Pasi Ketonen

Interaktiivisen tietokantasovelluksen toteutus WebSpeedillä

Tietotekniikan
pro gradu -tutkimus
7.2.2000

Jyväskylän yliopisto
Tietotekniikan laitos

SISÄLLYSLUETTELO

INTERAKTIIVISEN TIETOKANTASOVELLUKSEN TOTEUTUS WEBSPEEDILLÄ

Pasi Ketonen

Tietotekniikka

7.2.2000

Jyväskylän yliopisto

157

TIIVISTELMÄ

Tässä tutkielmassa tarkastellaan CGI-ympäristön käyttöä tietokantasovellusten tekemisessä. CGI-ympäristöä tutkitaan Progress Softwaren WebSpeed-sovelluskehittimellä ja tarvittaessa havaituista tuloksista lähetetään korjausehdotuksia WebSpeedin kehitysryhmällä. Tutkimus liittyy tekijän työskentelyyn Progress Software Oy:ssä.

Tutkimus koostuu neljästä osasta. Ensimmäisessä osassa määritellään WWW ja Internet sovellusten ajoympäristönä. Toisessa osassa määritellään tietokannan käsittelyyn liittyvät perusvaatimukset, Internet tietokannan ajoympäristönä sekä WebSpeed ohjelmointi- ja ajoympäristönä. Kolmannessa osassa tutkitaan esimerkkiohjelmilla tietokantasovellusten toteuttamista WebSpeedillä. Neljännessä osassa pohditaan tulevaisuuden tarpeita ja

näkymiä tietokantasovellusten tekemisessä webbiin sekä arvioidaan CGI-ympäristöä ja WebSpeediä tietokantasovellusten tekemisessä.

CGI-ympäristössä tehtyjen tietokantasovellusten keskeinen tekijä on ohjelman kontekstin säilyttäminen, jonka toteuttamiseksi WebSpeed sisältää erilaisia tekniikoita. Internet kasvattaa myös sovellusten käyttäjämäärää, mikä vaatii ajoympäristöiltä erityistä skaalautuvuutta, johon WebSpeedin ajoympäristö sisältää erilaisia sekä automaattisia että sovelluksen ohjelmointiin liittyviä menetelmiä.

AVAINSANAT

CGI, tietokantatransaktio, konteksti, monikielisyys

IMPLEMENTING INTERACTIVE DATABASE APPLICATION USING WEBSPEED

Pasi Ketonen

Computer Sciences

7.2.2000

University of Jyväskylä

157

PREFACE

This research is a study of using CGI-environment with database applications. The study is done by using Progress Software's WebSpeed as a case and it is related to author's work at Progress Software Oy. The results of the research are reported to the WebSpeed's development team if needed.

Research consists of four parts. The first part defines WWW and Internet as an application run-time and development environment. The second part defines basic needs for database usage, Internet as a database's run-time environment and WebSpeed as a development and run-time environment. The third part is a research of doing database applications with WebSpeed. It includes sample programs. The fourth part contains considerations of future WWW technology views and needs. There is also some evaluation of how well do CGI and

WebSpeed suit for doing database applications. It also includes thoughts of how future technologies will effect WWW programming.

An essential feature in CGI-programs is how to save program's context. WebSpeed has several techniques to save and pass program's context over HTTP requests. Internet as an application's run-time environment increases the number of users which requires scalability. WebSpeed has both automatic scalability options in run-time environment and programming techniques for scalability.

KEYWORDS

CGI, transaction, context, multilingual, scalability

1. Johdanto

Tämä tutkimus on Progress Softwaren kehittämällä WebSpeedillä toteutettu tapaustutkimus CGI-ympäristön (Common Gateway Interface, kts. Käsitteitä) soveltuvuudesta tietokantasovellusten tekemiseen. WebSpeed on WWW-sovellusten CGI-pohjainen kehitys- ja ajoympäristö, joka on suunniteltu tietokantasovellusten ohjelmointiin. Se perustuu Progressin 4gl-ohjelmointikieleen (4th generation language) ja käyttöliittymän tekemisessä käytettävään HTML-muotoilukieleen.

Tietokantasovelluksen toteuttamiseen vaikuttavat useat WWW:ssä käytettävät tekniikat, kuten HTTP (Hypertext Transfer Protocol, kts. Käsitteitä), tietokanta ja ajoympäristön suorituskyky. CGI-pohjaiseen tietokantasovellukseen kuuluvia ohjelmistokomponentteja ovat WWW-palvelin, selain, sovelluksen ajoympäristö ja tietokanta. Tutkimuksessa käydään läpi eri tekniikoiden ja komponenttien soveltuvuus maailmanlaajuisesti käytetyissä tietokantasovelluksissa.

Tietokantasovellusten käsittelyssä on olennaista sovelluksen tilan säilyttäminen. CGI-ympäristössä käytettävä HTTP-protokolla on kuitenkin tilaton, joka onkin keskeinen WWW-sovelluksiin vaikuttava seikka. Tutkimuksessa käydään läpi, kuinka sovelluksen tila voidaan säilyttää standardeilla WWW-tekniikoilla ja WebSpeed-ohjelmoinnilla. Tietokantojen ja WebSpeedin käsittelyn yhteydessä tutkitaan maailmanlaajuiseen käyttöön tehtäviin sovelluksiin vaikuttavia

seikkoja, kuten monikielisyys. Tavoitteena on tutkia ohjelman kontekstin säilyttämiseen käytettävissä olevia WWW- ja WebSpeed-tekniikoita sekä arvioida niiden soveltuvuutta tietokantaan päivittävissä sovelluksissa.

Tutkimus jakautuu neljään osaan. Ensimmäisessä osassa (luvut 2-4) tarkastellaan webbiä ja Internetiä sovellusten ajoympäristönä. Erityisesti tutkitaan kirjallisuuden pohjalta CGI- ja HTML-ympäristöä (Hypertext Markup Language, kts. Käsitteitä) ohjelmoinnin kannalta. Ensimmäisessä osassa määritellään CGI-ohjelmointiympäristön sisältämien tekniikoiden - kuten HTTP:n ja HTML:n - ohjelmoinnille asettamat rajoitukset. Toisessa osassa (luvut 5-6) määritellään tietokannan käsittelyyn liittyvät perusvaatimukset, Internetin ajoympäristöön kohdistamat lisätarpeet sekä WebSpeed ohjelmointi- ja ajoympäristönä. Kolmannessa osassa (luku 7) tutkitaan esimerkein WebSpeedillä sovelluksen kontekstin säilyttämistä – erityisesti tietokannan käsittelyn kannalta - sekä kansainvälisen monikielisen sovelluksen tekemistä. Neljännessä osassa (luvut 8-9) arvioidaan tulevaisuuden tarpeita ja näkymiä tietokanta- ja WWW-sovellusten tekemisessä. Lopuksi arvioidaan WebSpeedin soveltuvuutta sekä kehitystarpeita tietokantasovellusten tekemiseksi ja ajamiseksi. Viimeisessä osassa pohditaan lisätutkimusaiheita, jotka liittyvät webbiin tehtäviin tietokantasovelluksiin ja tähän tutkimukseen.

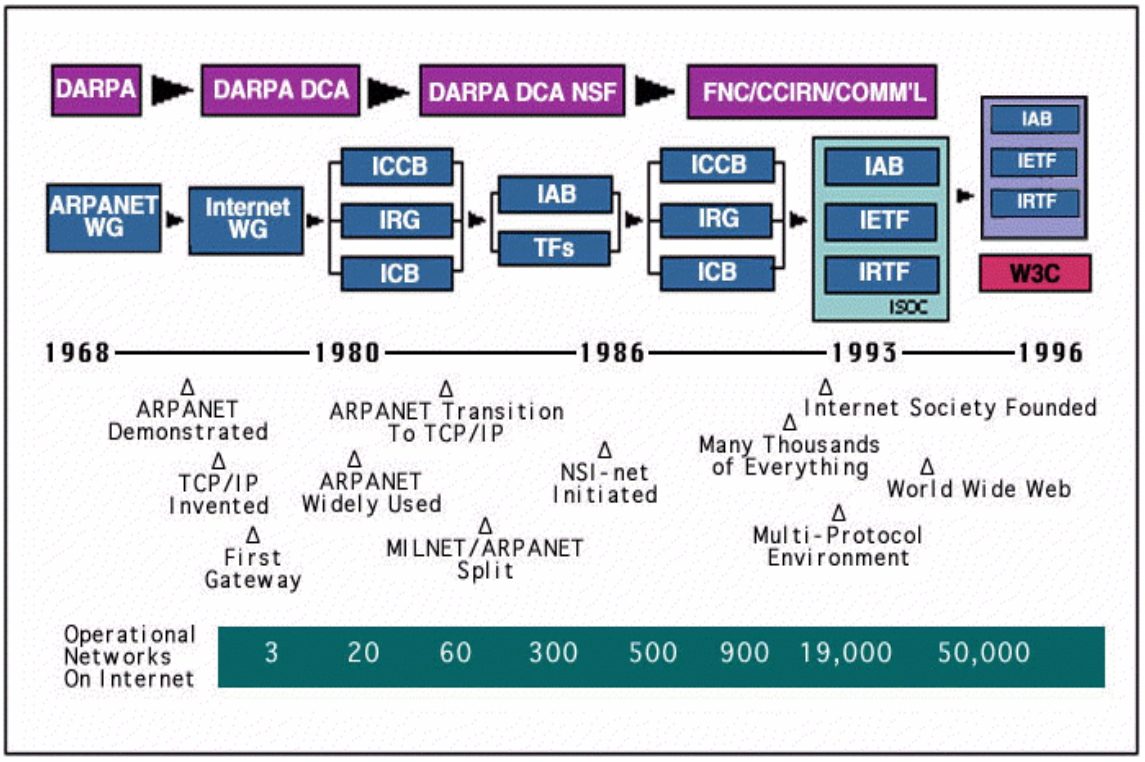
2. Internetin perusarkkitehtuuri ja –käsitteet

Internet on kymmenien tuhansien aliverkkojen muodostama maailmanlaajuinen tietokoneverkko, jonka takia se on houkutteleva vaihtoehto valittaessa uusille sovelluksille verkkoympäristöä. Niinpä siihen liitettyjen koneiden määrällä mitattuna Internet onkin kasvanut huimaa vauhtia. Vuoden -96 alussa verkossa oli yli 7 miljoonaa konetta ja käyttäjiä noin 20-30 miljoonaa. Asiantuntijat arvioivat Internetin käyttäjämäärän kaksinkertaistuvan vuosittain. (Sirola & Linjamaa 1996, 14.)

2.1 Historiaa

Internetin lähtökohtana oli Yhdysvaltojen puolustusvoimille ydinsodan tiedonvälitykseen kehitetty ARPANET-tietoverkko. Koska verkon täytyi toimia hyvin mahdollisen ydinsodan aikana, täytyi verkko tehdä riippumattomaksi yksittäisistä tietokoneista. ARPANET:ssä yksittäiset kiinteät linjat tietokoneiden välillä korvattiin useilla ristikkäisillä yhteyksille, jolloin yhden solmun poistuminen verkosta ei vaikuta muun verkon toimintaan, eli tiedon siirtäminen koneiden välillä oli mahdollista useampaa eri reittiä. Tällaisen verkon toimintaa varten täytyi kehittää myös uudenlainen tiedonsiirtoprotokolla, jolla koneiden lähettämät tiedot täytyi pystyä reitittämään oikeaan osoitteeseen riippumatta tiedon kulkureitistä verkossa. ARPANET:ä varten kehitettiin protokolla, joka pilkkoi siirrettävän tiedon osiin ja lähetti sen paloina linjaa pitkin. Vastaanottavan koneen täytyi osata koota eri reittejä, eri aikoihin tulevat paketit takaisin alkuperäiseksi viestiksi. Nykypäivänäkin Internetin toiminta perustuu samaan

malliin. (Staflin 1996, 11; Järvinen 1995; Leiner, Cerf, Clark, Kahn, Kleinrock, Lynch, Postel, Roberts & Wolff 1997.)



Kuva 2-1 Internetin kehityshistoria (Leiner, Cerf, Clark, Kahn, Kleinrock, Lynch, Postel, Roberts & Wolff, 1997)

2.2 Internetin toimintaperiaate

Internet muodostuu lähiverkoista, jotka koostuvat palvelinkoneista (*Server*) ja asiakaskoneista (*Client*). Lähiverkot kytkeytyvät toisiinsa toistimien, siltojen, reitittimien ja yhdyskäytävien avulla. Toistimen tehtävä on vahvistaa ja toistaa edelleen sen kautta verkossa kulkevat signaalit. Silta liittää yhteen kaksi erillistä verkkoa, jotka käyttävät samaa tekniikkaa. Reititin siirtää tietoja verkkojen

välillä. Jokaisella reitittimellä on oma osoite verkossa. Reitittimen tehtävä on nimensä mukaista kuljettaa tieto oikealle koneelle. Yhdyskäytävän tarkoitus on muuntaa erityyppisten verkkojen välillä siirrettävä tieto käsiteltäväksi siinä verkossa, jonne data on lähetetty. (Sirola & Linjamaa 1996, 19, 22-25.)

Internet toimii asiakas/palvelin-mallin mukaisesti. Asiakas/palvelin-mallissa asiakas- ja palvelin-koneet eivät ole jatkuvasti yhteydessä toisiinsa vaan ne kommunikoivat keskenään ainoastaan sillä hetkellä, kun ne siirtävät toisilleen tietoa. Internetissä palvelinkoneet sisältävät varsinaiset tiedot ja sovellukset kun taas asiakaskoneissa ajetaan käyttöliittymäsovellusta, jonka kautta loppukäyttäjät voivat käyttää Internetin tietoja tai ajaa Internet-sovellusohjelmia. Internetissä asiakas on työasemaassa ajettava asiakasohjelma - selain. (Sirola & Linjamaa 1996, 19-21.)

2.3 World Wide Web

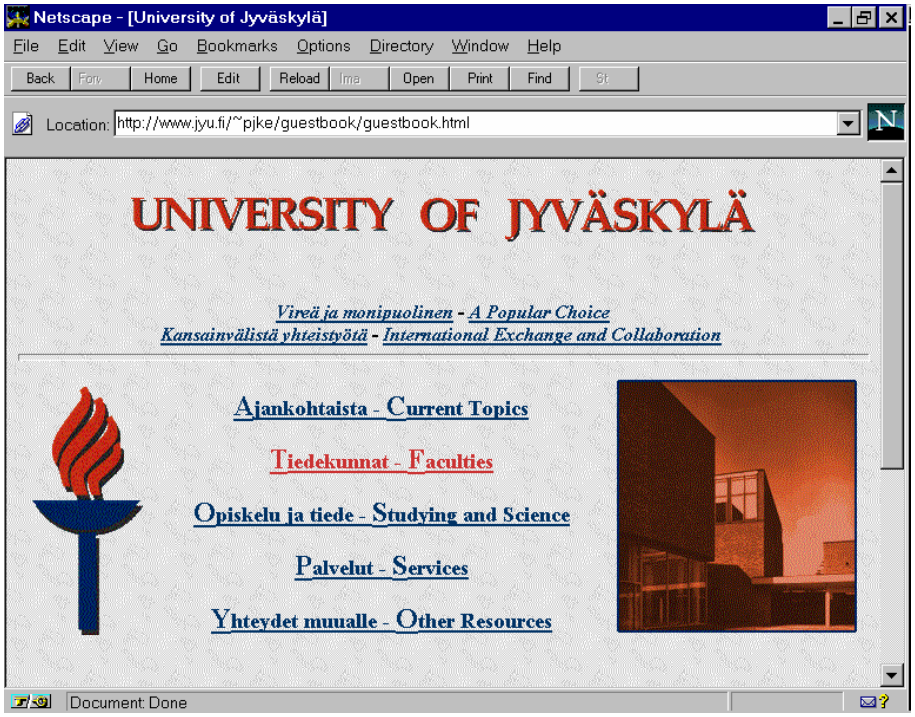
2.3.1 Historiaa

World Wide Web (‘maailmanlaajuinen seitti’, lyh. WWW, Web) sai alkunsa vuonna 1989 CERN:ssä. Aluksi WWW:tä käytettiin hypermediasovellusten testiympäristönä, joka käyttäjien toimesta laajeni nopeasti WWW:ksi. Tämän jälkeen WWW alkoi kehittyä omaa tahtiaan. (Staflin 1996, 14.)

Alunperin Internet oli täysin merkkipohjainen järjestelmä. Nykyisin World Wide Web on myös graafinen ja se sisältää lähes kaikki Internetin perinteiset

ominaisuudet. Internetin voimakas kasvuvauhti liittyy ennen kaikkea juuri World Wide Webiin, jonka etu on graafisen käyttöliittymän tuoma käytön helppous. Internetin kaupallinen hyödyntäminen perustuu nykypäivänä lähes ainoastaan sähköpostin ja WWW:n käyttöön. (Staflin 1996, 14.)

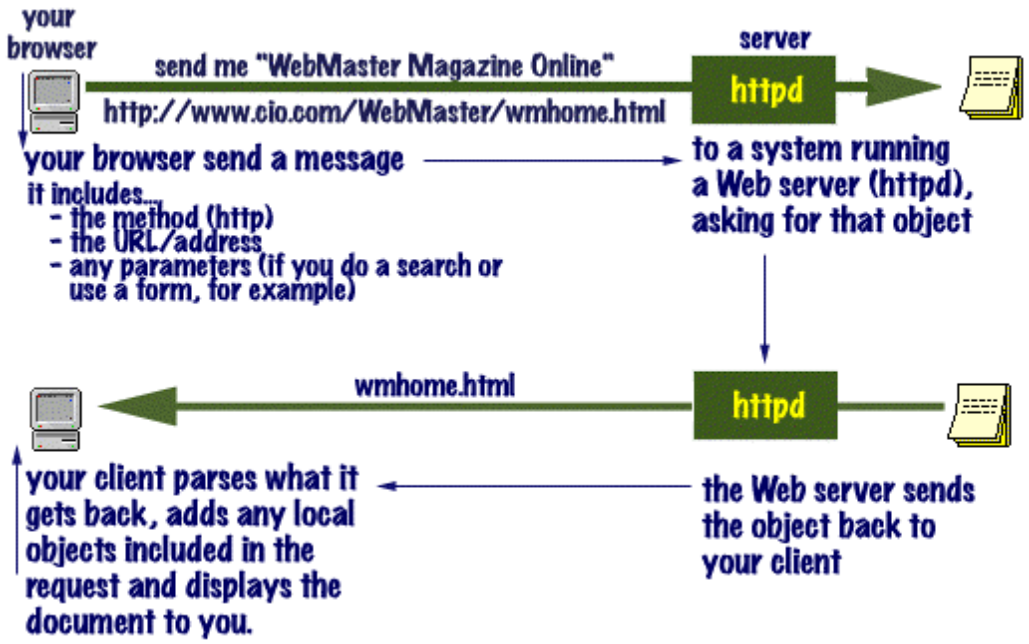
Alkuvaiheessa webbiä saattoi verrata jättimäiseen maailmanlaajuiseen kirjastoon. Kaupalliselta kannalta hyödyntäminen perustui ainoastaan mainostamiseen ja muuhun tiedottamiseen. Nykyisinkin suurin osa WWW-sivuista on staattisia. Näiden sivujen käyttöön ei sisälly minkäänlaista vuorovaikutusta eli käyttäjä ei voi vaikuttaa sivujen sisältöön (Kuva 2-2).



Kuva 2-2 Staattinen WWW-sivu

2.3.2 Yleistä

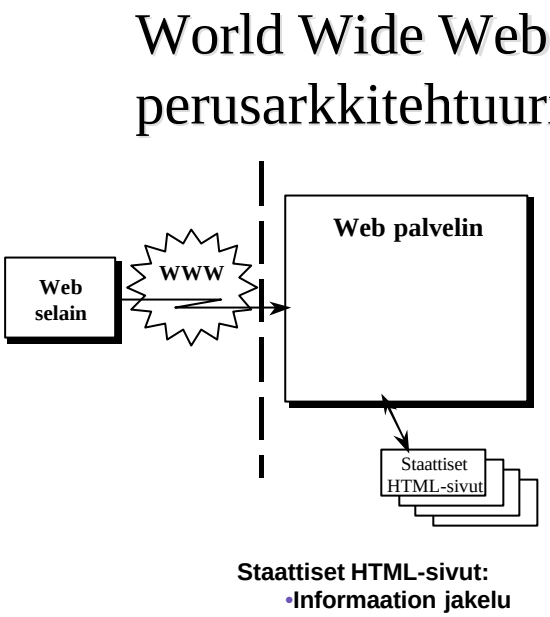
Webin avulla käyttäjät pystyvät käyttämään toisten tietokoneiden tietoja maailmanlaajuisesti (Zeltser 1995). Toiminta tapahtuu siten, että selain - HTML-sivujen katseluohjelma - ottaa yhteyttä palvelimella olevaan (www-)palvelinohjelmaan lähettämällä sille pyynnön halutusta URL-osoitteesta. Tämän jälkeen palvelin palauttaa selaimelle pyydetyn HTML-dokumentin, jonka sisällön pohjalta selainohjelmisto tulkkaa näytettävän HTML-sivun (Kuva 2-3).



Kuva 2-3 Web-kysely (WebMaster Magazine 1996)

Yhteydet työaseman ja palvelimen välillä perustuvat HTTP-protokollaan (HyperText Transfer Protocol). Hypertekstiasiakirjojen sijainti määräytyy yksikäsitteisen URL-osoitteen avulla.

Alussa World Wide Web (Kuva 2-4) koostui ainoastaan staattisista HTML-sivuista, joita käyttäjät saattoivat siis ainoastaan lukea tai joista käyttäjät saattoivat hypätä toisille HTML-sivuille seuraamalla hyperlinkkiä.



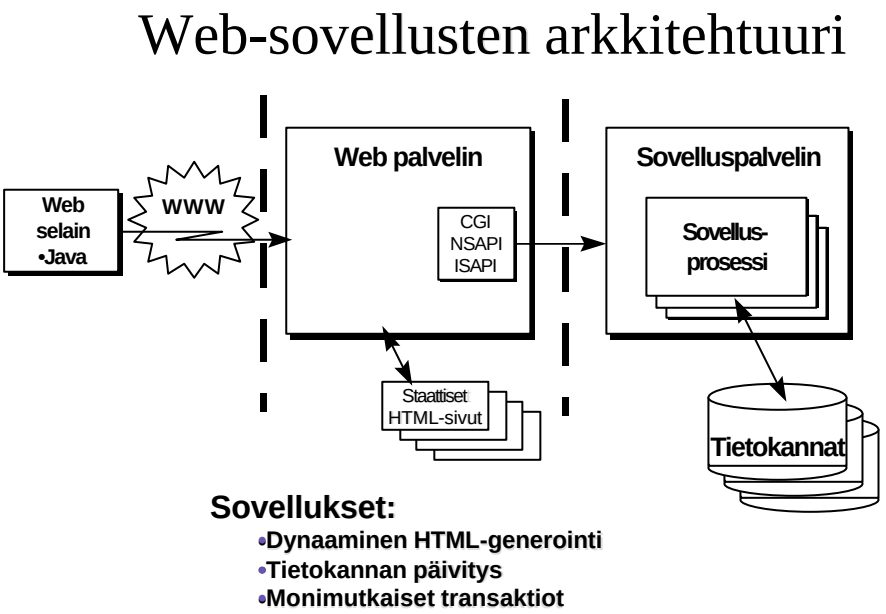
Kuva 2-4 World Wide Web -perusarkkitehtuuri (Progress Software Oy 1997)

2.3.3 Sovellukset

Webin kaupallinen hyödyntäminen vaatii vuorovaikutteisten sovellusten tekemistä. Interaktiivisuus mahdollistui, kun HTML-kieleen kehitettiin vuorovaikutusta tukevia ominaisuuksia, kuten FORM-elementit. WWW-palvelimiin taas kehitettiin yhdyskäytäviä (*gateway*), joiden välityksellä pystyttiin siirtämään tietoa WWW-palvelimilta sovelluksille ja päinvastoin. Web-palvelimien kanssa käytettäviä yhdyskäytäviä ovat CGI, NSAPI ja ISAPI. Näihin

tekniikkoihin perustuvat sovellukset suoritetaan palvelimessa, joka palauttaa selainohjelmalle ainoastaan HTML-dokumentin. Selaimissa suoritettavien tekniikoiden lisäksi voidaan nykyisin myös käyttää selaimessa ajettavia ohjelmia. Tällaisia tekniikoita ovat erilaiset skriptikielet, plug-init sekä Java (kts. Käsitteitä). Useat nykyisin käytössä olevat WWW-sovellukset yhdistävät sekä palvelimessa että selaimessa suoritettavia tekniikoita.

Kaupalliselta ja ylläpidon kannalta katsottuna WWW-sivujen sisältö täytyy muodostaa ajonaikaisesti (raportit, hintatiedot). Useimmiten sivujen sisältö muodostetaan tietokannoissa olevasta datasta. Kuva 2-5 näyttää WWW:n sovellusympäristön.



Kuva 2-5 Web-sovellusten arkkitehtuuri (Progress Software Oy 1997)

3. WWW-projekti

WWW-projekti koostuu kaikista työvaiheista sovelluksen suunnittelusta sen toteutukseen. Projektissa tulisi myös ottaa huomioon jatkossa tapahtuva ylläpito, mikä on tyypillisesti kaikista kallein vaihe. WWW-projekti muodostuu useista eri kokonaisuuksista, kuten WWW-pohjaisen verkkoratkaisun valinnasta, sovellusarkkitehtuuri, ohjelmointitekniikasta ja työväline. CGI-pohjaisen WWW-projektin ohjelmoinnissa voidaan lisäksi erotella palvelimessa tehtävän sovelluksen ohjelmointi ja HTML-pohjaisen (tai jonkin muun *ML-pohjaisen) käyttöliittymän tekeminen.

3.1 Projektin valmistelu

Laajoihin WWW-projekteihin kuuluu monien eri alojen asiantuntijoita, kuten HTML-sivun tekijä, käsikirjoittaja eli sisällön tekijä, ohjelmoija, graafikko ja erilaisia multimedia asiantuntijoita sekä yhteensovittamisesta vastaava projektipäällikkö. WWW-projektin suunnittelussa täytyy ottaa huomioon:

- kohderyhmä
- projektiryhmä
- aikataulu
- valmiin aineiston määrä: kuvat, äänet, tekstit, tietokanta, ohjelmat, yms.
- grafiikan määrä
- sovelluksen interaktiivisuus ja/tai logiikka
- ulkoiset linkit

- ulkoasu, laatuvaatimukset, yrityksen logot ym.
- tietoturva
- copyright-oikeudet
- esityksen rakenne
- ylläpito

(Sirola & Linjamaa 1996, 110).

3.2 WWW-sovelluksia

WWW-sovelluksien suunnittelussa tulee ottaa kaksi tekijää huomioon karkealla tasolla: sovelluksen vaativuus ja käyttäjät. Mitä varten WWW-sovellusta tehdään, vaikuttaa paljolti siihen, millä tavalla ja millaisella välineellä sovellus kannattaa tehdä. Välineen valinnassa tulee pohtia myös pitemmän tähtäimen suunnitelmia webin hyödyntämisessä, jotta vältyttäisiin uuteen sovelluskehitysvälineeseen – tai jopa ohjelmointikieleen - siirtymisestä aiheutuvista kustannuksista. Nykyiset WWW-sovellukset voivat olla hyvinkin erityyppisiä, osa on pelkkää tiedottamista ja osa saattaa olla vaativien tietokantatransaktioiden käsittelyä. Tiedottamiseen tehtävät sovellukset saadaan usein tehtyä lähes täysin ilman ohjelmointia käytettäessä soveltuvuutta kehitintä. Tietokantaa päivittävät sovellukset taas vaativat usein monimutkaistakin ohjelmointia.

WWW-sovelluksiin vaikuttavat olennaisesti samanaikaisten käyttäjien määrä, käyttäjien profiili (yrityksen omia työntekijöitä – kuka tahansa), sovelluksen monimutkaisuus, verkkoliikenteen määrä ja verkkoratkaisun avoimuus, johon

liittyy tietoturva. WWW-sovellukset voidaan avoimuuden mukaan jaotella Intra-, Extra- tai Internet-sovelluksiin, joista Internet-sovellukset ovat usein vaativimpia mahdollisesti suuren käyttäjämäärän ja tietoturvan takia. (Hurwitz & Ashton 1998.)

Yksinkertaisinta WWW:n hyödyntämistä on staattisiin HTML-dokumentteihin perustuva tiedotus, johon WWW soveltuukin hyvin, koska se tarjoaa valmiin maailmanlaajuisen kanavan. Kun dokumenttien määrä kasvaa, WWW-sovelluksiin tarvitaan kuitenkin ennen kaikkea ylläpidettävyyden takia lisää automaatiota. Sen sijaan, että tehdään useita erillisiä staattisia HTML-dokumentteja, tehdään niiden tulostaminen tietokannassa olevan tiedon pohjalta dynaamisesti, joka tehdään palvelimessa ajettavilla ohjelmilla. Staattisten dokumenttien käyttöä rajoittaa ainoastaan WWW-palvelimen kyky käsitellä palvelupyyntöjä, kun taas dynaamisia dokumentteja käytettäessä kuormitusta lisäävät myös sovelluslogiikan suorittaminen ja sen ajamiseen käytettävä prosessi. Tietokantaa käyttävissä sovelluksissa rajoittavaksi tekijäksi saattaa muodostua myös itse tietokannan suorituskyky. Nykyisin staattisina HTML-dokumentteina tehdään useimmiten organisaatioiden muuttumatonta tai harvoin muuttuvaa tietoa sisältävät dokumentit. Sen sijaan, esimerkiksi tuotekatalogit tehdään dynaamisesti tietokannasta, jossa tiedot saattavat muuttua päivittäisen toiminnan yhteydessä. (Hurwitz & Ashton 1998.)

Tulevaisuudessa nähdään mielenkiintoisena mahdollisuutena yritykset, jotka toimivat ensisijaisesti verkon kautta (Hurwitz & Ashton 1998). Webissä oleva

kauppa voi olla auki 24 tuntia joka päivä ja mahdolliset asiakkaat ovat kaikki ihmiset, joilla on WWW-yhteys. Asiakkaat taas hyötyvät saamalla tuotteista välittömästi tietoa, joutumatta odottamaan myyjän vapautumista. (Stanek 1996, 35.) Kirjakauppa *www.amazon.com* on hyvä esimerkki verkkokaupasta. Tällaisesta kirjakaupasta asiakas voi tilata lähes minkä kirjan tahansa, koska näytekirjat ovat tutkittavissa sähköisinä dokumentteina ja siten niiden näytteille asettaminen ei vaadi fyysistä tilaa. Kirjat tilataan ainoastaan, kun ne toimitetaan asiakkaille. Verkkokaupan suurimpia ongelmia tällä hetkellä on turvallisen ja helppokäyttöisen maksumenetelmän kehittäminen.

3.3 WWW-pohjaiset verkkoratkaisut

WWW:ssä käytettävä verkkoratkaisu voi olla joko Intra-, Extra-, tai Internet. Valintaan vaikuttavia tekijöitä voivat olla mm. tietoturva ja verkkoliikenteen määrä. Tarkastellaan seuraavaksi eri WWW-verkkoratkaisuita.

Intranet on yrityksen sisäinen WWW-tekniikkaan perustuva verkko, joka voi olla rakennettu Internetiin, jolloin se suljetaan ulkopuolisilta käyttäjiltä erilaisilla tieturvaratkaisuilla. Erityisesti kansainvälisille yrityksille Internetiin toteutetun Intranetin merkittävä etu on sovellusten saavutettavuus yrityksen sisäisen verkon ulkopuolelta (Järvinen 1996, 31), koska yhä kansainvälistyvässä maailmassa myös yritysten tietojärjestelmien tulee olla käytettävissä mistä päin maailmaa tahansa.

