TERMTOOL – termipankki
Työkalu kääntäjien terminologiseen työhön

Denis Mustonen
Tampereen yliopisto
Kieli- ja käännöstieteiden laitos
Käännöstiede (venäjä)
Pro gradu -tutkielma
Toukokuu 2009

Tampereen yliopisto
Käännöstiede (venäjä)
Kieli- a käännöstieteiden laitos

MUSTONEN, DENIS: TERMTOOL – termipankki, Työkalu kääntäjien terminologiseen työhön
Pro gradu tutkielma, 62 sivua + liitteet (4 kpl) (+ venäjänkielinen lyhennelmä 10 sivua)
Kevät 2009

Tutkielma tehtiin osana suurempaa projektia, jonka tavoitteena oli luoda terminologiseen työhön soveltuva termipankkisovellus ja kehittää olemassa olevaa termitietokantaa terminologiseen työhön sopivaksi. Projektin päätarkoituksena oli tarjota Tampereen yliopiston kieli- ja käännöstieteiden laitoksen käyttöön helppokäyttöinen ja monipuolinen työkalu terminologiapainotteisten töiden toteuttamiseen. Tutkielma on siis melko käytännönläheinen ja perustuu todelliseen sovellukseen.

Oikeanlaiset työkalut tehostavat työskentelyä merkittävästi ja terminologinen työ asettaa useita omanlaisia vaatimuksia termipankkia kohtaan, jotka eroavat muuhun käyttöön, kuten käännöstyökaluksi, suunnatuista termipankeista. Näitä eroavaisuuksia ja vaatimuksia käsitellään tässä tutkielmassa. Vaatimukset kohdistuvat pääosin termiartikkelien esitykseen, mutta riippuen työkalun pääasiallisesta käyttöfunktiosta muitakin eroavaisuuksia esiintyy.

Termipankkisovellus toteutettiin verkkosovelluksena käyttäen uusia sovelluskehityssuosituksia ja menetelmiä. Projektin toteutuksessa käytettiin useita erilaisia, mutta toisiaan tukevia ohjelmointi- ja merkintäkieliä (PHP, HTML, JavaScript, CSS ja SQL), mikä mahdollisti sovelluksen monipuolisemman kehityksen. Riippumatta teknisistä yksityiskohdista sovelluskehityksen tulisi kuitenkin aina olla käyttäjälähtöistä käytettävyyden ollessa sen tärkeimpänä osa-alueena. Sen vuoksi suunnitteluvaiheessa käytettiin myös paljon aikaa käytettävyyssuunnitteluun. Tehtyjä valintoja ja kohdattuja ongelmakohtia pyritään kuvaamaan ja perustelemaan tässä tutkielmassa.

Termipankkisovelluksessa on myös suuri jatkokehityksen tarve, koska projektin puitteissa ei toteutettu kaikkia toimintoja, jotka katsottiin olevan kriittisiä kokonaisuuden kannalta resurssien puutteen vuoksi. Jatkokehitysprojektit on pyritty ottamaan monipuolisesti huomioon toteutuksessa.

Avainsanat: *terminologia, termipankki, verkkosovellus, tietokanta, ohjelmointi*

SISÄLLYS

1. JOHDANTO

Teknologinen kehitys tuo ulottuvillemme jatkuvasti uusia mahdollisuuksia tehostaa työskentelyä tai työskennellä täysin uusilla tavoilla lähestulkoon kaikilla aloilla. Kääntäjien työssä viimeisen kymmenen vuoden aikana yhä enemmän suosiota ovat saaneet käännösmuistit, elektroniset sanakirjat ja muut työtä tehostavat välineet. Nykyään näiden välineiden käyttö on jo lähes vaatimuksena kustannustehokkaalle ammatin harjoittamiselle ja tutkimuksen tekemiselle. Tutkimustyössä työtä tehostavien työkalujen merkityksen ymmärtämisestä ja suosiosta viestii mm. korpusvetoisen tutkimuksen selkeä lisääntyminen viime vuosina. Tällainen kehitys tuo mukanaan paljon uusia mahdollisuuksia, mutta samalla asettaa jatkuvasti yhä kovempia tietotaidollisia vaatimuksia käyttäjiä kohtaan.

Tampereen yliopiston kieli- ja käännöstieteiden laitoksella tehdään jatkuvasti terminologiapainotteisia töitä, mutta tutkijoiden käyttöön ei ollut saatavilla helppokäyttöisiä ja tehokkaita apuohjelmia työn tehostamiseksi. Tämänkin projektin aikana toteutettiin rinnan kaksi terminologista pro gradu tutkielmaa (Olga Sakurina & Viktoria Abrosimova: *Разработка и составление финско-русского глоссария по теме "Строительно-отделочные и лакокрасочные материалы"* ja Merja Kerimaa: *Venäjän työoikeuden termien tarkastelua*) ja useita pienempiä töitä. Samalla kyseiset henkilöt toimivat järjestelmän testaajina, joiden palautteen perusteella järjestelmää pyrittiin muokkaamaan vastaamaan paremmin tutkijoiden tarpeita.

Alkuperäinen ajatus termipankin luomisesta tuli Mikhail Mikhailovilta ja oman kiinnostukseni pohjalta päätin lähteä mukaan projektiin, jossa tavoitteena oli luoda tutkimustyöhön soveltuva termipankkisovellus. Kiinnostukseni kautta minulla on usean vuoden kokemus internetsivustojen luomisesta ja eri tekniikoista niiden toteuttamiseen. Toisaalta kääntäjäopiskelijana minulla on myös selkeä käsitys terminologisesta työstä, sen erityispiirteistä ja siitä, mitä ominaisuuksia kääntäjäkollegani voisivat tarvita terminologisessa tutkimuksessa. Olen aina pitänyt itseäni enemmän websuunnittelijana (eng. web-designer) kuin varsinaisena ohjelmoijana ja sen vuoksi minulla on erityinen kiinnostus käytettävyyteen ja käyttäjälähtöiseen toteutukseen, mikä heijastui vahvasti sovelluksen suunnitteluun ja toteutukseen.

Toteuttamani termipankkiprojekti sijoitettiin Tampereen yliopiston ”mustikka” -palvelimelle (http://mustikka.uta.fi), jolla on käynnissä useita muitakin projekteja joissa pyritään luomaan ja kehittämään ennen kaikkea käännöstutkimusta helpottavia tietoteknisiä välineitä. Tällä tavoin toteutettu termipankki tuli osaksi suurempaa työkaluvalikoimaa.

Vaikka projekti toteutettiin pitkälti yksilöprojektina, en voi väittää tehneeni kaiken yksin. Ohjaajani Mikhail Mikhailovin aito kiinnostus aiheeseen edesauttoi projektin etenemistä merkittävästi ja hän oli tärkein lähteeni projektin monissa terminologisissa kysymyksissä hänen laaja-alaisen terminologisen ja leksikografisen osaamisen kautta. Myös ystäväni Kaarlo Lahtela tarjosi minulle tärkeää tukea erityisesti tietokanta ja JavaScript ohjelmointiin liittyvissä kysymyksissä. Kaarlo on ohjelmoinnin monitaituri ja hänen paneutuminen esittämiini ongelmiin ja pitkät keskustelumme auttoivat useista jopa umpikujalta näyttävistä tilanteista.

Sovelluskehitys on yksi nopeitten kehittyvistä aloista ja toteutustavat vanhentuvat hyvin nopeasti. Sen vuoksi monesti kirjoissa oleva tieto on vanhentunutta ja parhaimpia lähteitä ongelmaratkaisuun ovat erilaiset sovelluskehityksen ja suunnittelun ammattilaisille tarkoitetut blogit ja artikkelikokoelmat, ohjelmointifoorumit, ongelmanratkaisukeskukset ja postituslistojen arkistot. Näiden lähteiden käyttö vaatii tietysti laajaa pohjatietoa ja laaja-alaista tutkimustyötä, koska lähdekritiikin merkitys korostuu silloin kun lähteinä käytetään muita kuin tutkimustuloksiin perustuvaa kirjallisuutta. Projektin aikana käytettiin useita resursseja, joita ei voida suoraan merkitä lähteiksi, koska ne tarjosivat lähinnä ideoita ja yksittäisten ohjelmointi- tai muotoiluongelmien ratkaisukeinoja. Näiden resurssien tarjoama informaatio oli kuitenkin monessa suhteessa korvaamatonta projektin toteutuksen kannalta. Tällaisen tiedon etsiminen vaatii kuitenkin hakukoneiden tehokasta käyttöä ja saadun informaation tarkastamista useista paikoista.

2. KESKEISIÄ TERMEJÄ JA KÄSITTEITÄ

Luvussa käydään lyhyesti läpi työn kannalta keskeisiä termejä ja käsitteitä. Listan tarkoituksena on tarjota mahdollisimman lyhyt määritelmä annetuille termeille ja toimia ikään kuin muistilistana ja johdantona yksittäisiin käsitteisiin. Monet työssä esiintyvät käsitteet vaativat kuitenkin laajempaa paneutumista ja siten yksittäiset termit ja käsitteet käsitellään työn muissa osioissa.

2.1. Terminologiseen työhön liittyviä käsitteitä

(Termi)artikkeli; termitietue: Termipankin osa, joka sisältää yksittäiseen käsitteeseen välittömästi liittyvän tiedon. Tällainen tieto voi olla termi, määritelmä, esimerkit, vastineet jne.

Hakusana: Merkkijono, jolla haetaan tietoa termipankista.

Käsite: Käytetään tiedon jäsentämiseen. Käsitteet eivät välttämättä ole kielisidonnaisia, mutta yhteiskunta- ja kulttuurisidonnaiset tekijät vaikuttavat usein erilaisen käsitejaon syntymiseen eri

kielissä. Kaikille käsitteille ei välttämättä ole vakiintunutta niitä vastaavaa kielellistä ilmausta. (TSK 36, 2006.)

Leksikografia: Sanasto-oppi, jossa tutkitaan sanaston kuvauksen ja esittämisen periaatteita. Käytännössä leksikografi on sanakirjankirjoittaja. (Wikipedia.)

Määritelmä: Käsitteen kuvaus, jonka tulee erottaa käsite sen lähikäsitteistä (TSK 36, 2006).

Termi: Käsitteen kielellinen nimitys, jonka avulla voidaan lyhyesti viitata käsitteen koko sisältöön. (TSK 36, 2006.)

Terminologia; terminologiaoppi: Oppi käsitteiden ja termistöjen rakenteesta, muodostamisesta, kehityksestä, käytöstä ja käsittelystä eri erikoisaloilla (TSK 36, 2006).

Terminologinen työ: Työ, johon kuuluu tietyn erikoisalan käsitteitä ja niiden nimityksiä koskevan tiedon systemaattinen kerääminen, analysointi, kuvaaminen ja esittäminen (TSK 36, 2006).

Termipankki: Erikoisalan sanakirja, joka koostuu erikoisalojen käsitteisiin ja niiden nimityksiin liittyvästä tiedosta.

Vastine: Käsitteen vieraskielinen nimitys, jonka avulla voidaan lyhyesti viitata käsitteen koko sisältöön.

2.2. Sovelluskehitykseen liittyviä käsitteitä

Asiakas; pääte (eng. client): Tietokone, joka hakee tietoja palvelimilta ja esittää ne käyttäjille.

CSS-tyylimäärittely (eng. Cascade Style Sheet): Yleisesti sanottuna dokumentin ulkoasua koskeva ehdotus, joka on kirjoitettu erityisesti tähän tarkoitukseen kehitetyllä kielellä. Tarkemmin sanoen CSS on kieliperhe, johon kuuluvat nykyisin CSS1 ja sitä paljon laajempi CSS2; lisäksi CSS3 on valmisteilla. (Korpela, 2007.) Eri selainten tuki tyylimäärittelyille vaihtelee, mutta uusimmilla selaimilla CSS2 on jo laajasti tuettu, minkä vuoksi tässä työssä käytetään nimenomaan CSS2 mukaisia määrittelyjä.

Dynaaminen toiminto: Toiminnot, joilla luodaan vuorovaikutus käyttäjän ja sovelluksen välille. Toteutuksessa dynaamisuus tarkoittaa sitä, että verkkosivu luodaan vasta, kun selain sitä pyytää.

Eväste (eng. cookie): Tieto, jonka palvelin tallentaa käyttäjän tietokoneelle. Evästeet helpottavat resurssin käyttäjien yksilöimistä.

Hakemisto; kansio (eng. directory): Tietokonejärjestelmässä massamuistille luotu ja nimetty tiedostojen tallennuspaikka. Sisäkkäiset hakemistot muodostavat **hakemistorakenteen** eli hakemistopuun. (Wikipedia.)

Iteraatio: Pieni ohjelmistoprojekti, joka sisältää kaikki uusien toimintojen julkaisemiseksi tarvittavat tehtävät. Ohjelmistokehityksessä iteroinnilla tarkoitetaan prosessien toistamista.

JavaScript: Tulkattava komentosarjakieli, jota käytetään erityisesti Web-sovellusten dynaamisten selainelementtien (client-side scripting) luonnissa (Korpela, 2007).

jQuery: Erillinen JavaScript-kirjasto, jonka tarkoituksena on yksinkertaistaa sovelluskehitystä.

Järjestelmä (eng. system): Erillisistä sovelluksista ja tietovarastoista muodostuva kokonaisuus.

Käyttäjäprofiili: Sovelluksen kuvitteellisen käyttäjän kuvaus.

Käyttöliittymä (eng. UI; User Interface): Laitteen, ohjelmiston tai minkä tahansa muun tuotteen osa, jonka kautta käyttäjä käyttää tuotetta. Esimerkiksi tietokoneohjelmassa käyttöliittymä tarkoittaa sitä ohjelman osaa, jonka käyttäjä näkee tietokoneen näytöllä, ja sitä tapaa, jolla hän käyttää ohjelmaa. (Wikipedia.)

Käyttöskenaario: Sovelluksen kuvitteellinen käyttötilanne, joiden avulla sovelluksen toimintaa määritellään, suunnitellaan ja testataan käyttäjäkeskeisessä sovelluskehityksessä.

