Online; viitattu 10.1.2011].

[13] Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. http://www.w3.org/TR/2010/WD-CSS2-20101207, 2010. W3C Working Draft 07 December 2010.

[14] 3Com. The Net Impact of Thin Clients. Technical report, 3Com corporation, 1999. [Online; viitattu 4.1.2011].

[15] Jeremy Allaire. Macromedia Flash MX-A next-generation rich client. Technical report, Macromedia, 2002. [Online; viitattu 5.1.2011].

[16] Engin Bozdag, Ali Mesbah, and Arie van Deursen. A comparison of push and pull techniques for ajax. *CoRR*, abs/0706.3984, 2007.

[17] Jim Conallen. *Building Web Applications with UML*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.

[18] Phifer G. Driver M., Valder R. Rich Internet Applications are the next Evolution of the Web. Technical report, Gartner. [Online; viitattu 5.1.2011].

[19] Aviad Ezra. Twisting the MVC triad - model view presenter (MVP) design pattern. `http://aviadezra.blogspot.com/2007/07/twisting-mvp-triad-say-hello-to-mvpc.html`. Viitattu 20.11.2010.

[20] Martin Fowler. GUI architectures. `http://martinfowler.com/eaaDev/uiArchs.html`. [Online; viitattu 20.11.2010].

[21] Martin Fowler. Inversion of Control Containers and the Dependency Injection pattern. `http://www.martinfowler.com/articles/injection.html`, 2008. [Online; viitattu 24.1.2011].

[22] Erich Gamma, Richard Helm, Ralph E. Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[23] Jesse James Garrett. Ajax: A New Approach to Web Applications. `http://www.adaptivepath.com/ideas/essays/archives/000385.php`. [Online; viitattu 25.10.2010].

[24] Ian Hickson. Server-Sent Events. `http://www.w3.org/TR/2011/WD-eventsource-20110310/`. [Online; viitattu 15.3.2011].

[25] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[26] Federico Kereki. Web 2.0 development with the Google web toolkit. *Linux J.*, 2009(178):2, 2009.

[27] Pelastakaa Lapset. Manteli /tukichat. `http://www.pelastakaalapset.fi/toiminta/nuorisotoiminta/manteli-tukichat/`.

[28] Ali Mesbah and Arie van Deursen. Migrating multi-page web applications to single-page ajax interfaces. In *CSMR '07: Proceedings of the 11th European Conference on Software Maintenance and Reengineering*, pages 181–190, Washington, DC, USA, 2007. IEEE Computer Society.

[29] Tommi Mikkonen and Antero Taivalsaari. Web applications - spaghetti code for the 21st century. Sun Microsystems Research Tech Report TR-2007-166, Sun Microsystems, June 2007.

[30] Jack Moffitt. XMPP is Better With BOSH. `http://metajack.wordpress.com/2008/07/02/xmpp-is-better-with-bosh/`. [Online; viitattu 28.10.2010].

[31] Kevin Smith Peter Saint-Andre and Remko Troncon. *XMPP: The Definitive Guide*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2009.

[32] Mikko Pohja. Server push with instant messaging. In *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, pages 653–658, New York, NY, USA, 2009. ACM.

[33] Chris Ramsdale. Large scale application development and MVP. `http://code.google.com/intl/fi-FI/webtoolkit/articles/mvp-architecture.html`. [Online; viitattu 26.1.2011].

[34] David Reiss. Facebook Chat Now Available Everywhere. `http://blog.facebook.com/blog.php?post=297991732130`, 2010. [Online; viitattu 21.1.2011].

[35] Alex Russell. Comet: Low Latency Data for the Browser. `http://infrequently.org/2006/03/comet-low-latency-data-for-the-browser/`. [Online; viitattu 27.10.2010].

[36] Adobe Systems. What are local shared objects? `http://www.adobe.com/products/flashplayer/articles/lso/`, 2011. [Online; viitattu 20.3.2011].

[37] Cisco Systems. How NAT Works. `http://www.cisco.com/image/gif/paws/6450/nat-cisco.pdf`, 2010.

[38] Antero Taivalsaari, Tommi Mikkonen, Dan Ingalls, and Krzysztof Palacz. Web browser as an application platform. *Software Engineering and Advanced Applications, Euromicro Conference*, 0:293–302, 2008.

[39] A. van Kesteren. Cross-origin resource sharing (work in progress). `http://www.w3.org/TR/2010/WD-cors-20100727/`. [Online; viitattu 7.12.2010].

[40] W3C. HTML 4.01 specification. http://www.w3.org/TR/html401, 1999.

[41] W3C. XMLHttpRequest. http://www.w3.org/TR/2010/CR-XMLHttpRequest-20100803/, 2010.

[42] W3C. Web Storage. http://www.w3.org/TR/2011/WD-webstorage-20110208/, 2011.

[43] Wikipedia. Comet (programming) — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Comet_(programming)&oldid=408945208, 2011. [Online; viitattu 27.1.2011].