Yrityksen sisäisten järjestelmien toteuttaminen Intranetillä vähentää sovellusten ylläpidosta johtuvia kustannuksia, koska WWW-sovellukset asennetaan ja ylläpidetään keskitetysti. Intranet onkin usein yritysten ensimmäinen sovellusalue WWW:n hyödyntämisessä, koska niiden toteuttaminen on mm. tietoturvan suhteen suhteellisen helppoa. Usein ne ovat ainoastaan laajennus jo olemassa oleviin järjestelmiin - WWW-käyttöliittymä tietokantaan. Esimerkkejä erilaisista Intranet-sovellutuksia ovat mm.:

- yrityksen sisäiset raportit
- tuotannonohjaus, taloushallinta (tilaukset, varasto).

Extranet on tietyillä yrityksillä/organisaatioilla laajennettu Intranet (Järvinen 1996, 32). Nimetyt asiakkaat voivat käyttää suoraan yrityksen tietokantaa tai sen osia - esimerkiksi suurimmat asiakkaat voivat syöttää tilauksensa suoraan yrityksen tietokantaan, mikä säästää tilausten vastaanotosta aiheutuvia kustannuksia sekä parantaa palvelun laatua mm. nopeuttamalla tilausten käsittelyä. Asiakkaat voivat myös halutessaan seurata reaaliaikaisesti tilauksensa etenemistä. Extranet on usein yrityksellä seuraava vaihe Intranetin rakentamisen jälkeen, erityisesti tehtaiden ja tukkuliikkeiden järjestelmissä. Extranetin käyttäjät voivat olla sekä yrityksen työntekijöitä, asiakkaita että toimittajia, jolloin voidaan myös automatisoida yrityksen ulkoisten tekijöiden kanssa tapahtuvaa liiketoimintaa, yritysten välinen tiedonsiirto (business-to-business). Extranetin suuri etu Internet-kauppapaikkoihin verrattuna on hallittavissa oleva käyttäjäkunta, jolloin sekä käyttäjien että palveluntarjoajan välillä vallitsee luottamus, mikä helpottaa maksujärjestelmien ja

tietoturvaratkaisuiden toteuttamista. Toisaalta Extranettinä toteutetussa WWW-palvelussa täytyy olla hyvä tietoturva, jolla estetään ulkopuolisten luvaton käyttö (Hurwitz & Ashton 1998).

Internet-sovellukset poikkeavat ennen kaikkea käyttäjien suhteen merkittävästi Intra- ja Extranet-sovelluksista. Internet-sovellukset ovat julkisia, eli kuka tahansa voi käyttää palvelua. (Hurwitz & Ashton 1998.) Internet-sovellusten täytyy myös pystyä palvelemaan (ennakoimattomankin) suurta käyttäjämäärää ja niiden ajoympäristö täytyy olla helposti laajennettavissa, mikäli suorituskykyä täytyy tehostaa. Suuri käyttäjämäärä tuo myös nopeasti näkyviin tehottomasti tietokantaa käyttävät ohjelman osat, eli ohjelmointiin kohdistuu myös suuremmat vaatimukset - erittäin hyvin tehtyä tietokantaa, tehokasta tietokannan käyttölogiikkaa sekä nopeaa tietokantayhteyttä.

Internet-sovellusten käyttöön liittyy myös merkittäviä tietoturvakysymyksiä sekä palvelun tarjoajien että käyttäjien kannalta. Palvelun tarjoajien kannalta riskejä aiheuttavat järjestelmän suljettuihin osiin murtautumisen mahdollisuus sekä palvelun väärinkäytöstä aiheutuvat kustannukset. Käyttäjän kannalta taas riskejä ovat yksilön tietosuoja ja palvelun tarjoajan luotettavuus, erityisesti mikäli palvelun käyttö edellyttää maksua.

3.4 WWW-ohjelmointitekniikat

WWW-sovellukset voidaan jaotella selaimessa tai palvelimessa ajettaviin sovelluksiin. Usein suuret WWW-sovellukset koostuvat molemmista. WWW-

ajoympäristön yksi merkittävä etu on, että se mahdollistaa sovellusten keskitetyn ylläpidon riippumatta siitä tehdäänkö sovelluksesta selaimessa tai palvelimessa ajettava – tosin palvelinsovellus on kuitenkin vielä helpommin ylläpidettävissä, koska selaimissa ajettavissa sovelluksissa täytyy huomioida mahdolliset selainkohtaiset toiminnat, jotka voivat heikentää sovelluksen yleistä (Inter-/Extranet) käytettävyyttä. Selainsovellukset ovat parempia, kun halutaan tehdä ohjelmia, joiden suoritus on verkkoliikenteen kannalta järkevää tehdä työasemassa. Selaimeen sovelluslogiikkaa tehdään lähinnä JavaScriptillä, VBScriptillä, Javalla sekä Plug-In- ja ActiveX-tekniikoiden avulla. Kaikkien käyttöön tehtävien sovellusten tulisi perustua mahdollisimman pitkälle standarditekniikoihin, kuten HTML:ään, jolloin sovellus on selainriippumaton, joka on erityisesti Internet-sovelluksissa tärkeää. Selainriippumattomat tietokannan käyttöön perustuvat sovellukset on usein hyvä tehdä palvelinsovelluksina, jolloin myös tietokannan käsittely tehdään verkkoliikenteen kannalta parhaassa paikassa. Sovellus käyttää selainta ainoastaan datan esittämisessä ja lähettämisessä. Palvelinsovellukset tehdään usein CGI-tekniikkaan pohjautuvilla menetelminä, joissa ohjelmointikieli on varsin pitkälti vapaasti valittavasti.

Selainohjelmia voidaan tehdä skriptikielillä, jotka toimivat siten, että ne upotetaan HTML-dokumentin sisään (Esimerkki 1) (Stanek 1996, 383).

```
<SCRIPT LANGUAGE="JavaScript">
        <!--// Submit the form.
            function SubmitForm (todo) {
                var f = document.FileList;
```

```
                // Is a file selected?
                if ((f.Filename == null) || f.Filename.selectedIndex == -1)
                    window.alert('Please select a file.');
                // Confirm deletion?
                else if (todo != "Delete" || window.confirm("Are you sure you want to delete the selected files?")) {
                    f.FileAction.value = todo;
                    f.submit();
                }
            }
            // End of Script -->
</SCRIPT>.
```

Esimerkki 1.

Skriptikielillä tehdään esimerkiksi käyttäjän syöttötiedon tarkistavia ohjelmia, esimerkiksi haluaako käyttäjä varmasti tuhota tietueen tietokannasta. Palvelinsovelluksessa tämä täytyisi toteuttaa siten, että lähetettäisiin palvelimeen pyyntö ohjelman suorittamiseksi, joka lähettäisi asiakaskoneeseen vastauksena käyttäjältä varmistuksen kysyvän HTML-dokumentin. Vastauksen jälkeen ajettaisiin vielä erikseen ohjelma, joka joko tuhoaisi tietueen tai peruuttaisi päivityksen. Palvelinsovellukset ovatkin harvoin "puhtaita" vaan niihin liittyy usein selaimissakin ajettavia ohjelmia. Java-ohjelmointikieli on myös yksi merkittävä vaihtoehto sovellusten ohjelmointikieleksi WWW-projektissa. Java on laitteisto- ja käyttöliittymäriippumaton ja sillä voidaan toteuttaa sekä palvelin- että selainsovelluksia. Javan yksi vahvuus on, että sillä toteutetuissa WWW-sovelluksissa ohjelmoijan ei tarvitse opetella eri ohjelmointikieliä selainta ja

palvelinta varten. Se tarjoaa myös käyttöliittymän tekemisessä vaihtoehdon muotoilukielille.

Web-ohjelmointi menee yhä enemmän asiakas/palvelin-arkkitehtuuriin, jossa pystytään yhdistämään sekä palvelin- että selainpohjaisten sovellusten hyvät puolet. Tietokantaa käyttävä sovelluslogiikasta – liiketoimintalogiikka - suoritetaan palvelimelle ja käyttöliittymälogiikka selaimessa. Käytännössä suurin osa internetiin tehtävistä tietokantasovelluksista on osittain toteutettu tällä tavalla, erot ovat lähinnä painotuksissa.

Sovelluksen arkkitehtuuri riippuu pitkälti tarvittavasta sovelluksesta: mikäli sovelluksen pääasiassa katsellaan tietokannan tietoja, on palvelinsovellus hyvä ratkaisu, mikäli taas sovelluksessa suoritetaan monimutkaista logiikkaa, jossa myös päivitetään tietokannan tietoja, on asiakas/palvelin-sovellus hyvä vaihtoehto, mikäli taas sovellus tarvitsee palvelimelta suhteellisen vähän tietoa, soveltuu selainsovellus hyvin ratkaisuksi. Tietokannan käsittely tulisi useimmiten pyrkiä saamaan palvelimeen, koska tietoliikenne työaseman ja palvelimen välillä kannattaa pyrkiä minimoimaan - esimerkiksi kaikkien tuotteiden hinnan korotuksen tekevää sovellusta ei kannata ajaa asiakaskoneessa, koska se aiheuttaisi runsaasti turhaa verkkoliikennettä.

3.5 Käyttöliittymän tekeminen HTML-kielellä

Erityisesti palvelinpohjaisten sovellusten käyttöliittymä toteutetaan usein HTML-muotoilukielellä, jonka tärkeimpiä ominaisuuksia on selainriippumattomuus.

Selaimia on olemassa lähes kaikkiin koneisiin ja käyttöjärjestelmiin. HTML:n vahvuuksiin on luettava myös sen helppokäyttöisyys, esimerkiksi HTML-kielen syntaksi on varsin yksinkertaista aloittelijankin oppia. HTML-kielessä olennaista on myös eri versioiden yhteensopivuus, joka on välttämätön edellytys WWW-sovellusten käytettävyydellä. (Sirola, Linjamaa 1996, 214-215.)

HTML-dokumentti koostuu leipätekstistä ja tekstin sekaan upotetuista komennoista eli tageista. Tagit ovat aina ’<’ ja ’>’ -merkkien välissä, muu teksti on leipätekstiä (Sirola, Linjamaa 1996, 250). Dynaamisesti uusia HTML-dokumentteja tulostaviin ohjelmiin täytyy myös tagit tulostaa ohjelmallisesti. Ohjelmoijan kannalta tärkeitä HTML-tageja ovat ne, joilla saadaan aikaan toiminnallisuutta käyttäjän toimenpiteiden mukaan. Seuraavissa kappaleissa käsitellään tällaisia tageja.

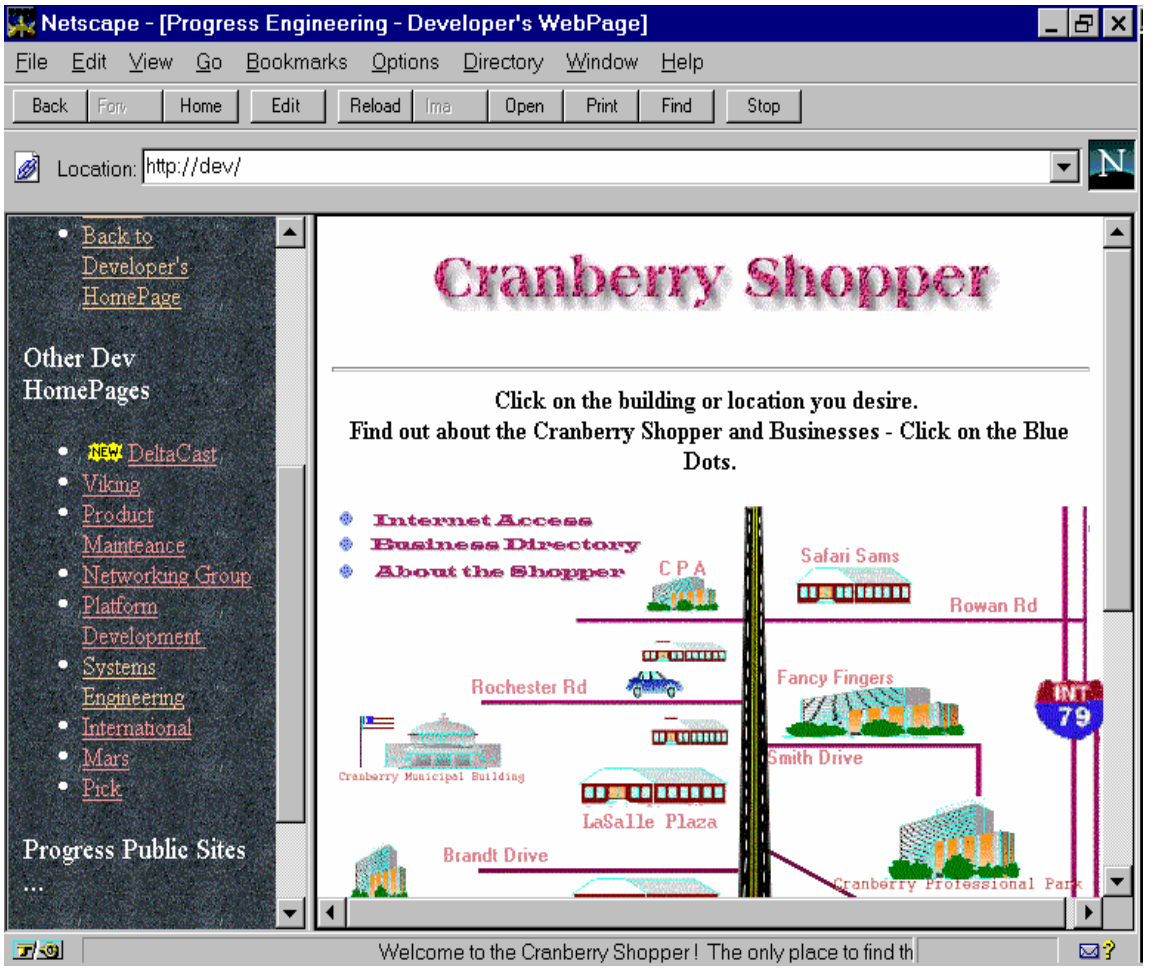
3.5.1 Linkit (<A>)

WWW-dokumentit voidaan liittää toisiinsa hypertekstilinkkien avulla. Linkkien avulla saadaan aikaiseksi vuorovaikutusta käyttäjien ja HTML-dokumenttien välille antamalla käyttäjän valita järjestyksen, jonka mukaan haluaa katsella dokumentteja tai dokumentin eri osia. Selain lähettää WWW-palvelimelle pyynnön linkissä määritellyn URL-osoitteen sisältämän dokumentin saamiseksi. URL-osoitteen mukana voidaan myös lähettää parametrejä, esimerkiksi seuraavasti CGI-ohjelmalle

http://www.webserver.fi/cgi-bin/bible.cgi/read.p?v_Rowid=0x000185e3, jossa CGI-ohjelmalle *read.p* viedään parametri *v_Rowid*, jonka arvo on *0x000185e3*.

3.5.2 Kehykset (<FRAMESET> ja <FRAME>)

Kehyksien avulla HTML-dokumentti voidaan jakaa osiin ,jolloin selainikkuna muodostuu erillisistä, itsenäisistä osista. Jokainen kehys voi sisältää eri HTML-dokumentin tai WWW-ohjelman. Kehyksien avulla voidaan tehdä käyttöliittymiä, joissa on esimerkiksi yhdessä kehyksessä jatkuvasti näkyvissä staattinen valikkorivi ja toisessa muuttuva sisältö (Kuva 3-1). Kuvassa oikeanpuoleisen kehyksen sisältöä voidaan muuttaa joko vasemmanpuoleisessa kehyksessä olevista linkeistä tai oikeanpuoleisessa dokumentissa itsessään olevista linkeistä. (Sirola, Linjamaa 1996, 283.)



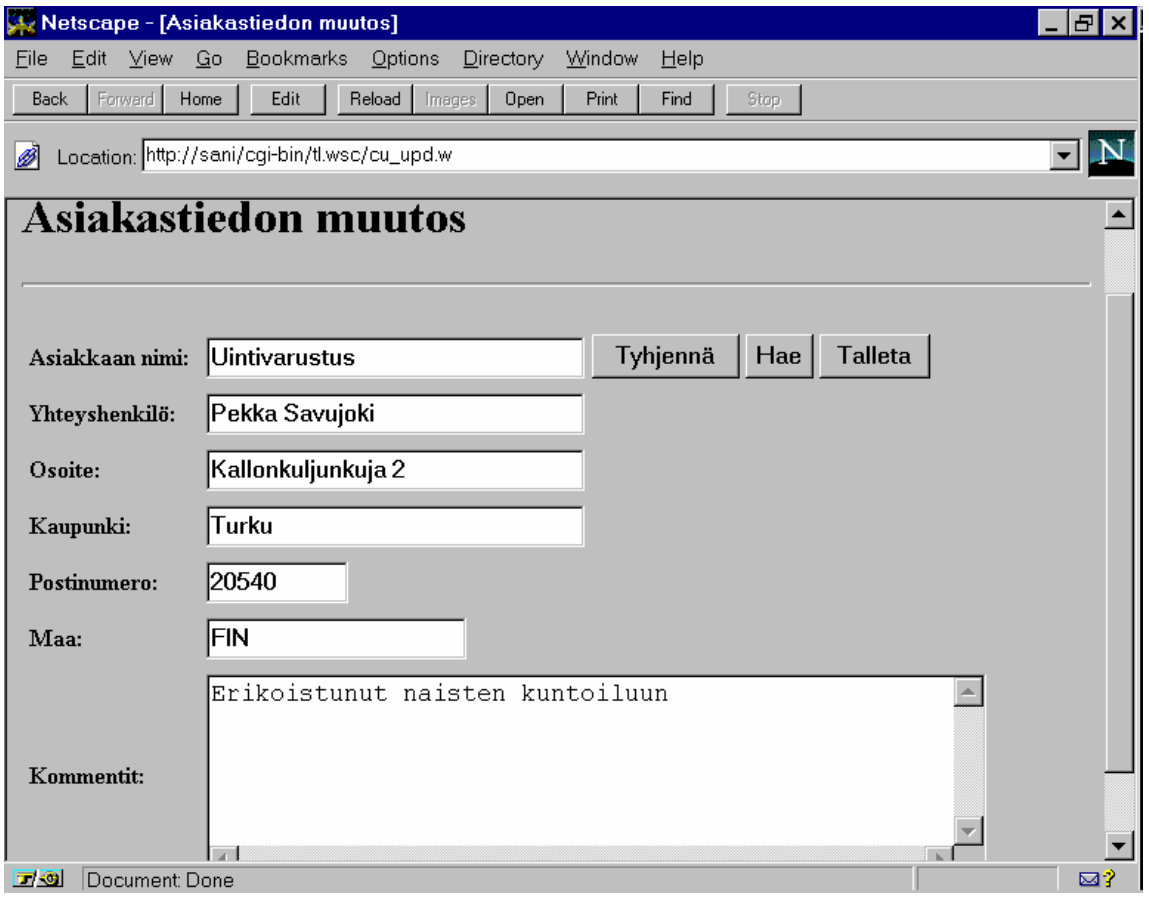
Kuva 3-1 Kehyksien avulla toteutettu HTML-dokumentti

Target-parametri

Joskus ohjelmoijan kannalta on hyödyllistä pitää useita ikkunoita auki samanaikaisesti. Target-parametrilla voidaan määrätä haluttaessa mihin ikkunaan käyttäjän valitseman URL:n osoittama dokumentti avataan. Target-parametri voidaan sisällyttää useisiin eri tageihin. (Sirola & Linjamaa 1996, 390-392.)

3.5.3 Lomake (<FORM>)

Lomakkeet ovat HTML-dokumenttien osia, jotka sisältävät graafisissa käyttöliittymissä tuttuja vuorovaikutusvälineitä, kuten nappuloita, tekstikenttiä ja valintaruutuja (Stahlin 1996, 14). Kuva 3-2 näyttää HTML-dokumentin, joka on toteutettu käyttäen lomaketta.



Kuva 3-2 Lomakkeena toteutettu HTML-dokumentti

Lomake on vuorovaikutteisen WWW-sovelluksen peruselementti. HTML-lomake koostuu toiminnallisesti kahdesta eri osasta: käyttäjälle käyttöliittymässä näkyvä lomake ja ohjelma tai skripti, joka suorittaa syöttötietoja käsittelevän

sovelluksen. Ohjelma voi olla palvelimella toimiva CGI-sovellus, dokumenttiin upotettu JavaScript, Java-applet tai jokin muu vastaava. (Sirola & Linjamaa 1996, 412.)

Lomake määritellään <*FORM*>- ja </*FORM*>-tageilla, joiden välissä olevat määrittelyt objektit kuuluvat lomakkeeseen. Lomakkeen objektit voivat olla tekstikenttiä (text), salasanakenttiä (password), valintaruutuja (checkbox), radionappula (radio), nappula (submit), piilotettuja kenttiä (hidden), editori (textarea), valintalista (select) sekä myös omia painikkeita (image), kuten kuvaa napsauttamalla käynnistettävät toiminnot.

FORM-tagiin kuuluu kaksi tärkeää määrettä: ACTION ja METHOD, jotka määräävät lomakkeen lähettämisessä käytettävän menetelmän ja tiedon käsittelyssä käytettävän ohjelman URL-osoitteen. Lomakkeen lähettämiseen on kaksi metodia: GET ja POST. GET-metodia käytetään CGI-sovelluksissa yleensä käynnistettäessä ohjelma ensimmäistä kertaa, jolloin sillä tehdään usein käyttöliittymän alustus. POST-metodi lähettää lomakkeeseen syötetyt tiedot URL-pyynnön jälkeen. Kun palvelin saa palvelupyynnön lomakkeen POST-metodilla lähetettynä, jää WWW-palvelin vielä odottamaan erikseen tulevia lomakkeeseen syötettyjä tietoja, jotka lähetetään palvelimelle FORM-elementin nimestä ja sen arvosta koostuvina pareina. WWW-palvelin siirtää saadut tiedot edelleen CGI-rajapintaa pitkin sovellukselle. POST-metodi on lomakkeiden käytössä suositeltava metodi. (Sirola & Linjamaa 1996, 412, 414, 415.)

Esimerkki

Esimerkissä 2 on HTML-koodi (kts. Kuva 3-2) FORM:n määrittelevästä koodista. Siinä määritellään POST-metodia käyttävä lomake, jonka lähettämät tiedot käsittelee ohjelma *cu_upd.w*. Kun annetaan ainoastaan ohjelman nimi, ajetaan ko. ohjelma samasta hakemistosta kuin nykyisen HTML-dokumentin tulostanut ohjelma. INPUT TYPE=”TEXT” -tageilla määritellään FORM:lle tekstikenttiä, josta on huomioitava, että ne määritellään FORM- ja /FORM-tagien välissä. INPUT TYPE=”SUBMIT” taas määrittelee painonapit, joiden jokaisen painallus aiheuttaa lomakkeen lähettämisen *cu_upd.w* -ohjelman käsiteltäväksi. Piilokentän (INPUT TYPE=”HIDDEN”) avulla viedään asiakasnumero parametrinä *cu_upd.w* -ohjelmalle.

```
<FORM METHOD="POST" ACTION="cu_upd.w">
          <B>Asiakkaan nimi: </B>
          <INPUT  TYPE="TEXT"  NAME="Name"  SIZE="30"  MAXLENGTH="30"
          VALUE="Uintivarustus">
          <INPUT TYPE="SUBMIT" NAME="Button" VALUE="Tyhjennä">
          <INPUT TYPE="SUBMIT" NAME="Button" VALUE="Hae">
          <INPUT TYPE="SUBMIT" NAME="Button" VALUE="Talleta">
          <B>Yhteyshenkil&ouml;:</B>
          <INPUT      TYPE="TEXT"      NAME="contact"      SIZE="30"
          MAXLENGTH="30" VALUE="Pekka Savujoki">
          <B>Kommentit: </B>
          <TEXTAREA NAME="Comments" ROWS="5" COLS="50">Erikoistunut
          naisten kuntoiluun</TEXTAREA>
          <INPUT TYPE="HIDDEN" NAME="cust-num" VALUE="0">
```

Esimerkki 2 HTML FORM-elementit

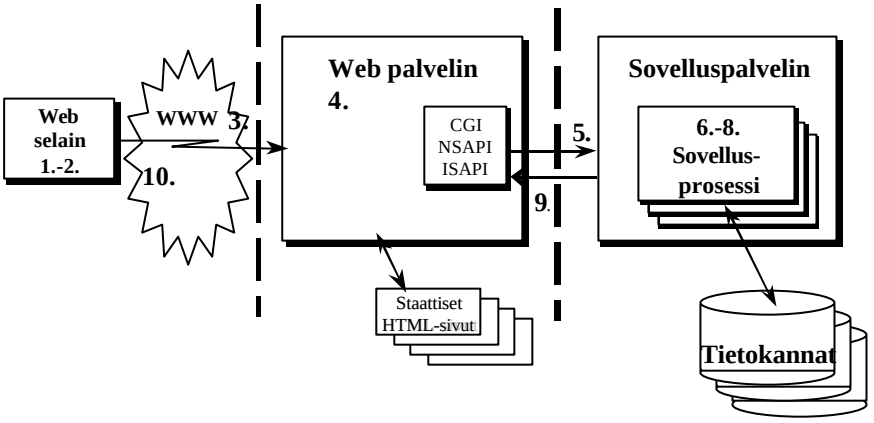
4. CGI-ohjelma

Palvelimessa ajettavat ohjelmat toteutetaan usein WWW-palvelimen CGI-rajapinnan avulla (Kuva 4-1). Käydään CGI-tapahtuma vaiheittain läpi:

1. Käyttäjä painaa HTML-dokumentista lähetä-nappulaa (data on HTML:n FORM-elementeissä). Sen seurauksena selain kerää FORM-elementtien sisältämät tiedot ja lähettää ne palvelimelle yhtenä merkkijonona, joka koostuu name/value –pareista erotettuna &-merkillä.
2. Selain käynnistää HTTP:n POST-metodin, jolloin FORM-elementtien data (merkkijono, kts. edellinen kohta) lähetetään palvelimelle HTTP:n runko-osassa.
3. HTTP-palvelin (WWW-palvelin) vastaanottaa viestin, josta selviää käytetty metodi sekä CGI-ohjelman nimi.
4. HTTP-palvelin asettaa ympäristömuuttujansa, joita käytetään HTTP-palvelimen ja CGI-ohjelman välisessä tiedonvaihdossa.
5. HTTP-palvelin käynnistää CGI-ohjelman, joka on nimetty URL:ssa.
6. CGI-ohjelma lukee palvelimen ympäristömuuttujissa olevat tiedot, josta ilmenee mm. käytettävä metodi.
7. CGI-ohjelma saa selaimelta lähetetyn runko-osan palvelimen STDIN:stä (Standar Input Pipe), joka sisältää FORM-elementtien datan merkkijonona.
8. CGI-ohjelma ajetaan ja se tekee ajon tuloksena HTML-dokumentin tai jonkin toisen hyväksyttävän MIME-tyypin tulostuksen.

9. CGI-ohjelma palauttaa tuloksen HTTP-palvelimelle käyttäen STDOUT:ia (Standard Output Pipe). Tämä päättää samalla CGI-ohjelman suorituksen.
10. HTTP-palvelin palauttaa tuloksen selaimelle, jolloin WWW-transaktio päättyy.

(Orfali, Harkey, Edwards 1999, 587-589.)



Kuva 4-1 CGI-ohjelman toimintaperiaate

4.1 HTTP-protokolla

CGI-ohjelmien toiminnan kannalta yksi keskeisimmistä tekniikoista on HTTP (HyperText Transfer Protocol), joka on tilaton hypertekstin siirtoa verkossa määrittelevä yhteyskäytäntö. HTTP-protokollan tapahtuma muodostuu neljästä osasta:

1. Yhteyden muodostus
2. Palvelupyyntö

3. Vastaus
4. Yhteyden sulkeminen

(Sirola & Linjamaa 1996, 51.)

Jokainen pyyntö ja vastaus koostuu kolmesta osasta: pyyntö tai vastausrivi, otsikko-osa ja dataosa. Selaimesta WWW-palvelimelle lähtevä pyyntö käsitellään seuraavasti:

1. Selainohjelma ottaa yhteyden palvelimen tiedonsiirtoporttiin, jonka jälkeen se lähettää palvelimelle käytettävän metodin, dokumentin URI:n ja HTTP:n version. Esimerkiksi:

   ```
   GET /scripts/wsisa.dll/Wservice=ws/cu_upd.w HTTP/1.0.
   ```

2. Seuraavaksi selainohjelma lähettää otsikkotiedot, joissa on tietoa selaimen asetuksista ja sen hyväksymistä dokumenttityypeistä.
3. Lopuksi selainohjelma voi lähettää muuta tietoa, joka useimmiten on POST-metodilla lähetettävää lomakkeen tietoa.

(Spainbour & Quercia 1997,153-154.)

Kun WWW-palvelin on saanut selaimelta kaikki tiedot, siirtyy käsittely WWW-palvelimen CGI-rajapinnan kautta ohjelmalle, joka lopuksi palauttaa tuloksensa WWW-palvelimelle. HTTP-transaktio päättyy, kun WWW-palvelin lähettää selainohjelmalle vastauksen seuraavasti:

1. Ensimmäiseksi selaimelle lähtee tilarivi, jossa on HTTP:n versio, protokollan tilakoodi ja tekstikuvaus. Tilakoodi on kolminumeroinen luku, joka ilmaisee palvelimen vastauksen pyyntöön. Tekstikuvaus kertoo saman kuin tilakoodi, mutta tekstimuodossa.

2. Seuraavaksi palvelin lähettää selainohjelmalle otsikkotiedot itsestään ja pyydetystä dokumentista.
3. Jos kaikki on onnistunut, lähetetään data, muutoin lähetetään tilarivin ja tekstitiedon määrittelemät virheilmoitukset.
4. HTTP 1.0:ssa yhteys palvelimen ja asiakaskoneen välillä katkaistaan pyydetyn datan lähettämisen jälkeen ellei otsikkotiedoissa ole lähetetty otsikkoa Connection: Keep Alive. HTTP 1.1:ssä palvelin pitää oletuksena yhteyden auki. Yhteyden katkaisu voi haitata, jos dokumentti koostuu useista dokumenteista, kuten kuvista, kehyksistä (frame) tai appleteista, jolloin jokaisen dokumentin haku käsitellään erillisinä pyyntöinä, eli jokaisen dokumentin yhteydessä käydään erikseen läpi edellä luetellut kohdat. Kun kaikki data on lähetetty, yhteys katkaistaan. Seuraava siirtotapahtuma täytyy tämän jälkeen aloittaa alusta, kuten otettaessa yhteyttä ensimmäistä kertaa. Toisaalta tämän toimintaperiaatteen ansiosta HTTP-protokolla on erittäin kevyt, koska palvelimen ei tarvitse ylläpitää mitään tietoa yhteyksistä (Stanek 1996, 16). (Spainbour & Quercia 1997,154.)

4.1.1 HTTP:n metodit

Metodi on HTTP-komento, joka kertoo palvelimelle mitä pyynnöllä tehdään. HTTP:n keskeisimmät ja parhaiten tuetut metodit ovat GET, HEAD ja POST. (Spainbour & Quercia 1997,155.)

GET:llä pyydetään palvelimelta URI:ssa sijaitsevaa dokumenttia, jonka yhteydessä selainohjelma ei lähetä palvelimelle dataa vaan ainoastaan pyynnön saada palvelimesta haluttu dokumentti (Spainbour & Quercia 1997,156). CGI-ohjelmissa GET-metodia käytetään useimmiten, kun halutaan ajaa jokin ohjelma ensimmäistä kertaa ja selaimelta viedään ohjelmalle vähän syöttötietoa. GET-metodissa tieto välitetään ohjelmalle WWW-palvelimelta QUERY_STRING-ympäristömuuttujassa (Black Belt Web Programming Methods 1997, Dutt, 51). HEAD toimii muuten samoin kuin GET sillä erolla, että palvelin ei lähetä asiakaskoneeseen muuta kuin otsikkotiedot (Spainbour & Quercia 1997,156).