Merkintäkieli (eng. markup language): Formaali kieli, jolla kuvataan tekstin rakennetta tai esitystapaa metainformaatiolla. Merkintäkielellä pyritään erottamaan tekstin looginen rakenne sisällöstä. Merkintäkielistä varmasti käytetyin on **HTML** (eng. Hypertext Markup Language). (Wikipedia.)

(Sovellus)moduuli: Itsenäinen osa, jollaisista voidaan koota erilaisia kokonaisuuksia. Tässä työssä moduuleilla tarkoitetaan sovelluksen toiminnallisia osioita, jotka rakentuvat rungon ympärille.

Palvelin (eng. server): Tietokone, joilla verkkoresurssit sijaitsevat.

PHP (eng. PHP: Hypertext Preprocessor): Tulkattava komentosarjakieli, jossa ohjelmakoodi tulkitaan vasta ohjelman suoritusvaiheessa. PHP on alustariippumaton ja sitä käytetään erityisesti Web-palvelinympäristöissä dynaamisten web-sivujen (server-side) luonnissa. (http://www.php.net)

Selain (eng. browser): Ohjelma, jota tarvitaan verkkoresurssien käyttöä varten.

Sovelluksen runko: Ohjelmistotuote, joka muodostaa rungon sen päälle rakennettavalle sovellukselle. Sovelluksen runkoa ei voida yleensä käyttää sellaisenaan.

SQL (eng. Structured Query Language): Yleisesti relaatiotietokannan käsittelyssä käytettävä kysely- ja määrityskieli (Lahtonen ym., 2003).

Tietokanta (eng. database): Tietotekniikassa käytetty termi tietovarastolle. Se on kokoelma tietoja, joilla on yhteys toisiinsa. (Wikipedia.) **Relaatiotietokanta** on yksi tietokantamalleista, joka on tällä hetkellä selvästi käytetyin malli (Lahtonen ym., 2003).

URI (eng. URI: Uniform Resource Identifier): Merkkijono, jolla kerrotaan tietyn tiedon paikka (URL) tai yksikäsitteinen nimi (URN). Erityisesti URI:n erikoistapausta URL:ää (eng. URL: Uniform Resource Locator) käytetään osoittamaan verkkosivuja. (http://tools.ietf.org/html/rfc2396)

3. TERMINOLOGINEN TYÖ

Sanastotyö eli **terminologinen työ** on työ, johon kuuluu tietyn erikoisalan käsitteitä ja niiden nimityksiä koskevan tiedon systemaattinen kerääminen, analysointi, kuvaaminen ja esittäminen. Sanastotyö perustuu terminologiaan ja koostuu usein monista työvaiheista, joita ovat esimerkiksi sanastus, käsiteanalyysi, määritelmien kirjoittaminen, sopivien nimitysten valinta ja termistön esittäminen terminologisena sanastona tai termitietokantana. (TSK 36, 2006.)

Sanastotyön päämääränä on yleensä alaa koskevan viestinnän tehostaminen. Ilman systemaattista sanastotyötä viestintä vaikeutuu merkittävästi, sillä erikoisaloilla täsmällisiin käsitteisiin viittaavat termit ovat hyvin yleisiä ja ominaisia. On hyvin tärkeätä, että esimerkiksi leikkaussalissa kaikki tietävät mitä välinettä tarkoitetaan tai, että oikeussalissa ei synny väärinkäsityksiä kun puhutaan eri asioista. Pahimmillaan tällaiset väärinkäsitykset voivat siis johtaa ihmisten kuolemaan tai vääränlaisiin tuomioihin.

Terminologinen työ voidaan jakaa kahdentyyppiseen sanastotyöhön sen luonteen perusteella:

- deskriptiiviseen, jossa pyritään kuvaamaan käytettävää termistöä ja käsitteistöä
- normatiiviseen, jossa pyritään selkeyttämään ja yhdenmukaistamaan termistön käyttöä

Jotta erikoisalan sanasto palvelisi tarkoitustaan, se on laadittava huolellisesti ja käyttäen hyväksi sanastonteon yleisiä periaatteita ja menetelmiä. Niinpä sanastotyössä tarvitaan paitsi alan itsensä myös terminologian asiantuntemusta, minkä vuoksi terminologinen työ toteutetaan yleensä

projektimuotoisena ja siihen osallistuu niin terminologeja, kuin kyseisen alan asiantuntijoita. (Sanastokeskus TSK.)

Yksikielisten erikoisalojen sanastojen merkitystä kääntäjien arjessa ei pidä missään nimessä väheksyä, koska kääntäjät ovat harvoin sen alan ammattilaisia, mitä käännökset koskevat, mutta erityisen tärkeässä roolissa käännöstyössä ovat monikieliset sanastot ja niiden laadinta. Monikielisissä sanastoprojekteissa tehdään myös systemaattista vastinetyötä, jossa pyritään sovittamaan erikoisalan erikieliset käsitejärjestelmät yhteen. Tällainen lähestymistapa on hyvin haasteellinen ja sen vuoksi kohdekielet jätetään yleensä pelkkien termivastineiden tasolle (Sanastokeskus TSK).

3.1. Mikä on termipankki?

Termipankki on käsitettävä pelkäksi työvälineeksi muiden ohella, se ei siis tarjoa tapaa automatisoida terminologista työtä tai käännöstyötä. Esimerkiksi erilaisia sovelluksia sanakirjojen automaattiseen keräämiseen kehitetään jatkuvasti, mutta tällaiset sovellukset ovat monesti osoittautuneet epäluotettaviksi varsinkin erikoisalojen kohdalla. Termipankin on toki tarkoitus helpottaa työskentelyä monella tavoin riippuen käyttötarkoituksesta. Terminologiseen työhön se tarjoaa mm. mahdollisuuden säilöä ja ylläpitää tietoa ja linkittää käsitteitä toisiinsa ja sanastojen käyttöön se tarjoaa mm. monipuoliset hakuominaisuudet tietojen hakuun. Termipankki käsitteenä on kuitenkin hyvin ongelmallinen ja sen suhde muihin hakuteoksiin, kuten sanakirjoihin, on vaikeasti määriteltävissä.

3.1.1. Termipankin ja sanakirjan ero

Tutkijoiden mukaan sanakirjalle ei ole tällä hetkellä olemassa yleispätevää määritelmää. Kudashev mainitsee useita syitä tällaiseen tilanteeseen, mutta tärkeimpänä voidaan pitää varsinkin 1900-luvun loppupuolella tapahtunutta nopeata kehitystä leksikografiassa, mistä johtuen perinteinen sanakirjan määritelmä teoksena, jossa määritellään sanojen merkitys, alkoi kadota. Tällä hetkellä monet asiantuntijat luokittelevat oikeastaan kaikki hakuteokset, joilla on sanakirjan rakenne sanakirjoiksi. (Kudashev, 2007.) Yhtenä syynä nopeaan kehitykseen pidetään tietoteknisten apuvälineiden nopeaa yleistymistä ja kehittymistä. Nämä apuvälineet muuttavat jatkuvasti sanakirjojen luonnetta ja lähentävät erilaisia sanakirjoja ja lähdeteoksia toisiinsa.

Jos ajatellaan painetun sanakirjan ja elektronisen sanakirjan eroja, on helppoa huomata näiden kahden välinen ero sanakirjan rakentumisen luonteessa ja rajoituksissa. Elektroninen versio poikkeaa painetusta muun muassa siinä suhteessa, että artikkelit päivitetään säännöllisesti ja että elektronisessa teoksessa voi olla perusartikkelien ja kuvien lisäksi erilaisia oheisaineistoja, kuten

kartastoja, ääni- ja videonäytteitä ym. (Honkala, 2007.) Muutosten tekeminen painettuun mediaan on hankalaa ja yleensä tapahtuu uusien painosten kautta, jolloin virheellinen tai vanhentunut tieto on painettujen sanakirjojen ja muiden hakuteosten yleinen ongelma. Elektronisten sanakirjojen ja hakuteosten perusajatus on muovautunut kohti jatkuvaa kehitystä, mikä tarkoittaa ideologian tasolla sitä, etteivät ne ole koskaan valmiita ja käytännön tasolla sitä, että virheet ja vanhentunut tieto on mahdollista korjata nopeasti, jolloin saatavilla oleva tieto on mahdollisimman ajan tasalla.

Painettu sanakirja asettaa myös useita rajoituksia muun muassa fyysisen kokonsa puolesta sisällön määrälle ja esitysasulle, tällaista rajoitusta elektronisissa sanakirjoissa ei suoranaisesti ole. Varsinkin tietoteknisen kehityksen tarjoamat monipuoliset haku- ja lajitteluominaisuudet, myös muulle sisällölle kuin pelkästään tekstille, poistavat täysin perinteisen kiinteän sanaston lajitteluperusteen valinnan ongelman. (Kudashev, 2007.)

Termipankki on pohjimmiltaan erikoisalan sanakirja, mutta käsitteenä se viittaa nimenomaan tietotekniseen ratkaisuun. Erikoisalan sanakirja on taas sanakirjan alakäsite, jolloin se pitää käsittää suuremman kokonaisuuden yhtenä tekijänä. Sanakirja käsitteenä viittaa kuitenkin yleensä leksikografiseen teokseen, joten erojen tarkastelu voidaan rajata leksikografian ja terminologian eroihin. Tutkijat jakavat leksikografian ja terminologian seuraavalla tavalla (Kudashev, 2007):

- leksikografiassa pyritään määrittelemään yleiskielen sanoja ja terminologiassa erikoiskielten sanoja
- leksikografisissa teoksissa lajittelu tapahtuu yleensä aakkosjärjestyksen mukaan ja terminologisissa ideografisesti
- leksikografia on tavallisesti deskriptiivinen ja terminologia preskriptiivinen
- leksikografiset teosten käyttäjäryhmä on tavalliset ihmiset ja terminologisten asiantuntijat
- leksikografisten teosten päätarkoitus on informaation tulkinta ja terminologisten informaation koodaus

Tarkemmin tutkittuna tällainen jako ei ole yleispätevä, mutta mielestäni riittävä erottamaan sanakirjan ja erikoiskielen sanakirjan erot. **Termipankki** käsitteenä on kuitenkin hieman harhaanjohtava, sillä se viittaa enemmänkin termikokoelmaan. Termit ovat kuitenkin vain yksi erikoissanaston osa (Kudashev, 2007). Termipankki tulisikin käsittää ennemmin erikoiskielen artikkelikokoelmana, jossa yksittäiset **artikkelit** (teknisesti: termitietueet) sisältävät termin ja siihen liittyvät muut tiedot, kuten määritelmän, esimerkit, vieraskieliset vastineet ja synonyymit. Haku tietokannasta ei siis tuota yksittäisiä termejä vaan artikkeleja, joissa termi toimii ikään kuin otsikkona.

3.1.2. Termipankin käyttöfunktio terminologisessa työssä

Termipankkeja ja sanastoja käytetään moneen eri tarkoitukseen: sinne tallennetaan tietoa, sieltä haetaan tietoa, saadun tiedon perusteella tehdään johtopäätöksiä ja ratkaisuja jne. Jotta oikeanlainen toiminta olisi mahdollista, termipankin tulisi tukea käyttötarkoitusta johon sitä käytetään, koska esimerkiksi tiedon vääränlainen esitysmuoto voi johtaa täysin vääriin johtopäätöksiin. Käyttötarkoituksen näkökulmasta termipankille on havaittavissa ainakin kaksi erilaista käyttöfunktiota:

- käsitteiden välisten suhteiden määrittely ja käsitejoukon hahmottaminen
- tietyn termin määritelmän, synonyymin, vieraskielisen vastineen ym. löytäminen tai määritelmän perusteella tietyn termin löytäminen

Ensimmäisessä käyttöfunktiossa huomion kohteena ovat käsitejoukot ja sitä kautta termipankin artikkeliryhmät ennalta määrätyn rajauksen perusteella. Tarve tällaiselle lähestymistavalle on tavallinen terminologisessa työssä ja sen vuoksi erityisen mielenkiintoinen tämän työn kannalta, koska toteutettava termipankki tulisi palvelemaan pääasiallisesti nimenomaan terminologisen työn tekijöitä.

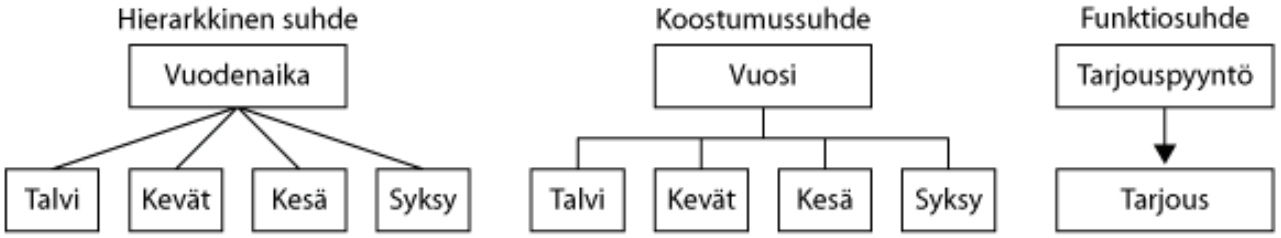
Toisena esitetyssä käyttöfunktiossa huomion kohteena ovat taas yksittäiset artikkelit. Tällainen lähestymistapa on tavallinen käännöstoiminnassa tai muussa tekstien laadinnassa, jolloin ongelma, jota pyritään ratkaisemaan, on muodostunut jonkun yksittäisen tiedon, kuten termin, puutteesta tai termin käytön epävarmuudesta.

Esitetyt käyttöfunktiot eivät ole toisiaan poissulkevia, vaan lähinnä täydentäviä, sillä kyllähän terminologisessa työssä kiinnostus voi kohdistua myös yksittäiseen artikkeliin ja taas käännöstyössä tai tekstin laadinnassa kysymykseen usein tulee käsitejoukon hahmottaminen. Termipankin ensisijainen käyttötarkoitus on kuitenkin otettava huomioon sovelluksen suunnittelussa, sillä vääränlainen tai puutteellinen työkalu voi työn tehostamisen sijan päinvastaisesti hidastaa tai vaikeuttaa työtä.

3.1.3. Käsitesuhteista muodostuva ongelma

Tavallisesti termipankissa olevia termejä yhdistetään toisiinsa synonyymien ja vieraskielisten vastineiden kautta puuttumatta niiden suhteeseen toisiinsa. Tästä aiheutuu vakava ongelma kokonaisuuden hahmottamisen näkökulmasta. Termipankin käyttö työkaluna vaatii tämän ongelman tiedostamista ja siihen paneutumista.

Termit ovat käsitteiden kielellisiä nimityksiä, joiden avulla voidaan lyhyesti viitata käsitteen koko sisältöön – edellyttäen, että se on tunnettu. Terminologisista työmenetelmistä tärkein on käsiteanalyysi, jossa selvitetään kunkin käsitteen olennainen sisältö ja käsitteiden väliset suhteet. Näiden suhteiden perusteella muodostetaan käsitejärjestelmiä. Käsiteanalyysissä eritellään yleensä kolmenlaisia käsitesuhteita (Kuva 3-1) (TSK 36, 2006):



Kuva 3-1. Käsitesuhteet

- **Hierarkkinen suhde** vallitsee laajemman yläkäsitteen ja sitä suppeamman alakäsitteen välillä. Alakäsite sisältää tällöin kaikki yläkäsitteen piirteet sekä vähintään yhden lisäpiirteen, ja sitä vastaa suppeampi joukko tarkoitteita kuin yläkäsitettä. Alakäsite voidaan siis ajatella yläkäsitteen erikoistapaukseksi.
- **Koostumussuhteessa** alakäsitteet ovat osia yläkäsitteenä olevasta kokonaisuudesta. Yläkäsitteen piirteet eivät kuitenkaan sisälly alakäsitteeseen kuten hierarkkisessa käsitejärjestelmässä.
- **Funktiosuhteina** kuvataan laaja joukko erilaisia käsitesuhteita, joita ei voida luokitella hierarkkisiksi tai koostumussuhteiksi. Niitä ovat esimerkiksi ajalliset, paikalliset, toiminnalliset, välineelliset sekä alkuperään ja syntyyn liittyvät suhteet. Funktiosuhteen tyyppi käy yleensä ilmi määritelmän kielellisestä muodosta.

Käsitejärjestelmät ovat tavallisesti moniulotteisia ja sekakoosteisia. Moniulotteisuudella tarkoitetaan sitä, että yläkäsitteestä voidaan päästä eri jaotteluperusteita käyttäen erilaisiin alakäsitevalikoimiin. Sekakoosteisuus puolestaan tarkoittaa sitä, että samassa käsitejärjestelmässä esiintyy useita eri käsitesuhdetyyppejä. (TSK 36, 2006.)

Monikielisissä sanastoprojekteissa käsitesuhteiden lisäksi termeillä on myös suhde niiden vieraskielisiin vastineisiin. Sanastotyössä vastineet eritellään yleensä kolmenlaisiksi vastineiksi (Kalliokuusi & Seppälä, 1999):

- **Termivastineet** ilmaisevat kohdekielessä samanlaista käsitettä vakiintuneen termin muodossa.

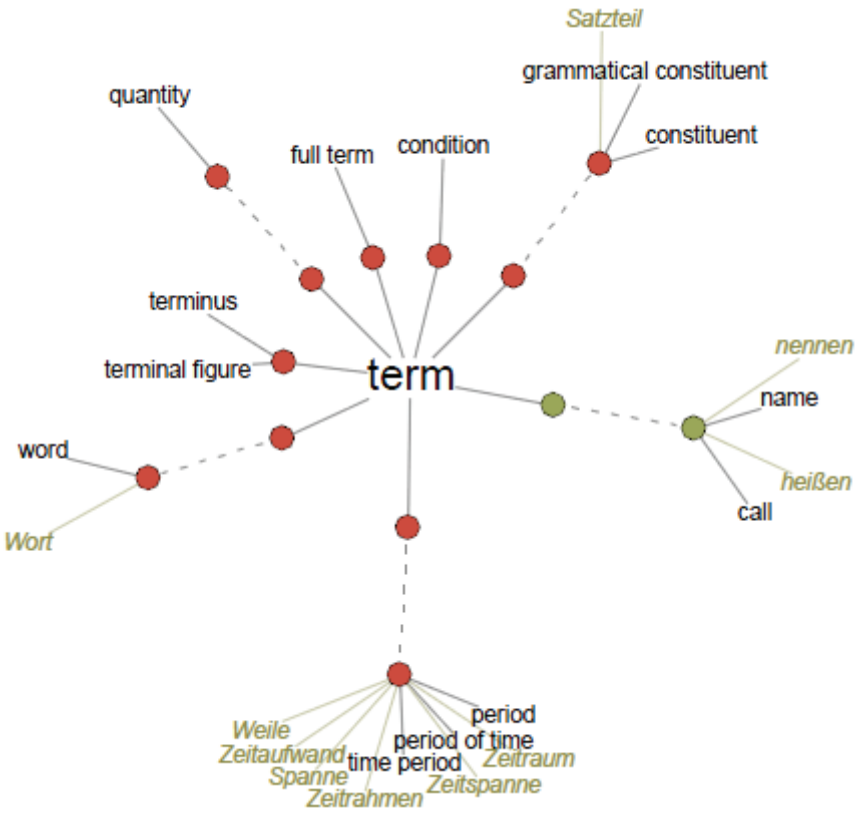
- **Käännösvastineet** muodostetaan ilmaisemaan lähdekielistä käsitettä silloin kun kohdekielestä puuttuu vastaava käsite ja termi.
- **Lähivastineet** taas ovat ilmauksia, jotka ilmaisevat kohdekielistä lähikäsitettä. Kohdekielisen lähikäsitteen ja lähdekielisen käsitteen välillä on tällaisessa tapauksessa riittävästi yhteisiä piirteitä erilaisista erottavista piirteistä huolimatta. Lähikäsitteet voivat olla ylä- tai alakäsitteitä tai vieruskäsitteitä tai sellaisia käsitteitä, jotka ovat jollakin tapaa merkittävässä funktiosuhteessa verrattavaan käsitteeseen nähden.

Edellä mainituista eriasteisista käsitesuhteista ja vastineiden luonteesta termien linkitys toisiinsa synonyymien ja vastineiden kautta täysin puuttumatta niiden keskinäisiin suhteisiin on ongelmallista ja harhaanjohtavaa. Ongelma tiedostetaan täysin, mutta oikeanlainen suhteiden kuvaus on teknisesti erittäin haastava tehtävä toteuttaa. Kyse ei ole pelkästään toteutuksen alla olevasta termipankista, vaan yleisesti ottaen kaikki termipankit toimivat tällä logiikalla. Parhaimmillaankin käsitejärjestelmäkaaviot ovat erillisiä osia, joita ei ole sidottu millään tavoin varsinaisiin artikkeleihin. Termipankkien kehityksessä tulisikin ottaa paremmin huomioon käsitejärjestelmät ja niiden integrointi osaksi sovelluksen toimintaa.

Sanastokeskus TSK:n suositus sanastojen talletukselle: ”*Erikoisalojen sanastot tulee tallentaa käsitekohtaisesti: kunkin käsitteen kaikki tiedot (määritelmät, erikieliset termit synonyymeineen jne.) sijoitetaan yhteen ns. termitietueeseen, ja kussakin termitietueessa kuvataan (määritellään) täsmälleen yksi käsite.*” eritasoisten vastineiden vuoksi on liian yksinkertaistettu lähestymistapa, sillä näin voisi menetellä vain termivastineiden kohdalla. Olenkin vahvasti sitä mieltä, että erikieliset artikkelit tulisi erottaa kokonaan omiksi artikkeleiksi. Näin on mahdollista antaa mahdollisimman tarkat määritelmät termeille (vertaa: Venäjän duuma ja Suomen eduskunta) ja samalla voidaan välttää se vaara, että vieraskielinen käsite saisi määritelmän toisesta ympäristöstä (esimerkiksi Venäjän duuma kuvattaisiin Suomen eduskunnaksi). Tarkemmin ajateltuna ylipäätänsä hyvin harva termi saa toisessa kielessä tarkan termivastineen johtuen kulttuurisista ja järjestelmien välisistä eroista. Riippuen tarkastelun tarkkuudesta myös hyvin tavalliset asiat sisältävät maakohtaisia eroavaisuuksia. Esimerkiksi pistorasia on erimallinen joissakin maissa (vertaa: Iso-Britannia ja Suomi) tai se tarjoaa eri käyttöjännitteen (vertaa: USA ja Suomi). Mitä abstraktisemmista asioista on kyse, sitä enemmän nämä erot korostuvat (vertaa: peruskoulu Suomessa ja Venäjällä).

Yksi ratkaisu termien suhteiden määrittelyn problematiikkaan olisi luopuminen synonyymeistä ja vastineista kokonaan, jolloin termien linkitys toteutettaisiin graafisesti käsitekaavioiden tapaan. Tällaisessa ratkaisussa tarkastelun alla olevan artikkelin termi olisi aina graafisen kaavion keskellä

ja linkitys muihin termeihin tapahtuisi käsitejärjestelmän ja vastinesuhteiden kautta. Vastaavanlainen ongelma on havaittavissa myös leksikografiassa ja sen ratkaisemiseksi on kehitetty mm. thesaurus-tyyppisiä sanakirjoja ja erityisesti sähköisessä mediassa esimerkiksi WordNet (http://wordnet.princeton.edu) ja FrameNet (http://framenet.icsi.berkeley.edu) -tapaisia sovelluksia. Tässä suhteessa leksikografissa kehitys on siis paljon pidemmällä kuin terminologiassa ja sieltä voi saada paljon tietoa ongelman ratkaisemiseksi (Kuva 3-2).



Kuva 3-2. Kaksikielinen (englanti-saksa) esimerkki leksikografiasta hakusanalla ”term”: Tapa kuvata termien suhteita toisiinsa termikeskeisesti (http://www.visualthesaurus.com)

Luvussa kuvattua toimintaa ei toteutettu tässä termipankkiprojektissa sen teknisen monimutkaisuuden ja käyttäjäystävällisen toteutuksen vaikeuden vuoksi (graafinen esitys asettaa korkeat tietotaitovaatimukset käyttäjiä kohtaan – niin tiedon tallentamisen kuin myös tiedon lukemisen ja tulkinnan suhteen). Graafisen kuvauksen toteuttaminen vaatisi kokonaan oman ohjelman, jota ei ole mahdollista toteuttaa niillä ohjelmointikielillä, joita käytettiin tässä projektissa (tarkemmin: *4.5 Mikä on verkkosovellus?*). Tärkeintä on kuitenkin ongelman tiedostaminen, niin sovelluskehittäjien kuin myös käyttäjien toimesta. Projektin termipankin sovelluksessa ongelma on ratkaistu toteutuksessa keinotekoisesti eli kommentti-kenttään voi lisätä artikkelia koskevat kommentit, joita voivat olla mm. termin suhde toisiin termeihin.

4. TERMIPANKIN TOTEUTUKSEN TAUSTATEKIJÖITÄ

Sovelluskehitysprojekti määräytyy aina tietyn taustatiedon perusteella ja siinä noudatetaan tiettyjä periaatteita ja metodeja pitääkseen projektin ”kasassa” ja johdonmukaisena. Jotta voisi ymmärtää sovelluskehityksen taustalla olevia prosesseja ja niitä ratkaisuja joita tehdään sovelluksen toteutuksen yhteydessä, on tiedettävä sovelluskehityksen perusperiaatteet ja erityisesti verkkosovelluksen rakentumis- ja toimintamekanismit. Ilman tällaista pohjatietoa toteutusvaiheiden ja sovelluksen toiminnan tason hahmottaminen voi olla hankalaa ja osittain jopa mahdotonta. Luvun tarkoitus on tarjota pohjatietoa projektin määrittelystä, sovelluksesta yleisesti, sovelluskehityksestä ja niistä sovelluskehityksen periaatteista, joita sovellettiin termipankin toteutuksen yhteydessä. On myös muistettava, että projektissa tehdyt valinnat eivät ole se ainoa oikea ratkaisu, sillä sovelluskehitysmetodeja on useita erilaisia ja ne usein ovat jopa ristiriidassa toistensa kanssa.

4.1. Nykyinen järjestelmä

Sovelluksen toteutus alkaa projektia vastaavien olemassa olevien työkalujen kartoituksesta, joista projektia edeltävän järjestelmän arviointi on selkein ja helpoiten rajattavissa oleva kokonaisuus. Tampereen yliopiston käännöstieteiden laitoksella oli jo olemassa ennen projektin alkua termipankin tietokanta ja alkeellinen verkkosovellus sen rajoitettuun käyttöön. Verkkosovellus käsitti vain sisään kirjautumisen, termien lisäämisen ja rajoitetun haun termeistä. Siitä siis puuttuvat täysin mm. laajemmat hakuominaisuudet, raportointityökalut ja ohjeet. Nykyisen verkkosovelluksen arkkitehtuuri ei kuitenkaan mahdollistanut sen jatkokehitystä toivotulla tavalla, koska sovelluksen runko rajoittaa jatkokehitysmahdollisuuksia ja käytössä olevat vanhat ohjelmointitavat ovat ristiriidassa nykyisten suositusten ja standardien kanssa. Toisaalta olemassa olevan sovelluksen alkeellisuudesta johtuen sitä ei myöskään ollut mielekästä lähteä muuttamaan suunnitelman mukaiseen järjestelmään sopivaksi.

4.2. Katsaus olemassa oleviin sovelluksiin

Kääntäjät ovat sovelluskehityksessä melko marginaalinen ryhmä, joten työtä helpottavien työkalujen valikoima on melko pieni verrattuna moneen muuhun alaan. Monen olemassa olevan työkalun käyttöfunktio on myös eri verrattuna projektin aikana toteutettavaan sovellukseen, sillä markkinoilla olevat sovellukset ovat suunniteltu pääasiallisesti ammattikääntämiseen käsitteiden yhdenmukaisen nimeämisen turvaamiseksi ja suunnitteilla olevan sovelluksen pääasiallinen käyttötarkoitus on terminologinen työ (tarkemmin: *3.1.2 Termipankin käyttöfunktio terminologisessa työssä*). Termityökaluilla on kuitenkin paljon yhteistä ja analysoimalla olemassa

olevia työkaluja on usein myös mahdollista löytää toimivia ratkaisuja projektin toteutuksen kannalta.