POST-metodissa lähetetään pyynnön yhteydessä lomakkeen data WWW-palvelimelle, josta se ohjataan suoraan ACTION-attribuutissa määritellyn ohjelman käsiteltäväksi. POST-metodia käytetään useimmiten lomakkeista lähetettyjen tietojen käsittelyn yhteydessä. POST-metodissa lomakkeiden data lähetetään palvelimelle lomake-elementin nimi - arvo -pareina käyttäen STDIN:ä (standard input) (Dutt 1997, 51).

4.1.2 HTML-dokumentin tulostaminen CGI-ohjelmasta

Kun CGI-ohjelma on suoritettu, sen täytyy palauttaa tieto HTML-dokumenttina selaimelle. CGI-ohjelmat tulostavat tiedon STDOUT-tulostusvirtaan (standard output) kaksiosaisen tietolohkon. Ensimmäinen osa on HTTP-otsikko (header), jossa kerrotaan tulevan datan muoto - HTML, ASCII, GIF, tms.. Otsikkolohko

loppuu tyhjään riviin, jonka jälkeen tulostetaan runko (body) otsikon kertomassa formaatissa, esimerkiksi HTML-formaatissa lähetettävä dokumentti voisi näyttää seuraavalta:

Sivu HTML-koodina:

```
<HTML>
<HEAD><TITLE>Tervetuloa</TITLE></HEAD>
<BODY>
<H1>Tervetuloa ohjelmaan, Pasi Ketonen !</H1>
</BODY>
</HTML>
```

Sivun HTTP-protokollan mukainen otsikkolohko:

```
HTTP-otsikko (header):
HTTP/1.0 200 OK
Date:  Wednesday 9-Sep-98 14:00 GMT
Server: Microsoft-PWS/2.0
Content-type:  text/HTML
Content-legth:  100
```

HTTP-runko (body):

```
<HTML>
<HEAD><TITLE>Tervetuloa</TITLE></HEAD>
<BODY>
<H1>Tervetuloa ohjelmaan, Pasi Ketonen !</H1>
</BODY>
</HTML>
```

Otsikossa on tiedot tiedonsiirtoprotokollasta ja tässä HTTP-protokollan tilakoodi (200 ok), päiväys, palvelinohjelmiston nimi ja versio, datatyyppi sekä runkoasiakirjan pituus tavuina. (Spainbour & Quercia 1997, 80-81,171,173-174.)

4.1.3 HTTP-protokollan heikkoudet ja vahvuudet

Internetin käytön yhteydessä usein esiin nousevana ongelmana pidetään tietoturvaa, joka nousee esiin erityisesti pyrittäessä hyödyntämään webbiä kaupallisesti. Esimerkiksi lähetettäessä luottokortin numeroa palvelun tarjoajalle, täytyy käyttäjän olla varma, että ainoastaan lähettäjä ja vastaanottaja pystyvät katsomaan tiedon selkokielisenä. HTTP-protokollaa pyritäänkin kehittämään turvallisemmaksi, josta esimerkkinä mainittakoon SHTTP (Secure Hypertext Transfer Protocol). HTTP:n tilattomuus saattaa heikentää sovellusten suorituskykyä, mikä johtuu siitä, että palvelupyynnöissä täytyy ensin muodostaa yhteys asiakaskoneen ja palvelimen välille ja usein ajettavat ohjelmat täytyy alustaa parametrien avulla haluttuun tilaan. Toisaalta tilattomuus on HTTP:n vahvuus, koska se kuormittaa palvelinta mahdollisimman vähän, jolloin pystytään palvelemaan useampia samanaikaisia käyttäjiä. mikä on WWW-ympäristössä erityisen merkittävää. (Stanek 1996, 15-16.)

Tilan säilyttävät protokollat ylläpitävät sovelluksen kontekstia muistissa, mikä toisaalta kuluttaa järjestelmän resursseja, mutta toisaalta taas nopeuttaa ohjelman toimintaan - muistin käyttö on suorituskyvyn kannalta huomattavasti

tehokkaampaa kuin kovalevyn. Tilattomat protokollat - kuten HTTP - ovat taas erittäin kevyitä palvelimelle, koska yhteyttä selaimen ja palvelimen sekä ohjelmien kontekstia ei ylläpidetä muistissa, jolloin palvelinkoneen resursseja varataan vain yhden palvelupyynnön käsittelyn ajaksi. HTTP-protokollaa perustuvat sovellukset käyttävät huomattavasti tilan säilyttäviä protokollia enemmän kovalevyä, joka on selkeä pullonkaula suoritustehossa, vaikkakin verkkosovelluksissa eniten ongelmia aiheuttaa verkon tiedonsiirtokapasiteetti. Useat ovat havainneet myös webbi-sivuja selaillessaan kuinka selaimesta tunnutaan ottavan useita kertoja yhteyksiä palvelimeen. Tämä on HTTP:n ominaisuus, joka joissain tilanteissa on heikkous. Esimerkiksi kaikki kuvat HTML-sivulla ovat erillisiä dokumentteja, jotka selain hakee erikseen. Tällaisissa tapauksissa tiedonsiirto olisi luonnollisesti nopeampaa, jos kaikki data lähetettäisiin yhdessä palvelussa. (Held 1997, 11.) (Stanek 1996, 16-17.)

4.2 CGI-sovelluksen tilan välittäminen

Olennaista CGI-ohjelmien toiminnassa on, että jokainen pyyntö-vastaus -pari käsitellään erillisinä tapahtumina, jonka takia CGI-sovelluksen tila täytyy välittää jokaisen pyynnössä ja vastauksessa selaimen ja WWW-palvelimen välillä. Välitettävillä parametreillä sovellus joko alustetaan siihen tilaan johon se jäi edellisen palvelupyynnön aikana tai sille etsitään palvelimella käyttäjälle lukittu ja aktiiviseksi jätetty sovellusprosessi. CGI-sovelluksen tila säilytetään välittämällä erillisten palvelupyyntöjen yli parametrejä. Parametrejä voidaan välittää selaimen ja palvelimen välillä piilotettujen kenttien, URL:n tai Cookieiden avulla. CGI-ohjelmien parametrit siirretään joko URL-osoitteen

mukana tai lomakkeen tekstirungossa, mikä määrittyy lomakkeen METHOD-attribuutin arvosta. GET-metodilla tiedot siirtyvät URL-osoitteen mukana, kun taas POST-metodi siirtää tiedot HTTP-protokollan runko-osassa. CGI-sovellus saa selaimelta lähetetyn datan WWW-palvelimen ympäristömuuttujista. (Spainbour & Quercia 1997, 80.)

CGI-sovellukselle voidaan välittää dataa käyttäen URL:ää WWW-palvelimen QUERY_STRING:llä. URL:a voitaisiin käyttää esimerkiksi seuraavasti, jossa etunimi ja sukunimi viedään CGI-ohjelmalle:

```
http://kone/cgi-bin/ohjelma?etunimi=Pasi&sukunimi=Ketonen.
```

Kysymysmerkki aloittaa parametri-osion, jossa annetaan parametrit arvoineen pareina, parametrit erotetaan &-merkillä. URL:n käytön heikkous parametrien välityksessä ovat sen rajallinen koko – korkeintaan 255 merkkiä - sekä mahdollisten erikoismerkkien - kuten ääkkösten - käyttö. URL:ssä voi suoraan käyttää ainoastaan 'amerikkalaisia' aakkosia, numeroita ja merkkejä ’$, -, _, . ja +’. Kaikkia muita merkkejä käytettäessä merkit täytyy ensin koodata URL:n hyväksymiksi merkeiksi, joka vastaavasti CGI-ohjelman täytyy purkaa, esimerkiksi 15/02/70 olisi koodattuna 15%2F02%2F70. Sekä URL:n kokorajoitus että erikoismerkkien käyttö vaikeuttavat URL:n käyttöä parametrien välityksessä erityisesti, jos parametrien arvot tulevat dynaamisesti, esimerkiksi tietokannasta. Käyttäjät voivat myös itse kirjoittaa suoraan URL:n parametreille arvot ja ohittaa siten ohjelman välittämien parametrien arvot, mikä vaatii erityistä tarkkuutta virheenkäsittelyyn. (Spainbour & Quercia 1997, 82.)

Käyttäjän syöttämät tiedot välitetään CGI-ohjelmalle käyttäen lomaketta, josta tiedot lähetetään elementin nimi-arvo -pareina WWW-palvelimelle ja edelleen sovellukselle. Lomaketta voidaan käyttää myös ohjelman sisäisten parametrien välityksessä FORM:n piilokentillä, jotka ovat nimensä mukaisesti FORM:iin upotettuja elementtejä, jotka eivät näy käyttöliittymässä. Käyttäjä voi kuitenkin katsoa niiden arvoja dokumentin lähdekoodista. Piilotettujen kenttien etu on, että ne siirtyvät muiden lomakkeen tietojen mukana, jolloin niiden käsittely ei edellytä erityistä ohjelmointia. Käyttäjät eivät myöskään pääse muokkaamaan piilotettujen kenttien arvoja, kuten URL:ssa. (Sirola & Linjamaa 1996, 418.)

Cookiet (keksit, eväste) on mekanismi, jolla voidaan tallettaa pysyvää tietoa työasemaan. Cookiet asetetaan palvelimessa, joka lähettää sen selainohjelmalle, joka tallettaa cookien työasemaan. Cookie on käytännössä lyhyt ASCII-tiedosto asiakaskoneen kovalevyllä ja sitä voidaan käyttää hyvin esimerkiksi käyttäjän tunnistuksessa, koska cookie voidaan määrätä olemaan voimassa halutuksi ajaksi. (Sirola & Linjamaa 1996, 409.) Internet-sovellukset koostuvat useimmiten useasta eri ohjelmasta, joita jokaista voidaan ajaa suoraan kirjoittamalla ohjelman URL-osoite. Esimerkiksi tällaisissa tapauksissa on järkevää tehdä käyttäjän kirjoittautuminen ohjelmaan kerran asettamalla sen yhteydessä cookie, jota käytetään muissa ohjelmissa tarkistettaessa käyttäjän oikeuksia ohjelman käyttöön. Internet-ohjelmissa on huomattavaa, että käyttäjäoikeuden tarkistus täytyy tehdä jokaisen palvelupyynnön yhteydessä, muuten käyttäjäntunnistuksen voi ohittaa siirtymällä suoraan tunnistuksen ohi

URL-osoitteen avulla. Cookieilla voidaan hyvin ylläpitää myös muita käyttäjäkohtaisia asetuksia.

Kun CGI-ohjelma tunnistaa uuden käyttäjän, se lähettää asiakkaalle lähtevään vastaukseensa ylimääräistä otsikkotietoa. Tämä tieto tallennetaan asiakaskoneen kovalevylle cookies-tiedostoon. Cookieiden käyttö edellyttää selainta, joka tukee niiden käyttöä, sekä sitä, että selain on asetettu hyväksymään cookieiden käyttö. Cookieiden asettamisen jälkeen kaikki asiakaskoneelta cookien asettaneelle palvelimelle lähtevät pyynnöt sisältävät otsikkotiedoissa cookiessa olevat tiedot. Koska cookiet ovat asiakaskoneen kovalevyllä, tiedot voidaan säilyttää myös eri istuntojen välillä niin haluttaessa. Cookie luodaan, kun palvelin lähettää sen selaimelle, joka tallentaa cookien. Seuraavien palvelupyyntöjen yhteydessä CGI-ohjelma tutkii, onko haluttu cookie voimassa, jolloin sitä ei tarvitse asettaa uudestaan, muutoin CGI-ohjelma lähettää asiakaskoneelle vastauksen yhteydessä set-cookie -otsikon, joka tekee ylimääräisen otsikkotiedon cookien asettamiseksi. Ylimääräinen otsikkotieto koostuu seuraavista tiedoista:

- muuttuja = arvo (tai cookie = arvo)
    - arvo ei saa sisältää sarkainta, välilyöntiä tai puolipistettä muuten kuin koodattuina, kuten URL:kin
- expires = cookien vanhenemispäivämäärä
- path = palvelimen hakemistot, joissa cookie on voimassa
- domain = koneiden toimialue, joihin cookie lähetetään

- esimerkiksi firma.com, jolloin cookie lähetettäisiin kaikkiin sellaisiin koneisiin, jotka kuuluvat ko. toimialueeseen, kuten WWW.firma.com ja dev.firma.com. Palvelinmäärittelyn määrittely tarvitaan, jotta cookie voidaan yksilöidä käytettäväksi oikean palvelimen kanssa.

- secure-attribuutti määrittää, voiko asiakas lähettää cookien ainoastaan salattuna SHTTP:tä tai SSL:ää käyttäen

Cookiet täytyy asettaa HTML-dokumenttien HEADER-osiossa, eli ennen varsinaista loppukäyttäjälle näytettävää tietoa. Cookieissa huomioitavaa on myös, että uusi esiintymä korvaa vanhan, mutta ainoastaan mikäli ne on asetettu samaan palvelimeen ja path-määrittely on sama. Hyvänä tyylinä cookien asettamisessa voidaan useimmiten pitää, että samannimiset tulisi tuhota ennen uuden asettamista. Cookieiden käytölle on olemassa seuraavat rajoitukset:

- yksittäinen cookie voi olla kooltaan korkeintaan 4 KB
- yhdessä toimialueessa tai palvelimessa voidaan ylläpitää korkeintaan 20 cookieta
- selain voi käsitellä korkeintaan 300 cookieta

(Sirola & Linjamaa 1996, 410; Client Side State – HTTP Cookies 1997; Spainbour & Quercia 1997, 97-99.)

4.3 CGI-ympäristön puutteita

Puhtaat CGI-sovellukset ajetaan kokonaisuudessaan palvelimessa, mikä ei ole aina esimerkiksi verkkoliikenteen kannalta järkevää – kuten syöttötiedon tarkistuksessa. Selaimeen tehtävä toiminnallisuus taas joudutaan usein

ohjelmoimaan jollakin toisella ohjelmointikielellä kuin varsinainen CGI-sovellus, mikä luonnollisesti vaatii lisätaitoja.

WWW-palvelin välittää pyynnön ohjelmalle CGI-prosessilla, jonka käynnistäminen aiheuttaa palvelimelle kuormitusta, koska prosessi käynnistetään ja päätetään yhden palveluprosessin aikana ja jokainen prosessi voi palvella ainoastaan yhtä palvelupyyntöä (Blum 1997, 67). Tätä on tosin parannettu ISAPI- (Microsoft) ja NSAPI-palvelimissa (Netscape), joissa yksi prosessi voi palvella useita pyyntöjä. ISAPI- ja NSAPI-palvelimet ovatkin paremmin skaalautuvia, eli ne pystyvät käsittelemään useampia palvelupyyntöjä kuin CGI-palvelimet (Blum 1997, 68). Toisaalta taas ISAPI-prosessin kaatuminen aiheuttaa koko webbi-palvelinohjelmiston kaatumisen, koska yksi prosessi hoitaa kaikkia palvelupyyntöjä (Blum 1997, 68). Erityisesti ISAPI-tekniikassa rajoituksia aiheuttaa myös käyttöjärjestelmäriippuvaisuus, ainoastaan Windows NT.

CGI-rajapinta voi välittää tietoa WWW-palvelimelta kahdella eri tavalla: ympäristömuuttujilla ja ohjelman vakiosyöttövirralla. Vastaavasti ajettava ohjelma palauttaa tietoa pääsääntöisesti vakiotulostusvirrassa. Ongelmia aiheutuu mm., jos ajettava ohjelma ei pysty käsittelemään vakiosyöttö- ja vakiotulostusvirtaa. Myös välitettävän tiedon määrä on rajoittava tekijä, koska sekä palvelimen ympäristömuuttujilla voidaan siirtää rajallinen määrä tietoa, kuten QUERY_STRING 255 merkkiä. (Blum 1997, 66-67.)

5. Tietokannan käsittely internetissä

5.1 Erilaiset tietokantajärjestelmän arkkitehtuurit

Tietokantajärjestelmien rakentamisessakin on arkkitehtuurin valinta tärkeää. Tietokantasovelluksen arkkitehtuuri koostuu toiminnallisesti katsottuna kolmesta osasta:

- Käyttöliittymälogiikka, joka sisältää kaiken käyttöliittymän käsittelyssä tarvittavan koodin.
- Liiketoimintalogiikka, joka voi olla mm. kuukauden myynti, myyntiedustajan laskutusten summa, eli algoritmit ja ohjelmat, jotka hakevat ja muokkaavat tietokannasta halutut tiedot.
- Tiedon hallintajärjestelmä, joka on tyypillisesti tietokanta.

(Gorman 1994, 189.)

Arkkitehtuurilla valitaan suoritetaanko tietokantasovellus kokonaan palvelimessa, kokonaan asiakkaassa vai hajautetusti asiakkaassa ja palvelimessa. Seuraavissa kappaleissa käydään läpi erilaiset arkkitehtuurit arvioiden niiden vahvuuksia ja heikkouksia. Myös WWW-ympäristössä toteutettavat tietokantasovellukset voivat käyttää samoja arkkitehtuureita kuin muutkin tietokantasovellukset. Puhtaat CGI-sovellukset vastaavat kuitenkin lähinnä keskitettyä arkkitehtuuria – tosin palvelimella tapahtuvaa sovelluslogiikan suoritusta voidaan hajauttaa useille eri palvelimille.

5.1.1 Keskitetty järjestelmä (host-based)

Keskitetty, ”tyhmiin” päätteisiin perustuva järjestelmä on ensimmäinen kaupalliseen käyttöön tullut tietokanta-arkkitehtuuri. Siinä palvelin suorittaa kaiken ohjelmalogiikan. Työasema näyttää ainoastaan palvelimen lähettämän datan. (Gorman 1994, 187.)

Vahvuudet

Keskitetty arkkitehtuuri on edullinen ja hyvä sovellusten ylläpidon kannalta, koska sovellus on keskitetysti yhdessä paikassa. Asiakas/palvelin-arkkitehtuuriin verrattuna keskitetty järjestelmä vähentää myös verkkoliikennettä, koska työasemalle lähetetään ainoastaan näytölle tuleva tieto. (Gorman 1994, 191.)

Heikkoudet

Arkkitehtuurin käyttää työasemaa pelkkänä näyttölaitteena, jolloin jokainen käyttäjä kuormittaa palvelimen prosessoria ja muistia. Sen seurauksena käyttäjien täytyy jonottaa prosessoria ohjelmansa suorittamiseksi. Nykyisillä palvelimilla tämä tosin näkyy vasta useilla sadoilla käyttäjillä. (Gorman 1994, 191-192.)

5.1.2 Asiakas/palvelin (Client/Server)

Asiakas/palvelin-arkkitehtuurissa sovelluslogiikan suorittaminen tapahtuu työasemassa. Palvelin ainoastaan välittää tietokannasta tietoa työaseman siitä

pyytäessä. Ensisijaisena tavoitteena on hajauttaa sovelluksen suorittamisesta aiheutuvaa kuormitusta. (Gorman 1994, 187.)

Vahvuudet

Merkittävin etu asiakas/palvelin-arkkitehtuurissa on, että jokaisella käyttäjällä on oma prosessori ja muisti omassa käytössä sovelluslogiikan suorittamiseen (Gorman 1994, 190). Toinen etu on, että palvelin ei rajoita käyttöliittymän merkkipohjaisuutta tai graafisuutta.

Heikkoudet

Asiakas/palvelin-arkkitehtuurin suuri heikkous verrattuna keskitettyyn järjestelmään on ylläpito. Jo useiden kymmenien työaseman järjestelmissä, uusien versioiden ja sovellusten asentaminen vaatii paljon työtä. Asennusta helpottamaan onkin kehitetty useita erilaisia sovelluksia, jotka työntävät uudet ohjelmat ja asetukset työasemiin. Arkkitehtuurin ehkä kaikista merkittävin heikkous on sen aiheuttama suuri verkkoliikenne palvelimen ja työaseman välillä. Esimerkiksi haluttaessa laskea jokaisen kuukauden tilausten summa, tarvitsisi jokaisen tilaustietueen summa-tieto tuoda verkon yli työasemaan, jossa ne vasta laskettaisiin yhteen - keskitetyssä järjestelmässä verkon yli tulisi ainoastaan yksi summa. Verkon nopeuden parantaminen laitteiston avulla on lisäksi kallista, erityisesti jouduttaessa käyttämään WAN:a (Wide Area Network). (Gorman 1994, 190.)

5.1.3 Monitasoarkkitehtuuri (n-tier)

Monitasoarkkitehtuurissa asiakas ajaa palvelimessa ohjelman, joka suorittaa tietokannan käsittelylogiikan – ns. liiketoimintalogiikka - esimerkiksi kaikkien kuukauden laskut hakemisen. Asiakas saa verkon yli jo palvelimella osittain käsitellyt tiedot, joita asiakaskoneessa ajettava ohjelma käsittelee edelleen. (Gorman 1994, 187.)

Vahvuudet

Monitasoarkkitehtuuri on suorituskyvyltään paras, koska sovelluksen logiikkaa suoritetaan siellä, missä se on toiminnallisuuden kannalta järkevää. Massiiviset tiedon käsittely- ja päivitysoperaatiot suoritetaan palvelimella, jolla saadaan minimoitua verkkoliikenne. Näytön käsittely taas tehdään kokonaisuudessaan työasemassa, jolloin myös käyttöliittymän käsittely tehdään sekä verkkoliikenteen että suorituksen kannalta parhaassa paikassa. (Gorman 1994, 192.) Arkkitehtuurin vahvuus on myös sen skaalautuvuus (Özsu, Valduriez 1999, 18). Asiakassovellukset käyttävät tietokantaa sovelluspalvelimen kautta, joita voidaan helposti lisätä kuormituksen kasvaessa.

Heikkoudet

Ylläpito on jossain määrin heikkous verrattuna keskitettyyn järjestelmään, koska käyttöliittymälogiikka asennetaan työasemiin, eli ylläpitoon menee lähes sama työmäärä kuin asiakas/palvelin-arkkitehtuurissakin. Tosin sovelluksen keskeisin osa tietokannan käsittelylogiikka sijaitsee keskitetysti palvelimella. Ohjelmoinnin

kannalta ympäristö on myös jonkin verran monimutkaisempi. Ohjelman suunnittelu vaatii huolellista työtä, jotta logiikan suorittaminen tapahtuu oikeassa paikassa. Usean käyttäjän ympäristöissä tietokantatransaktioiden hallinta voi myös olla haastavampaa - erityisesti mahdollisten virhetilanteiden kohdalla. (Gorman 1994, 192.)

5.2 Hajautettu tietokanta

Tietokannan hajauttaminen voidaan jakaa kahteen pääluokkaan: partitiointiin eli tietokannan osittamiseen tai replikointiin eli datan kopiointiin tietokantojen välillä (Özsu, Valduriez 1999, 20). Tyypillisiä syitä tietokannan hajauttamiseen ovat suorituskyvyn ja/tai toimintavarmuuden parantaminen (Özsu, Valduriez 1999, 16). Partitioinnilla pyritään ensisijaisesti suorituskyvyn parantamiseen jakamalla tietokannan käsittelystä aiheutuva kuormitus usealle eri prosessorille. Replikoinnilla taas voidaan pyrkiä sekä parempaan toimintavarmuuteen että suorituskykyyn. Replikoinnin toteutus sinänsä voi olla hyvinkin haastava tehtävä, joka on oma tutkimusaiheensa. Karkeasti replikoinnin vaativuus voidaan jakaa helpoimmasta vaativaan seuraavasti:

1. Käyttäjät tekevät muutokset keskitetysti yhteen tietokantaan, josta muutokset kopioidaan muihin ainoastaan tietueiden lukemiseen käytettäviin tietokantoihin.
2. Kaikki käytettävät tietokannat ovat identtisiä keskenään ja käyttäjät voivat päivittää niistä jokaista samanaikaisesti. Tässä mallissa tiedon yhteentörmäysten riski on suuri ja sen toteuttaminen on vaativa algoritminen ongelma.

5.3 Tietokannan toimintavarmuuden takaaminen

Tietokantojen sisältämä data on usein yritysten ja organisaatioiden tärkein omaisuus. Tänä päivänä tietokannan toimintavarmuus on usein ehdoton edellytys yritysten ja organisaatioiden päivittäisessä toiminnassa. Tietokannan toimintavarmuus ei tarkoita ainoastaan palvelun käynnissä pysymistä vaan myös sen oikeaa toimintaa useiden käyttäjien ympäristöissä. Tietokantasovelluksissa mahdollisimman monen täytyy pystyä samanaikaisesti käyttämään samoja tietueita. Toisaalta taas tietokannasta täytyy saada aina oikeita tietoja eli keskeneräisiä tietueiden päivityksiä ei näytetä. Edelliset kaksi seikkaa ovat tietokantaympäristöissä toistensa vastavoimia, joiden välillä ohjelmoija joutuu valitsemaan painotuksen. Toimintavarmuuden yhteydessä puhutaan transaktioiden (tietokantatyön yksikkö) ACID-ominaisuuksista sekä lukituksista. ACID-ominaisuudet varmistavat tietokannasta tulevan tiedon oikeellisuuden. (Özsu, Valduriez 1999, 283.)

Atomicity merkitsee, että transaktiota käsitellään yksikkönä. Transaktio joko toteutetaan kokonaisuudessaan tai ei lainkaan. Tällöin tietokantaan ei voi tallentua keskeneräisiä transaktioita. Tätä ominaisuutta kutsutaan myös ’kaikki tai ei mitään’ –ominaisuudeksi. (Özsu, Valduriez 1999, 283-284.)

Consistency tarkoittaa transaktion oikeellisuutta. Tietokannan päivitys muuttaa sen ’ehjästä’ tilasta uuteen ’ehjään’ tilaan. (Özsu, Valduriez 1999, 284.)

Isolation tarkoittaa, että toiset käyttäjät eivät voi nähdä keskeneräisiä tietueen päivityksiä vaan ainoastaan valmiit tietueet - joko uuden tai vanhan. Muut käyttäjät eivät voi katsella päivitettävän tietueen tai tietueiden dataa - jokainen transaktio on eristetty. (Özsu, Valduriez 1999, 285.)

Durability on, että tietokantatransaktion päättyminen tekee muutoksista pysyviä. Muutokset voidaan peruuttaa ainoastaan aloittamalla uusi transaktio. (Özsu, Valduriez 1999, 288.)

Lukitukset ovat mekanismi, jonka avulla estetään eri käyttäjiä samanaikaisesti päivittämästä yhtä tietuetta. Niiden toiminta toteutetaan sekä ohjelmoinnilla että tietokannan ominaisuuksilla. Lukituksia käytettäessä tärkeä huomioitava seikka on, että transaktiossa käsiteltävät tietueet pysyvät lukittuina koko transaktion keston ajan. Monen käyttäjän ympäristöissä pyritään usein minimoimaan tietueiden lukitukset, jotta tietueet olisivat mahdollisimman monen käyttäjä käytössä. Lukitusten ja transaktioiden ohjelmointi on keskeinen kysymys tietokantasovellusten toteuttamisessa. Niiden toteuttamista tutkitaankin tarkasti WebSpeedillä toteutettavan ohjelmointia käsittelevissä luvuissa.

Suurten käyttäjämäärien ympäristöissä on tiedon päivityksen hallinta erityisen tärkeä tekijäksi. Ohjelmoijan täytyy päättää, täytyykö käyttäjän nähdä tietueesta

- viimeisin versio,
- käyttäjäkohtaisesti määritelty versio,
- säännöllisesti ja automaattisesti haettava uusin versio vai

- kullakin hetkellä saatavissa oleva versio ?

Käyttäjille voidaan haluta myös näyttää aina uusi tietue, mikäli se muuttuu tietokannassa. Tämä voidaan toteuttaa Internetissä selaimen avulla tietyin aikavälein tapahtuvalla tietueen automaattisella haulla. Selaimen avulla toteutettuna huonoksi puoleksi, varsinkin suurilla käyttäjämäärillä, on runsaasti turhaa verkkoliikennettä. Paras tapa olisi, että tietokanta käynnistäisi tietueen muutoksen yhteydessä käyttäjille muuttuneet tiedot työntävän ohjelman. (Özsu, Valduriez 1999, 596.)

5.4 Internet tietokannan toimintaympäristönä

Internetin merkittävä etu on maailmanlaajuisesti käytettävissä oleva verkko. Maailman muuttuessa yhä kansainvälisemmäksi toimivat myös yritykset ja organisaatiot yli maantieteellisten rajojen. Kansainvälistyminen tarkoittaa tietokannoille ja sovelluksille monikielisyyttä sekä käyttäjä- ja tietomäärältään entistä suurempaa kokoa.

Sovellusten suurista tieto- ja käyttäjämääristä johtuen WWW-sovellukset pitää usein suunnitella minimoimaan tietoliikenne selaimen ja palvelimen välillä. Suuri käyttäjämäärä vaatii myös tietokannan ajoympäristöltä skaalautuvuutta kuormituspiikkeihin. Siinä ongelma ole ainoastaan tekninen vaan myös kaupallinen: tietokannan lisensointi. Teknisesti suuriin käyttäjämääriin voidaan valmistautua käyttämällä rinnakkaisarkkitehtuuria. Siinä samaa järjestelmää ajetaan eri prosessoreilla ja kovalevyillä suorituskyvyn parantamiseksi. Tietokanta voi olla joko ositettu tai niitä voi olla useita replikoimalla

ylläpidettäviä. Rinnakkaisarkkitehtuuria voidaan käyttää myös järjestelmän jatkuvakäyttöisyyden (24x7) sekä laajennettavuuden ratkaisemisessa. (Özsu, Valduriez 1999, 420-423.)

5.4.1 Monikielisyys

Internetissä käytettävän tietokannan täytyy usein pystyä tukemaan monikielisyyttä, jolloin tietokannan täytyy pystyä käyttämään eri kielien merkistöjä. Unicode on merkistö, joka sisältää kaikkien yleisesti käytettyjen kielien merkistöt. Useimmat monikielisyyttä tukevat tietokannat käyttävätkin merkkien tallentamisessa unicodea. Myös monikielisten sovellusten täytyisi pystyä käyttämään unicodea.

5.5 Tietoturva

Tietoturvan merkitys kasvaa yhä merkittävämmäksi, mitä avoimemmin sovellus ja tietokanta ovat käytettävissä Internetistä. Järjestelmässä täytyy pystyä sekä estämään luvaton käyttö että salaamaan käyttäjän käsittelemä tieto.

Adam Cain, National Center for Supercomputer Applications, jaottelee webin tietoturvan neljään luokkaan:

- Järjestelmän tietoturva ja yhtenäisyys
- Käyttäjän tunnistus
- Käyttöoikeuden hallinta
- Yksityisyyden suojaaminen, tiedon salaus

(Floyd 1997, 249.)