EU:n vaikutus näkyy selkeästi terminologisen työ tarpeessa, koska useita termien yhdenmukaistamisprojekteja ja termipankkeja on yhtäaikaisesti käynnissä ympäri Eurooppaa EU:n tukemana. Suurin osa kartoituksen aikana löydetyistä projekteista olivat jonkun tietyn erikoissanaston monikielisiä kokoelmia. Tällaisista erikoissanastoprojekteista voidaan merkittävimmiksi erottaa InterActive Terminology for Europe (IATE) (http://iate.europa.eu), EuroTermBank (http://www.eurotermbank.com), NORDTERM:in termipankkiprojektit (http://www.nordterm.net) ja suomen mittakaavassa Sanastokeskus TSK:n termipankkiprojektit (http://www.tsk.fi) ja monialainen valtioneuvoston termipankki VALTER (http://www.valter.fi).

Yhteistä kaikille edellisille projekteille oli se, että ne olivat suljettuja eli ennalta valitun ryhmän kokoamia erikoisalojen sanastoja. Myös kokoelmien kielivalikoimassa näkyi EU:n vaikutus eli ne käsittivät vain joitakin EU:n maiden kieliä. Ratkaisu on täysin ymmärrettävä, sillä termipankin sisällön oikeellisuuden ja luotettavuuden varmistaminen onnistuu vain koordinoidusti ja ammattilaisten toteuttamana.

Kartoituksen aikana löytyi myös muutama omien termipankkien luomiseen tarkoitettu termipankkisovellus. Näistä toteutettavaa projektia parhaiten vastasivat Tanskan kansallisen terminologiakeskuksen (DANTERMcentre) I-Term (http://www.i-term.dk), The Star Group:in WebTerm (http://www.star-group.net) ja SDL MultiTerm Online (http://www.sdl.com) verkkosovellukset.

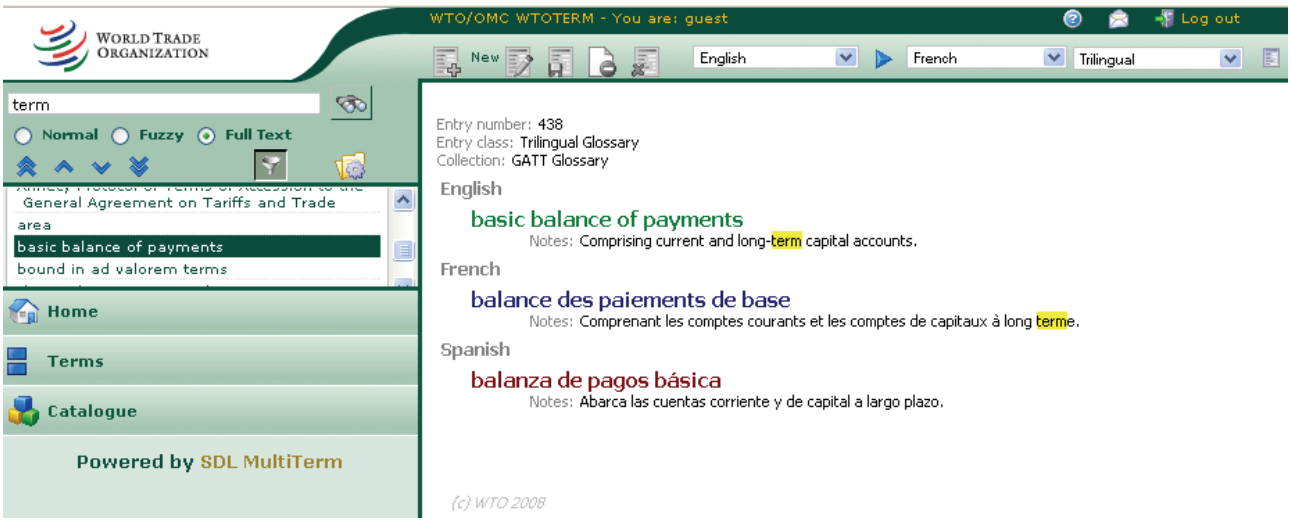
Kaikki verkkosovellukset olivat hyvin monipuolisia ja sisälsivät monia edistyksellisiä elementtejä, kuten I-Termin käsitekaavioiden luomiseen tarkoitettu työkalu. Jo aikaisemmin mainittu sovellusten pääasiallinen käyttötarkoitus on kuitenkin tekstien kääntäminen, mistä viestii jo se, että WebTerm ja MultiTerm ovat molemmat oheistuotteita valmistajilta, joiden päätuotteita ovat käännösmuistiratkaisut, mikä on myös toisaalta näiden tuotteiden vahvuus käännöstyössä, koska termipankit ovat integroitavissa käännösmuistien toimintaan, jolloin termipankkien tiedot ovat suoraan käytettävissä käännöksissä.

Kaikki kolme tuotetta ovat yksilöräätälöitäviä kokonaisuuksia, joten voi olla että niihin on saatavissa myös erilaisia toimintalogiikoita, mutta ne versiot joita pääsin kokeilemaan soveltuvat parhaiten monikielisten termisanakirjojen laadintaan ja tutkimuskäyttöön ne eivät mielestäni sellaisenaan sovellu. Näiden sovellusten käyttöä myös rajoittaa niiden maksullisuus ja lisenssien rajoitteet. Vain I-Termin hinta oli ilmoitettu: 10000€ / enintään 10 käyttäjää + 1340 € vuosittainen

ylläpitomaksu, joten ei voi tehdä johtopäätöksiä muiden sovellusten hintatasosta, mutta uskon, että ne ovat vielä kalliimpia. Suurin ongelma on kuitenkin juuri käyttäjämäärän rajoitus, koska se ei mahdollista väliaikaisten projektien toteutusta, mikä on esimerkiksi yliopistoympäristössä hyvin tärkeä osa tutkimustyötä.

Etsinnän aikana löytyi myös useita palveluja, jotka tarjoavat, ilmaiseksi tai maksua vastaan, alustan käyttäjien termipankkiprojekteille. Pääsin kokeilemaan vain täysin avoimia ja ilmaisia projekteja, koska löydetyistä maksullisista sovelluksista ei löytynyt kokeiluversioita. Ulkopuolisen palvelun käyttö omissa termipankkiprojekteissa on kuitenkin aina riskialtista, mikäli palveluntarjoaja ei ole ennestään tunnettu ja vakavarainen yhtiö, koska esimerkiksi kaikissa esiin tulleissa ratkaisuissa termipankki sijoitettiin palvelutarjoajan palvelimille ja tietokannan ollessa kyseessä, sen siirtäminen palvelimelta toiselle voi olla hyvin hankalaa, jolloin termipankki voi syystä taikka toisesta pahimmassa tapauksessa kadota osittain tai kokonaan, jolloin jopa monien vuosien työ voi mennä hukkaan.

Yhteenvetona voi sanoa, että kaikki kartoitusvaiheessa esiin tulleet luotettavat termipankit ovat suljettuja projekteja ja lähes kaikkien projektien ja sovellusten pääkäyttötarkoitus on yksittäisten termin käännösvaihtoehdon tai määritelmän tarjoaminen. Tästä johtuen lähestulkoon kaikki termipankkisovellukset toimivat sanakirjatyyppisesti (Kuva 4-1 ja Kuva 4-2), jolloin ne soveltuvat hyvin yksittäisten artikkelien hakuun, mutta esimerkiksi terminologiseen työhön tällainen lähestymistapa on mielestäni täysin väärä.



Kuva 4-1. SDL MultiTerm Online (http://wtoterm.wto.org)

Kuva 4-2. Kielikone OY:n NET MOT-ohjelmarunko termipankkityökalussa (http://www.tsk.fi/tepa)

Sanakirjatyyppinen toimintaperiaate, jossa hakujoukko esitetään tyypillisesti vasemmalla olevaan valikkoon, josta voi valita haluttu artikkeli, minkä jälkeen valitun termin artikkeli esitetään oikealla puolella olevassa ikkunassa, ei mahdollista samanaikaista työskentelyä artikkelinjoukon kanssa. Sanakirjatyyppisen toimintaperiaatteen suurin etu on helppo luettavuus, jolloin saa hyvän yleiskuvan hakujoukosta. Tässä projektissa tämä ongelma kuitenkin pyritty ratkaisemaan dynaamisten toimintojen avulla (tarkemmin: *5.6.2. Tietojen esitys*).

4.3. Projektin tavoitteiden määrittely

Tarve ohjelmistoprojektille ja projektin toteutuksen sisältö määräytyy pitkälti jo olemassa olevien työkalujen kartoittamisen kautta. Tässä tulee tarkastella niin projektia edeltävää järjestelmää kuin myös muita markkinoilla olevia työkaluja, jotka potentiaalisesti voisivat soveltua niihin tarkoituksiin, mitä varten sovellusta suunnitellaan. Projektin toteutus on kannattava vain siinä tapauksessa, jos tulevalle sovellukselle asetettuja vaatimuksia (helppokäyttöisyys, toiminnot, hinta, tehovaatimukset jne.) ei ole mahdollista täyttää edeltävän järjestelmän tai jonkin muun olemassa olevan ratkaisun avulla. Tavoitteiden määrittely on siis vahvasti sidoksissa vaihetta edeltäneen työkalujen kartoitukseen. Toteutettavalle projektille määriteltiin alustavasti kaksi tavoitetta:

1. Olemassa olevan termipankin tietokannan kehitys terminologiseen työhön sopivaksi.
2. Uuden verkkosovelluksen luominen termipankin tietokannan käyttöä varten.

Jo alun perin tavoitteiden määrittelyn yhteydessä oli tiedossa, ettei verkkosovellusta ole mahdollista toteuttaa kokonaisuudessaan tämän projektin puitteissa resurssien rajallisuuden vuoksi. Tarkempi

rajaus ei kuitenkaan ollut mahdollinen ennen sovelluksen varsinaista suunnittelua, koska kaikkia tarvittavia sovelluksen osia ei vielä tässä vaiheessa ollut tiedossa ja mahdollisten osien toteutuksen vaativuustasoa ei ollut määritelty. Tarkempi sovelluksen toimintojen rajaus projektin alkuvaiheessa olisi myös mahdollisesti estänyt havaitsemasta sellaisten toimintojen tarvetta, joita ei otettu alussa huomioon.

Sovellus päätettiin toteuttaa mahdollisimman modulaarisena eli sovellus koostuisi rungosta ja itsenäisistä sovellusosista. Tällainen toteutustapa mahdollistaa toimivan sovelluksen, jonka toimintoja on mahdollista laajentaa kehityksen myötä ja mahdollisimman vaivattoman jatkokehityksen. Vaivattoman jatkokehityksen mahdollistaminen oli erityisen tärkeässä roolissa, koska sovelluksen kaikkia toimintoja ei ollut mahdollista toteuttaa tämän projektin puitteissa.

Projektia edeltävä sovellus on puutteestaan huolimatta täysin käyttökelpoinen, siksi ensisijaisesti pyrittiin luomaan uusia moduuleita, joita ei vanhassa järjestelmässä ole ja jotka ovat termipankkityöskentelyn kannalta kriittisiä. Vanha järjestelmä on ainakin toistaiseksi suunniteltu käytettäväksi rinnan uuden kanssa.

Järjestelmälle asetettiin useita vaatimuksia, jotka muovautuivat projektin edetessä, mutta näistä tärkeimpinä olivat käyttäjäystävällisyys, alhainen oppimiskynnys ja käyttäjien terminologisen työn helpottaminen parhaalla mahdollisella tavalla. Yhteistä näille kaikille vaatimuksille on käyttäjien käyttäytyminen, sillä huonosti toteutettua palvelua ei käytetä enempää kuin on välttämätöntä. Vaikka sovelluksen käytettävyys oli projektin yksi tärkeimmistä mittareista, en halua puhua tässä vaiheessa käytettävyydestä, koska se on mielestäni liian laaja käsite vaatimusten määrittelyn suhteen. Vaikka 90 % internetsivustoista on huonosti toteutettuja, se ei tarkoita, että käyttäjät käyttäisivät 90 % ajastaan niiden käyttöön. Todellisuudessa he käyttävät 90 % ajastaan hyvin toteutettujen sivustojen käyttöön ja vain 10 % huonosti toteutettuihin. Tällä tavoin hyvin toteutetut palvelut saavat helposti lojaaleja käyttäjiä. (Nielsen, 1998.) Näin ollen teknisesti hyvin toteutettu sovellus ei välttämättä ole käyttäjäystävällinen, jolloin käyttäjät pyrkivät löytämään muita ratkaisuja ongelman ratkaisemiseksi. Kappaleessa kuvatut vaatimukset pyrittiin ottamaan huomioon toteutuksen kaikilla osa-alueilla suunnittelusta testaukseen.

Ei voi liikaa korostaa, että tässä projektissa ollaan tekemisissä oikeiden ihmisten käyttäjätietojen kanssa ja sovellus mahdollistaa melko laajan vuorovaikutuksen asiakkaan ja palvelimen välille. Sen vuoksi tietoturva on erityisen tärkeä osatekijä ja vaatimus projektin toteutuksessa. Tietoturva on otettava huomioon monella eri tasolla niin teknisessä toteutuksessa, jossa varmistetaan sovelluksen tietoturvallinen toiminta, kuin myös projektin kuvauksissa ja ohjeissa. Sen vuoksi monia projektin

liittyviä asioita, joiden katsotaan vaarantavan tietoturvallisuutta, ei kuvata tässä työssä enempää kuin on välttämätöntä.

4.4. Relaatiotietokanta

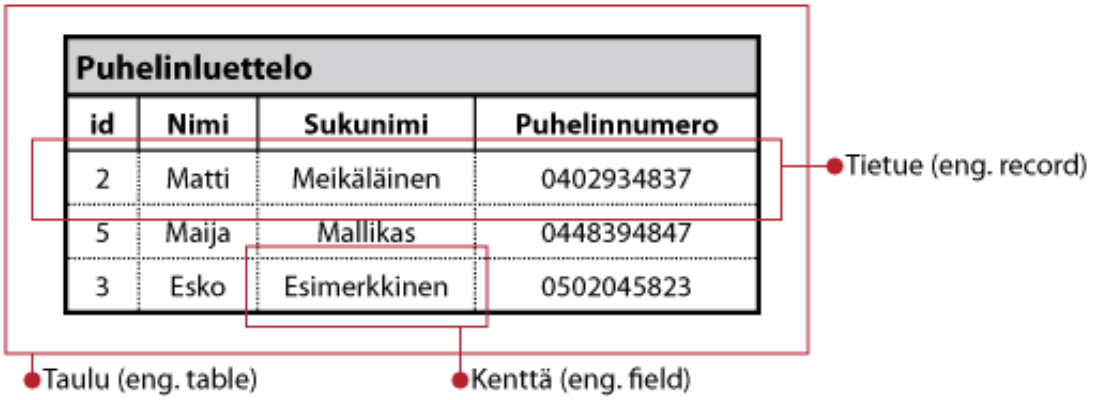
Tietokanta on kokoelma yhteen liittyvää tietoa eli tosiasioita, jotka voidaan kirjata ja joilla on jokin merkitys. Tietokantoja on monenlaisia ja laajuudeltaan erikokoisia ja jokapäiväisen elämän esimerkeistä tavallista puhelinnumeromuistiota voidaan pitää hyvin pienenä ja yksinkertaisena tietokantana. (Lahtonen ym., 2003.) Kaikkia tietokantoja yhdistävät kuitenkin seuraavat ominaisuudet (Lahtonen ym., 2003):

- Tietokannalla on jokin lähde, josta sen sisältämä tieto on peräisin.
- Tietokannalla on jotain tekemistä todellisen maailman tapahtumien kanssa.
- Tietokannalla on käyttäjiä, jotka ovat kiinnostuneita sen sisällöstä.

Termipankin toteutuksessa on käytössä relaatiotietokanta, joka on yksi yleisimmistä tietokantamalleista verkkosovellusten toteutuksessa ja sen hallintajärjestelmänä on PostgreSQL (tarkemmin: *4.5.1. Tietokantataso ja sen rakenne*).

4.4.1. Relaatiotietokannan rakenne

Relaatiotietokannassa tiedot esitetään **tauluina** (engl. table), joita kutsutaan myös relaatioiksi. Tietokannassa voi olla yksi tai useampia tauluja. Yhtä riviä kutsutaan **tietueeksi** (engl. record) ja taulut voivat sisältää tai olla sisältämättä tietueita (tyhjä taulu). Taulun jokaisella rivillä on yhtä monta tietoa eli **kenttää** (engl. field) eli tietue koostuu yhdestä tai useammasta kentästä. (Kuva 4-3) (Lahtonen ym., 2003.)



Kuva 4-3. Tietojen esitys relaatiotietokannassa

Jokaiselle taululle pitää määritellä yksikäsitteinen **perusavain** (engl. primary key), joka yksilöi kyseisen taulun sisältämät tietueet. Jokaisella taulussa olevalla tietueella pitää olla jokin yksilöllinen (eng. unique) kenttä tai kenttien yhdistelmä, jollaista ei ole yhdelläkään toisella samassa taulussa

olevalla tietueella. (Lahtonen ym., 2003.) Puhelinluetteloesimerkissä (Kuva 4-3) perusavaimeksi voisi sopia puhelinnumero, koska eri ihmisillä ei voi (yleensä) olla samaa puhelinnumeroa, kun taas esimerkiksi sukunimi voi olla useammalla henkilöllä samanaikaisesti sama. Yksiselitteisin tapa määritellä perusavain on kuitenkin luoda sille oma kenttä ja esimerkiksi termipankkisovelluksen jokaisen taulun perusavain on selvyyden vuoksi id-kenttä.

Perusavaimen määrittäminen ei ole pakollista, mutta kuuluu hyvään ohjelmointitapaan, sillä se mahdollistaa taulujen ja tietojen linkittämisen ja niihin viittaamisen yksilöllisesti. Järjestetyn taulun käsitteleminen on myös huomattavasti tehokkaampaa kuin järjestämättömän ja tietokannan hallintajärjestelmä pitää taulun järjestyksessä perusavaimen mukaan (Lahtonen ym., 2003).

Taulun kentistä voidaan viitata myös jonkin toisen taulun tietoihin **viiteavaimen** (eng. foreign key) avulla. Yleensä tällainen viiteavainkenttä viittaa perusavaimeen, niin että jokaista viittaavassa taulussa esiintyvää viiteavaimen arvoa pitää vastata sama perusavaimen arvo viittauksen kohteena olevassa taulussa. (Lahtonen ym., 2003.) Tällainen tietokannan toiminta tulee kysymykseen esimerkiksi silloin kun artikkelille määritellään kieli, jolloin artikkelitietueessa olevaa viiteavainta vastaa kielitaulussa oleva kielen tietue (tarkemmin: *4.5.1. Tietokantataso ja sen rakenne*). Viiteavaimet ovat tärkeä osa tietokannan viite-eheyttä (eng. referential integrity), millä pyritään siihen, että tietokannan rakenne on looginen, eikä sisällä ylimääräisiä arvoja. Viite-eheys määrää, että jokaista viittaavassa taulussa esiintyvää viiteavaimen arvoa pitää vastata sama perusavaimen arvo viittauksen kohteena olevassa taulussa (Lahtonen ym., 2003).

Tietokannan avaimet eivät näy suoranaisesti loppukäyttäjille millään tavoin, mutta ovat tärkeä osa teknistä toteutusta eli vaikka termipankin käyttäjät eivät hae tietoa tietokannasta avainten avulla moni tieto haetaan teknisellä tasolla juuri avainviittausten mukaisesti, mikä mahdollistaa monipuolisemman tietokantarakenteen.

4.4.2. Kyselyehtojen määrittely

SQL (Structured Query Language) on standardoitu ja laajimmin käytössä oleva relaatiotietokantojen yhteydessä käytettävä kieli (Lahtonen ym., 2003). Relaatiotietokannan rakenne määritellään ja sinne lisättävää tietoa lisätään, muokataan, haetaan ja poistetaan kyselyiden (eng. query) avulla. Kyselyt ovat tarkoin määriteltyjä SQL-lauseita, joita yhdistämällä on mahdollista toteuttaa hyvin monipuolisia toimintoja (Listaus 4-1). Tässä työssä keskitytään kuvaamaan ainoastaan tietojen tietokannasta hakua ja siihen vaikuttavia tekijöitä, koska loppukäyttäjän kannalta erityisen haasteellista on hahmottaa hakuehtojen suhde toisiinsa. Loppukäyttäjän näkökulmasta tällaista ongelmaa ei esimerkiksi ole havaittavissa tietojen lisäyksen yhteydessä, koska siinä on

käytössä kiinteä lomake ja kenttien suhde on käyttäjälle toisarvoista. Hakuehtojen suhteen hahmottaminen vaatii paneutumista SQL-lauserakenteeseen ja SQL-kielen perusperiaatteisiin.

```
/*
Peruslause, jossa valitaan kaikki kentät taulusta ”taulu”.
HUOM! Hyvän ohjelmointitavan mukaisesti *-merkkiä ei tulisi käyttää, vaan
jokainen kenttä on osoitettava erikseen.
*/
SELECT * FROM taulu;

/*
Tarkemmin määritelty lause, jossa valitaan kaksi kenttää (kentta1, kentta2)
taulusta ”taulu”, jotka vastaavat hakuehtoa kentta3 on 1 ja kentta2 sisältää
kirjainyhdistelmän ”haku”. Tieto lajitellaan kentän ”kentta1” mukaan laskevaan
järjestykseen.
*/
SELECT kentta1, kentta2 FROM taulu WHERE kentta3 = 1 AND kentta2 LIKE '%haku%'
ORDER BY kentta1 DESC;
```

Listaus 4-1. Esimerkki SQL-kyselylauseesta

SQL-ehtolauseet (`WHERE`-osuus) rakennetaan erilaisten operaattoreiden avulla. Termipankin toteutuksessa käyttäjän kannalta oleellisia operaattoreita ovat:

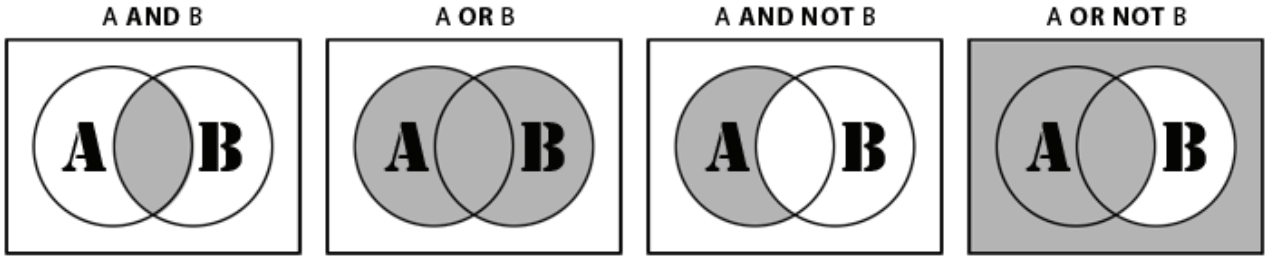
- vertailuoperaattorit (eng. comparison operators) (= yhtä kuin, < pienempi kuin, > suurempi kuin, <= pienempi tai yhtä suuri kuin, >= suurempi tai yhtä suuri kuin ja <> erisuuri kuin)
- laajennetut vertailuoperaattorit (eng. extended comparison operators) (`LIKE`, `IN`, `BETWEEN` ja `IS NULL`)
- loogiset operaattorit (eng. logical operators) (`AND`, `OR`, `NOT`)

Vertailuoperaattoreiden avulla voidaan hakea arvojoukkoja asetetulla välillä. Termipankissa laajassa käytössä on =, <= ja >= operaattorit ja niitä käytetään erityisesti eksaktin arvon (esim. käyttäjä = <käyttäjänimi> tai luontipäivämäärä = <PP.KK.VVVV>) hauissa ja aikavälien rajauksissa (esim. luontipäivämäärä <= <PP.KK.VVVV>). Vertailuoperaattorit myös esitetään käyttäjille niiden merkkimuodossa (=, <=, >=) yksiselitteisen ja lyhyen esitystavan vuoksi. Oletettavasti käyttäjät tuntevat nämä symbolit muista yhteyksistä (matematiikka, algebra jne.).

Laajennetuista vertailuoperaattoreista on käytössä ainoastaan `LIKE`. `LIKE` -operaattorin avulla voidaan hakea kirjoitusasultaan toisiaan muistuttavia rivejä (eng. pattern matching) ja sen vuoksi se on käytössä kaikissa hauissa, joissa käytetään hakusanoja. `LIKE` -operaattorin yhteydessä käytetään jokerimerkkejä (eng. wildcards), joiden avulla määritetään minkä tyyppistä vastaavuutta haetaan. Mahdollisia jokerimerkkejä on kaksi: % (prosenttimerkki) vastaa mitä tahansa merkkijonoa ja _ (alaviiva) vastaa yhtä merkkiä. Näistä termipankissa käytössä on vain prosenttimerkki, joka

sijoitetaan hakusanan molemmin puolin. Esimerkiksi SQL-lause: `SELECT kentta1 FROM taulu WHERE kentta1 LIKE '%pyörä%'` tuottaa kaikki tulokset, joissa esiintyy sana ”pyörä” kentässä ”kentta1” taulussa ”taulu” eli seuraavat tulokset ovat mahdollisia: polku*pyörä*, polku*pyörä*n vanne, etu*pyörä*veto, jako*pyörä*stö jne.

SQL-kielessä hakuehtojen yhdistäminen toisiinsa perustuu pohjimmiltaan loogisille operaattoreille. Loogisten operaattoreiden toiminnan ymmärtäminen on tärkeä tehokkaan hakutyöskentelyn kannalta, oli kyseessä sitten hakukoneella toteutettavat haut taikka termipankin toiminta. Logiikasta peräisin olevat operaattorit on mahdotonta korvata, joten ne ovat aina läsnä käyttäjien tekemissä hakuratkaisuissa, joko suoraan tai taustalla. Esimerkiksi Google-haku hakusanoilla `koira kissa` on todellisuudessa `koira AND kissa`, koska Google lisää automaattisesti `AND` operaattorin hakusanojen väliin (http://www.google.fi/intl/fi/help/basics.html).



Kuva 4-4. Loogiset operaattorit

Loogiset operaattorit (Kuva 4-4) kuvataan yleensä kahden hakusanoilla rajatun arvojoukon avulla (kuvan ympyrät). Hakusanoja tässä kuvaavat A ja B, joita voivat olla myös mitkä tahansa muut hakusanat kuten X ja Y tai konkreettisimmista vaikkapa Kissa ja Koira. Nämä kaksi arvojoukkoa ovat osa suurempaa epämääräistä arvojoukkoa (kuvan laatikko), joka voi olla myös ääretön. Epämääräinen arvojoukko voi olla kaikki mahdolliset tietokannan arvot taikka esimerkiksi internetsivujen hakukoneen, kuten Googlen, ollessa kyseessä kaikki mahdolliset tiedossa olevat internetsivustot (mitkä ovat myöskin tietokannassa olevat arvot).