6. WebSpeed ohjelmointityökaluna ja ajoympäristönä

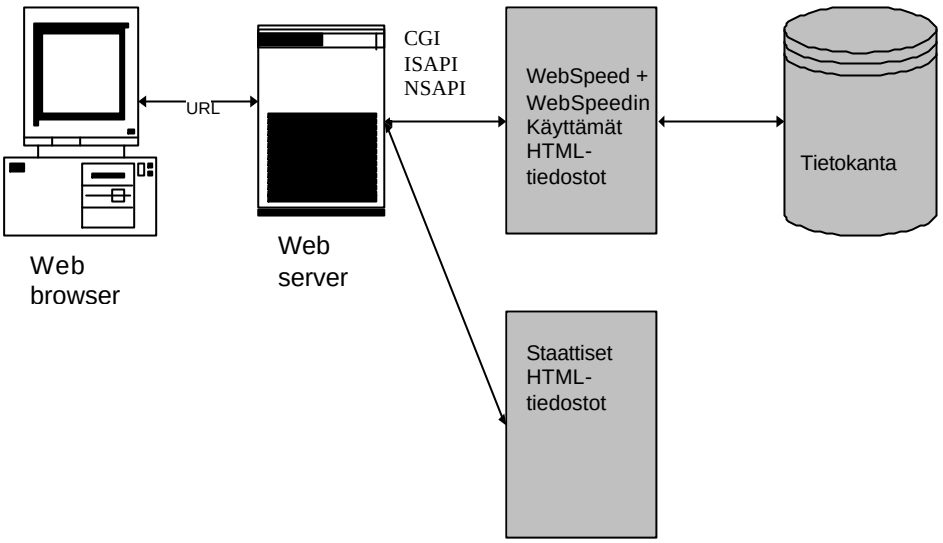
WebSpeed on Progress Software Corporationin WWW-ympäristöön kehittämä sovellusten kehitystyökalu ja ajoympäristö. WebSpeedillä kehitetään CGI-, NSAPI- tai ISAPI-pohjaisia sovelluksia webiin. Työkalu on suunniteltu ensisijaisesti yritysten tietokantasovellusten tekemiseen ja sovellusarkkitehtuuriltaan se on keskitetty. Laitteistoriippumattomuus on yrityssovelluksissa ja siten myös WebSpeedissä olennainen asia. Ohjelmointikieli on Progress Softwaren oma ajonaikaisesti tulkattava 4GL-ohjelmointikieli – WebSpeedissä kieltä kutsutaan SpeedScriptiksi. Sovellusten käyttöliittymä toteutetaan HTML:llä tai muulla muotoilukielellä.

WebSpeedin ajoympäristö perustuu pitkälti WWW:ssä käytettäviin standardeihin, joka takaa mahdollisimman hyvän yhteensopivuuden muihin WWW:ssä käytettäviin tekniikoihin, kuten WWW-palvelinohjelmistoihin ja selaimiin.

6.1 WebSpeedin arkkitehtuuri

WebSpeed koostuu kahdesta osasta: sovellusten ajo- ja kehitysympäristöstä, joka on WebSpeedillä tehty sovellus. Sovellusten ajoympäristöön kuuluvat WebSpeedin lisäksi selain, WWW- ja tietokantapalvelin, jotka voidaan sijoittaa eri koneisiin. WebSpeed-ohjelmisto sijaitsee toiminnallisesti web-palvelimen ja tietokannan välissä (Kuva 6-1). WebSpeedin komponentteja ovat adminserver,

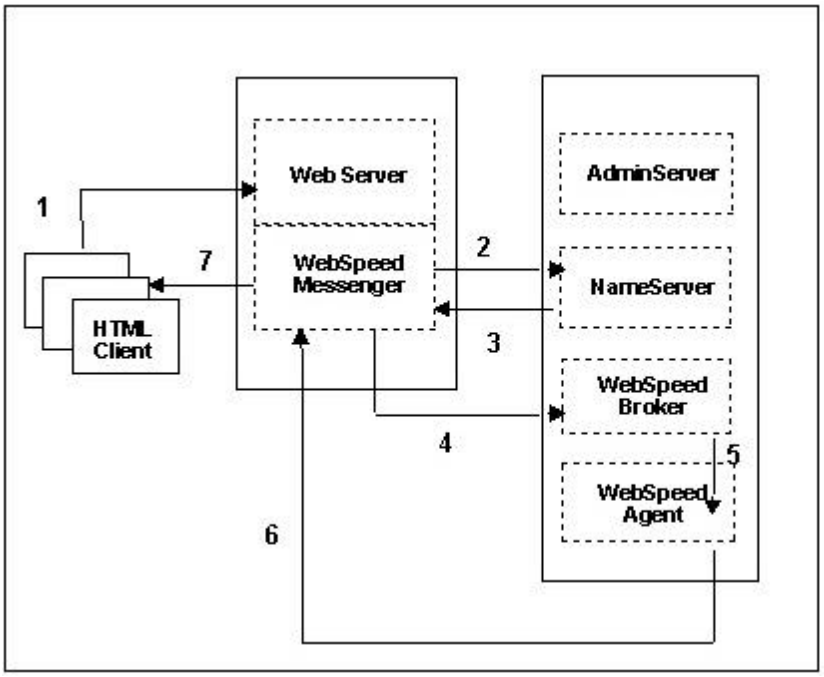
nameserver, messenger, broker ja agentit (Kuva 6-2) (WebSpeed Developer's Guide 1998. 1-5, 10-2).



Kuva 6-1 WebSpeed-sovellusten ajoympäristön arkkitehtuuri (Progress Software Oy 1997).

Adminserver on ajoympäristön hallintaympäristö, josta käynnistetään ja hallitaan eri sovellus- ja tietokantapalveluita (brokerit) (WebSpeed Installation and Configuration Guide for Windows NT 1998. 2-4).

Nameserver on palvelupyyntöjen ensisijainen vastaanottaja, joka ohjaa selaimilta tulevat palvelupyynnöt oikeaan sovelluspalveluun (WebSpeed Installation and Configuration Guide for Windows NT 1998. 2-2).



Kuva 6-2 Selaimelta lähetetyn palvelupyynnön käsittely WebSpeedin ajoympäristössä (WebSpeed Installation and Configuration Guide for Windows NT 1998. 2-7).

Messenger on joko CGI-, ISAPI- tai NSAPI- prosessi, riippuen käytetystä WWW-palvelimesta. Messenger välittää palvelupyyntöjä ja dataa WWW-palvelimen ja WebSpeedin välillä. WebSpeedissä CGI-prosessi toimii ainoastaan tiedon välityksessä sovelluksen ja WWW-palvelimen välillä. (WebSpeed Installation and Configuration Guide for Windows NT 1998. 2-2.)

Broker on ”johtajaprosessi”, joka ohjaa selaimelta messengerin välityksellä lähetetyt palvelupyynnöt vapaalle agentin. Broker tietää käytettävissä olevat agentit ja niiden tilat, joiden avulla se hoitaa myös ajonaikaisen kuormituksen hallinnan. Käyttäjämäärän kasvaessa suuremmaksi kuin mitä käytössä olevat

agentit pystyvät hoitamaan, broker lisää automaattisesti uusia agenttiprosesseja lisenssin rajoissa. Myös kuormituksen laskiessa agenttiprosesseja tuhotaan palvelimen muistin vapauttamiseksi.

Agentti on WebSpeedin suorittava osa, joka hoitaa tietokantayhteyden ja ohjelman suorittamisen (WebSpeed Installation and Configuration Guide for Windows NT 1998. 2-2). Agenttiprosessit ovat tietokannan asiakkaita, jotka pysyvät jatkuvasti sovelluspalvelimen muistissa. Ne myös tyypillisesti luovat tietokantayhteyden käynnistyksen yhteydessä, mikä toteutetaan brokerille annetuilla käynnistysparametreillä. Tosin ohjelmoija voi halutessaan luoda tietokantayhteyden myös ohjelmallisesti ajon aikana.

Agenttiprosessin pitäminen muistissa ja jatkuva tietokantayhteys vähentävät ajonaikaisesti ohjelman suoritukseen kuluvaa aikaa, koska niillä voidaan vähentää kovalevyn käyttöä (käyttämällä muistia) ja verkon kuormitusta. Se myös antaa ohjelmoijalle paremmat mahdollisuudet säilyttää sovelluksen tila eri palvelupyyntöjen yli. Agenttiprosessi ei vaikuta CGI-prosessien (messenger) käyttäytymiseen, joiden toiminnallisuuden hoitaa WWW-palvelin.

Broker antaa ohjelman suorituksen käytettävissä olevalle agentille. Agentti taas ilmoittaa aina brokerille tilansa muutoksesta. Agenttien tilat ovat:

- **Available:** Agentti on valmis suorittamaan ohjelman (WebSpeed Developer’s Guide 1998. 2-10).

- **Busy:** Agentti on suorittamassa ohjelmaa ja näin ollen ei muiden käyttäjien käytettävissä
- **Locked:** Agentti on lukittu ainoastaan yhden nimetyn selaimen käyttöön
- **Limbo (unix):** Agentti on vaihtamassa tilaansa. Jos tila säilyy pidempää, se tarkoittaa, että agentti on virhetilassa
- **Starting (unix):** Broker on käynnistänyt agentin, mutta se ei ole vielä suorittanut alustusta
- **Notstarted (nt):** Agentille on varattu muistista tila, mutta Broker ei ole käynnistänyt sitä

(WebSpeed Installation and Configuration Guide for Windows NT 1998. 2-2, WebSpeed Developer's Guide 1998. 2-10.)

6.1.1 Ajoympäristön hajautus

Ajoympäristön hajauttaminen eri koneille voi olla tarpeellista, jos käyttäjämäärä palvelimelle aiheutuva kuormitus muodostuu suorituskyvyn kannalta pullonkaulaksi. Hajautetussa ympäristössä palvelimien välille tulee rakentaa nopein mahdollinen verkko, jottei siitä aiheutuisi pullonkaulaa. Alla luetellaan yksittäisen WebSpeedin ajoympäristön vaatimukset:

- Nameserver voi olla omalla koneella.
- Brokerin ja sen agenttien täytyy olla samassa koneessa.

(WebSpeed Developer's Guide 1998. 2-11.)

Taulu 1 näyttää suurimman mahdollisen WebSpeedin ajoympäristön hajautuksen verkossa. Lisäksi ympäristöä voidaan edelleen laajentaa erillisillä sovelluspalvelimilla ulkoisten aliohjelmien ajamiseen (WebSpeed Developer's

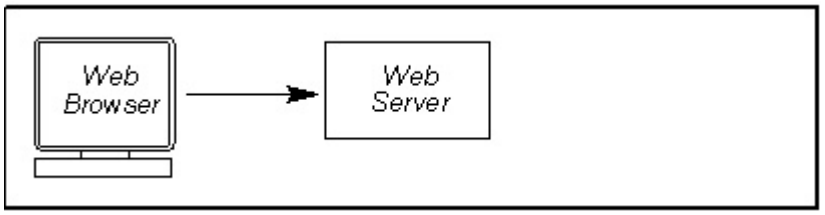
Guide 1998. 2-11). Ajoympäristön hajautusta mietittäessä kannattaa aina pitää mielessä, että verkkoliikenteen kannalta optimaalisinta on suorittaa mahdollisimman moni prosessi samassa palvelimessa.

Kone 1	Kone 2	Kone 3	Kone 4	Kone 5
Web selain	• Web palvelin • WebSpeed Messenger	NameServer	• WebSpeed Broker • WebSpeed Agent • WebSpeed Management Utilities	Tietokanta-palvelin

Taulu 1: Suurin mahdollinen ajoympäristön hajautus (WebSpeed Developer's Guide 1998. 2-11).

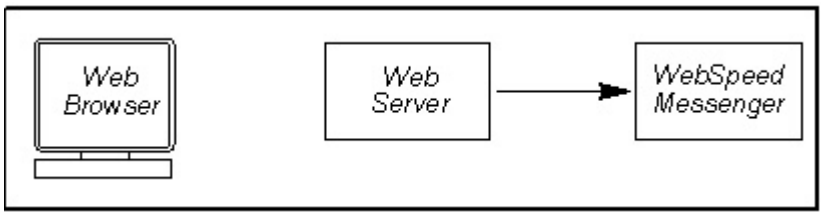
Toinen vaihtoehto hajautukselle on laittaa sama sovellus useaan palvelimeen. NameServer mahdollistaa automaattisen samaa sovellusta ajavien ajoympäristöjen (eri brokereiden) kuormituksen tasaamisen, jolloin NameServer ohjaa palvelupyynnöt brokereille niille asetettujen prioriteettien perusteella. (WebSpeed Installation and Configuration Guide 1998. 4-10.)

6.2 HTTP-palvelupyynnön käsittely WebSpeedissä



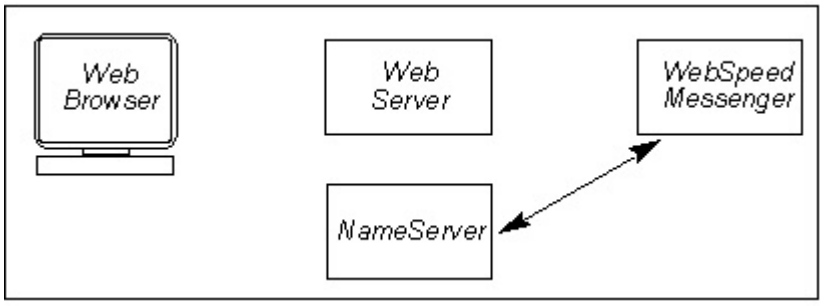
Kuva 6-3 (WebSpeed Developer's Guide 1998. 2-2).

1. Web-palvelin vastaanottaa palvelupyynnön selaimelta URL:n välityksellä.



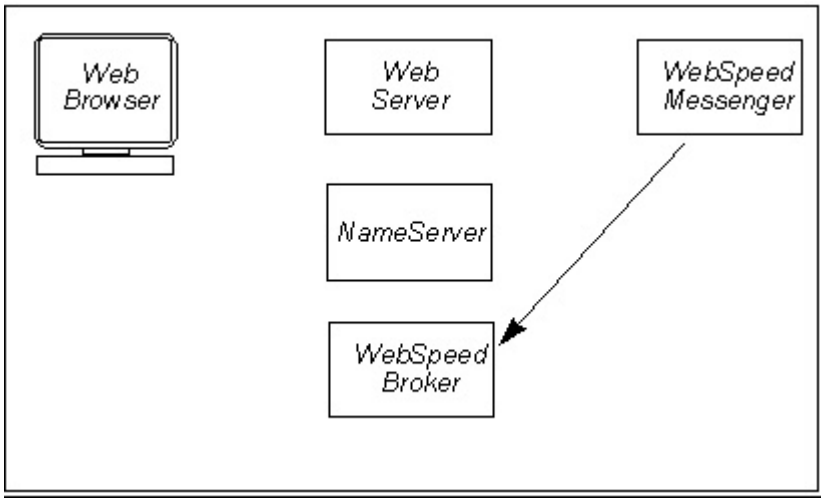
Kuva 6-4 (WebSpeed Developer's Guide 1998. 2-2).

2. Web-palvelin käynnistää CGI- tai aktivoi ISAPI- tai NSAPI-messengerin.



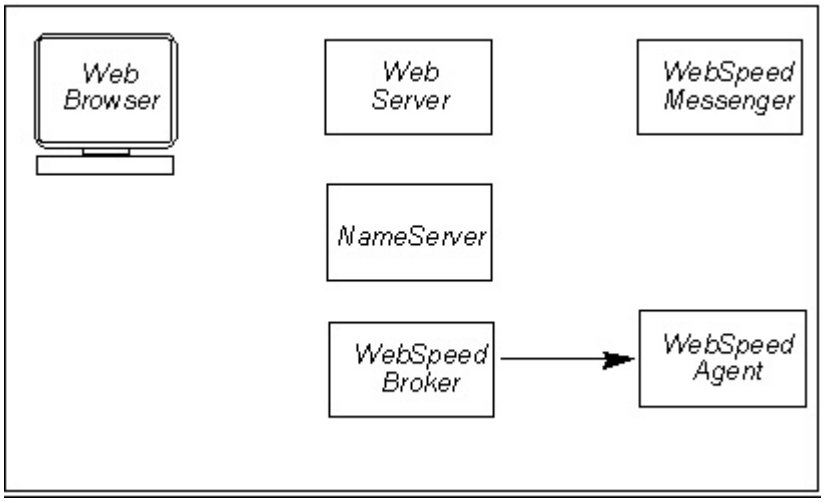
Kuva 6-5 (WebSpeed Developer's Guide 1998. 2-3).

3. WebSpeedin messenger lähettää palvelupyynnön NameServerille, joka etsii halutun palvelupyynnön käsittelyyn käytettävissä olevan WebSpeedin brokerin.



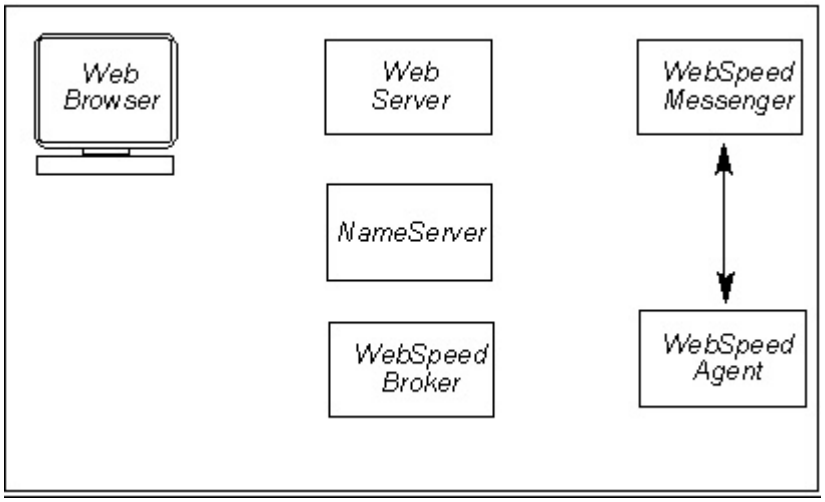
Kuva 6-6 (WebSpeed Developer's Guide 1998. 2-3).

4. NameServer hakee halutun brokerin ja yhdistää messengerin brokeriin. Messenger antaa brokerille palvelupyynnön.



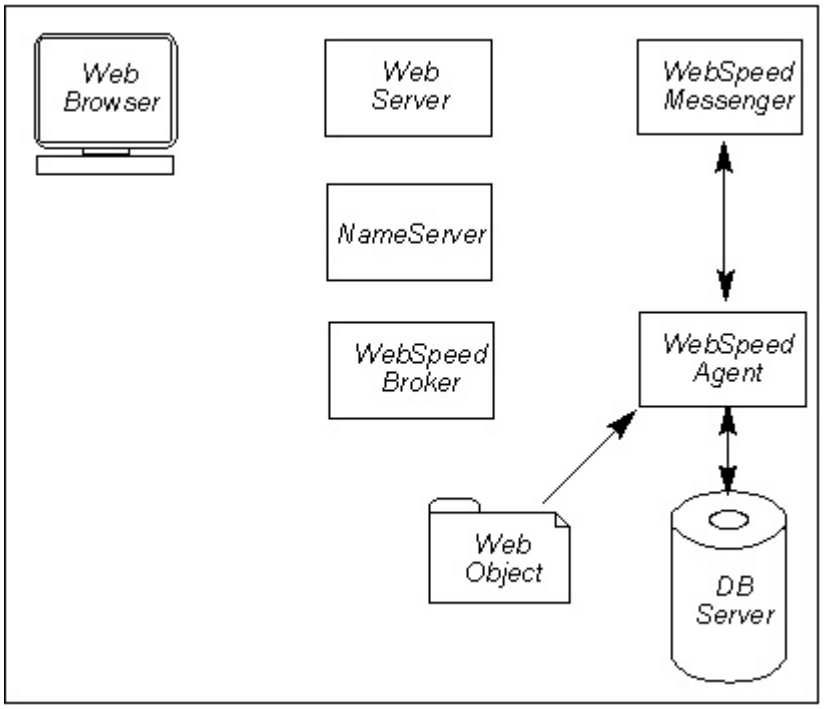
Kuva 6-7 (WebSpeed Developer's Guide 1998. 2-4).

5. WebSpeedin broker etsii vapaana olevan agentin ja antaa palvelupyynnön sen käsiteltäväksi.



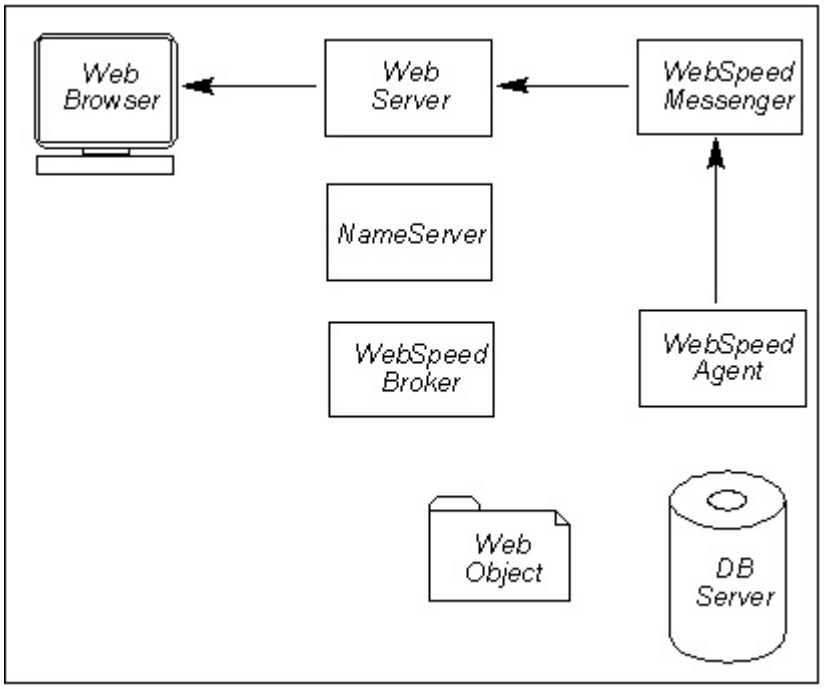
Kuva 6-8 (WebSpeed Developer's Guide 1998. 2-4).

6. Agentti ja messenger luovat yhteyden. Palvelupyynnössä halutun web-objektin nimi ja CGI-ympäristön tiedot välitetään agentille.



Kuva 6-9 (WebSpeed Developer's Guide 1998. 2-5).

7. WebSpeed agentti suorittaa web-objektin (ohjelman), mukaan lukien tarvittavat tietokantaluvut ja –kirjoitukset. Agentti ylläpitää yhteyden messengeriin koko web-objektin suorituksen ajan.



Kuva 6-10 (WebSpeed Developer's Guide 1998. 2-6).

8. WebSpeedin agentti palauttaa ajon tuloksena HTML-sivun messengerille, joka lähettää sen edelleen WWW-palvelimen kautta selaimelle. Agentti katkaisee yhteyden messengeriin ja päivittää tilansa brokerille.

HTTP-palvelupyynnön käsittelyn päätyttyä CGI-pohjainen messenger tuhoutuu - ISAPI- tai NSAPI-messengerit pysyvät muistissa mennen passiivisiksi. Periaatteeltaan kaikki palvelupyynnöt käsitellään edellä käsitellyllä tavalla. (WebSpeed Developer's Guide 1998. 2-2 – 2-6.)

6.2.1 WebSpeedin transaktiot

WebSpeedin transaktio on aika, jonka agentti on yhden selaimen käytössä. Agentin hallitaan ohjelmilla, joiden toimintaa voidaan kontrolloida ohjelmoimalla.

WebSpeedin transaktio voi kestää ainoastaan normaalin HTTP-transaktion ajan – WebSpeedin oletus - tai agentti voidaan lukita yhden selaimen käyttöön useiksi palvelupyynnöiksi, jolloin sovelluksen konteksti voidaan säilyttää samantapaisesti kuin tilan säilyttäviä protokollia käyttävissä sovelluksissa. (WebSpeed Developer's Guide 1998. 2-9.)

Toiminnallisesti WebSpeedin transaktio toteutetaan seuraavasti:

1. WebSpeed messengerin yhteyden luominen agenttiin tapahtuu, kun agentin hallintaohjelma vastaanottaa palvelupyynnön halutulle web-objektille. Web-objektille välitetään samalla myös CGI-ympäristön välittämät tiedot.
2. Hallintaohjelma suorittaa web-objektin.
3. Jos web-objekti lukitsee agentin, jää agentin hallintaohjelma web-objektin suorittamisen jälkeen odottamaan seuraava palvelupyyntöä samalta selaimelta tai lukituksen lopettavaa aikakatkaisua, joka asetetaan ohjelmallisesti. (WebSpeed Developer's Guide 1998. 2-9.)

Jos agentin hallintaohjelma lukitsee agentin, lähetetään brokerille viesti, että agentti on lukittuna (locked) ja se vastaanottaa palvelupyyntöjä ainoastaan nimetyltä selaimelta (cookie). WebSpeedin transaktion päätyttyä, agentti lähettää brokerille viestin, että se on jälleen muiden palvelupyyntöjen käytettävissä (available). Sovellusten lisensoinnin ja skaalautuvuuden takia pyritään lukittuja agentteja usein välttämään. (WebSpeed Developer's Guide 1998. 2-10, 8-2.)

6.3 Ohjelmointi WebSpeedillä

WebSpeedin kehitystyökalut koostuvat ohjelmointia tukevat editorin, koodia tuottavia velhoja (wizard), valmiita ohjelmapohjia (template) ja tietokannan tekemiseen tarvittavat työkalut. HTML-dokumentit luodaan erillisillä HTML-kehittimillä tai käyttäen ohjelmoijan valmiita HTML-dokumenttia pohjana. Kehitystyökaluilla tehdyt ohjelmat tallennetaan suoraan palvelimelle, jolloin ohjelmia voidaan myös suoraan testata kehityksen aikana. Ohjelmointityökalu toimii ainoastaan Windowsissa, tosin ohjelmia voidaan kehittää editorilla myös merkkipohjaisissa ympäristöissä, kuten unixissa.

Seuraavaksi luetellaan kaikki WebSpeed-ohjelmointiin kuuluvat ominaisuudet, joista osaa käsitellään vielä tarkemmin jatkossa:

- **SpeedScript:** Käytettävä 4GL-ohjelmointikieli. SpeedScript sisältää tekniikat WWW-palvelimen I/O-operaatioiden käyttöön. Niiden avulla saadaan CGI-ympäristön tiedot ohjelmien käyttöön ja vastaavasti myös WWW-palvelimelle.
- **WebSpeedin globaalit muuttujat:** Joukko muuttujia, joiden avulla saadaan tietoa web-palvelimesta, agentista ja CGI-ympäristömuuttujista.
- **WebSpeedin esiprosessorit:** Joukko esiprosessoreita ohjelmoinnin – erityisesti web-tulostuksen – helpottamiseksi.
- **API funktiot:** Alhaisen tason tehtäviä suorittavia funktioita, kuten URL:n formatointi tai yksittäisen CGI-muuttujan käsittely.
- **Method procedures:** WebSpeedin oletustoiminnan toteuttavat aliohjelmat, joista muutamat ovat muokattavissa.

- **Control handlers:** Mapped web object –tekniikassa käytettäviä tietyissä tapahtumissa suoritettavia erityisiä aliohjelmia.
- **Tagmap.dat:** HTML:n form-elementtejä vastaavat SpeedScriptin käyttöliittymäobjektit määrittelevä tiedosto, joka sisältää tiedon lukemisen ja tulostamisen kyseisten elementtien välillä käsittelevän tiedoston nimen.
- **Web-disp.p:** Agenttien hallintaohjelma, joka on keskeinen WebSpeedin toiminnalle.

(WebSpeed Developer's Guide 1998. 4-7 – 4-8).

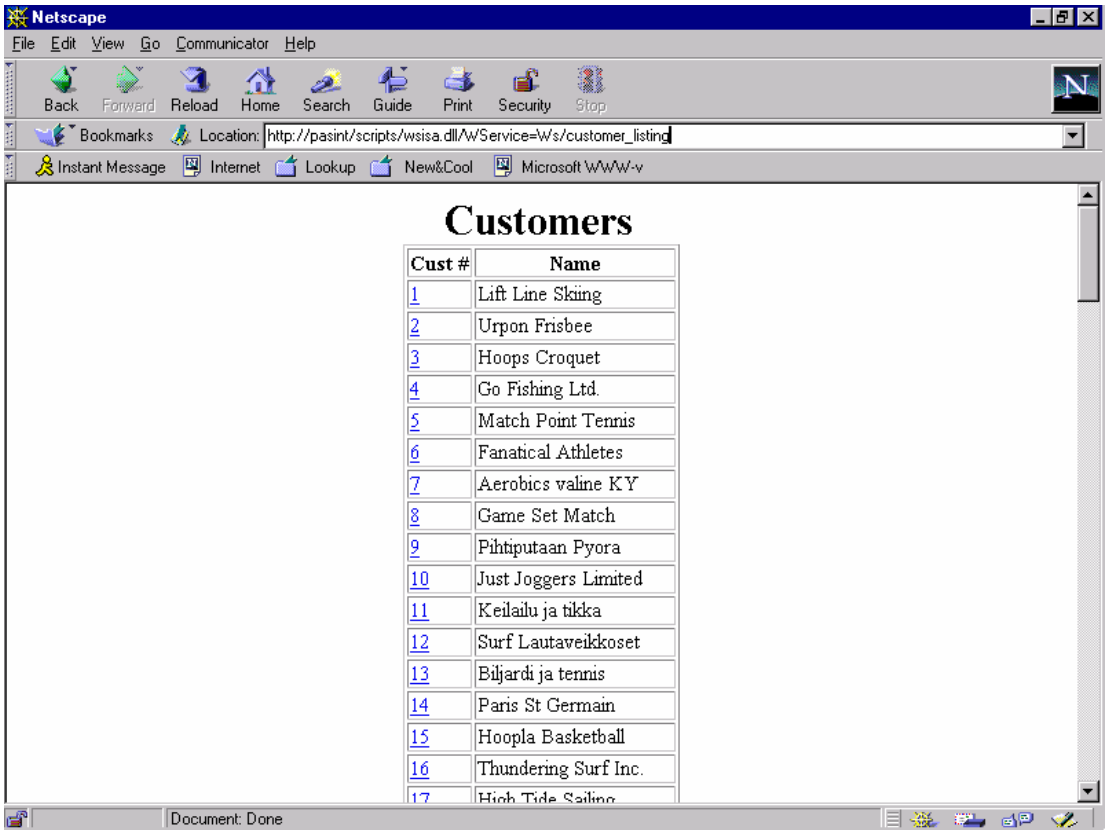
WebSpeedillä voidaan tehdä ohjelmia kolmella ohjelmointitekniikalla, jotka käydään läpi seuraavissa kappaleissa. Tekniikat ovat upotettu SpeedScript, HTML mapping ja CGI wrapper.

6.4 Upotettu (embedded) SpeedScript

Upotetussa SpeedScriptissä sovelluslogiikka koodataan HTML-dokumenttiin (WebSpeed Developer's Guide 1998, 5-2). Esimerkissä on upotettu sovelluslogiikka merkitty kursivoituna (valmis ohjelma Kuva 6-11):

```
<HTML><BODY bgcolor="white">
<CENTER>
<FONT SIZE=+3><B>Customers</B></FONT>
<TABLE BORDER=1>
<TR> <TH>Cust #</TH> <TH>Name</TH> </TR>
<!--WS4GL For each customer no-lock: -->
<TR>
  <TD> <A HREF=customer_update.r?CustID=`custnum`>`custnum`</A> </TD>
```

```
  <TD> `name` </TD>
</TR>
<!--WS4GL End. -->
</TABLE>
</CENTER>
</BODY>
```



Kuva 6-11

Upotettu SpeedScript näyttää tekniikkana samannäköiseltä esimerkiksi JavaScriptin kanssa, mutta niiden toimintaperiaatteessa on kuitenkin olennainen ero. JavaScript on osa HTML-dokumenttia, joka tulkitaan selaimessa. Sen sijaan SpeedScriptiä sisältävä HTML-sivu ajetaan WebSpeed-

palvelimessa, joka tulostaa HTML-sivun. (WebSpeed Developer's Guide 1998. 5-2.) WebSpeed tekee siis upotettua SpeedScriptiä sisältävästä HTML-dokumentista palvelimessa ajettavan WebSpeed-objektin. Ohjelman kääntämisessä WebSpeed kääntää koodin 'ylösalaisin', jolloin HTML-tagit ovat upotettuina SpeedScriptin sekaan (Liite 1). Käännösvaiheessa WebSpeed lisää automaattisesti myös valmiita aliohjelmia, jotka tulostavat mm. HTML-dokumentin otsikkotiedot. Ohjelmakoodiin upotetut HTML-tagit tulostetaan ajonaikaisesti, ja selaimelle lähetetään ainoastaan HTML-tagit ja data, kuten edellisestä esimerkistä (Kuva 6-11) otettu lähdekoodi näyttää (Kuva 6-12) (vertaa HTML-koodissa custnum ja name kenttiin).

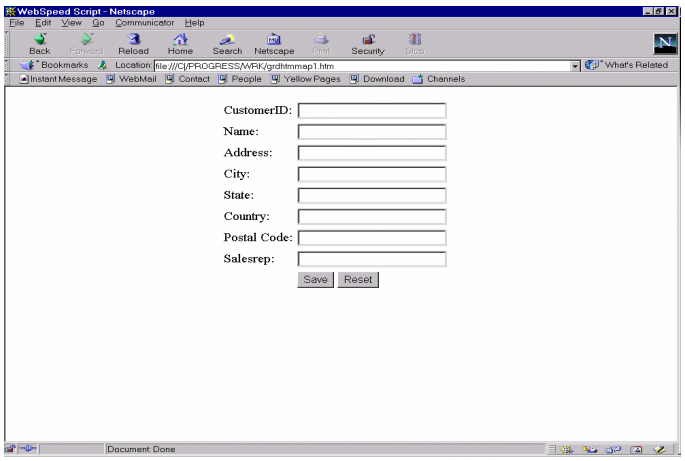
Kuva 6-12

WebSpeedin velhot tuottavat aina upotettua SpeedScriptiä ja jonkin verran JavaScriptiä yksinkertaista selaimessa tapahtuvaa validointia varten. Upotettu SpeedScript soveltuu parhaiten yksinkertaisiin, vähän sovelluslogiikkaa sisältäviin tietokantaohjelmiin.

6.5 HTML mapping

HTML mapping –tekniikka mahdollistaa sovelluslogiikan ja käyttöliittymän erottamisen erillisiin tiedostoihin (WebSpeed Developer's Guide 1998, 8-16). Ohjelmoija saa tai tekee erikseen HTML-sivun (FORM-elementteineen) (Kuva 6-13), josta WebSpeedillä kartoitetaan FORM-elementit muuttujiksi (tallennetaan offset-tiedostoon). Kartoituksen jälkeen sovelluslogiikka tehdään omaan tiedostoon. HTML-mapping tekniikka käyttää kolmea tiedostoa: HTML-dokumenttia, offset-tiedostoa ja sovelluslogiikkaa.

```
<HTML><head>
<meta name="AUTHOR" content="Your Name">
<title>WebSpeed Script</title>
</head><body>
<form method="post"><center><table>
<tr><td></td>
<td><input                    type="hidden"
name="RowID"></td>
</tr><tr><td></td>
```



Kuva 6-13 staattinen HTML-dokumentti ja sen lähdekoodi

Offset-tiedosto (Liite 2) sisältää HTML:n FORM-elementeistä nimen, FORM-elementin, tyypin, SppedScriptin vastaavan käyttöliittymäobjektin (graafisessa/merkkipohjaisessa ympäristössä) sekä FORM-elementin koordinaatit. Tietokannan kenttiin yhdistettyjen elementtien nimiksi tulee ohjelmissa suoraan tietokannan kenttien nimet. (WebSpeed Developer's Guide 1998, 8-29.) Kartoitetuista FORM-elementeistä tulee tavallisia ohjelmointikielen käsiteltäviä muuttujia. (WebSpeed Developer's Guide 1998, 8-29.)

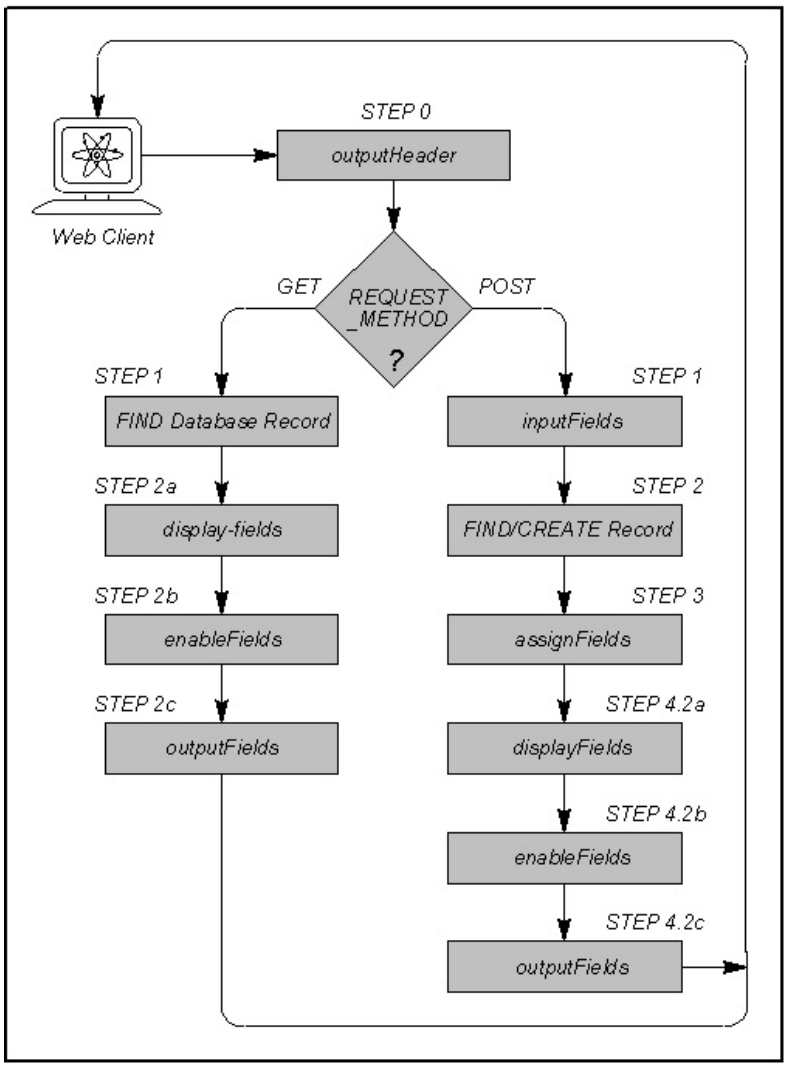
Myös varsinainen sovelluslogiikka tulee WebSpeedin luomaan erilliseen tiedostoon. WebSpeed generoi seuraavista osioista koostuvan sovelluslogiikka:

- **OutputHeader** tulostaa HTTP-otsikkotiedot ja asettaa web objektin haluttuun tilaan (lukittu vai ei) (WebSpeed Developer's Guide 1998, 8-16).
- **Process-web-request** käsittelee selaimelta lähetetyn datan (WebSpeed Developer's Guide 1998, 8-16). Ohjelmoijan tekemät muutokset logiikan suorittamisessa tehdään pääsääntöisesti process-web-request-aliohjelmaan.
- **HtmOffset** (ainoastaan luettavissa) assosioi HTML:n form-kentät WebSpeedin vastaavaan käyttöliittymäobjektiin (WebSpeed Developer's Guide 1998, 8-16).
- **Definitions** sisältää koko tiedostossa voimassa olevat muuttujamäärittelyt, ohjelman vastaanottamat parametrit ja esiprosessorimäärittelyt. Huomioitavaa on, että parametrejä voidaan ottaa vastaan ainoastaan toiselta 4GL-ohjelmalta, ei selaimesta lähetettävästä datasta ! (WebSpeed Developer's Guide 1998, 8-16.)
- **Main Code Block** on pääohjelma, jossa alustetaan web objekti ja käynnistetään process-web-request-aliohjelma (WebSpeed Developer's Guide 1998, 8-16).
- **Control Handlers** ovat erikoisaliohjelmia, jotka suoritetaan ainoastaan tiettyjen tapahtumien yhteydessä. Niillä voidaan ylittää FORM-elementin lukemisen tai kirjoittamisen oletustoiminta. Jokaisella FORM-elementillä on oma control handler. (WebSpeed Developer's Guide 1998, 8-16.)

- **ADM metodit (event procedures)** määrittävät kartoitetun web objektin oletustoiminnot. Ohjelmien toimintaa muokataan usein tekemällä näistä ohjelmista paikallisia versioita, joko täydentämään oletustoimintaa tai korvaamaan sen kokonaan (Liite 3). (WebSpeed Developer's Guide 1998, 8-16.)

Control handlereiden oletustoiminto määräytyy muokattavasta tagmap.dat-tiedostossa (Liite 6), jossa määritellään niiden oletustoiminnan tekevät apuohjelmat. Mikäli halutaan tehdä oma FORM-elementin tulostuksen käsittely, tehdään paikallinen control handler tai mikäli halutaan muuttaa koko ajoympäristössä tapahtuva FORM-elementin käsittelyn oletustoiminta, muutetaan tagmap.dat-tiedostossa määriteltyä apuohjelmaa. (WebSpeed Developer's Guide 1998, 8-26, 8-30.)

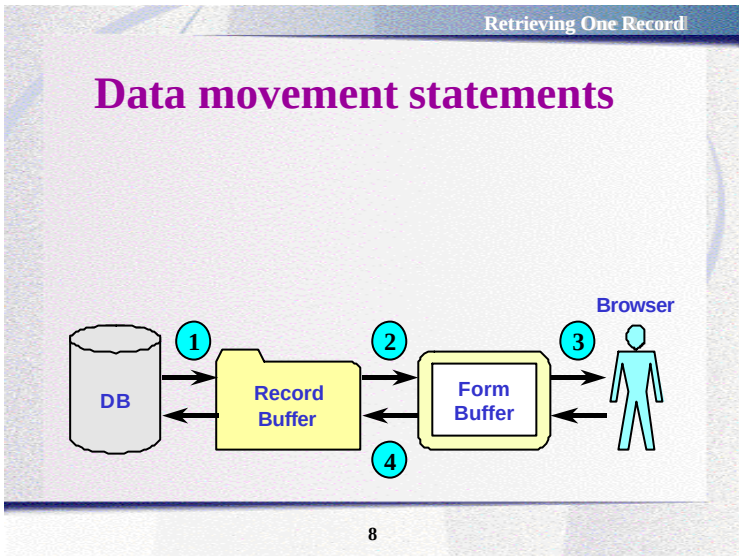
Process-web-request-aliohjelma (Kuva 6-14) muodostaa rungon HTML mapping -tekniikan toiminnalle. Process-web-request sisältää selaimelta palvelimelle lähetetyn HTML-dokumentin käsittelyssä tarvittavan logiikan oletustoimintoina. Se toiminnallisuus jakautuu kahteen haaraan GET- tai POST-metodin perusteella. Tyypillisesti GET-haara ajetaan, kun ohjelmaa kutsutaan ensimmäistä kertaa, tavallisesti ohjelmaan on tällöin viitattu suoraan URL:sta. POST-haaraan taas mennään, kun käyttäjä lähettää selaimesta FORM-kenttiin syöttämänsä tiedot.



Kuva 6-14 process-web-request algoritmi (WebSpeed Developer's Guide 1998, 8-17).

Datan liikkuminen selaimen ja tietokannan välissä tapahtuu tapahtumaproseduureilla (event procedure). Asian selventämiseksi käydään ensin läpi Progressin muistinkäsittely. Progress jakaa muistin kahteen puskuriin: form- ja record-puskuriin (buffer) (Kuva 6-15). Form-puskuri sisältää näyttömuistissa olevan datan. Record–puskuri on ohjelman sisäisesti käyttämä muistialue. Kun tietokantaa päivitetään, muutetaan tietuetta aina record-

puskurissa. Tietojen siirtyminen näiden buffereiden välillä tapahtuu SpeedScriptin lauseilla ja/tai tapahtumaproseduureilla. Ohjelmoija voi käsitellä dataa molemmissa puskureissa.



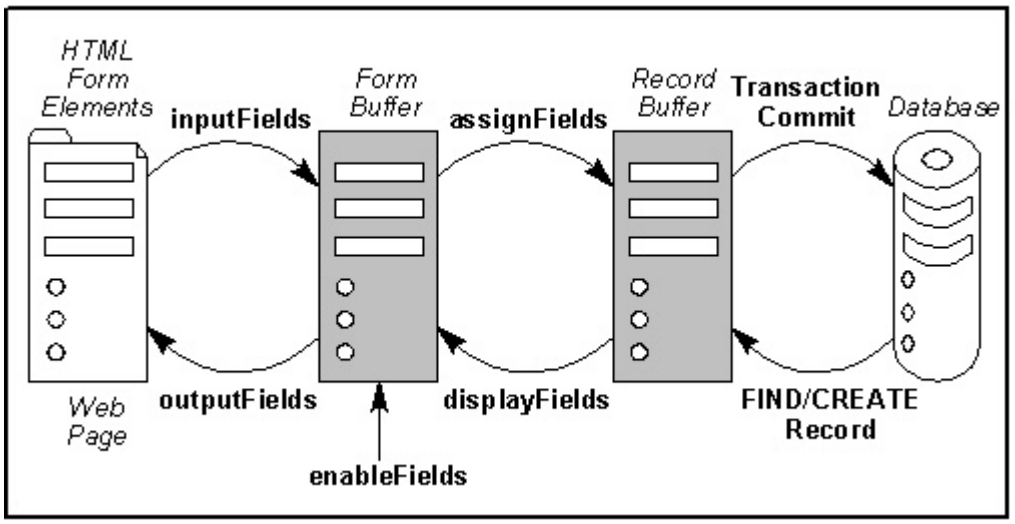
Kuva 6-15 Tiedon liikkuminen puskureiden välillä.

Tapahtumaproseduureja ovat:

- **outputHeader:** Tulostaa HTTP otsikon.
- **inputFields:** Siirtää selaimelta tulevien form-elementtien arvot 4gl:n vastaavien käyttöliittymäobjektien arvoiksi form bufferissa. 4gl:n vastaavat käyttöliittymäobjektit määräytyvät tagmap.dat tiedostossa.
- **findRecords:** Hakee oletuksena tietokantakyselyn ensimmäisen tietueen ja tuo tietueen record bufferiin.
- **assignFields:** Siirtää näytöltä inputFieldsillä tulleen datan muuttujien arvoiksi record bufferiin.
- **displayFields:** Siirtää datan record bufferista form bufferiin..

- **enableFields:** Aktivoi form bufferissa olevat objektit, joiden sensitive attribuutin arvoksi on asetettu true.
- **outputFields:** Tulostaa form bufferissa olevat arvot vastaaviin HTML-elementteihin. HTML-sivu määrää tulostuksen ulkoasun, tosin mm. sensitive-attribuutilla voidaan ohittaa input-määreen tulostaminen.

HTML mapping –tekniikassa datan liikkuminen toteutetaan pääsääntöisesti tapahtumaproseduureilla (Kuva 6-16). Ohjelman muokkaaminen tehdään useimmiten muuttamalla tapahtumaproseduureja ja/tai niiden suoritusjärjestystä (esimerkki Liite 4). (WebSpeed Developer's Guide 1998, 8-18, 8-20 – 8-22.)



Kuva 6-16 Event procedures (WebSpeed Developer's Guide 1998, 8-20).

Mapped web object -sovelluksissa käytetään usein vakiomuotoista käyttöliittymää, eli kartoitettua HTML-sivua.

CGI wrapper (Liite 5) ohjelmapohja sisältää tulostettavat HTML-tagit, siksi menetelmää voisi sanoa upotetun SpeedScriptin vastakohdaksi. Kuten upotetussa SpeedScriptissä myös CGI wrapperia käytettäessä ohjelmoijan täytyy hallita sekä HTML-kieli että SpeedScript. Tyypillisesti CGI wrapperia käytetään jos käyttöliittymä muodostetaan aina dynaamisesti, kuten erilaiset raportit ja ostoskorit. CGI wrapper antaa ohjelmoijalle joustavimman tavan tulostaa ajonaikaisesti erilaisia käyttöliittymiä, joka on hyödyllistä mm. tehtäessä kansainvälisiä, monikielisiä sovelluksia. Tekniikan huono puoli on, että se yhdistää käyttöliittymän ja sovelluslogiikan yhteen tiedoston. (WebSpeed Developer's Guide 1998, 8-15.)

6.6 Tiedon liikuttaminen API-funktioilla

Upotetulla SpeedScriptillä ja cgi wrapperilla tehdyissä ohjelmissa täytyy selaimelta lähetetty syöttö- ja tulostustieto ohjelmoida API-funktioilla. WebSpeedissä voidaan käyttää mm. get-value() funktiota, jolle annetaan parametrinä selaimesta merkkijonona lähtevästä datasta halutun muuttuja *name* (kts. Luku 4). API-funktiot suorittavat tiedon liikuttamisen WWW-palvelimen ja WebSpeed-ohjelman välillä. HTML-mapping -tekniikassa hyödynnettiin tapahtumaproseduureihin ohjelmoitua oletustoimintaa, jotka käyttävät samoja API-funktioita. (WebSpeed Developer's Guide 1998, 8-26.)

6.7 WebSpeedin transaktioiden ohjelmointi

WebSpeedin transaktio kesti oletuksena yhden HTTP-transaktion. Ohjelmoija pystyy myös määrittelemään WebSpeedin transaktion pituuden, joka

toteutetaan cookie-tekniikan avulla lukitsemalla agenttiprosessin yhdelle selaimelle. Tilan säilyttävää agenttia voidaan käyttää, jos sovelluksen halutaan toimivan kuten tilan säilyttäviä protokollia käyttävät sovellukset, eli agentti säilyttää yksittäisen käyttäjän ajaman sovelluksen kontekstin. Agentin lukinneen selaimen kaikki pyynnöt menevät samalle agentille siitä riippumatta, ovatko muut ajettavat ohjelmat tilan säilyttäviä vai eivät. Jos selaimelta lähetetään palvelupyyntö samalle brokerille käyttäen kuitenkin eri messengeriä, toteutetaan palvelupyyntö toisella agentilla. (WebSpeed Developer's Guide 1998. 8-2, 8-3, 8-6.)

Esimerkki

Tehdään ohjelma, jolla voidaan selailla ja päivittää asiakastietueita. Käyttäjän selaillessa asiakastietueita, saadaan tilan säilyttävissä protokollissa seuraava tietue suoraan hakemalla muistissa olevasta tietueesta seuraava. Tilattomassa ohjelmassa taas täytyy ensiksi hakea edellisellä kerralla käytetty tietue, jonka jälkeen vasta voidaan hakea seuraava. Alla on esimerkkikoodit molemmista tapauksista ja koodista, kun käyttäjä painaa next-nappulaa seuraavan tietueen hakemiseksi tietokannasta:

Tilan säilyttävässä ohjelmassa

```
FIND NEXT customer.
```

Tilattomassa ohjelmassa

```
FIND customer WHERE customer.custnum = get-value(”custnum”).
IF AVAILABLE customer THEN FIND NEXT customer.
```

Get-value on API-funktio, jolla saadaan selaimesta lähetetty custnum-tieto.

Tilan säilyttävän sovelluksen suurin hyöty on muuttujien ja tietorakenteiden arvojen säilymisessä useiden palvelupyyntöjen yli, mistä on erityisesti hyötyä haluttaessa päivittää tietokantaa useista palvelupyynnöistä muodostuvassa tietokantatransaktiossa. Sen käyttö myös minimoi verkossa liikuteltavan datan määrän, koska käytettävä data pidetään palvelimen muistissa niin pitkään kuin sitä tarvitaan. (WebSpeed Developer's Guide 1998. 8-4.)

Esimerkki

Käyttäjä haluaa ostaa internet-kaupasta tietokoneen, jossa hänen tulee määritellä itse haluamansa osat. Osien valinta tehdään usealla eri ohjelmalla, joista jokaisesta saadaan valituksi yksi koneen osa. Jotta tietokoneen tilaus voidaan hyväksyä, täytyy käyttäjän tehdä jokaisen osan kohdalla valinta. Käyttäjän tilaus lähtee kauppaan vasta siinä vaiheessa, kun käyttäjä on koonnut hyväksyttävän paketin ja hänellä voidaan näyttää lopullinen hinta. Tässä tapauksessa tietokantaa päivitetään ainoastaan, jos käyttäjä tekee lopullisen hyväksynnän, eikä jokaisen osan kohdalla erikseen. Tilaus viedään tietokantaan joko kokonaisuudessaan tai ei lainkaan. Jokainen valittu osa pysyy nyt agentin muistissa, kunnes tietokantatransaktio päätetään.

Edellisen esimerkin tekeminen tilattomalla agentilla vaatii ohjelman kontekstin säilyttämistä joko viemällä kaikki käyttäjän valinnat jokaisen palvelupyynnön yhteydessä selaimen ja palvelimen välillä tai tekemällä tietokantaan roskataulu

kontekstin tallentamiseksi. Tilatonta agenttia käyttämällä ei kuitenkaan ole mahdollista lukita tietueita yhden palvelupyynnön yli.

Agentin lukitsemisen haittapuoliakin taas ovat:

- Agentti varataan yhden selaimen käyttöön, jolloin se ei voi palvella muita käyttäjiä. Tällöin ajoympäristössä saatetaan tarvita useampia agentteja, mikä lisää järjestelmän hintaa ja palvelimeen kohdistuvia vaatimuksia.
- Ohjelmoijan täytyy olla huolellisempi tietueiden hallinnassa – lukituksissa ja tietokantatransaktioissa.

Usein ohjelmoitaessa Internetiin tulisi sovelluksen skaalautuvuuden takia välttää lukittuja agentteja. (WebSpeed Developer's Guide 1998. 8-4.)

WebSpeedin transaktioiden hallinnan tekee agentin hallintaohjelmalla, jonka toimintaa voidaan kontrolloida ohjelmoimalla. Agentti lukitaan ohjelmallisesti set-web-state –metodilla, joka ajetaan ennen HTTP:n otsikkotietojen lähettämistä. Metodi tekee kaksi asiaa:

1. Asettaa cookien, jolla broker tunnistaa käytetyn selaimen, agentin ja ohjelman, johon WebSpeedin transaktio liittyy.
2. Asettaa aikakatkaisun agentin vapauttamiseksi, mikäli käyttäjä ei tee mitään.

(WebSpeed Developer's Guide 1998. 8-2.)

WebSpeed transaktio päätetään samalla metodilla, jolla se käynnistettiinkin laittamalla transaktion kestoksi 0, tai odottamalla aikakatkaisua. (WebSpeed Developer's Guide 1998. 8-3.)

6.8 Kansainvälisyys

Kansainvälisen sovelluksen tekeminen edellyttää tietokannalta ja sovellukselta eri kielten käyttämien merkistöjen tukea. Progressin tietokanta käyttää unicodea, joka mahdollistaa kaikkien yleisten kielten käytön. Myös sovellusta ajavan komponentin – WebSpeedissä agentin - täytyisi pystyä käyttämään unicodea, mikä ei kuitenkaan ole tällä hetkellä vielä mahdollista.

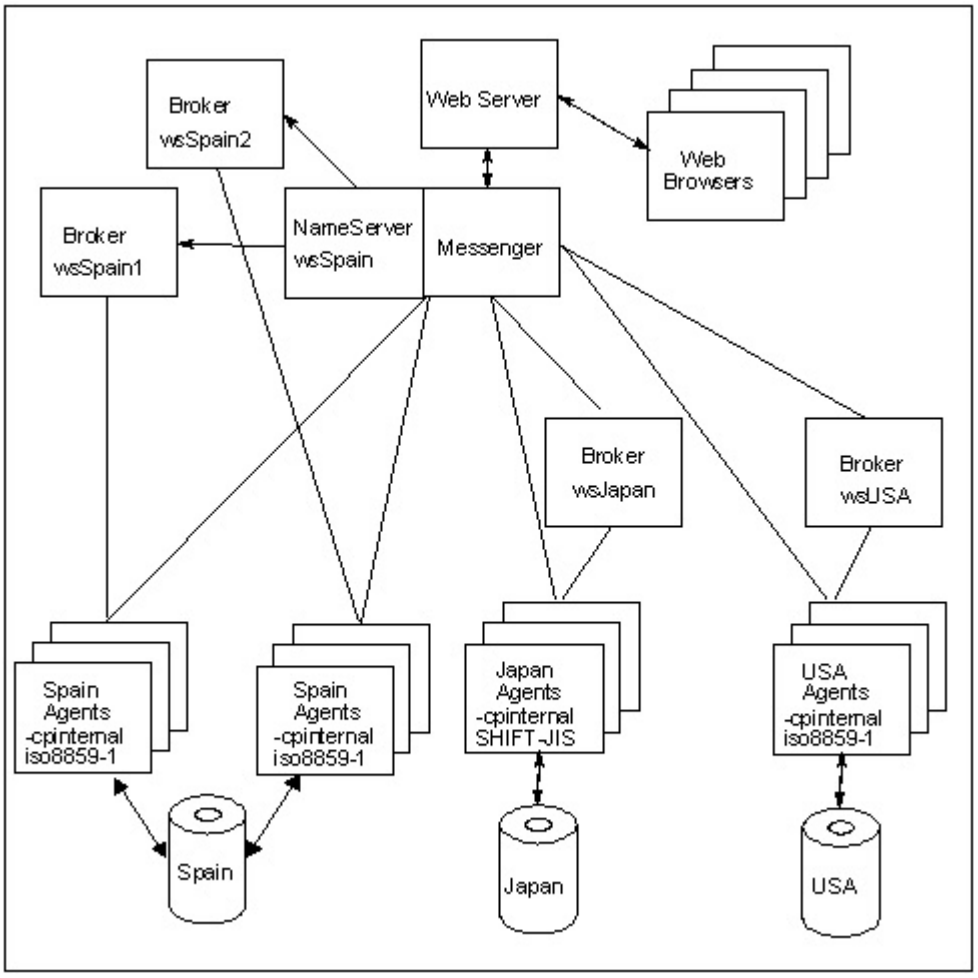
Monikielinen eri merkistöä käyttävä WebSpeed-sovellus täytyy toteuttaa eri ajoympäristöillä (Kuva 6-17), koska agentti pystyy käyttämään ainoastaan yhtä merkistöä. Erikielisiin brokereihin viitataan erilaisilla URL-osoitteilla:

japani: http://mars/wsnsa.dll/WService= wsJapan/iwebapp

espanja: http://mars/wsnsa.dll/WService= wsSpain/iwebapp

englanti: http://mars/wsnsa.dll/WService= wsUSA/iwebapp.

Selkeä heikkous WebSpeedillä toteutetuissa monikielisissä sovelluksissa on ajoympäristön moninkertainen ylläpito. (WebSpeed Installation and Configuration Guide 1998, 5-8, 5-11.)



Kuva 6-17 Monikielisen sovelluksen konfiguraatio (WebSpeed Installation and Configuration Guide 1998, 5-9).

6.9 Tietoturva

WebSpeediin ei varsinaisesti ole rakennettu tietoturvaratkaisua vaan lähtökohtana on, että tietoturvaratkaisu toteutetaan ulkopuolisilla välineillä lähinnä WWW-palvelimen ja selaimen välissä. WebSpeed-käyttäjällä on periaatteessa mahdollisuus ajaa kaikkia ohjelmia, jotka löytyvät brokerin polusta. On myös mahdollista, että käyttäjä tekee itse ajettavan WebSpeed-ohjelman. Käynnistysparametreillä voidaan estää kuitenkin kääntämättömien

ohjelmien ajamisen, tosin tästä voi olla joissain tapauksissa haittaa ohjelmoinnillisissa ratkaisuissa. Tietoturvassa lähtökohtana täytyy olla, että brokerin polussa on ainoastaan ’turvallisia’ ohjelmia.

Usein tietokantasovellusten käyttämiseen vaaditaan käyttöoikeus. Käytettäessä HTTP-protokollaa täytyy käyttöoikeus tarkistaa jokaisen palvelupyynnön yhteydessä. Sovelluksen tasolla tehtynä tämä on työlästä, koska tarkastus täytyisi tehdä jokaisen ohjelman yhteydessä. Käyttöoikeuden tarkastus toteutetaankin usein webbipalvelimessa. Tämän ratkaisun heikkous on kuitenkin, että se on palvelinohjelmistokohtainen. Internet-tietoturvassa on olennaista myös verkossa liikuteltavan tiedon salaus – kryptaus. Tämä voidaan myös toteuttaa WWW-palvelimen ja selaimen välillä. (WebSpeed Developer’s Guide 1998. 10-5).

7. Monikielinen interaktiivinen tietokantasovellus WebSpeedillä

Tietokantaa päivitettävässä sovelluksessa kontekstin säilyttäminen on erityisen tärkeää. Koska HTTP-protokolla on tilaton, ei tietokannasta näytöllä näkyvä tieto ole välttämättä sama kuin samanaikaisesti tietokannassa oleva. Seuraavissa kappaleissa tarkastellaan tietokantasovellusten ohjelmointia WebSpeedin ero tekniikoilla.

7.1 Lukitukset

Tilan säilyttävässä ohjelmassa tietokannan lukituksia käytetään kuten tavallisissa graafisissa tai merkkipohjaisissa asiakas/palvelin-sovelluksissa, eli ohjelmoijalla on käytettävissä kaikki tietokannan normaalit lukitusmekanismit. Progressin relaatiotietokannassa on kolme lukitustasoa: NO-LOCK, SHARE-LOCK ja EXCLUSIVE-LOCK (Taulu-1). Vastaavat lukitukset ovat myös muissa relaatiotietokannoista.

NO-LOCK:ssa tietue ei ole lukittu, vaan se on haettu tietokannasta ainoastaan luettavaksi, eli tietuetta ei voida päivittää. Tietue saadaan aina luettua NO-LOCK:lla. SHARE-LOCK estää muita käyttäjiä päivittämässä tietuetta samanaikaisesti ja se voidaan tarvittaessa ylentää päivityslukitukseen. Tietue saadaan SHARE-LOCK:iin, mikäli yhdelläkään toisella käyttäjälle ei ole tietuetta EXCLUSIVE-LOCK:ssa. EXCLUSIVE-LOCK on päivityslukko.

	NO-LOCK	SHARE-LOCK	EXCLUSIVE-LOCK
NO-LOCK	Ok	Ok	Ok
SHARE-LOCK	Ok	Ok	Yhteentörmäys
EXCLUSIVE-LOCK	Ok	Yhteentörmäys	Yhteentörmäys

Taulu-1 Lukitusten väliset yhteentörmäykset

7.1.1 Optimistinen lukitusstrategia

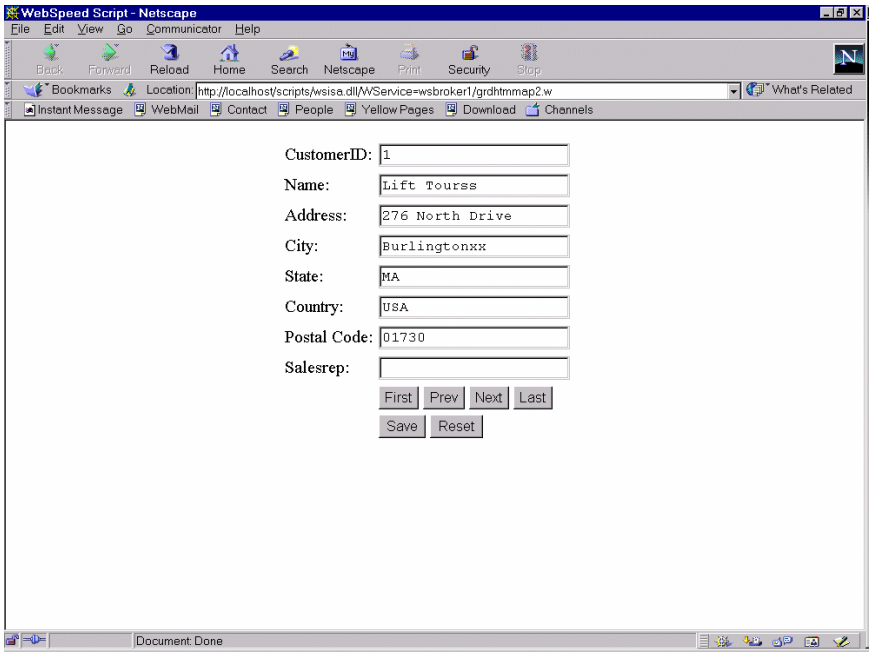
Monen käyttäjän ympäristössä ohjelmoijan tulee useimmiten pyrkiä pitämään tietueet mahdollisimman monen käyttäjän käytössä samanaikaisesti. Lukitusten käytössä pitäisi pyrkiä pitämään aina mahdollisimman alhaisen tason lukitus, eli ylemmän tason lukitus otetaan mahdollisimman myöhäisessä vaiheessa ja mahdollisimman lyhyeksi ajaksi. Usein tietueen lukitus kannattaa tehdä vasta, kun käyttäjä on syöttänyt tiedot ja lähettää päivitykset tietokantaan (Campbell 1997,84).