Loogisia operaattoreita on kaksi `AND` ja `OR` ja niillä on myös käänteiset vastineet `AND NOT` ja `OR NOT` eli termipankin käytön kannalta perusoperaattoreita on yhteensä neljä. Kun käytössä on `AND`, molempien ehtojen on täytyttävä ja `OR` operaattorin kohdalla vain toisen ehdon on täytyttävä. Sama pätee myös näiden käänteisiin versioihin. Kuvasta (Kuva 4-4) voimme kuitenkin huomata, että viimeinen `OR NOT` operaattori on epälooginen käytön kannalta. Tästä syystä erilaisissa toimintakuvauksissa ja manuaaleissa loogiset operaattorit rajataankin yleensä kolmeen, jolloin `AND NOT` operaattori kuvataan yleensä yksinkertaistetusti: `NOT` operaattorina. `OR NOT` on kuitenkin täysin mahdollinen ja sitä kautta sen ottaminen mukaan tarkasteluun selventää mielestäni loogisten

operaattorien toimintaperiaatetta. `OR NOT` ei ole myöskään estetty termipankin hakutoiminnossa, jolloin sitä on mahdollista käyttää, vaikkei se ole tuettu toiminto ja näin ollen se ei ole kuvattu esimerkiksi käyttöohjeissa.

Kun hakuehtoja on enemmän kuin kaksi, niiden toteutumisen järjestystä joudutaan usein säätelemään, koska järjestys vaikuttaa lopputulokseen (Listaus 4-2). Järjestyksen määritys tapahtuu sulkeiden avulla samalla tavoin kuin esimerkiksi matematiikassa.

```
(A AND B) OR C = x
A AND (B OR C) = y

(2 * 3) + 4 = 10
2 * (3 + 4) = 14
```

Listaus 4-2. Sulkeiden käyttö hakuehtojen yhteydessä ja matemaattisissa yhtälöissä

Sulkeiden käyttöön pätee myös samanlaiset erityispiirteet sulkeiden avaamisen kohdalla kuin matematiikassakin eli esimerkiksi De Morganin lain mukaisesti käänteisten operaattoreiden kohdalla sama lopputulos voidaan saavuttaa kahdella erilaisella lausekkeella (Listaus 4-3).

```
NOT (A OR B) = (NOT A) AND (NOT B)
NOT (A AND B) = (NOT A) OR (NOT B)
```

Listaus 4-3. De Morganin laki: OR ja AND operaattorien suhde toisiinsa negaatiossa (Wikipedia)

Sulkeiden käyttö ei suoranaisesti näy käyttäjälle hakujen yhteydessä, mutta teknisesti on haastavaa arvata etukäteen missä järjestyksessä käyttäjä haluaisi hakusanat käsiteltäväksi. Sen vuoksi hakutoimintalogiikka on rakennettava aina käyttöskenaarioiden perusteella.

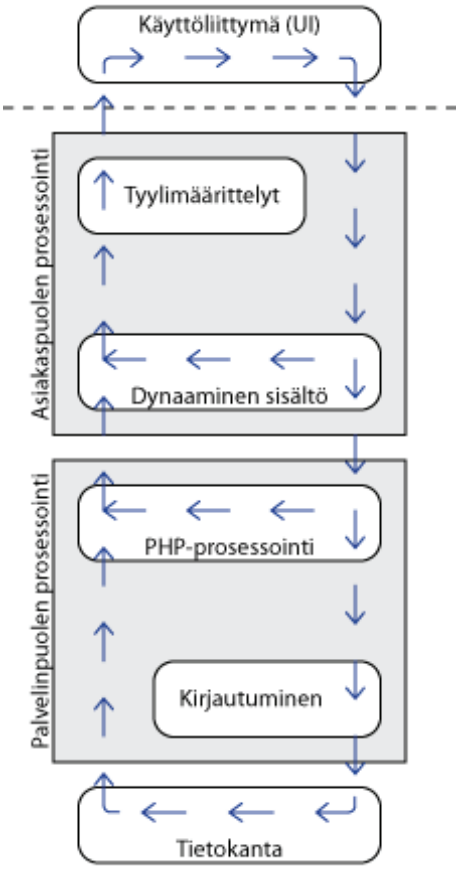
4.5. Mikä on verkkosovellus?

Ennen kuin voi puhua sovelluskehitysmetodeista, pitää määritellä verkkosovellus sellaisessa muodossa, missä se esiintyy tässä projektissa. Internet yhdistää kahdentyyppisiä tietokoneita toisiinsa: **palvelimia** (eng. server), joilla resurssit sijaitsevat ja **asiakkaita** (päätteitä, eng. client), jotka hakevat tietoja ja esittävät ne käyttäjille. Palvelimella tapahtuvaa toimintaa sanotaan **palvelinpuolen prosessoinniksi** (eng. server-side) ja asiakkaalla tapahtuvaa toimintaa **asiakaspuolen prosessoinniksi** (eng. client-side). Jotta asiakas voisi käyttää palvelimella olevia internetresursseja, se tarvitsee **internet selaimen** (eng. browser). Tällä hetkellä on saatavilla useita eri selaimia, joilla on toisistaan poikkeavia toiminnallisuuksia ja jotka perustuvat eri selainytimiin

(eng. layout engine). (Kennedy & Musciano, 2006.) Kuuluisimpia selaimia ovat Mozillan *Firefox*, Microsoftin *Internet Explorer*, Googlen *Chrome*, Opera Softwaren *Opera* ja Applen *Safari*.

Internetresurssit voidaan jakaa karkeasti kahteen ryhmään: verkkosivustoihin, jotka käyttäytyvät sanomalehden tavoin eli ovat paikkoja joissa luetaan ja verkkosovelluksiin, jotka toimivat sovellusten tavoin eli ovat paikkoja joissa käyttäjä on aktiivinen vaikuttaja. Sovelluskehityksessä **verkkosovellukseksi** (eng. web application; webapp) sanotaan internetissä tai intranetissä selaimen avulla käytettävää sovellusta. Verkkosovellusten tärkeimmät edut verrattuna perinteisiin ns. työpöytäsovelluksiin (eng. desktop application) on mahdollisuus päivittää ja ylläpitää sovelluksia asentamatta niitä asiakkaan tietokoneelle. (Wikipedia.) Parhaimmillaan verkkosovellus voi tarjota vaihtoehdon työpöytäsovellukselle, esimerkiksi Horde ja Gmail -sähköpostisovellukset tarjoavat verkkokäyttöisen vaihtoehdon MS Outlook tai Mozilla Thunderbird -työpöytäsovelluksille. Verkkosovelluksen ja verkkosivuston ero on kuitenkin häilyvä, sillä yhä useammalla verkkosivustolla esiintyy sovellusosia eri toimintojen toteutuksissa.

Projektin termipankki luokitellaan verkkosovellukseksi. Teknisenä toteutuksena se on melko monimutkainen kokonaisuus ja käsittää sovelluksen rungon ja erilaiset ohjelmamoduulit sen ympärille. Sovelluksen toteutuksessa on käytetty useampaa ohjelmointi- ja määrittelykieltä eri toiminnallisuuksien aikaansaattamiseksi. Tavallinen käyttäjä näkee kaikesta tästä vain pienen osan eli käyttöliittymän ja todellisuudessa yksinkertaiseltakin näyttävän toiminnon taustalla voi olla monimutkainen funktiorypäs huolehtimassa sen toiminnasta.



Kuva 4-5. Teknisen toteutuksen toimintamalli

Projektin suunnitteluvaiheen alussa oli luotu järjestelmän teknisen toteutuksen toimintamalli (Kuva 4-5), jossa kuvataan yksittäisen asiakaspyynnön (esim. kirjautuminen järjestelmään tai haku tietokannasta) kiertoa järjestelmässä. Kaavion tarkoitus oli selventää järjestelmän teknisten osien tarkoitusta. Kaavion mukaisesti asiakaspyynnöt pyritään ohjaamaan mahdollisimman pientä kiertoa käyttäen, jolloin monet toiminnoista voidaan toteuttaa kokonaan asiakaspuoleisesti ottamatta yhteyttä palvelimeen. Toteutuksen mukainen kuormituksen jakaminen

asiakkaille vähentää mm. palvelimen rasitusta.

Järjestelmä koostuu neljästä itsenäisestä tasosta:

- relaatiotietokannasta
- palvelinpuolen prosessoinnista
- asiakaspuolen prosessoinnista
- käyttöliittymästä

4.5.1. Tietokantataso ja sen rakenne

Relaatiotietokantaa tarvitaan tietojen säilyttämistä varten (tarkemmin: *4.4. Relaatiotietokanta*). Relaatiotietokannan toteutukseen ja sen toimimiseen vaaditaan relaatiotietokantaohjelma (eng. relational database program). Tällaista ohjelmaa kutsutaan myös relaatiotietokannan hallintajärjestelmäksi (eng. relational database managment system). Koska termipankilla oli jo olemassa valmis tietokanta, projektin relaatiotietokannan hallintajärjestelmänä käytetään PostgreSQL -hallintajärjestelmää. Vaikka erilaiset tietokantajärjestelmät (Oracle, MS SQL-server, MySQL, PostgreSQL jne.) ovat pohjimmiltaan samanlaisia (Lahtonen ym., 2003) niissä on joitakin sovelluskohtaisia eroja (esimerkiksi lainausmerkkien käyttö tai sovelluskutsut muista sovelluksista). Myös eri tietokantajärjestelmien tuki SQL-standardin mukaisille käskyille ja toiminnoille poikkeaa toisistaan – toiset tukevat niitä vain osittain ja toiset laajentavat toimintoja omilla käskyillä. Edellä mainituista seikoista johtuen, yhdelle tietokantajärjestelmälle suunniteltu sovellus ei välttämättä toimi suoraan toisessa järjestelmässä (Listaus 4-4).

```
// PostgreSQL sovelluskutsu
$dbconn = pg_connect("dbname=tietokanta user=kayttaja password=salasana");

// mySQL sovelluskutsu
$dbconn = mysql_connect('localhost', 'kayttaja', 'salasana');
$db_selected = mysql_select_db('tietokanta', $dbconn);
```

Listaus 4-4. PHP: Käskyjen eroavaisuudet eri tietokantojen kanssa (esimerkissä PostgreSQL ja mySQL)

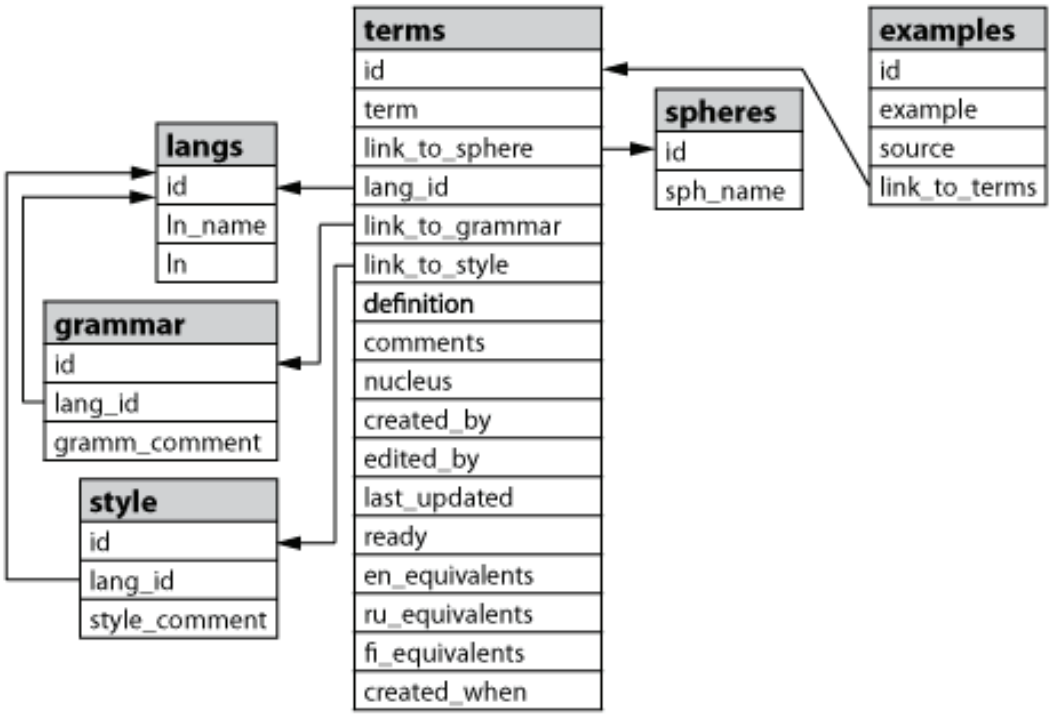
Termipankkisovellus pyrittiin toteuttamaan mahdollisuuksien mukaan tietokantariippumattomaksi, jotta se olisi mahdollista siirtää tarvittaessa myös toiseen palvelinympäristöön. Tällainen kehitys mahdollistui PHP-alustan PEAR::MDB2-laajennuksella ja SQL-standardin mukaisilla lausekkeilla. Huonona puolena MDB2-laajennuksen käytössä on se, että se hidastaa sovelluksen tietokantatyöskentelyä. Rasitustesteissä ei kuitenkaan havaittu merkittävää vaikutusta työskentelyyn, mutta samanaikaisten käyttäjien määrän ja tietokannan koon kasvaessa, tästä voi tulevaisuudessa muodostua ongelma.

Termipankin tietokanta kuvataan erillään toteutuksesta, koska vaikka se on kiinteä osa järjestelmää ja siten vaikuttaa vahvasti sovelluksen toteutukseen, se tulisi nähdä ennen kaikkea yhtenä tiedon säilytyspaikkaratkaisuista. Termipankkisovelluksen taustalla pyörii kaksi tietokantaa, jotka ovat välttämättömiä sovelluksen toiminnan kannalta. Toinen tietokanta koostuu käyttäjätiedoista (nimet, käyttäjätunnukset, salasanat, pääsytasot jne.), eikä se poikkea normaalista käyttäjätietokannasta, joten sitä ei sen tarkemmin käydä tässä työssä läpi. Toteutetun sovelluksen kannalta käyttäjätietokannan ainoat pakolliset tiedot ovat: käyttäjätunnus ja salasana kirjautumista varten ja käyttäjäkohtaiset pääsytasot ohjelman eri toimintojen käyttämistä varten oman pääsytason rajoissa. Mikäli pääsytasoa ei ole määritelty, käyttäjä saa automaattisesti saman pääsytason kun ei kirjautunut käyttäjä (tarkemmin: *5.3. Auktorisointi*).

Termipankkisovellus on mahdollista toteuttaa täysin ilman käyttäjätietotietokantaa, mutta silloin erilliset pääsytasot, jotka sallivat tietyt toiminnot vain erikseen määritellyille ryhmille ei ole mahdollista toteuttaa tietoturvallisesti. Nykyisessä ratkaisussa jokainen käyttäjä vastaa omalla käyttäjätilillä tapahtuvasta toiminnasta ja kaikki käyttäjien toimet on mahdollista yhdistää tiettyyn käyttäjätiliin, jolla kyseinen toiminta on tapahtunut.

Toinen tietokanta taas sisältää kaikki syötetyt artikkelit ja niihin liittyvät tiedot. Vaikka tietokannan rakennetta on mahdollista teknisesti muuttaa sen luonnin jälkeen, tietokannan muokkaaminen jälkeenpäin on ongelmallista ja aikaa vievää. Tietokannan kentät ovat sidoksissa toisiinsa, joten muutosten yhteydessä viittaussuhteiden muuttaminen vaatii tarkkuutta ja erillistä testausta. Suurin ongelma tietokannan muutoksissa tulee kuitenkin sen sisällöstä, koska tiedot joudutaan muutosten yhteydessä päivittämään vastaamaan uutta rakennetta (uuden kentän lisääminen vaatii myös kenttää vastaavan tiedon lisäämisen, kentän poistaminen vaatii kentässä olevan tiedon mahdollista siirtoa muualle jne.). Edellä mainituista syistä tietokannan rakenne pitää olla hyvin selvillä ennen kuin käyttäjät alkavat syöttämään tietoja tietokantaan (esim. keräämään aineistoa termipankkiin) ja sen vuoksi etukäteistutkimus ja tietokannan rakenteen toimivuuden testaus kehitysvaiheessa ovat hyvin kriittisiä vaiheita toimivan termipankin luomisessa.

Mikhail Mikhailovin luoma alustava versio tietokannasta oli jo koekäytössä projektin alkuvaiheessa ja tarkoituksena oli määritellä sen lopullinen rakenne projektin mahdollisimman aikaisessa vaiheessa. Mikhailovin luoma tietokanta osoittautui erittäin hyvin suunnitelluksi ja toimivaksi kokonaisuudeksi, joten pieniä muutoksia lukuun ottamatta se jäi miltei sellaisenaan käyttöön. Itse vaikutin tietokannan rakenteeseen lähinnä suosituksin ja kehotuksin, jolloin sen teknisestä ylläpidosta vastasi täysin Mikhail Mikhailov. Suurimmat muutokset kohdistuivat termipankin tietokannan kenttien nimeämiskäytäntöihin sovelluksen toimintalogiikan näkökulmasta.



Kuva 4-6. Termipankin tietokannan rakenne

Termipankkitietokanta rakentuu kuudesta erillisestä taulusta, jotka ovat sidoksissa toisiinsa (Kuva 4-6). Jokainen taulu sisältää useita kenttiä, joihin varsinaiset tiedot tallennetaan. Tietokannan jokainen osa pyrittiin nimeämään semanttisesti ja läpinäkyvästi, myöhempää jatkokehitystä ajatellen. Nimeämiskäytäntöihin vaikutti myös tarve kenttien myöhempään lisäykseen, minkä vuoksi jotkut kentät nimettiin tietyn logiikan mukaisesti, jotta suuremmilta sovellusmuutoksilta vältyttäisiin ennakoiduissa tapauksissa. Tällaisia kenttiä ovat mm. vastine- ja synonyymikentät, joiden nimeäminen perustuu langs-taulun ln-kentän sisällön ja ”_equivalents”-tekstin yhdistämiseen (esim. en_equivalents). Tällä tavoin sovellus voi dynaamisesti rakentaa SQL-lauseet ilman, että kaikki kentät olisivat määriteltyinä lähdekoodissa.

Selvyyden vuoksi termipankin tietokannan kaikissa tauluissa perusavaimena toimii id-kenttä (tarkemmin: *4.4.1. Relaatiotietokannan rakenne*). Tässä tietokannassa perusavain on kasvava kokonaisluku, joka luodaan automaattisesti uuden tietueen luonnin yhteydessä.

Termipankin päätaulu on terms, sillä se sisältää varsinaiset termiartikkelit. Muut taulut sisältävät termiartikkelien kannalta tarpeellista ryhmittävää tietoa, joista ainoana poikkeuksena on examples-taulu, jonka erityispiirteet kuvataan seuraavassa kappaleessa. Ryhmittävää tietoa tarvitaan sen vuoksi, koska ryhmittäväksi tiedoksi luokitellun tiedon määritys erikseen jokaiselle artikkelille olisi epälooginen ja vaikeasti hallittava ratkaisu. Myös jatkokehityksen kannalta muutokset ryhmittävässä tiedossa vaikuttavat automaattisesti kaikkiin artikkeleihin, joita muutokset koskevat ja siten mahdollistavat muutosten joustavan tekemisen koskematta varsinaiseen artikkelien sisältöön.

Examples {*id, example, source, link_to_terms*} -taulu sisältää termeihin liittyvät esimerkit. Taulu koostuu neljästä kentästä, joita ovat:

- tietueen perusavain *id*
- esimerkin sisältävä *example*
- esimerkin mahdollisen lähteen sisältävä *source*
- viittausavain *link_to_terms* (viittaa terms-taulun termitietueen perusavaimeen *id*)

Esimerkkien vieminen omaan tauluun mahdollistaa monen erillisen esimerkin tallentamisen jokaiselle termille ilman jatkuvia tietokannan rakenteellisia muutoksia. Samasta syystä taulun viittaussuhde on hieman erikoinen ja poikkeaa muista: viittaus suoritetaan päätauluun päin ja varsinaisessa artikkelitaulussa (terms-taulu) ei ole minkäänlaisia viittauksia esimerkkeihin. Tällainen lähestymistapa on kuitenkin ongelmallinen viite-eheyden kannalta, koska se joudutaan tarkastamaan ohjelmallisilla keinoilla. Myös esimerkkien lisääminen artikkeliin tapahtuu erillisellä haulla, koska artikkelitaulussa ei ole niihin viittauksia.

Spheres {*id, sph_name*} taulu sisältää tiedon eri artikkelien luokituksesta eli minkä alan sanastoon ne kuuluvat. Luokituksen avulla on mahdollista sisältää loogisesti termipankkiin eri asioita tarkoittavia samannimisiä termejä (vertaa: työurakan määritelmä juridisessa ja rakennusalan termistössä). Spheres-taulu koostuu kahdesta kentästä, joita ovat:

- tietueen perusavain *id*
- luokituksen nimitys *sph_name*

Kielikohtaiseen luokkien ryhmittelyyn ei haluttu lähteä, koska kielikohtainen jako luokkiin olisi vaatinut erikoisratkaisuja lokalisoinnin suhteen. Tulevaisuudessa kielivalikoiman kasvaessa tällainen lähestymistapa tulee olemaan erittäin ongelmallinen (ts. millä kielellä/kielillä tulisi tallentaa luokkien nimet?). Yhtenä toimivana ratkaisuna olisi artikkelien sijoittaminen erillisiin luokanmukaisiin taulukkoihin, jolloin luokanmukaisten taulukoiden rakenne olisi identtinen. Tällaisella ratkaisulla voisi antaa jokaiselle taulukolle oma lokalisoitu nimi ja samalla voisi jo tietokannan tasolla estää samannimisten artikkelien tallentamisen, mutta se vaatisi melko oleellisia muutoksia itse sovelluksen toimintaan.

Langs {*id, ln_name, ln*} -taulu sisältää käytettävissä olevien kielten nimet ja niiden ISO-standardin mukaisen lyhenteen (esim. ru, fi). Lang-taulu koostuu kolmesta kentästä, joita ovat:

- tietueen perusavain *id*
- halutunkielinen kielen nimitys *ln_name*

- kielen ISO-standardin mukainen lyhenne *ln*

Termipankin toiminnan kannalta erityisen tärkeä on kielen ISO-standardin mukainen lyhenne, koska sen avulla muussa sovelluksessa on mahdollista ennakoida myös tulevien kielten lisäyksen kautta syntyvät tarpeet. Lyhenne on myös se osa mikä on havaittu toimivammaksi sovelluksen käytön kannalta, sillä tarvittaessa näin kielen nimi tulee lokalisointitiedostosta.

Grammar {*id, lang_id, gramm_comment*} -taulu sisältää mahdollisia kielioppimerkintöjä artikkelin termistä. Grammar-taulu koostuu kolmesta kentästä, joita ovat:

- perusavain *id*
- halutunkielinen kielioppimerkintä *gramm_comment*
- viittausavain *lang_id* (viittaa langs-taulun kielitietueen perusavaimeen *id*)

Kielioppimerkinnät päätettiin erottaa kielikohtaisiksi, koska eri kielialueilla on toisistaan poikkeavat kielioppisäännöt ja sitä kautta vieraalla kielellä luodut kielioppimerkinnät johtaisivat väärinkäsityksiin ja joissakin tapauksissa yksikielinen menettely ei edes olisi mahdollinen. Ratkaisu ei ole vedenpitävä, sillä tässä ratkaisussa erikielisiä artikkeleja ei voida yhdistää kielioppimerkintöjen pohjalla toisiinsa. Esimerkiksi useimmat sanakirjat ovat päätyneet käyttämään palvelun ensisijaiseksi suunnitellun kielen kielioppimerkintöjä. Yhtenä vaihtoehtona olisi lokalisointiratkaisu, jossa käytetyt kielioppimerkinnät tulevat sovelluksen käyttökielen mukaan, mutta se vaatisi muutoksia tietokantaan ja erillistä sovellusmoduulia toiminnon toteuttamiseksi.

Style {*id, lang_id, style_comment*} -taulu sisältää mahdollisia tyylimerkintöjä artikkelin termistä. Style-taulu koostuu kolmesta kentästä, joita ovat:

- perusavain *id*
- halutunkielinen tyylimerkintä *style_comment*. Tyylimerkintöjä voivat olla esimerkiksi vanhentunut termi, alatyylinen ilmaisu, arkityylinen ilmaisu jne.
- viittausavain *lang_id* (viittaa langs-taulun kielitietueen perusavaimeen *id*)

Tyylimerkinnät päätettiin erottaa kielikohtaisiksi lähinnä artikkelien kieliasun säilyttämiseksi. Ratkaisu ei kuitenkaan ole vedenpitävä samoista syistä kuin grammar-taulussakin ja ehdotettu ongelmanratkaisu olisi vastaava.

Terms {*id, term, link_to_sphere, lang_id, link_to_grammar, link_to_style, definition, comments, nucleus, created_by, edited_by, last_updated, ready, en_equivalents, ru_equivalents, fi_equivalents, created_when*} -taulu on sisältää varsinaiset termiartikkelit. Terms-taulu koostuu 17 kentästä, joita ovat:

- perusavain *id*
- termi *term*
- termin määritelmä *definition*
- kommentit *comments*. Kommenttikenttään on tarkoitus tallentaa kaikki mahdolliset artikkelia koskevat huomautukset eli tälle kentälle ei ole olemassa tarkkaa sisältörajausta muista kentistä poiketen. Mahdollisia kommentteja voivat olla huomautukset termin vastinesuhteista, artikkelin tekijän omat näkemykset jne.
- termin nukleus *nucleus*. Nukleuksella tarkoitetaan sanan ydintä ja tässä tapauksessa termin ydintä. Nukleuksen avulla on tarkoitus ryhmitellä termiartikkeleja pienempiin joukkoihin niiden kantasanan perusteella. Esimerkiksi erilaiset lakat on mahdollista tarkastella omana ryhmänä nukleuksen avulla, sillä nukleus ”lakka” esiintyy mm. seuraavissa termeissä: *alkydilakka, epoksireaktiolakka, katalyyttilakka, lakka* jne.
- viittausavaimet termin erikielisiin vastineisiin, joita ovat tällä hetkellä *en_equivalents*, *ru_equivalents* ja *fi_equivalents*. Vastinekenttien nimityksessä ei oteta huomioon artikkelin kieleen ja sitä kautta siihen, viittaavatko näihin kenttiin tallennetut avaimet synonyymeihin vai vieraskielisiin vastineisiin. Ero synonyymeihin ja vastineisiin on loogisempaa ja vaivattomampaa toteuttaa muilla tavoin sovelluksen toiminnassa.
- viittausavaimet *lang_id* (viittaa langs-taulun kielitietueen perusavaimeen *id*), *link_to_sphere* (viittaa sphere-taulun luokitustietueen perusavaimeen *id*), *link_to_grammar* (viittaa grammar-taulun kielioppitietueen perusavaimeen *id*) ja *link_to_style* (viittaa style-taulun tyylitietueen perusavaimeen *id*)
- muokkaustiedot artikkelin luoja *created_by*, luontipäiväys *created_when*, viimeisin muokkaaja *edited_by*, viimeisin muokkauspäiväys *last_updated*
- artikkelin valmiusaste *ready*. Koska termipankin toiminta perustuu käyttäjien yhteistyöhön, artikkeleja on mahdollista muokata. Näin voidaan varmistaa tehokas virheiden korjaus ja artikkelien kehittyminen. Tästä kuitenkin seuraa se, että termipankki sisältää eri valmiusasteen omaavia artikkeleja. Termipankin sisällön luotettavuuden säilyttämiseksi artikkelien työversiot on erotettava valmiista artikkeleista. Artikkelien jako työversioihin on monitasoinen (0–3), jolloin lähdekritiikki jää osittain käyttäjien vastuulle. Jakoasteikkoa voidaan myös joustavasti laajentaa tarpeen mukaan, jolloin termipankissa olevat artikkelit pitää siirtää uudelle (korkeammalle) tasolle loogisesti uudelleenarvioinnin kautta. Luotettavuuden säilyttämiseksi korkeinta valmiusastetta ei tulisi asettaa artikkeleille liian kevyin perustein ilman monivaiheista useamman käyttäjän suorittamaa arviointia (tarkemmin: *3. TERMINOLOGINEN TYÖ*).

Artikkelien syöttövaiheessa kaikkien kenttien täyttäminen ei ole pakollista, vaan käyttäjät voivat valita itselleen tarpeelliset kentät. Kenttiä voi myös myöhemmässä vaiheessa päivittää ja lisätä niihin tietoa, mikä tukee termipankin jatkuvan kasvun periaatetta.