Optimistisen lukitusstrategian kääntöpuolena on, että käyttäjä voi joutua syöttämään tiedot uudestaan. Ongelmakohtia optimistisessa lukituksessa ovat:

1. Toinen käyttäjä käy päivittämässä tietueita, jolloin toinen tekee muutokset vanhojen tietojen pohjalta.
2. Toinen käyttäjä lukitsee tietueen.
3. Toinen käyttäjä tuhoaa tietueen.

7.2 Esimerkkiohjelma

Käsitellään tietueiden päivitystä esimerkkiohjelman avulla: käyttäjä käy läpi asiakastietueita ja päivittää niitä halutessaan. Tekniikkana käytetään HTML-mappingiä (HTML-sivu: Kuva 7-1).



Kuva 7-1 Esimerkkiohjelman HTML-sivu

7.2.1 Esimerkkiohjelman ohjelmointi lukitulla agentilla

Lukittua agenttia käytettäessä ohjelmoijan täytyy valita tietueiden lukitusstrategia. Koska tyypillisesti internetissä on useita samanaikaisia käyttäjiä, kannattaa useimmiten valita optimistinen lukitusstrategia, jolloin estetään yksittäisestä käyttäjästä mahdollisesti aiheutuvat ongelmat. Tietueiden lukemisessa käytetään oletuslukituksena NO-LOCK:ia ja kun käyttäjä haluaa päivittää tietuetta, haetaan se uudestaan käyttäen EXCLUSIVE-LOCK:ia. Aluksi ohjelmasta tehdään tilan säilyttävä, mikä tehdään HTTP:n otsikkotiedot

asettavassa outputHeader–aliohjelmassa asettamalla agentti lukituksi parametrillä määrättäväksi ajaksi ajamalla `setWebState (2.0)`.

Tietueiden navigointi toteutetaan lokalisoimalla findRecords-tapahtumaproseduuri:

```
/*------------------------------------------------------------------
  Purpose:     Super Override
  Parameters:
  Notes:
------------------------------------------------------------------*/
/* SUPER avaa tietokantakyselyn. Tässä ohjelmassa käytetään lukittua
  ** agenttia, joten kysely avataan vain kerran */
IF REQUEST_METHOD = "GET":U THEN RUN SUPER.
CASE get-value("NavButton"):
      WHEN "First" THEN
        RUN fetchFirst.
      WHEN "Next" THEN
        RUN fetchNext.
      WHEN "Prev" THEN
        RUN fetchPrev.
      WHEN "Last" THEN
        RUN fetchLast.
END CASE.
END PROCEDURE.
```

Muutetaan process-web-request –aliohjelmaa siten, että se pystyy päivittämään tietuetta lisäämällä sinne seuraava koodi:

```
IF get-value("button") = "Save" THEN RUN assignFields.
```

Lopuksi lokalisoidaan assignFields:

```
/*------------------------------------------------------------------
  Purpose:     Super Override
  Parameters:
  Notes:
------------------------------------------------------------------*/
DEF VAR rowCustomer AS ROWID NO-UNDO.
ASSIGN rowCustomer = ROWID(customer).
/* Täytyy tutkia onko joku toinen käyttäjä muuttanut tietuetta. */
FIND CURRENT customer EXCLUSIVE-LOCK NO-ERROR NO-WAIT.
    IF AVAILABLE customer THEN
        IF CURRENT-CHANGED(customer) THEN DO:
              {&OUT} "Record changed by another user !  Displaying  new
                     record. <BR> Please re-write your modifications.".
            FIND CURRENT customer NO-LOCK NO-ERROR NO-WAIT.
            RETURN.
        END.
/*  SUPER  tekee  varsinaisen  tietuen  päivityksen.  Se  myös  selvittää
**  miksi tietuetta ei saatu. Jos ei, onko tuhottu vai lukittu ? */
RUN SUPER.
/* Virhe päivityksessä, tietuetta ei saatu record bufferiin. */
IF NOT AVAILABLE customer THEN
   FIND customer WHERE ROWID(customer) = rowCustomer NO-LOCK NO-ERROR.
END PROCEDURE.
```

AssignFields hoitaa tietueen varsinaisen päivityksen ja virheenkäsittelyn. Koska lukitusstrategiana käytetään optimistista lukitusta, täytyy haettaessa tietue EXCLUSIVE-LOCK:iin tarkistaa

1. Saatiinko tietue ? Jos ei, niin miksi ? Onko se tuhottu vai lukittu ?
2. Onko tietuetta muutettu ?

Nämä tarkistukset täytyy aina tehdä varsinaisen päivityksen yhteydessä. Ne pitäisikin olla WebSpeedin assignFieldsin oletustoiminnassa, mutta siellä ilmeni seuraavassa päivityksessä korjattava ohjelmointivirhe.

Erityisesti on myös huomioitava vapauttaa tietueen lukitus päivityksen jälkeen. Muussa tapauksessa tietueeseen jää ns. left-over lukko, joka pitää tietueen SHARE-LOCK:ssa, kunnes se vapautettaisiin aikakatkaisun seurauksena tai käyttäjä hakisi seuraavan tietueen. Edellisessä esimerkissä asiaa yksinkertaistaa jokaisen tietueen päivittäminen erillisessä tietokantatransaktiossa. Toinen huomioitava seikka tilan säilyttävä ohjelmassa on, kun samalta työasemalta päivitetään kahdella samalla ohjelmalla samaa tietuetta. Tällöin ohjelmista ei luoda agentissa erillisiä esiintymiä, joilla olisi oma konteksti, joten CURRENT-CHANGED –funktio ei toimi. Tämä johtuu siitä, että selaimen tunnistus tapahtuu cookien avulla, joka sisältää työaseman IP-osoitteen ja ajettavan web-objektin nimen.

7.2.2 Esimerkkiohjelman ohjelmointi lukitsemattomalla agentilla

Valitulla lukitusstrategialla ei ole vastaavaa merkitystä, jos ohjelmasta ei tehdä tilan säilyttävää. Tällöin toiminta menee HTTP-protokollan oletustoiminnan mukaisesti eli selaimelta lähetetään palvelupyyntö, palvelin toteuttaa sen ja yhteys katkeaa. Käyttäjä ei voi vaikuttaa lukituksen kestoon. Toisaalta sovellus ei myöskään säilytä kontekstiään, vaan sovelluksen tila täytyy välittää jollakin tekniikalla seuraavaan sovelluksen palvelupyyntöön. Sovelluksen tilan säilyttäminen voidaan toteuttaa WWW-ympäristön perustekniikoilla, kuten hidden-field, URL ja cookie. Tietokantasovelluksissa ohjelman tilan voidaan säilyttää myös tietokantaan tehdyllä 'roskataululla'.

Ainoastaan tietueiden selailuunkin käytettävälle ohjelmalle täytyy palvelupyyntöjen mukana välittää aina yksiselitteisesti käytetyn tietueen identifioiva tieto. Progress-tietokannassa voidaan käyttää ROWID-arvoa, joka on tietueen osoite kovalevyllä. Tällainen tieto voidaan helposti viedä hidden-fieldin avulla.

Tehdään vastaava ohjelma kuin edellisessä esimerkissä (Luku 7.2.1) käyttäen tilatonta ohjelmaa. Määritellään HTML-sivuun piilokenttä, johon tallennetaan käsiteltävän tietueen ROWID. HTML-sivun kartoituksen jälkeen piilokenttää voidaan käyttää, kuten muuttujaa. Ohjelman muokkaaminen aloitetaan tekemällä vastaava lokaali versio findRecordsista.

```
/*------------------------------------------------------------------
Purpose:     Super Override
```

```
  Parameters:
  Notes:
------------------------------------------------------------------------*/
/* Code placed here will execute PRIOR to standard behavior. */
RUN SUPER.
/* Code placed here will execute AFTER standard behavior.    */
CASE get-value("NavButton"):
      WHEN "First" THEN
        RUN fetchFirst.
      WHEN "Next" THEN
        RUN fetchNext.
      WHEN "Prev" THEN
        RUN fetchPrev.
      WHEN "Last" THEN
        RUN fetchLast.
END CASE.
END PROCEDURE.
```

Erona lukitun agentin käyttöön on, että nyt täytyy huolehtia myös tietueen ROWID-arvon välittämisestä selaimelle aina, kun HTML-sivu tulostetaan. Tämä voidaan tehdä lokalisoimalla displayFields.

```
/*------------------------------------------------------------------------
  Purpose:     Super Override
  Parameters:
  Notes:
------------------------------------------------------------------------*/
/* Code placed here will execute PRIOR to standard behavior. */
/* Asetetaan muuttujan ab_unmap.RowID arvoksi record bufferissa
  ** tällä hetkellä olevan tietueen ROWID ennen kuin
```

```
  ** arvot siirretään form bufferiin. */
ASSIGN ab_unmap.RowID = STRING(ROWID(customer)).
RUN SUPER.
/* Code placed here will execute AFTER standard behavior.    */
END PROCEDURE.
```

Vastaavasti tietueen ROWID-arvo täytyy lukea selaimelta lähetetystä tiedosta. ROWID:n avulla haetaan record-puskuriin selaimessa käsitelty tietue. Sen arvon lukeminen tehdään control handlerilla, joka laukeaa ajettaessa inputFields tapahtumaproseduuri.

```
PROCEDURE ab_unmap.RowID.input .
/*------------------------------------------------------------------
Purpose:     Assigns form field data value to frame screen value.
  Parameters:  p-field-value
  Notes:
------------------------------------------------------------------*/
  DEFINE INPUT PARAMETER p-field-value AS CHARACTER NO-UNDO.
  DO WITH FRAME {&FRAME-NAME}:
      /* store the rowids that are used in fetchcurrent */
      setCurrentRowids (p-field-value).
  END.
END PROCEDURE.
```

Ohjelma toimii nyt tietueiden selailuun. Laitetaan myös päivitys toimimaan ihanteellisissa olosuhteissa. Ihanteelliset olosuhteet tarkoittavat, että samaa tietuetta ei päivitä useampi käyttäjä samanaikaisesti. Muutetaan process-web-request –aliohjelmaan seuraavasti:

```
    ASSIGN chrAction = get-value("button").
    /* STEP 2 - Open the database or SDO query and and fetch the first
     * record. */
    RUN findRecords.
    /* STEP 3 - AssignFields will save the data in the frame.
      * (it automatically upgrades the lock to exclusive while doing
     * the update)
    */
    IF chrAction = "Save" THEN RUN assignFields.
```

Lokalisoidaan assignFields:

```
/*------------------------------------------------------------------------
  Purpose:     Super Override
  Parameters:
------------------------------------------------------------------------*/
  /*  Haetaan  tietue  record  bufferiin  käyttäen  control  handlerissa
  ** asetettua ROWID-arvoa. */
  RUN fetchCurrent.
  /* Code placed here will execute PRIOR to standard behavior. */
  RUN SUPER.
  /* Code placed here will execute AFTER standard behavior.    */
  /*  Jos  assignFields  epäonnistui,  haetaan  tietue  uudestaan  HTML-
  ** tulostusta varten record bufferiin. */
  IF NOT AVAILABLE customer THEN RUN fetchCurrent.
END PROCEDURE.
```

Nyt ohjelmaa voidaan käyttää tietueiden selailuun ja päivitykseen ihanteellisissa oloissa. Päivitys voi kuitenkin aiheuttaa ongelmia, jos useampi käyttäjä päivittää

samanaikaisesti samaa tietuetta. Tilattomassa ohjelmassa CURRENT-CHANGED –funktio ei toimi, koska ohjelmalla ei ole vertailuun tarvittavaa tietuetta muistissa, jolloin CURRENT-CHANGED ei aiheuttaisi koskaan virhetilannetta. Alla on ongelma kuvattuna vaiheittain:

1. Ohjelma hakee tietokannasta tietueen ja tulostaa datan HTML-sivuksi. Tämän jälkeen ohjelma konteksti tuhoutuu.
2. Käyttäjä päivittää tietoja ja lähettää HTML-formin ohjelmalle.
3. Ohjelma käynnistyy uudelleen ja hakee päivitettävän tietueen tietokannasta. Ohjelmalla ei ole mitään tietoa siitä, millainen tietue oli silloin, kun se tulostettiin alun perin käyttäjälle, jolloin se kirjoittaa uudet tiedot tietueeseen korvaten ilmoittamatta mahdollisesti muutetut tiedot !

Tällaisen tilanteen käsittely täytyy tehdä ohjelmallisesti. Toteutus voidaan tehdä siten, että alkuperäinen tietue lähetetään selaimelle piilokenttään tai cookieen talletettuna. Seurauksena on kuitenkin ylimääräinen verkkoliikenne jonka määrä riippuu tietueen koosta. Yksi mahdollisuus olisi tehdä tietokantaan roskataulu, johon talletetaan yksiselitteisesti käyttäjäntunnistava tieto ja alkuperäinen tietue. Tällöin selaimelle lähetetään tieto, jonka avulla roskataulusta voidaan löytää alkuperäinen tietue. Käyttäjän tunnistetieto on tyypillisesti istuntokohtainen, joten se kannattaa asettaa cookien avulla.

Tehdään tietokantaan taulu, joka sisältää seuraavat kentät:

- TableName: tietueen taulun nimi, character
- RecordID: päivitettävän tietueen yksiselitteinen tunnistin, recid

- DayRecordCreated: tietueen luontipäivämäärä, date
- OrigRecord: alkuperäinen tietue binäärimuodossa, raw
- ModifiedRecord: muutettu tietue binäärimuodossa, raw
- TimeRecordCreated: tietueen luontiaika, integer
- UsrID: käyttäjän tunniste, character
- TransID: transaktion tunniste, integer

TableName-kenttää voidaan tarvita, mikäli tietokannan päivityksessä muutetaan useita tauluja. RecordID:n avulla löydetään yksiselitteisesti sama tietue, kuin mitä tällä hetkellä päivitetään. DayRecordCreated- ja TimeRecordCreated-kenttiä voidaan tarvita, mikäli tietokannan päivitykset halutaan tehdä aikajärjestyksessä. Niitä tarvitaan myös roskataulun siivoamisessa, joka perustuu tietueiden tallennushetkeen. OrigRecord sisältää tietueen ennen muutoksia. Sen avulla tehdään vertailu muutettavan ja alun perin näytölle tulostetun tietueen välillä. Koska OrigRecord on binäärinen, se ei ole sidoksissa minkään taulun tietorakenteeseen eli se on tauluriippumaton. ModifiedRecord sisältää muutetun tietueen, mikäli halutaan tehdä monesta palvelupyynnöstä muodostuva tietokantatransaktio. Monen sivun tietokantatransaktiota käsitellään myöhemmin tässä luvussa. UsrID on käyttäjän tunniste, jonka avulla etsitään monen käyttäjän tilassa käyttäjän alkuperäinen ja muokattu tietue. TransID-kentän avulla voidaan tunnistaa käyttäjän samanaikaisesti päivittämät tietueet, tällöin saman TransID:n omaavat tietueet ovat eri tauluista.

Algoritmi tietueiden päivitykseen yhden sivun tietokantatransaktiossa:

1. Kun tietue tulostetaan HTML-sivuksi,

1.1. luodaan roskatauluun tietue, joka sisältää kyseisen tietueen.

1.2. asetetaan roskatietueen tunnistetiedot.

1.3. tuhotaan vanha roskatietue.

2. Kun tietuetta päivitetään,

2.1. haetaan roskataulusta päivitettävän tietueen sisältämä tietue.

2.2. verrataan roskatauluun talletettuun tietueeseen tällä hetkellä tietokannasta löytynyttä tietuetta.

2.3. jos tietueet ovat samat,

2.3.1 tehdään päivitys.

2.3.2 muutoin tehdään haluttu virheenkäsittely.

Toteutetaan tietueen päivitys esimerkkiohjelmassa muokkaamalla edelleen lokalisoitua assignFields-tapahtumaproseduuria.

```
/*------------------------------------------------------------------
Purpose:     Super Override
  Parameters:
  Notes:
------------------------------------------------------------------*/
  DEF VAR logSame AS LOGICAL INITIAL TRUE NO-UNDO.
  /*  Haetaan  tietue  record  bufferiin  käyttäen  control  handlerissa
  ** asetettua ROWID-arvoa. */
  RUN fetchCurrent.
  /*   Haetaan   roskataulusta   tietue,   joka   sisältää   alkuperäisen
  ** käyttäjälle näytetyn tietueen */
  FIND FIRST scratch WHERE Scratch.UsrID = "Pasi" EXCLUSIVE-LOCK.
```

```
  /*  Luetaan binäärimuodossa oleva tietue ohjelman TEMP-TABLE -
  **  muuttujaan. Samalla se muunnetaan oikeisiin tietotyyppeihin.
  **  Ongelma: temp-table täytyy olla määritelty sisältämään oikeat
  ** kentät, vaihtoehto merkkijonolista tai dynaaminen temp-table. */
  IF AVAILABLE customer THEN DO:
    RAW-TRANSFER Scratch.OrigRecord TO original NO-ERROR.
    BUFFER-COMPARE customer TO original SAVE RESULT IN logSame.
    IF NOT logSame THEN
        {&OUT} "Record updated by another user. Update cancelled ! "
        "Displaying modified record, please re-write your changes.".
    ELSE RUN SUPER.
  END.
  /*  Jos assignFields epäonnistui, haetaan tietue HTML-tulostusta
  ** varten record bufferiin. */
  IF NOT AVAILABLE customer THEN RUN fetchCurrent.
END PROCEDURE.
```

Esimerkin TEMP-TABLE:n käytössä ongelmana on saada aliohjelmasta geneerinen, koska TEMP-TABLE on WebSpeedissä staattinen muuttuja, eli se määritellään käännösaikana. WebSpeedin seuraavassa versiossa TEMP-TABLE on kuitenkin jo dynaaminen objekti, jolloin esimerkin aliohjelmasta saadaan helposti geneerinen.

Seuraavaksi täytyy vielä muokata lokalisoitua displayFieldsiä:

```
/*------------------------------------------------------------------
  Purpose:     Super Override
  Parameters:
```

```
  Notes:
------------------------------------------------------------------------*/
  /* Code placed here will execute PRIOR to standard behavior. */
  /* Asetetaan muuttujan ab_unmap.RowID arvoksi record bufferissa
  ** tällä hetkellä olevan tietueen ROWID ennen kuin
  ** arvot siirretään form bufferiin. */
  ASSIGN ab_unmap.RowID = STRING(ROWID(customer)).
  RUN SUPER.
  /* Code placed here will execute AFTER standard behavior.    */
  IF chrAction <> "Save" THEN
  DO TRANSACTION:
    /* Puhdistetaan roskataulua tuhoamalla käytetyt roskatietueet */
    FOR EACH scratch WHERE Scratch.UsrID = "Pasi":
        DELETE scratch.
    END.
    /*  Luodaan  uusi  roskatietue,  johon  tallennetaan  selaimelle
    **  lähetettävä  tietue,  sekä  erilaiset  tunnistetiedot:  käyttäjä,
    ** tietue, aika, taulu, transaktio. */
    CREATE scratch.
    RAW-TRANSFER customer TO Scratch.OrigRecord.
    ASSIGN
        Scratch.DayRecordCreated = TODAY
        Scratch.RecorID = RECID(customer)
        Scratch.TableName = "customer"
        Scratch.TimeRecordCreated = TIME
        Scratch.TransID = 1
        Scratch.UsrID = "Pasi".
  END. /* DO TRANSACTION: */
END PROCEDURE.
```

Nyt ohjelma toimii tietokannan päivityksissä myös monen käyttäjän ympäristössä.

7.3 Tietokantatransaktioiden hallinta

WebSpeedissä oletuksena tietokantatransaktio voi olla korkeintaan yhtä pitkä kuin HTTP:n transaktiokin, eli yksi palvelupyyntö. Lukittua agenttia käytettäessä tietokantatransaktio voidaan helpoimmin laajentaa usean palvelupyynnön mittaiseksi, jolloin tietokantaa päivitetään useilla eri HTML-sivuilla, mutta muutokset viedään tietokantaan kerralla joko kokonaisuudessaan tai ei lainkaan. Monisivuiset tietokantatransaktiot tehdään WebSpeedin API-funktioilla.

Lukitulla agentilla tietokantatransaktio aloitetaan ja lopetetaan WebSpeedin API-funktiolla `RUN set-transaction-state IN web-utilities-hdl (INPUT "START":U)`, joka mahdollistaa useasta erillisestä palvelupyynnöstä koostuvan transaktion. Ohjelmoija voi kontrolloida transaktion päättymistä edellisellä API-funktiolla.

Lukitsemattomalla agentilla ei voida WebSpeedin valmiilla tekniikoilla tehdä lainkaan useista palvelupyynnöistä muodostuvia tietokantatransaktioita. Aiemmin käydyissä kappaleissa käytiin läpi, kuinka tietokantaan voidaan tehdä roskataulu kontekstin säilyttämiseksi eri palvelupyyntöjen yli. Samalla menetelmällä voitaisiin toteuttaa myös usean sivun tietokantatransaktio, jolloin

tietokannan varsinainen päivitys tehtäisiin yhden palvelupyynnön aikana. Roskataulun avulla toteutettava transaktioiden käsittelyssä merkittävä puute olisi, että tietokannan lukituksia ei voitaisi lainkaan hyödyntää. Mikäli tiedon päivityksessä halutaan tehdä automaattinen yhteentörmäysten käsittely, voi tekniikan käyttäminen vaatia huomattavasti ohjelmointia. Virheiden käsittelyssä voidaan hyödyntää, esimerkiksi tietueeseen asetettavalla aikaleimalla. Käyttäjien tekemät päivitykset voidaan myös tallentaa välitietokantaan, josta ne siirrettäisiin eräajona tietokantaan ja mahdolliset virhetilanteet käsiteltäisiin manuaalisesti. Viimeisin malli tosin on toteutettavissa ainoastaan pienelle tietomäärälle.

Suuri WebSpeed-sovellus sisältää tyypillisesti sekä tilan säilyttäviä että tilan välittäviä ohjelmia. Sovellusten suunnittelussa täytyykin miettiä skaalautuvuuden ja optimaalisen suorituskyvyn painotusta. Lukittu agenttia tarjoaa parhaan mahdollisen suorituskyvyn, koska sovelluksen tila pysyy muistissa. Lukitsematon taas on paras tekniikka skaalautuvuutta ajatellen. Ohjelmissa voidaan myös laittaa käyttäjät eriarvoisiksi, esimerkiksi tehokäyttäjille ajaisivat sovellusta tilan säilyttävänä ja muut käyttäjät käyttävät tilan välittävinä. Tällä toimintamallilla voidaan sekä turvaamaan palvelun käytettävyys kaikille käyttäjille että tarjoamalla tehokäyttäjille tehokkaimman. Toimintamallin valinta on hyvä ottaa huomioon jo sovellusta suunniteltaessa, koska se antaa hyvän pohjan arvioida tarvittavat lisenssit ja laiteympäristön.

7.4 Erillinen liiketoimintalogiikka

WebSpeed-ympäristössä voidaan myös liiketoimintalogiikka laittaa haluttaessa erilliseen ohjelmaan. Liiketoimintalogiikka sisältää tietokannan käsittelyssä ja laskennassa käytettävän koodin. Liiketoimintalogiikkaobjektit voidaan ajaa erillisessä sovelluspalvelimessa, jolloin ne eivät sisällä lainkaan käyttöliittymälogiikkaa. Sovelluspalvelin on ohjelmisto, jota voidaan ajaa samassa tai eri koneessa kuin WebSpeediä. Tällainen monitasomalli parantaa myös osaltaan sovelluksen skaalautuvuutta hajauttamalla raskaat tietokantaoperaatiot erillisille prosessoreille.

Kuinka monitasomallilla voitaisiin parantaa kontekstin säilyttämistä on tällä hetkellä tutkittavana oleva aihe, jonka käyn tässä tutkielmassa ainoastaan lyhyesti periaatteiltaan läpi. Jos ohjelman konteksti säilytetään tietokantaan, on ilmiselvää, että suorituskyky on heikompi verrattuna tavallisiin asiakas/palvelin-sovelluksiin, joissa sovelluksen konteksti säilytetään koneen muistissa. Erilliset liiketoimintaobjektit voivat hyödyntää palvelinten jaettua muistia kontekstin säilyttämiseen, jolloin palvelimen täytyisi ainoastaan tietää käyttäjäkohtaiset muistiosoitteet sovelluksen kontekstiin.

7.5 Käyttäjäkohtaiset asetukset

Sovelluksissa täytyy usein pystyä määrittelemään käyttäjäkohtaisia asetuksia, joissa voidaan mm. määritellä käyttöliittymän ulkoasua. WebSpeed-sovelluksissa käyttäjäkohtaiset asetukset voidaan toteuttaa käyttäen tietokantaa tai cookieita, mikäli jokaiselle käyttäjälle ei haluta tehdä tietokantaan omaa

tietuetta. Cookiet voidaan myös jättää voimaan halutuksi ajaksi, tosin käyttäjä voi myös käydä tuhoamassa olemassa olevat cookiet, koska ne tallennetaan selainten koneisiin.

7.5.1 Kansainvälisyys

Internet-sovellusten käyttäjät tulevat usein eri maista. Jotta sovellus saadaan mahdollisimman käyttäjäystävälliseksi, tulisi sovelluksen käyttöliittymä olla käyttäjän omalla kielellä. Tämä ongelma on varsin haastava, koska kieliin vaikuttavat monet eri seikat. Yksi ongelma on kielten käyttämät erilaiset merkistöt (kts. Luku 6.8).

Usein monikieliset sovellukset toteutetaan yksinkertaisesti tekemällä sovelluksesta ja HTML-dokumenteista erikielisiä versioita sekä myös erilliset ajoympäristöt. Sovelluksen ylläpidon ja tekemisen kannalta tämä on kuitenkin hankalaa ja työlästä. Jos HTML-sivut tulostetaan ohjelmallisesti, voidaan samaa merkistöä käyttävän sovelluksen monikielisyys toteuttaa tietokantaan tallennettujen kielien avulla. WebSpeedillä se onnistuu helpoiten cgi wrapperilla. Kielestä johtuvia muokkaustarpeita voidaan minimoida käyttämällä kuvia tekstin sijasta. Kuvien käytössä tulee pyrkiä mahdollisimman yleisesti alalla käytettyihin kuvioihin. Ohjelmoinnissa taas tulee pyrkiä minimoimaan sovellukseen tarvittavat muutokset erikielisten versioiden aikaansaaminen (Design an International Application 1998, 1-3).

Monikielisen sovelluksen tekeminen yksinkertaistuu merkittävästi, jos kielet pystyvät käyttämään samaa merkkikoodistoa, kuten suomi, ruotsi ja englanti. Jos ohjelman käyttöliittymä tulostetaan dynaamisesti, on erikielisten käyttöliittymien tekeminen yksinkertaista, esimerkiksi tietokantaan talletettujen erikielisten tekstien avulla. Tilanne kuitenkin vaikeutuu, jos ohjelmista halutaan tehdä päivittäviä. Tämä näkyy erityisesti nappuloiden kanssa työskenneltäessä. HTML:n syntaksi nappuloiden määrittelyllä on `<INPUT TYPE=”submit” NAME=”button” VALUE=”Save”>`. Ongelma on, että nappulan arvo on sama kuin näytöllä näkyvä teksti. WebSpeedissä taas ohjelmoija on saattanut kirjoittaa koodin `IF get-value(”button”) = ”Save” THEN…` . Suomalaisessa versiossa HTML-sivulla nappulan VALUE tulisi olla *talleta* ja ohjelmakoodi vastaavasti `IF get-value(”button”) = ”Talleta” THEN…` . HTML-kielen puute on, että se erottele nappulan arvoa ja sen näytöllä näkyvää tekstiä. Ongelma voidaan kiertää nimeämällä jokainen nappula erinimiseksi NAME-attribuutilla, jolloin sovellus voidaan ohjelmoida riippumattomaksi näytöllä näytettävästä tekstistä. Tällöin jokaiselle nappulalle tehdään oma IF-käsittely: sitä nappulaa on painettu, jolla on jokin arvo. Esimerkiksi

```
IF get-value("ButtonFirst") <> "" THEN RUN fetchFirst.
ELSE IF get-value("ButtonPrev") <> "" THEN RUN fetchPrev.
ELSE IF get-value("ButtonNext") <> "" THEN RUN fetchNext.
ELSE IF get-value("ButtonLast") <> "" THEN RUN fetchLast.
```

Erillisten IF-lauseiden tekeminen ei ole suorituskyvyn kannalta optimaalista, koska jokaisen nappulan tarkastelu täytyy käydä lukemassa API-funktiolla avulla (vrt. 7.2.2).

Suorituskyvyn kannalta olisi parempi käyttää tietokantaan talletettuja erikielisiä merkkijonoja ja vastaavaa CASE-rakennetta kuin aiemmissa esimerkkiohjelmissa. Kieli välitettäisiin palvelupyyntöjen mukana – esimerkiksi cookiella – jolloin ohjelmissa tehdään vertailu käyttäjän kielen teksteillä. Esimerkiksi alla olevan mallin mukaisesti, jossa *chrNav* sisältää käytetyn kielen merkkijonot.

```
DEF VAR chrNav AS CHAR EXTENT 4 INITIAL ["First", "Prev", "Next",
  "Last"] NO-UNDO.

IF get-value(”lang”) = ”FIN” THEN ASSIGN
      ChrNav[1] = ”Ensimmäinen”
      ChrNav[2] = ”Edellinen”
      ChrNav[3] = ”Seuraava”
      ChrNav[4] = ”Viimeinen”.

CASE get-value("NavButton"):
      WHEN chrNav[1] THEN
          RUN fetchFirst.
      WHEN chrNav[3] THEN
          RUN fetchNext.
      WHEN chrNav[2] THEN
          RUN fetchPrev.
      WHEN chrNav[4] THEN
```

```
            RUN fetchLast.
END CASE.
```

7.5.2 Monikielinen sovellus HTML-mapping-tekniikalla

HTML-mapping –tekniikka perustuu malliin, jossa HTML-sivu ja sovelluslogiikka ovat erillisissä tiedostoissa. Ohjelmoinnin kannalta tämä malli automatisoi kaikista eniten ohjelman ja käyttöliittymän tekemistä. Monikielisyys vaatisi, että HTML-dokumenttia voitaisiin vaihtaa ajonaikaisesti halutun kieliseen versioon, mutta WebSpeed ei tue tätä toimintoa mahdollista.

WebSpeedissä monikielisen ohjelman käyttöliittymä täytyy tulostaa dynaamisesti. Jos haluttaisiin tehdä HTML mapping –tekniikalla monikielinen sovellus, täytyisi WebSpeedin pystyä vaihtamaan käytettävää HTML-sivua ajonaikaisesti eri kieliversioihin. WebSpeed ei kuitenkaan tue ajonaikaista HTML-dokumentin nimen vaihtamista. Teknisesti asian muuttaminen ei olisi vaikeaa. Mapped web objekti koostui kolmesta tiedostosta: HTML-dokumentti, offset-tiedosto ja ohjelmatiedosto. Erikielisissä versiossa ainoastaan HTML-dokumentti muuttuu, eli ajonaikaisesti ainoataan se pitäisi pystyä vaihtamaan. Sen toteuttaminen vaatii ainoastaan vähän muutoksia WebSpeedin käyttämiin ohjelmiin.

Vertailun vuoksi käydään läpi monikielisen sovelluksen toteuttaminen Javalla. Java-sovellusten käyttöliittymälogiikka suoritetaan selaimessa ajettavalla sovelmalla – appletilla. Java-sovelluksissa ei tarvitse käyttää

käyttöliittymäkomponentteina HTML:n FORM-elementtejä, jolloin monikielisyys voidaan toteuttaa helposti ohjelmalla, kun käyttöliittymäkomponenteista tarvitsee ainoastaan vaihtaa otsikon (label) arvo (Jaworski 1998, 102). Käyttöliittymän käsittely on olennaisesti helpompaa, kun käyttöliittymässä näkyvä teksti ei samalla ole käyttöliittymäobjektin arvo.

7.6 Paperitulostus

Paperitulostus on merkittävä ongelma CGI-pohjaisissa webbisovelluksissa, kuten WebSpeedissä. Kirjoittimien hallintaa ei pystytä tekemään palvelinsovelluksesta, koska selaimet käyttävät tulostukseen työasemassa olevia kirjoitinasetuksia. Tulostuksessa ongelmiksi nousevatkin mm. sivutus ja sivun ulkoasu. Tulostuksen ohjaamiseen täytyisi käyttää jotain selaimessa ajettavaa tekniikkaa: java, skriptikielet tai plug-init.

Kuitenkin jos CGI-sovellusta käytetään ainoastaan yrityksen sisäisessä verkossa olevassa intranetissä, voidaan tulostus toteuttaa, kuten muissakin keskitettyä arkkitehtuuria käyttävissä ajoympäristöissä. Palvelimella on tieto, kaikista verkossa olevista kirjoittimista, jolloin tulostus ohjataan käyttäjäkohtaisten asetusten perusteella lähimmälle verkkokirjoittimelle.

7.7 Microsoft Active Server Pages 3.0 - tietokantatransaktioiden hallinta

Käydään vertailun vuoksi läpi Microsoftin Active Server Pages (ASP) -tekniikan käyttö tietokantasovelluksen toteuttamisvälineenä WWW:ssä. ASP koostuu arkkitehtuuriltaan sekä palvelimessa että selaimessa ajettavista komponenteista, joiden toiminnallisuudesta suurin osa tapahtuu palvelimessa.

Palvelimessa ajettavat komponentit ovat request, response, application, session, server ja ASPError objekteista. Objetien toiminta:

- Request objekti välittää ohjelman käyttöön selaimesta lähetetyt tiedot.
- Response objekti huolehtii sovelluksen tuloksena saadun datan lähettämisestä www-palvelimelle. Se tuo ohjelman käyttöön valitun www-palvelinohjelmiston ympäristömuuttujat.
- Application objekti luodaan, kun ASP-sivua pyydetään ensimmäistä kertaa. Sitä voidaan käyttää kaikkien käyttäjien käytettävissä olevilla ASP-sivuilla käytettävien muuttujien arvojen ja objektiviittausten tallettamiseen.
- Session objekti luodaan jokaiselle käyttäjälle. Sitä voidaan käyttää käyttäjäkohtaisen kontekstin säilyttämiseen ja se pysyy muistissa määritellyn ajan.
- Server objektia käytetään ASP-skriptien ajamisessa. Niiden avulla voidaan käyttää COM-objekteja (Component Object Model).
- ASPError objektin avulla voidaan toteuttaa ASP-ympäristössä tapahtuneet virhetilanteet.

Lisäksi sovelluksissa käytetään ADO-objekteja (ActiveX Data Objects), jotka ovat käyttöliittymäriippumattomia tietokannan käsittelyssä käytettävän logiikan suorittavia ohjelmia, joista on sovelluksille standardi rajapinta. Myös ADO-objektit suoritetaan palvelimessa, jolloin minimoidaan verkon käyttö ja niiden toimintaa ohjataan käyttöliittymäohjelmasta asetettavien parametrien avulla. (Anderson, Blexrud, Chiarelli, Denault, Homer, Esposito, Francis, Gibbs, Kropog, McQueen, Reilly, Robinson, Schenken, Sonderegger, Sussman, 1999, 35, 314-315.)

Myös ASP:ssa sovelluksen tilan välittäminen voidaan toteuttaa lukitsemalla palvelimessa prosessi yhden käyttäjän käyttöön, jolloin selaimesta lähetetään cookiella tunnistetieto, jonka avulla palvelimelta etsitään käyttäjän istunnon kontekstin sisältämä objekti. Ohjelman kontekstin säilyttämisessä voidaan käyttää kolmea tapaa: konteksti välitetään selaimen ja palvelimen välissä palvelupyyntöjen mukana, konteksti tallennetaan session objektiin tai tietokantaan. (Anderson, Blexrud, Chiarelli, Denault, Homer, Esposito, Francis, Gibbs, Kropog, McQueen, Reilly, Robinson, Schenken, Sonderegger, Sussman, 1999, 547.)

Jos sovelluksen tila tallennetaan session objektiin, on konteksti saatavissa nopeasti muistista ohjelman käyttöön, joka vie toisaalta palvelimen muistia heikentäen skaalautuvuutta. Session objektiin voidaan tallentaa muuttujien nimiä sekä niiden arvoja ja se pysyy muistissa ennalta määrätyn ajan tai kunnes se vapautetaan ohjelmallisesti. Kaikkien käyttäjien yhteisesti käyttämää

kontekstia voidaan tallentaa application objektiin, joka pysyy aktiivisena niin pitkään, kun yksikin käyttäjä käyttää sovellusta. Sekä session että application objektien konteksti on käytettävissä yhdessä palvelimessa. Ohjelman kontekstin tallentaminen tietokantaan mahdollistaa eri sovelluspalvelimien käytön. Se myös parantaa ajoympäristön skaalautuvuutta minimoimalla muistin käytön, mutta toisaalta taas heikentää suorituskykyä. Selainsovellusta käyttäminen kontekstin säilyttämisessä vähentää palvelimen kuormitusta, mutta haittapuolena on, että konteksti täytyy välittää jokaisen palvelupyynnön ja vastauksen mukana kuormittaen verkkoa. (Anderson, Blexrud, Chiarelli, Denault, Homer, Esposito, Francis, Gibbs, Kropog, McQueen, Reilly, Robinson, Schenken, Sonderegger, Sussman, 1999, 110-116, 548.)

Kontekstin välittämiseen käytettävän menetelmän valintaan vaikuttavat suorituskyky, skaalautuvuus ja sovelluksen käytettävyys (Anderson, Blexrud, Chiarelli, Denault, Homer, Esposito, Francis, Gibbs, Kropog, McQueen, Reilly, Robinson, Schenken, Sonderegger, Sussman, 1999, 547). Session ja Application objektien käytön hyviä puolia ovat helppokäyttöisyys, välimuistin käyttö kontekstin tallentamiseen ja se, ettei tarvitse erikseen ohjelmoida sovelluksen tilan välittämistä. Vastaavasti objektien käytön huonoja puolia ovat:

- vaativat cookieden käyttöä, selaimet lähettävät cookiet vain tiettyihin URL-osoitteisiin, palvelupyynnöt suoritetaan jonossa - voi haitata käytettäessä HTML:n <FRAME>-tekniikkaa
- kuluttavat palvelimen muistia ja hidastavat ASPin toimintaa
- palvelimen virhetilanteissa konteksti menetetään

- aikakatkaisu hävittää kontekstin
- objektit jäävät ’roikkumaan’ palvelimelle, vaikka käyttäjä ei niitä enää käytä.

(Anderson, Blexrud, Chiarelli, Denault, Homer, Esposito, Francis, Gibbs, Kropog, McQueen, Reilly, Robinson, Schenken, Sonderegger, Sussman, 1999, 1033-1034.)

Tietueiden päivityksessä mahdollisesti tapahtuvien konfliktien hallintaan käytetään ASP-tekniikassa Remote Data Services (RDS) –palveluita, jotka ovat ADO-komponenttien osia. RDS:ää käytetään, kun dataa siirretään palvelimen ja selaimen välillä. RDS koostuu sekä asiakkaassa että palvelimessa olevista komponenteista. Palvelimessa olevia RDS-komponentteja ovat DataFactory ja Custom Component. DataFactory muodostaa tietokantayhteyden ja huolehtii datan siirtämisestä palvelimen tietokannasta asiakkaalle ja toisinpäin. Custom Component on COM-komponentti, joka sisältää datan siirrossa käytettävät metodit. Custom Componentteja käytetään, kun DataFactoryn oletustoiminnot eivät ole riittäviä. Selaimen RDS-komponentit ovat DataSpace, Data Source, Client Data Cache ja Data Binding Manager. DataSpace vastaa palvelimessa olevaa DataFactorya tai Custom Componenttia, ja se huolehtii palvelimen ja selaimen välisestä tietoliikenteestä. Data Source Object (DSO) säilyttää selaimelle tuodun datan, eli se sisältää ADO-objektin tietuejoukon (recordset) ja huolehtii datan käsittelystä Client Data Cachen kanssa. Data Binding Manager yhdistää tietueiden kentät HTML-kontrolleihin. (Anderson, Blexrud, Chiarelli, Denault, Homer, Esposito, Francis, Gibbs, Kropog, McQueen, Reilly, Robinson, Schenken, Sonderegger, Sussman, 1999, 390-391.)

Tietueiden päivityksessä tapahtuvien konfliktien käsittely sisältyy SubmitChanges-metodin toimintaan, jolla lähetetään selaimelta kaikki päivitetyt tietueet palvelimelle, jossa tietokannan varsinainen päivitys tapahtuu. Metodi myös tarkistaa eri virhetilanteet, jolloin peruutetaan kaikki tietokannan muutokset. Konfliktien käsittely on mahdollista, koska DSO sisältää sekä alkuperäisen, muutetun että tietokannassa olevan tietueen. Varsinainen tietueen päivityksen epäonnistumisen syyn selvittäminen tehdään ohjelmoimalla. RDS-komponenttien heikkous on, että se perustuu täysin Microsoftin tekniikkaan, selain mukaan lukien (Anderson, Blexrud, Chiarelli, Denault, Homer, Esposito, Francis, Gibbs, Kropog, McQueen, Reilly, Robinson, Schenken, Sonderegger, Sussman, 1999, 392, 413).

8. Tulevaisuus

8.1 Suorituskyky

Käyttäjämäärän jatkuva kasvu webissä johtaa siihen, että Internetiin tehtyjen WWW-sovellusten täytyy minimoida verkon ja palvelimen kuormitus. Kehitys näyttäisikin samantyyppiseltä kuin tietokantasovelluksissakin on tapahtunut: ensin oli keskitetyt järjestelmät, jonka jälkeen kuormitusta hajautettiin työasemalle ja palvelimelle ja nykyisin ollaan siirrytty monitasoarkkitehtuuriin. WWW-sovellusten logiikkakin tullaan todennäköisesti hajauttamaan siten, että asiakaskoneessa suoritetaan käyttöliittymälogiikka ja palvelimissa tai erityisissä tietokantaa lähellä nopean verkkoyhteyden päässä olevissa sovelluspalvelimissa suoritetaan liiketoimintalogiikka, jolloin pystytään parhaiten vähentämään verkkoliikennettä ja palvelimen kuormitusta.

WWW:n haasteet tietokannoille muodostuvat pitkälti suurista käyttäjämääristä, 24x7-toiminnasta sekä tietokantojen hajautuksesta, joita voidaan teknisesti ratkoa mm. paremmalla suorituskyvyllä, moniprosessoritekniikoilla, toissijaisilla varatoiminnoilla ja replikoinnilla. Yhtenä mielenkiintoisena mahdollisuutena tänä päivänä tutkitaan myös oliotietokantoja, joka tuo lisämahdollisuuksia mm. tietokannan hajauttamiseen verkossa ja WWW:ssä käytettävien monimutkaisten tietorakenteiden, kuten kuvien, äänen tai oliorakenteita käsittelyyn (Orfali, Harkey, Edwards 1999, 529).

Laitteiston kannalta kriittisimmät kohdat ovat verkkotekniikka, prosessori, kovalevyn suorituskyky ja koko sekä muistin määrä (Held 1997, 4-10), joiden teknologinen kehitys vaikuttaa olennaisesti, millaisia sovelluksia webissä voidaan ajaa, kuten vaikka televisiota. Tiedonsiirron perusteknologioiden, kuten protokollien, kehittyminen on myös olennaisen tärkeää.

Ajoympäristöjen suorituskyky voi parantua kehittämällä välimuistin käyttöä sekä webbi- että sovelluspalvelimessa. Muistilla voidaan pyrkiä minimoimaan kovalevyn käyttöä parantaen siten ajoympäristön suorituskykyä. Välimuistin käytön kehittäminen edellyttää tehokkaita algoritmeja siihen, mitä tietoja säilytetään ja mitkä tuhotaan muistista. (IBM High-Volume Web site team 1999.)

8.2 Uudet sovellukset ja WWW-tekniikat

Tällä hetkellä nopeimmin kasvava sovellusalue on sähköinen kaupankäynti – ns. e-commerce, joka tarkoittaa sekä yritysten välistä tiedonvälitystä että kuluttajille suunnattujen kauppapaikkojen toteuttamista. Internetin perusajatus on maailmanlaajuisuus ja avoin, standardi teknologia, joiden tuloksena Internetistä on muodostunut nopeimmin kasvava sovellusten ajoympäristö. Sähköisen kaupankäynnin Internetissä edellyttää, että sovellus voidaan integroida muissa sovelluksissa käytettyihin teknologioihin. Tämä voi onnistua vain, jos webbi-sovellukset perustuvat standardeihin tekniikoihin (Clinton, Gore 1997).

Yhä useampi kuluttaja haluaa tehdä ostoksensa itse, ilman apua. Internet tarjoaa tällaiseen toimintaan hyvin soveltuvan ympäristön. Kauppapaikkojen toteuttajille Internet tarjoaa myös muita hyviä etuja, kuten tehokkaasti kohderyhmillä suunnatun myynnin. Suunnatun myynnin voidaan toteuttaa keräämällä asiakkaiden ostoskäyttäytymisestä tietoa, jonka pohjalta voidaan jatkossa suunnata tarjoukset ja mainokset tehokkaasti juuri niitä todennäköisimmin haluaville asiakkaille, mikä on usein samalla kuluttajienkin etu.

Menestyäkseen Internetissä toimivan yrityksen täytyy myös pystyä tekemään eroa kilpailijoihinsa verrattuna. Ensiarvoisen tärkeää on kuluttajien luottamuksen säilyttäminen, jolloin ajoympäristön ja sovelluksen toiminnan täytyy olla ehdottoman luotettavaa ja tehokasta. Laadukas palvelun kattaa koko tavaroiden toimitusketjun ja sovelluksen luotettavan sekä tehokkaan toiminnan. Sovelluksissa keskeisiä tekijöitä ovat tietoturva, tietokantatransaktioiden hallinta ja kaikkien toimitusketjuun osallistuvien yritysten järjestelmien tehokas integroituminen, jotta tavaroiden toimitus voidaan toteuttaa nopeasti ja luotettavasti. Internet-sovellusten täytyy myös pystyä muuntautumaan käyttäjäkohtaisesti, esimerkiksi kielen suhteen, koska käyttäjäthän valitsevat vapaassa, avoimessa verkossa kaikista miellyttävimmän palvelun. (Haughey 1999.)

Myös yritysten välinen liiketoiminta internetissä sisältää suuria mahdollisuuksia, jossa yhtenä keskeisenä tavoitteena on automatisoida yritysten välinen

tiedonvaihto. Tämä on ohjelmoinnissa keskeinen haaste, koska usein eri yritysten järjestelmät perustuvat erilaisiin teknologioihin. Eri tekniikoilla toteutettujen järjestelmien välinen kommunikointi edellyttää tiedonsiirtoon EDI:ä parempaa standardia tekniikkaa, jonka moni uskoo olevan XML (extensible markup language). Teknisesti liikuteltava tietomäärä voi aiheuttaa yritysten välisessä tiedonsiirrossa kuluttajille suunnattuun tiedonsiirtoon verrattuna merkittäviä lisähaasteita, koska tietomäärä voi olla huomattavasti suurempi. (Haughey 1999.)

Tiedon suojaamiseen käytettävissä olevat menetelmät vaikuttavat paljolti kaupallisiin Internet-sovelluksiin. Muun muassa Euroopan unionin ministerikokous 6.-8.7. 1997 Bonnissa toteaa julkilausumassaan (Ministerial Conference Bonn 1997): ”Tietoturva on yksi maailmanlaajuisten tietoverkkojen menestymisen avaintekijöistä” ja, että ”luotettava salaustekniikka on edellytys elektroniselle kaupankäynnille.”. Internet-kauppapaikkojen yleistyminen edellyttääkin luotettavia ja helppokäyttöisiä tilaus- ja maksumenetelmiä. Myös WWW:hen rakennettujen yritysten Extra-/Intranetien täytyy olla hyvin suojattuja luvattomalta ulkopuolisten käytöltä, jolloin tietoturva tarkoittaa sekä tiedon salaamista että luvattoman käytön estämistä.

Internetin maailmanlaajuisuus vaatii sovelluksilta myös kykyä toimia kansainvälisessä ympäristössä, joka sisältää kielet, kulttuurit ja lainsäädännöt. Jo tällä hetkellä voidaan tehdä monikielisiä ja –kulttuurisia sovelluksia, mutta lainsäädäntöjen käsittely vaatii teknisten toimenpiteiden lisäksi ennen kaikkea

kansainvälisen lainsäädännön kehittämistä. Monikielisten ja –kulttuuristen sovellusten toteuttaminenkaan ei ole tällä hetkellä erityisen helppoa, koska sekä niiden tekeminen että ylläpitäminen ovat työlästä. Usein ainoa tapa ratkaista monikielisyydestä ja –kulttuurisuudesta johtuvat ongelmat on tehdä ohjelmista erillisiä versioita. Monikielisten sovellusten pohjaksi kehitetäänkin jo aiemmin mainittua unicode-tekniikkaa. W3C määrittelee monikielisen tekniikan ytimeksi (Internationalization: W3C activities 1999) ajatusta, että tiedon välityksen lisääntyessä tulee web nähdä yhtenä sovelluksena, jolloin lähetettäessä tekstiä verkon yli, tulee lähettäjän ja vastaanottajan ensiksi sopia käytettävä merkistö (Internationalization: W3C activities 1999). (Internationalization: W3C activities 1999.)

8.3 Ylläpito

Yksi merkittävimmistä asiakas/palvelin-sovellusten ongelmista on kallis ja työläs ylläpito. Internetissä työasemia on käytännöllisesti katsoen mahdotonta ylläpitää muilla kuin automaattisesti tapahtuvilla tekniikoilla. Ylläpidon täytyykin tapahtua keskitetysti, jolloin käyttäjien täytyy saada työasemassa suoritettavat ohjelmat automaattisesti palvelimelta sovelluksen ensimmäisen käytön yhteydessä – vastaavasti myös päivitykset jatkossa.

Tulevaisuudessa yhä enemmän niin webbi kuin muidenkin sovellusten täytyy myös pystyä hyödyntämään jo olemassa olevia sovelluksia, koska uusien sovellusten lähtökohtana ei aina voi olla korvata kaikki se mitä aiemmin on tehty uudella tekniikalla. Tulevaisuudessa tarvitaan varmastikin uusia sovelluksia

vanhojen lisäksi ja niinpä vanhojen sovellusten hyödyntämiseen kehitelläänkin ratkaisuksi ns. middleware-tuotteita, kuten CORBA:a, jonka tarkoitus on olla sovellusten välisenä tulkkina.

8.4 Object Web

Orfali, Harkey ja Edwards määrittelevät Object Webin WWW-sovellusten seuraavaksi kehitysvaiheeksi (Orfali, Harkey ja Edwards 1999). Object Webissä WWW:ssä käytettävistä dokumenteista tehdään objekteja, jotka toteutetaan XML:llä, DOM:lla (Document Object Model) ja XSL:llä (Extensible Style Sheet) (623). Object Webin perustekniikoita ovat Java - joka sisältää menetelmät sovellusten kehittämiseen, ylläpitoon ja jakeluun - sekä CORBA:aan, joka täydentää Javaa tarjoamalla standardin sovellusten väliseen kommunikointiin. Microsoft kehittää CORBA:lle kilpailevaa tekniikkaa COM+:aa, jonka olennainen ero CORBA:aan verrattuna on, että se edellyttää Microsoftin tekniikan käyttämistä niin palvelimessa kuin työasemassakin. Object Webin arkkitehtuuri on kolmitasoinen, joka koostuu seuraavista:

1. taso muodostuu selaimista ja muista web-pohjaisista käyttöliittymistä (esim. Windows 98). Käyttöliittymä toteutetaan JavaBeaneillä, jotka pystyvät kommunikoimaan muiden asiakaskoneessa tai palvelimessa olevien ”papujen” kanssa. Lisäksi myös palvelimessa olevat JavaBeanit voivat käyttää asiakaskoneessa olevien JavaBeanien metodeita CORBA-tapahtumilla.
2. taso ajetaan palvelimessa, joka pystyy kommunikoimaan sekä HTTP- että CORBA-asiakkaiden kanssa. Se sisältää myös sovelluspalvelimen, joka

käynnistää objektiryhmiä, tasaa kuormitusta, takaa virheistä toipumisen ja koordinoi useista komponenteista koostuvia transaktioita. Ajettavat komponentit ovat EJB:tä (Enterprise Java Bean) tai CORBA Beanejä.

3. taso on mitä tahansa, mitä CORBA-objektit voivat käyttää, kuten tietokannat ja muut sovellukset.

(Orfali, Harkey, Edwards 1999, 616-617,621-622, 630.)

9. Johtopäätökset

9.1 Tietokannan käyttö CGI-ympäristössä

Tietokantojen käytössä sovelluksen tilan säilyttäminen on keskeinen tekijä, jolla hallitaan tietokannan käyttöä monen käyttäjän ympäristöissä. Sekä useasta työaseman ja palvelimen välisestä tapahtumasta koostuva tietokantatransaktio että tietueiden lukitukset vaativat kontekstin säilyttämistä. Niinpä tietokannan käyttäminen WWW-ympäristössä vaatiikin usein tekniikan HTTP-protokollan tilattomuuden kiertämiseksi.

WebSpeedissä HTTP:n tilattomuus voidaan helposti kiertää lukitulla agentilla ja cookieilla. Ratkaisun huono puoli on sen ajoympäristön skaalautuvuutta huonontavat vaikutukset. Lukittujen agenttien käyttö johtaa aikanaan varsin todennäköisesti samaan tilanteeseen, jossa ajoympäristön kuormitusta täytyy pystyä hajauttamaan eri koneisiin (vrt. asiakas/palvelin tai monitasoarkkitehtuuri). Toisaalta jos tietokannan päivitys voitaisiin virheettömästi toteuttaa eräajona, voitaisiin sovellukset ohjelmoida myös tilattomiksi. WebSpeedissä ohjelmoijalle tietueen päivityksen käsittely on automatisoitu ainoastaan lukittua agenttia käyttävissä sovelluksissa. Yhdessä HTTP-transaktiossa tehtävän tietokannan päivityksen toteuttaminen vaatii ohjelmoinnilla tehtävää ratkaisua, esimerkiksi roskataululla tai HTML/HTTP-tekniikoilla. Myös lukitsematonta agenttia käyttävissä tietokantaa päivittävissä sovelluksissa olisi hyvä olla olemassa standardoitu tekniikka, joka vähentää

mm. eri ohjelmoijien saman asian uudelleen keksimistä sekä sovellusten siirtämistä tulevaisuudessa mahdollisiin uusiin tekniikoihin.

Ohjelman kontekstin säilyttäminen voidaan nykyisessä versiossa (V3.0) toteuttaa ainoastaan käyttämällä palvelimen kovalevyä – lähinnä tietokantaa. Agenttiprosessit voisivat kuitenkin hyödyntää myös palvelimen jaettua muistia jolloin agentin muistialuetta ei varattaisi yhdelle käyttäjälle, vaan se voisi jakaantua kaikkien yhteiseen ja käyttäjäkohtaisiin osiin (vrt. Microsoft Active Server Pages). Muistin tehokkaammalla hyödyntämisellä voidaan parantaa sekä suorituskykyä että skaalautuvuutta.

9.2 CGI-ajoympäristö WWW:ssä

Tietokantoihin kohdistuvia haasteita Internetissä ovat entistä suurempi samanaikaisten käyttäjien määrä sekä 24x7 tapahtuva toiminta, joista todennäköisesti seuraa myös koon kasvaminen. Hajautustekniikoilla voidaan ratkoa käyttäjämäärästä ja koosta aiheutuvia ongelmia. Oliotietokannat ovat nimenomaan HTTP-protokollaan perustuvassa WWW-ympäristössä mielenkiintoinen mahdollisuus erityisesti, koska niiden avulla toivotaan saatavan ratkaisu myös tietokannan tilan hallintaan.

WWW:ssä erityisesti sovellusten kuormituspiikit voivat olla suuria, jolloin ajoympäristöiltä – niin tietokannalta kuin sovelluspalvelimeltakin - vaaditaan suorituskykyä ja luotettavuutta. Ajoympäristön täytyy pystyä automaattisesti käynnistämään varatoimintoja ja skaalautumaan kuormitukseen. WebSpeedissä

kuormituksen tasaus on toteutettu NameServerillä, jolla sovelluksen ajoympäristön voidaan hajauttaa useaan koneeseen, jolloin myös sovelluksen käytettävyys ei ole riippuvainen yhdestä brokerista. WebSpeedin ajoympäristössä NameServer, jolle ei ole olemassa varatoimintoa, jää kuitenkin kriittiseksi kohdaksi, koska sen toiminta on välttämätön. Kuormitukseen skaalautumisen hoitaa taas broker, joka automaattisesti käynnistää ja sulkee agentteja kuormituksen mukaan. Skaalautuvuudessa rajoitukset muodostuvat lähinnä laitteistosta ja lisensoinnista.

CGI-pohjaiseen tekniikkaan pohjautuvien sovellusten selkeä heikkous on keskitetty arkkitehtuuri. Internet-sovelluksia ajetaan usein pitkien etäisyyksien päästä WAN:in (Wide Area Network) yli. Siksi Internet-sovellusten tulee kuormittaa verkkoa mahdollisimman vähän. Sovellukset tulisi suorittaa siellä, missä se on verkkoliikenteen kannalta järkevintä, eli käyttöliittymälogiikka - kuten syöttötiedon tarkistus ja paperitulostus - tehdään selaimessa ja tietokannan käsittelyyn liittyvä logiikka palvelimessa. CGI-sovellukset ovatkin enää harvoin ”puhtaita” CGI-sovelluksia, vaan ne sisältävät usein myös selaimessa ajettavia ohjelmia. Myös WebSpeed-sovelluksiin voidaan liittää selaimessa suoritettavaa ohjelmakoodia sisällyttämällä se HTML-tulostukseen. Selaimen ohjelmointia ei kuitenkaan voida tehdä SpeedScriptillä.

9.2.1 Monikielisyys

Haluttaessa maailmanlaajuisesti käytettävien sovellusten täytyy olla monikielisiä - sekä data että käyttöliittymä. Monikielisyyden toteuttaminen edellyttää standardeja, koska siinä tarvitaan eri tekniikoiden - kuten selain, tietokanta ja WWW-palvelin - yhteistoimintaa. Unicode-merkistö on monikielisyyden toteuttamiseksi kehitetty standardi, jota WebSpeedin agentti ei kuitenkaan pysty tällä hetkellä käyttämään, eli tässä kehityksen tarve on ilmeinen. WebSpeedillä monikielinen eri koodistoa käyttävä sovellus täytyy toteuttaa tekemällä jokaiselle erikieliseen sovellukseen oma broker ja agentit, joka on sovellusten ylläpidon kannalta ilmeisien hankalaa. Monikielinen käyttöliittymä vaatii, että sovellus pystyy tulostamaan käyttäjäkohtaisesti HTML-dokumentin. Jos käyttöliittymä tehdään ajonaikaisesti, on erikielisten HTML-dokumenttien tulostaminen suhteellisen helppoa. WebSpeedissä ohjelmointia eniten automatisoiva tekniikka on HTML mapping, jossa HTML-tiedostoa ei kuitenkaan voida asettaa ajonaikaisesti. Se olisi kuitenkin toteutettavissa suhteellisen pienillä muutoksilla, jotka helpottaisivat selkeästi erikielisten käyttöliittymien tekemistä.

9.3 Standardit

Erityisesti WWW:ssä ohjelmointitekniikoiden tulee perustua mahdollisimman pitkälti yleisesti käytettyihin standardeihin, koska maailmanlaajuiseen käyttöön tehtävien sovellusten täytyy pystyä toimimaan useisiin eri teknologioilla perustuvien järjestelmien kanssa. WebSpeed pohjautuu alalla vallitseviin standardi WWW-tekniikoihin, jolloin sillä tehdyt sovellukset ovat selain, WWW-

palvelin ja tietoturvaratkaisuriippumattomia (yhteensopivia). Tulevaisuuden haasteet aiheutuvat uusien kehittyvien standardien - kuten XML, CORBA/IIOP ja IDL - integroimisesta nykyiseen ajoympäristöön vaatien mahdollisimman vähäisiä muutoksia vanhoihin sovelluksiin.

9.4 Webbi-sovellusten tutkimuskohteita

WWW tulee epäilemättä kasvamaan edelleen uusien sovellusten verkkoratkaisuna. Webin kansainvälisyys luo runsaasti uusia vaatimuksia sekä tekniselle ympäristölle että sovellusten käyttöön liittyviin asioihin. Tekniseltä kannalta suuria ratkaistavia kysymyksiä ovat tietoturva, luotettava maksujärjestelmä ja verkon nopeus. Kaikki nämä täytyy ratkaista ennen kuin voidaan odottaa verkon vielä laajamittaisempaa käyttöä liiketoiminnassa ja muussa tiedonvälityksessä.

Alalla voimakkaasti kasvava sovellusten vuokrauksen tekninen ja kaupallinen toteuttaminen on hyvä ja tarpeellinen tutkimuskohde, jossa tietoturva on yksi keskeisistä kysymyksistä. Vuokraajien kannalta lienee ilmeistä, että eri sovellusten tekninen ylläpito perustuisi mahdollisimman pitkälti standardeihin, jotka tutkimuksella pyrittäisiin etsimään.

WWW aiheuttaa suuria haasteita myös kansallisiin lainsäädäntöihin, joissa on ilmeistä, että mm. verotusta joudutaan yhä enemmän yhdenmukaistamaan

kansainvälisesti. Yksi varsin selkeästi nähtävät tutkimuskohde on kansainvälinen verotus.

Intimiteettisuoja on muodostunut yksittäisen ihmisen kannalta tärkeäksi seikaksi. Intimeettisuoja sisältää mm. seuraavat asiat: mitä tietoja yrityksillä on oikeus kerätä käyttäjiltään ja kuinka viranomaiset voivat valvoa sitä. Viranomaisten valvonta on suuri kysymysmerkki myös laajemmassa mittakaavassa - tuleeko verkolla olla valvontaa vai ei ?

Sovelluskehittimien ja tietokantojen toimittajille jatkuvasti tutkittava asia on lisenssien hinnoittelu. Hinnoittelu ei saisi olla verkon hyödyntämisen esteenä, ja toisaalta teknologian kehittäjien täytyy hyötyä panostuksistaan.

Käsitteitä

WWW-palvelin (WebServer)

WWW-palvelin ymmärretään usein sekä palvelinkoneeksi että siellä olevaksi WWW-palvelinohjelma. Webbipalvelin vastaanottaa selainten pyynnöt, hakee halutut tiedostot tai suorittaa CGI-ohjelman ja palauttaa sen sisällön selaimelle (Spainbour & Quercia 1997,261).

URL - Universal Resource Locator

Jokaisella WWW-dokumentilla tai sovelluksella on oma yksikäsitteinen URL-osoite, jonka avulla jokainen dokumentti löytyy webistä. URL-osoite koostuu kolmesta osasta: skeema (määrittelee protokollan), palvelinkoneen osoite ja tiedoston suhteellinen polku. Esimerkiksi *http://WWW.jyu.fi/index.html*. URL:n syntaksi:

http://<host>:<port>/<path>?<searchpart>

Host on WWW-palvelimen Internet-osoite

Port kertoo portin, johon selainohjelma ottaa yhteyden

Path määrittelee palvelimelle halutun tiedoston

*Searchpart*in avulla voidaan välittää tietoa palvelimelle, esimerkiksi jollekin sovellukselle. (Baker 1996.)

Selain (Browser) **eli WWW-client**

Selain on ohjelma, joka kääntää palvelimelta tulevan HTML-sivun käyttäjälle esitettävään muotoon. Selainohjelma muodostaa Webin käyttöliittymän. Selaimia on tehty useaan eri laiteympäristöön.

Plug-in

Plug-init ovat WWW-selaimien laajennuksia. Ne ovat dynaamisia koodimoduuleja, joiden käynnistämistä ohjaa selain ja ne toimivat samassa koneessa, missä selainta ajetaan. Plug-inien käyttö pohjautuu MIME-tietotyyppiin, jonka avulla selain pystyy tunnistamaan minkä ohjelman käynnistää. Plug-inistä luodaan aina uusi ilmentymä, kun selain kohtaa sitä vastaavan MIME-tietotyypin.

Plug-inien käytössä olennainen asia on, että useimmat niistä on laitteistoriippuvaisia, eli jokaiseen laiteympäristöön täytyy tehdä erillinen versio. Plug-init voidaan upottaa HTML-sivuun, jolloin ne käynnistyvät osana sivua tai ne voidaan ajaa kokonaan omassa ikkunassaan, jolloin ne käynnistetään omasta URL-osoitteesta. Useimmiten Plug-Init tehdään C/C++:lla. Tyypillisiä laajennuksia ovat videoiden ja äänen esittämiseen kehitetyt plug-init. (Sirola & Linjamaa 1996, 98-99; Plugins Design 1997; Plugins Overview 1997.)

CGI - Common Gateway Interface

CGI on WWW-palvelimen rajapinta, joka mahdollistaa ohjelmien ajamisen WWW-palvelimella. CGI-ohjelma on palvelinkoneella toimiva ohjelma. CGI-rajapinta muuntaa selaimelta tulevan tiedot ohjelman ymmärtämään muotoon ja samoin toisinpäin. (Zeltser 1995; Webmaster Magazine 1996; Sirola & Linjamaa 1996, 578.)

Yhdyskäytävä (Gateway)

Yhdyskäytävä on järjestelmäksi, joka muuntaa tiedot sovellukselta toiselle ymmärrettävään muotoon (Sirola & Linjamaa 1996, 24).

ISAPI - Internet Server API (Application Programming Interface)

ISAPI on Microsoftin Windows NT:lle kehittämä CGI-liittymän korvaava järjestelmä. ISAPI on NT:ssä CGI:tä tehokkaampi menetelmä. (Microsoft Corporation 1997.)

NSAPI - Netscape Server API

NSAPI on Netscapen CGI:n korvaava menetelmä. Voidaan käyttää ainoastaan Netscape Serverin kanssa. (The Server-Application Function and Netscape Server API 1997.)

Java

Java on Sun Microsystemsin kehittämä ohjelmointikieli, joka muistuttaa C++:aa. Laitteistoriippumatonta Java-ohjelmointikieltä käytetään WWW-ohjelmoinnissa sekä palvelimessa että selaimessa. (Sirola & Linjamaa 1996, 24.)

JavaScript

JavaScript on Netscapen Javasta kehittämä skriptikieli, joka liitetään suoraan HTML-dokumentteihin. Javascriptillä tehdään selaimessa ajettavia ohjelmia ja sitä tulkataan selaimessa. Javascriptejä käytetään esimerkiksi käyttäjän syöttötiedon validoinnissa. (Sirola & Linjamaa 1996, 24)

ActiveX

ActiveX on Microsoftin kehittämä komponenttipohjainen teknologia, jota voidaan hyödyntää sekä Microsoftin Internet Serverin että selainohjelman kanssa.

VBScript

VBScript on Microsoftin Visual Basiciin perustuva skriptikieli, joka toimii ainoastaan Microsoftin selaimen kanssa (Blum 1997, 176).

CORBA - Common Object Request Broker Architecture

CORBA on yli 700 yrityksen sopima standardi siitä, kuinka eri sovellusten tulisi kommunikoida keskenään. Englanninkielisellä termillä puhutaan *middleware*-tuotteesta, joka voidaan määritellä eri tekniikoilla tehtyjen sovellusten välissä

käytettäväksi tekniikaksi. CORBA on tällä hetkellä vielä kehitysvaiheessa oleva standardi, joten se ei vielä ole laajamittaisessa käytössä.

DCOM - **Distributed Component Object Model**

DCOM on Microsoftin kilpaileva *middleware*-tuote CORBA:lle. DCOM pohjautuu Microsoftin ActiveX-tekniikkaan.

SGML – **Standardized General Markup Language**

SGML on ISO:n hyväksymä standardi rakenteisten dokumenttien kuvaamiseen. Se on myös muotoilukielien määrittämisessä käytettävä kieli.

HTML - **HyperText Markup Language**

HTML on SGML:ään pohjautuva WWW-sivujen ulkoasun tekemisessä käytettävä kieli. Dokumentin ulkoasu määritellään ns. tageilla, esimerkiksi seuraava otsikkoteksti:

<H1>Otsikko</H1>.

DHTML - Dynaaminen HTML

HTML-kielen laajennus, joka mahdollistaa selaimen päässä tapahtuvan tiedon käytön ilman, että kaikki sivulla tapahtuvat muutokset käytäisiin hakemassa palvelimesta. Tarkoituksena on myös erottaa tieto sivun ulkoasusta. Tällä menetelmällä HTML-pohjaiset sovellukset saadaan toimimaan nopeammin, kun asiakkaan ja palvelimen välisiä yhteyksiä saadaan vähennettyä. (Dobson 1998.)

XML – Extensible Markup Language

XML on määrittelykieli strukturoidun tiedon esittämiseen. Siinä esitetään tiedosta sekä sen rakenne että varsinainen data. Rakenne esitetään XML-kielellä luotavien tageilla. XML tuo standardoidun menetelmän datan välittämiseen dokumenteissa. (What is XML ? 1999.)

WML – Wireless Markup Language

WML pohjautuu XML-standardiin ja sitä käytetään WAP selaimille tehtävien dokumenttien tekemiseen. Sen tavoitteena on minimoida verkkoliikenne, joka on langattomassa tiedonsiirrossa suurimpia pullonkauloja. (WAP.NET 1998-1999.)

Liite 1

Upotetulla Speedscriptillä tehty ohjelma käännöksen jälkeen

```
/*E4GL-W*/
{src/web/method/e4gl.i}  Lisää e4gl.i tiedoston sisällön.
{&OUT} '<HTML><BODY bgcolor="white">~n'.
{&OUT} '<CENTER>~n'.
{&OUT} '<FONT SIZE=+3><B>Customers</B></FONT>~n'.
{&OUT} '<TABLE BORDER=1>~n'.
{&OUT} '<TR> <TH>Cust #</TH> <TH>Name</TH> </TR>~n'.
 /*Tag=<!--WS4GL*/  For each customer no-lock: /*Tag=-->*/
{&OUT} '<TR>~n'.
{&OUT} '   <TD> <A HREF=customer_update.r?CustID=' /*Tag=`*/ custnum
/*Tag=`*/ '>' /*Tag=`*/ custnum /*Tag=`*/ '</A> </TD> ~n'.
{&OUT} '  <TD> ' /*Tag=`*/ name /*Tag=`*/ ' </TD>~n'.
{&OUT} '</TR>~n'.
 /*Tag=<!--WS4GL*/  End. /*Tag=-->*/
{&OUT} '</TABLE>~n'.
{&OUT} '</CENTER>~n'.
{&OUT} '</BODY>~n'.
/************************ END OF HTML ************************/
/********************* Internal Definitions ********************/
/* This procedure returns the generation options at runtime.
   It is invoked by src/web/method/e4gl.i included at the start
   of this file. */
PROCEDURE local-e4gl-options :
```

```
  DEFINE OUTPUT PARAMETER p_version AS DECIMAL NO-UNDO
    INITIAL 2.0.
  DEFINE OUTPUT PARAMETER p_options AS CHARACTER NO-UNDO
    INITIAL "web-object":U.
  DEFINE OUTPUT PARAMETER p_content-type AS CHARACTER NO-UNDO
    INITIAL "text/HTML":U.
END PROCEDURE.
/* end */
```

Liite 2

Offset-tiedosto

```
/* HTML offsets */
htm-file= D:\WEBSPEED\WRK\customer_upd.html
version= WDT_v2r1
field[1]= "RowID,input,hidden,fill-in,12,5,12,38"
field[2]= "Mode,input,hidden,fill-in,13,5,13,37"
field[3]= "ID,input,text,fill-in,17,17,17,55"
```

Tietokantaan liitettyjä FORM-elementtejä

```
field[4]= "Customer.Name,input,text,fill-in,21,17,21,66”
field[5]= "Customer.Address,input,text,fill-in,25,17,25,69"
field[6]= "Customer.City,input,text,fill-in,29,17,29,66"
field[7]= "Customer.State,input,text,fill-in,33,17,33,67"
field[8]= "Customer.Country,input,text,fill-in,37,17,37,81"
field[9]= "Customer.postalcode,input,text,fill-in,41,17,41,73"
field[10]= "Customer.salesrep,input,text,fill-in,45,17,45,71"
```

Liite 3

HTML-dokumentin pohjalta luotu logiikkatiedosto pääkohtineen

```
Väliaikaistaulu, johon talletetaan ne HTML-elementit, joita ei
kartoitettu tietokannan kenttiin. Käytetään, kuten normaaleja
muuttujia, mutta viitataan kuin taulun kenttään.
/* Temp-Table and Buffer definitions */
DEFINE TEMP-TABLE ab_unmap
       FIELD Mode AS CHARACTER FORMAT "X(256)":U
       FIELD RowID AS CHARACTER FORMAT "X(256)":U .

/*********************  Preprocessor Definitions  ******************/
&Scoped-define WEB-FILE grdhtmmap1.htm
/* Esiprosessori, joka määrää HTML-dokumentin, jota käytetään
** käyttöliittymänä. Asetetaan käännösvaiheessa.  */

/* ********************  Frame Definitions  ********************** */
DEFINE FRAME Web-Frame
/* Määrittelee ’4GL-käyttöliittymän’ eli FORM bufferin. */
     Customer.Address
          &IF '{&WINDOW-SYSTEM}' = 'TTY':U &THEN AT ROW 1 COL 1
          &ELSE AT ROW 1 COL 1 &ENDIF NO-LABEL
          VIEW-AS FILL-IN
          &IF '{&WINDOW-SYSTEM}' = 'TTY':U &THEN SIZE 20 BY 1
          &ELSE SIZE 20 BY 1 &ENDIF
     Customer.City
          &IF '{&WINDOW-SYSTEM}' = 'TTY':U &THEN AT ROW 1 COL 1
```

```
          &ELSE AT ROW 1 COL 1 &ENDIF NO-LABEL
          VIEW-AS FILL-IN
          &IF '{&WINDOW-SYSTEM}' = 'TTY':U &THEN SIZE 20 BY 1
          &ELSE SIZE 20 BY 1 &ENDIF
     ab_unmap.Mode
          &IF '{&WINDOW-SYSTEM}' = 'TTY':U &THEN AT ROW 1 COL 1
          &ELSE AT ROW 1 COL 1 &ENDIF HELP
          "" NO-LABEL FORMAT "X(256)":U
          VIEW-AS FILL-IN
          &IF '{&WINDOW-SYSTEM}' = 'TTY':U &THEN SIZE 20 BY 1
          &ELSE SIZE 20 BY 1 &ENDIF
      .
      .
      .
    WITH 1 DOWN KEEP-TAB-ORDER OVERLAY
         SIDE-LABELS
         AT COL 1 ROW 1
         SIZE 80 BY 20.

/* ******************* Included-Libraries ************************ */
{src/web2/HTML-map.i}

/* ********************  Main Code Block  ************************ */
{src/web2/template/hmapmain.i}

/* ******************  Internal Procedures  ********************** */
PROCEDURE htmOffsets :
/*------------------------------------------------------------------
  Purpose:     Runs procedure to associate each HTML field with its
```

```
                corresponding widget name and handle.
-------------------------------------------------------------------------*/
  RUN readOffsets ("{&WEB-FILE}":U).
  RUN htmAssociate
    ("Address":U,"Customer.Address":U,Customer.Address:HANDLE IN FRAME
    {&FRAME-NAME}).
  RUN htmAssociate
    ("City":U,"Customer.City":U,Customer.City:HANDLE IN FRAME
    {&FRAME-NAME}).
  RUN htmAssociate
    ("Country":U,"Customer.Country":U,Customer.Country:HANDLE IN FRAME
    {&FRAME-NAME}).
  .
  .
  .
END PROCEDURE.

PROCEDURE outputHeader :
/*-------------------------------------------------------------------
  Purpose: Output the MIME header, and any "cookie" information needed
                by this procedure.
  Parameters:  <none>
  Notes:   In the event that this Web object is state-aware, this is
           a good place to set the WebState and WebTimeout attributes.
-------------------------------------------------------------------------*/
  output-content-type ("text/HTML":U).
END PROCEDURE.
```

Liite 4

Process-web-request-aliohjelmasta

```
PROCEDURE process-web-request :
/*------------------------------------------------------------------------
  Purpose:     Process the web request.
  Notes:
------------------------------------------------------------------------*/

  /* STEP 0 -
   * Output the MIME header and set up the object as state-less or
   * state-aware.
   * This is required if any HTML is to be returned to the browser.
   *
    * NOTE: Move 'RUN outputHeader.' to the GET section  below if you
   * are going
   * to simulate another Web request by running a Web Object from this
   * procedure.  Running outputHeader precludes setting any additional
   * cookie information.
   */
  RUN outputHeader.

  /* Describe whether to receive FORM input for all the fields.  For
   * example,
    * check particular input fields (using GetField in web-utilities-
   * hdl). Here we look at REQUEST_METHOD.
   */
```

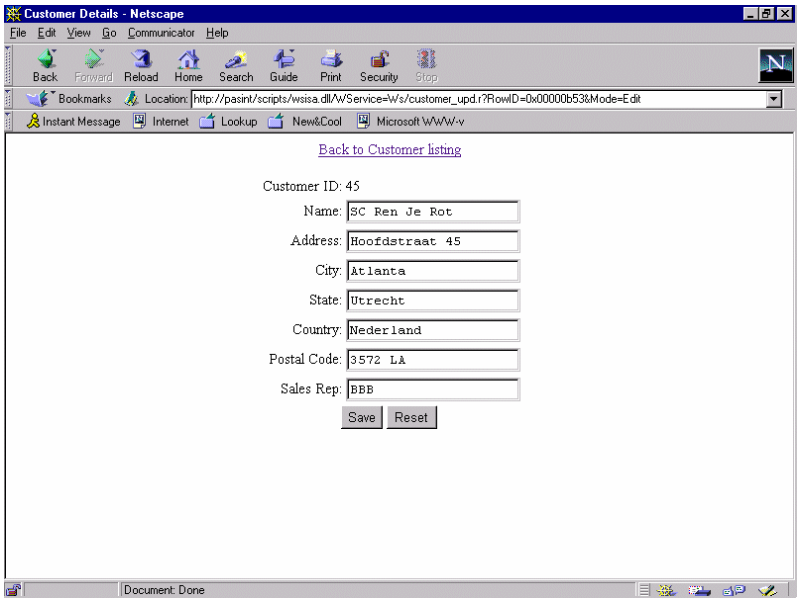
```
IF REQUEST_METHOD = "POST":U THEN DO:
  /* STEP 1 -
   * Copy HTML input field values to the Progress form buffer. */
  RUN inputFields.

  /* STEP 2 -
    * Open the database or SDO query and and fetch the first record.
  */
  RUN findRecords.

  /* STEP 3 -
   * AssignFields will save the data in the frame.
    * (it automatically upgrades the lock to exclusive while doing
   * the update)
   *
   *  If a new record needs to be created set AddMode to true before
   *   running assignFields.
   *     setAddMode(TRUE).
   * RUN assignFields. */

  /* STEP 4 -
     * Decide what HTML to return to the user. Choose STEP 4.1 to
   * simulate
    * another Web request -OR- STEP 4.2 to return the original form
   * (the default action).
   *
   * STEP 4.1 -
    * To simulate another Web request, change the REQUEST_METHOD to
   * GET and RUN the Web object here.  For example,
```

```
    *
    *  ASSIGN REQUEST_METHOD = "GET":U.
    *  RUN run-web-object IN web-utilities-hdl ("myobject.w":U).
    */

  /* STEP 4.2 -
     * To return the form again, set data values, display them, and
     * output them to the WEB stream.
     *
     * STEP 4.2a -
     * Set any values that need to be set, then display them. */
  RUN displayFields.

  /* STEP 4.2b -
     * Enable objects that should be enabled. */
  RUN enableFields.

  /* STEP 4.2c -
     * OUTPUT the Progress form buffer to the WEB stream. */
  RUN outputFields.

END. /* Form has been submitted. */

/* REQUEST-METHOD = GET */
ELSE DO:
   /* This is the first time that the form has been called. Just
    * return the
    *  form.   Move 'RUN outputHeader.' here if you are going to
    * simulate another Web request. */
```

```
    /* STEP 1 -
     * Open the database or SDO query and and fetch the first record.
    */
    RUN findRecords.

    /* Return the form again. Set data values, display them, and
     * output them to the WEB stream.
     *
     * STEP 2a -
     * Set any values that need to be set, then display them. */
    RUN displayFields.

    /* STEP 2b -
     * Enable objects that should be enabled. */
    RUN enableFields.

    /* STEP 2c -
     * OUTPUT the Progress from buffer to the WEB stream. */
    RUN outputFields.
  END.

  /* Show error messages. */
  IF AnyMessage() THEN
  DO:
      /* ShowDataMessage may return a Progress column name. This means
you
       * can use the function as a parameter to HTMLSetFocus instead of
       * calling it directly.  The first parameter is the form name.
```

```
        *
        * HTMLSetFocus("document.DetailForm",ShowDataMessages()). */
      ShowDataMessages().
  END.
END PROCEDURE.
```

Oletuksena saatu ohjelma näyttää seuraavalta:



Liite 5

CGI wrapper

```
/* ************************ Included-Libraries ****************** */
{src/web2/wrap-cgi.i}

/* ************************  Main Code Block  ********************* */
/* Process the latest Web event. */

Tämä on varsinainen ohjelma, joka käsittelee selaimesta lähetetyn
dokumentin.
RUN process-web-request.

/* ***********************  Internal Procedures ******************** */
Tulostaa HTML-dokumentin otsikkotiedot.
PROCEDURE outputHeader :
/*-------------------------------------------------------------------
   Purpose:      Output the MIME header, and any "cookie" information
                 needed by this procedure.
  Parameters:  <none>
  Notes:   In the event that this Web object is state-aware, this is
           a good place to set the webState and webTimeout attributes.
-------------------------------------------------------------------*/
   output-content-type ("text/HTML":U).   Tulostaa HTML-dokumentin.
 END PROCEDURE.
```

```
Process-web-request aliohjelmassa tehdään mahdollinen oma dynaaminen
tulostus.
PROCEDURE process-web-request :
/*------------------------------------------------------------------
  Purpose:     Process the web request.
  Parameters:  <none>
  Notes:
------------------------------------------------------------------*/
   RUN outputHeader.

  {&OUT}
    "<HTML>":U SKIP
    "<HEAD>":U SKIP
    "<TITLE> {&FILE-NAME} </TITLE>":U SKIP
    "</HEAD>":U SKIP
    "<BODY>":U SKIP
    .

  /* Output your custom HTML to WEBSTREAM here (using {&OUT}). */

  {&OUT}
    "</BODY>":U SKIP
    "</HTML>":U SKIP
    .

END PROCEDURE.
```

Jos halutaan tulostaa muissa liitteissä olevan esimerkin mukaisesti kaikkien asiakkaiden asiakasnumero ja nimi, lisätään process-web-request-aliohjelmaan seuraava koodi:

```
{&OUT} '<TABLE BORDER=1>
<TR> <TH>Cust #</TH> <TH>Name</TH> </TR>'.
For each customer no-lock:
{&OUT} '<TR><TD>
      <A  HREF=customer_update.r?CustID='  custnum  '>'  custnum  '</A>
      </TD><TD>' name '</TD></TR>'.
End.
{&OUT} '</TABLE>'.
```

Liite 6

tagmap.dat

```
# Default data mappable fields
#
# Do not move the first line below from its position.  The first line
is the
#  default field type for fields missing TYPE=.

input,,text,fill-in,web/support/webinput.p
input,,checkbox,toggle-box,web/support/webtog.p
input,,hidden,fill-in,web/support/webinput.p
input,,password,fill-in,web/support/webinput.p
input,,radio,radio-set,web/support/webradio.p
select,/select,,selection-list,web/support/weblist.p
textarea,/textarea,,editor,web/support/webedit.p

#Custom Tag that can be used to support HTML Tables and 3rd Party
controls
!--WSTAG,,,fill-in,web/support/tagrun.p

#Custom Tag that can be used to notify an application to output
messages
#messages that have queued up using the queue-message function
!--WSMSG,,,fill-in,web/support/webmsg.p
```

webinput.p

```
/*********************************************************************
**  Program: webinput.p
*********************************************************************/

/* for creating a <INPUT> field of type hidden,password, or text */
PROCEDURE web.output:
  /* Progress field containing the data */
  DEFINE INPUT PARAMETER hWid     AS HANDLE    NO-UNDO.
  /*the initial definition as contained in the HTML field definition*/
  DEFINE INPUT PARAMETER fieldDef AS CHARACTER NO-UNDO.

  /* local variable definitions */
  DEFINE VARIABLE gtPos        AS INTEGER   NO-UNDO.
  DEFINE VARIABLE resultString AS CHARACTER NO-UNDO.
  DEFINE VARIABLE screenValue  AS CHARACTER NO-UNDO.
  DEFINE VARIABLE startPos     AS INTEGER   NO-UNDO.

  ASSIGN gtPos = INDEX(fieldDef, ">":U).

  IF INDEX(fieldDef,"<INPUT":U) = 0 OR gtPos = 0 THEN
    RETURN. /* some error, get out! */

  /* If it had a value, then replace it.  Otherwise just put one in
  ** at the end of the definition before the enclosing ">".  */
  ASSIGN
    startPos = INDEX(fieldDef, "value=":U).
  IF startPos = 0 THEN
```

```
    ASSIGN
      startPos = INDEX(fieldDef, "value =":U).

  ASSIGN
    resultString = IF startPos > 0 THEN
                     SUBSTRING(fieldDef,1,startPos - 1,"CHARACTER":U)
                   ELSE
                     SUBSTRING(fieldDef,1,gtPos - 1,"CHARACTER":U).

/* OK, now we have either skipped over the original value attribute
** setting if there was one there. Lets put a new value= setting.
** Typical field def looks like this –
** <INPUT type=text name="Dave" value="Hello">. For now,
** just pass along the whole thing, and replace the value.
** For things like height, width, multiple, we need to determine if we
** want to inherit that from the HTML field def or from the Progress
** field definition.
** Note that we need to convert the contents from ascii to HTML
** in case any invalid characters are in the widget itself.
**
** If the PROGRESS widget is NOT enabled, just output the value.
*/
  screenValue  =
           IF hWid:SCREEN-VALUE = ? THEN ""
           ELSE hWid:SCREEN-VALUE.
  RUN AsciiToHTML IN web-utilities-hdl
           (screenValue, OUTPUT screenValue).
  resultString =
           IF hWid:SENSITIVE EQ NO THEN (" ":U + screenValue + " ":U)
```

```
           ELSE (resultString + ' VALUE="':U + screenValue + '">':U).
{&OUT} resultString.
END PROCEDURE. /* web.output*/

/*---------------------------------------------------------------------*/
/* for assigning the screen value from an INPUT field */
PROCEDURE web.input:
  /* Progress field containing the data */
  DEFINE INPUT PARAMETER hWid     AS HANDLE    NO-UNDO.
  /* the field value as obtained from an CGI post */
  DEFINE INPUT PARAMETER fieldVal AS CHARACTER NO-UNDO.

  ASSIGN hWid:SCREEN-VALUE = fieldVal.
END PROCEDURE. /* web.input */
/* webinput.p - end of file */
```

Lähdeluettelo

Anderson, R., Blexrud, C., Chiarelli, A., Denault, D., Homer, A., Esposito, D., Francis, B., Gibbs, M., Kropog, B., McQueen, G., Reilly, G., Robinson, S., Schenken, J., Sonderegger, D., Sussman, D. 1999. Professional Active Server Pages 3.0. Wrox Press. 35, 110-116, 314-315, 390-392, 413, 547-548, 1033-1034.

Baker, D. W. 1996. A Guide to URL's. [Viitattu 27.6.1997]. <URL:http://WWW.netspace.org/users/dwb/url-guide.html>

Blum, A. 1997. ActiveX Web Programming: ISAPI, Controls, and Scripting. John Wiley & Sons, Inc. 66-68,176.

Campbell, J. 1997. Making Good Progress. White Star Software. 84.

Clinton, W. J., Gore, A. Jr. 1997. A Framework For Global Electronic Commerce. W3C. [Viitattu 14.1.2000] <URL:http://WWW.w3.org/TR/NOTE-framework-970706.html>

Dobson R., Data Binding in Dynamic HTML. DBMS, Vol 11, Num 3, 1998, 47-59.

Dutt, G. D., Debugging CGI Gateways. Teoksessa: Black Belt Web Programming Methods: Servers, Security, Databases, and Sites. Artikkeli lehdestä: Dr.Dobb's Journal. Web Development Masters Collection, R&D Books Lawrence, Miller Freeman 1997, 51.

Floyd M., Security on the Web. Teoksessa: Black Belt Web Programming Methods: Servers, Security, Databases, and Sites. Artikkeli lehdestä: WEBTechniques. Web Development Masters Collection, R&D Books Lawrence, Miller Freeman 1997, 249.

Gorman, M. M. 1994. Enterprise Database in a Client/Server Environment. John Wiley & Sons, Inc.. 187,189-192.

Haughey, W. 1999. Strategic Approach to Value Chain Integration October 18, 1999. Object Management Group, Inc.. [Viitattu 14.1.2000] <URL:http://WWW.omg.org/omg/strategy.html>

Held, G., Revving Up Your Web Server. Teoksessa: Black Belt Web Programming Methods: Servers, Security, Databases, and Sites. Artikkeli lehdestä: Network. Web Development Masters Collection, R&D Books Lawrence, Miller Freeman 1997, 11.

Hurwitz, J., Ashton, H., E-Business Software. Enterpise Manager on DBMS, Vol 11, Num 8, 1998, 8-12.

IBM High-Volume Web site team 1999. [Viitattu 14.1.2000] <URL:http://WWW-4.ibm.com/software/developer/library/scalability>

Jaworski, J. 1998, Java 1.2 Unleashed. Sams. 102.

Järvinen, P. 1995. Internet Verkkojen verkko. WSOY.

Järvinen, P. 1996. Internet muutostekijä. WSOY. 31-33.

Leiner, B. M., Cerf, V. G., Clark D. D., Kahn, R. E., Kleinrock, L., Lynch, D. C., Postel, J., Roberts, L. G., Wolff, S. 1997. A Brief History of the Internet V3.1. [Viitattu 20.2.1997] <URL:http://info.isoc.org/internet-history/>

Netscape Communications Corporation. 1997. Plugins Design. [Viitattu 9.7.1997]
<URL: http://home15.netscape.com/eng/mozilla/20/handbook/plugins/ad3.html>

Netscape Communications Corporation. 1997. Plugins Overview. [Viitattu 7.7.1997]
<URL: http://home.netscape.com/eng/mozilla/3.0/handbook/plugins/pl1.html>

Netscape Communications Corporation. 1997. The Server-Application Function and Netscape Server API. [Viitattu 9.7.1997] <URL:http://home15.netscape.com/newsref/std/server_api.html>

Netscape Communications Corporation. 1997. Client Side State – HTTP Cookies. [Viitattu 9.7.1997] <URL:http://home15.netscape.com/newsref/std/cookie_spec.html>

Microsoft Corporation. 1997. ISAPI Overview. [Viitattu 8.7.1997] <URL:http://WWW.microsoft.com/win32dev/apiext>

Ministerial Conference Bonn 6-8 July 1997. Global Information Networks Conference 1997 – Ministereiden julistus. [Viitattu 14.1.2000] <URL:http://WWW2.echo.lu/bonn/finalfi.html>

Orfali, R., Harkey, D., Edwards, J. 1999. Client/Server Survival Guide Third Edition. John Wiley & Sons, Inc.. 529, 587-589, 616-617,621-622, 630.

Progress Software Corporation. 1998. WebSpeed Developer's Guide.

Progress Software Corporation. 1998. WebSpeed Installation and Configuration Guide for Windows NT.

Progress Software Corporation. 1998. Designing an International Application.

Sirola, H., Linjamaa, T. 1996. Internet tuottajan paketti. Teknolit Oy Gummerus Kirjapaino Oy Jyväskylä. 14,19-24,51,98-99,110,214-215,250,283,390-392,409-410,412,414-415,418,578.

Spainbour, S., Quercia, V. 1997. Webmaster Tehokäyttäjän opas. Gummerus Kirjapaino Oy. 77,80-82,97-99,153-156,171,173-174,261.

Staflin, R. 1996. HTML-ohjelmointi. Pagina AB. 11,14.

Stanek, W. R. 1996. HTML, JAVA, CGI, VRML, SGML WebPublishing Unleashed. Sams.net Publishing. 15-17,35,383,427.

W3C. 1999. Internationalization: W3C activities. [Viitattu 14.1.2000] <URL:http//WWW.w3.org/international/Activity.html>

WAP.NET 1998-1999. [Viitattu 9.12.1999]
<URL:http://WWW.option.com/techno/wap.htm>

Webmaster Magazine. 1996. Overview of the World Wide Web. [Viitattu 27.6.1996] <URL:http://WWW.cio.com/WebMaster/sem2_process.html>

What is XML ? 1999. [Viitattu 7.12.1999]
<URL:http://xml.com/xml/pub/98/10/guide1.html>

Zeltser L. 1995. The World-Wide Web: Origins And Beyond. [Viitattu 21.4.1995] <URL:http://WWW.seas.upenn.edu/lzeltser/WWW>

Özsu, M. T., Valduriez, P. 1999. Principles of Distributed Database Systems. Prentice-Hall, Inc.. 16, 18-20, 62, 283-285, 288, 358-359, 420-423, 596.