4.5.2. Palvelinpuolen prosessointi

Palvelinpuolen prosessointi toteutettiin kokonaisuudessaan PHP-ohjelmointikielellä. Palvelinpuolen prosessoinnilla tarkoitetaan niiden sivun dynaamisten elementtien tarkastusta, käsittelyä ja luontia, jotka on välttämätöntä toteuttaa palvelinpäässä. Tällaisia elementtejä ovat mm. tietokannan manipulaatio, kuten tietojen tietokantaan tallentaminen ja tietokannasta haku. Vaikka kaaviossa (Kuva 4-5) kirjautuminen eli auktorisointi on kuvattu erikseen, todellisuudessa sekin on toteutettu PHP:llä. Kirjautuminen erotettiin kaavioon omaksi osa-alueeksi sen vuoksi, koska haluttiin tarkentaa tapaa, jolla on mahdollista työskennellä tietokantajärjestelmän kanssa eli joka kerta kun tietokantaan otetaan yhteys, asiakkaan pääsytaso tarkastetaan poikkeuksetta auktorisoinnin kautta (tarkemmin: 5.*3. Auktorisointi*) tietoturvan varmistamiseksi.

4.5.3. Asiakaspuolen prosessointi

Sivujen dynaamiset elementit kuten dynaaminen hakulomake ja tietokannasta haettujen artikkelien esitys on mahdollista toteuttaa palvelinpuolen tai asiakaspuolen prosessoinnilla. Sovelluksen suunnitellun arkkitehtuurin mukaisesti tämä päätettiin toteuttaa mahdollisuuksien mukaan asiakaspuoleisena toimintona. Näin voidaan vähentää palvelinkuormaa, koska jokainen käyttäjä toteuttaa nämä muutokset HTML-sivussa täysin omalla työpäätteellään ottamatta yhteyttä palvelimeen.

Tavallisesti tällaiset elementit toteutetaan JavaScript-komentosarjakielellä, koska se on laajasti tuettu uusimmissa selaimissa. JavaScript mahdollistaa hyvin laajan toiminnan ja sen vuoksi sillä on mahdollista myös suorittaa haitallista koodia asiakaspäässä, mistä johtuen joillakin käyttäjillä se on kytketty pois toiminnasta. Vaikka JavaScriptillä on tällainen negatiivinen puoli, sen käyttö on usein välttämättömyys monipuolisten dynaamisten toimintojen toteutuksessa, jolloin poiskytkettynä nämä toiminnot lakkaavat toimimasta ja pahimmillaan koko sovellus muuttuu käyttökelvottomaksi. Termipankin toteutuksessa käytetään jQuery JavaScript-kirjastoa, jonka tarkoituksena on yksinkertaistaa JavaScript-elementtien luontia, mikä vaikuttaa merkittävästi tuotettuun ohjelmakoodiin ja sen luettavuuteen. Esimerkeissä (Listaus 4-5 ja Listaus 4-6) toteutetaan hyvin yksinkertainen ja verkkosovelluskehityksessä tavallinen klikkaustapahtuma jokaiselle sivulla esiintyvälle linkille (Skinner, 2007).

```
var external_links = document.getElementById('external_links');
var links = external_links.getElementsByTagName('a');
for (var i=0;i < links.length;i++) {
    var link = links.item(i);
    link.onclick = function() {
        return confirm('Olet siirtymässä sivulle: ' + this.href);
    };
}
```

Listaus 4-5. Toteutus puhtaalla JavaScriptillä ja DOM-komentosarjalla

```
$('#external_links a').click(function() {
    return confirm('Olet siirtymässä sivulle: ' + this.href);
});
```

Listaus 4-6. Toteutus jQuerylla

Projektina jQuery on melko tuore, joten se herättää kysymyksen projektin kehitystyön jatkuvuudesta. jQuery-kirjasto on kuitenkin käytössä mm. sellaisilla suurilla internet-vaikuttajilla kuin Google, IBM, Intel, Oracle jne. (http://docs.jquery.com/Sites_Using_jQuery) ja sille on olemassa kattava dokumentaatio, joten uskon, että jQuery-kirjasto tulee tyydyttämään myös jatkokehityksen tarpeet.

Verkkosovelluskehityksessä semanttisuus on tärkeä peruskäsite ja se tulisi ymmärtää laajempana kokonaisuutena kuin pelkästään sen alkuperäisessä lingvistisessä merkityksessä. Semanttisen merkintätavan (eng. semantic markup) mukaisesti ulkoasu ja sisältö on pidettävä erillään (tarkemmin: 4.*8.1. Semanttinen merkintätapa*) ja sen vuoksi kaikki ulkoasumääritykset on toteutettu CSS-tyylimäärittelyillä CSS2.1 spesifikaation mukaisesti (W3C: CSS2.1 Specification, 2009). Erillisillä tyylimäärittelyillä ja semanttisesti rakennetuilla HTML/XHTML-sivuilla on mahdollista muuttaa monipuolisesti ja kustannustehokkaasti käyttöliittymän ulkoasua, mikä osoitetaan mm. http://www.csszengarden.com/ -sivustolla.

Käyttöliittymä on teknisen toteutuksen toimintamallin (Kuva 4-5) mukaisen tapahtumaketjun tuote eli asiakas näkee XHTML-sivuna omassa selainikkunassa ne elementit, jotka on ensin tuotettu palvelinpäässä mahdollisesti tietokannan tietojen perusteella ja sitten muokattu ja muotoiltu tietyn näköisiksi ja mallisiksi asiakaspäässä. Näkemänsä XHTML-sivun asiakas on kuitenkin itse aktiivisesti tuottanut omalla toiminnallaan. Dokumenttien XHTML-merkintä on toteutettu XHTML1.0 spesifikaation mukaisesti (W3C: XHTML 1.0 The Extensible HyperText Markup Language, 2002), mikä on laajennus HTML 4.0 spesifikaatiolle.

4.5.4. Toteutuksen rajoitteet

Verkkosovellusten käyttö tapahtuu selaimen kautta, joten työkalujen ja ohjelmointikielten valinnassa tulisi aina ottaa huomioon mm. käyttäjien käyttämät internetselaimet ja niissä oleva sisäänrakennettu tuki sovelluksen toteutusmenetelmille. Vaikka toteutus tapahtuisi täysin spesifikaatioiden mukaisesti, toteutuksen elementit eivät välttämättä näy suunnitellulla tavalla, joko jonkun tietyn selaimen puutteellisen tuen tai spesifikaatiovastaisen selaimen toiminnan vuoksi. Ongelma kasvaa entisestään vanhempien selainversioiden kohdalla, sillä niissä ei ole otettu huomioon nykyisiä toteutustekniikoita.

Verkkosovellusten optimointi mahdollisimman monelle selaimelle (eng. cross-browser optimization) on kuitenkin hyvin paljon resursseja vievä toimenpide, siinä usein joudutaan toimimaan spesifikaatioiden vastaisesti eikä se läheskään aina ole tarkoituksenmukaista. Tarkoituksenmukaisuudella tässä tarkoitetaan tilannetta, jossa voidaan selkeästi rajata käyttäjien suosimat selaimet, sillä tällöin ei ole mielekästä kuluttaa resursseja optimointiin sellaisille selaimille joita ei käytetä.

Termipankki on pyritty tekemään toimivaksi mahdollisimman monella selaimella, mutta sen optimointi rajoitettiin vain Mozilla Firefox 3 ja Microsoft Internet Explorer 7 -selaimille. Tällainen periaatteellinen rajaus tehtiin eri selainten markkina-asemien perusteella (LIITE 3: Selainten markkina-asemat Suomessa ja maailmalla).

4.6. Projektin iteratiivinen toteutustapa

Haikalan mukaan perinteinen sovelluskehitysprojekti voidaan jakaa karkeasti viiteen vaiheeseen (Huttunen, 2006):

1. Määrittely
2. Suunnittelu
3. Ohjelmointi (toteutus)
4. Testaus
5. Käyttöönotto ja ylläpito

Tunnetuin ja käytetyin perinteisen sovelluskehitysprojektin vaiheita noudattava malli on vesiputousmalli. ”Vesiputous” kuvaa hyvin mallin perusajatusta: jokainen vaihe seuraa tiukasti edellistä ja koska vesi ei virtaa koskaan ylöspäin, edelliseen vaiheeseen ei periaatteessa voi palata. (Huttunen, 2006.) Vesiputousmallin mukaan kehitettävä sovellus nojaa tarkasti laadittuihin sopimuksiin ja kaikenkattavaan dokumentaatioon, sillä jokainen vaihe toteutetaan

kokonaisuudessaan kerralla ja palaaminen edeltäviin vaiheisiin on epäsuotavaa. Vesiputousmallin mukaisesti käyttöönottovaiheessa asiakkaalle toimitetaan täysin valmis tuote.

Modulaarisen sovelluksen kehityksessä perinteinen sovelluskehitysmalli on kuitenkin hyvin ongelmallinen, sillä moduulit toteutetaan itsenäisinä osina ja näin ollen ns. valmiiseen tuotteeseen on mahdollista lisätä toiminnallisuuksia myös jälkeenpäin. Tällaisessa toteutuksessa sovellus ei siis ole koskaan valmis perinteisen näkemyksen mukaisesti. Poikkeuksia lukuun ottamatta sovelluksia ylläpidetään, modernisoidaan ja niistä korjataan mm. ohjelmointivirheitä, joten voiko sovellusta ylipäätänsä pitää koskaan valmiina (Goodliffe, 2007)? Toteutettavan termipankkisovelluksen rakenne päätettiin pitää mahdollisimman modulaarisena jo senkin takia, koska tämän projektin puitteissa ei ollut tarkoitusta toteuttaa kaikkia suunniteltuja toimintoja resurssien puutteen vuoksi.

Ketteristä sovelluskehitysmenetelmistä (eng. agile methods) peräisin oleva iteratiivinen toteutustapa vastaa juuri näihin haasteisiin ja sen vuoksi projekti toteutettiin iteraatioiden kautta. Iteratiivisessa toteutuksessa sovelluskehityksen vaiheet menevät osittain päällekkäin ja kaikki vaiheet toistuvat erisisältöisinä useamman kerran projektin aikana. Kukin **iteraatio** on kuin pieni ohjelmistoprojekti, joka sisältää kaikki uusien toimintojen julkaisemiseksi tarvittavat tehtävät.

Iteratiivisella toteutustavalla on useita etuja verrattuna perinteiseen vesiputousmalliseen sovelluskehitykseen (Kroll, 2004):

- Reagointi muutoksiin on nopeaa, koska sovellukselle asetetut vaatimukset arvioidaan jokaisessa iteraatiossa uudelleen.
- Yksittäisten toiminnallisten osien määrittely ja suunnittelu on helpompaa pienemmissä kokonaisuuksissa kuin mikäli sovellus suunniteltaisiin kerralla.
- Pienemmissä kokonaisuuksissa on helpompi tunnistaa tarpeen uusille ohjelmakokonaisuuksille, joita ei välttämättä huomata projektin alkuvaiheessa.
- Iteraatioissa sovelluksen osien integrointi tapahtuu vaiheittain eikä yhdellä kerralla (eng. big bang), jolloin osien puutteet ja sopimattomuus muuhun järjestelmään nähden on mahdollista paikallistaa ja korjata mahdollisimman aikaisessa vaiheessa, mikä parantaa sovelluksen laatua ja vähentää tarvittavien resurssien määrää.
- Vaiheittainen kehitys mahdollistaa paremman kommunikoinnin asiakkaan ja muiden projektiryhmään osallistuvien kanssa, jolloin tarvittavat muutokset saadaan järjestelmään välittömästi palautteen kautta.

Iteraatio ei kuitenkaan tarkoita kehitysprojektin jakamista mahdollisimman pieniin osiin, sillä esimerkiksi testaukseen kulutettu aika ei ole suoraan verrannollinen osan kokoon ja liian pienten

osien testaus voi olla myös täysin tarkoituksenvastainen. Toisaalta liian isojen osien hallinta on hankalaa, vaikka tällainen osa sisältäisi vain yhden moduulin. Iteraatio rajoitetaankin ketterässä sovelluskehityksessä ajallisesti ja yksi iteraatio kestää yleensä n. 1–6 viikkoa. Tämän projektin kannalta mielekkääksi yksittäisen iteraation pituudeksi koettiin n. kahden viikon ajanjakso, jota myös pyrittiin noudattamaan.

Iteratiivisen toteutustavan vuoksi termipankin sovelluksen toteutusta ei myöskään ole mahdollista kuvata kronologisessa järjestyksessä, minkä vuoksi kuvaus pyritään jakamaan järjestelmän toiminnallisiin osiin. Kokonaisuuden kannalta on kuitenkin hyvin tärkeää tiedostaa, että kaikkien sovelluksen osien toteutus sisälsi erisisältöisinä kaikki sovelluskehityksen vaiheet määrittelystä käyttöönottoon.

4.7. Puolustautuva ohjelmointi

Ohjelmistoja kehittäessä muodostuu helposti joukko oletuksia siitä, miten ohjelman tulisi toimia ja kuinka sitä tulisi käyttää. Usein tällaiset oletukset eivät kuitenkaan toteudu sellaisinaan käyttäjien toimesta, mikä voi johtaa ohjelman toimintahäiriöihin, kaataa koko ohjelman tai luoda tietoturva-aukkoja suunnitellun toiminnan vastaisella käytöllä. Tavallisimpia oletuksia, joita sovelluskehittäjät voivat tehdä (Goodliffe, 2007):

- Tätä toimintoa ei koskaan tulla käyttämään tällä tavoin.
- Tämä toiminto toimii vuorenvarmasti, eikä koskaan tuota virheitä.
- Tätä toimintoa ei käytetä, jos mainitsen, ettei se ole tuettu.

Puolustautuvan ohjelmoinnin (eng. defensive programming) perusajatuksena on välttää kaikennäköisten oletusten tekemistä, sillä vaikka juuri tällä hetkellä oletukset pitäisivätkin paikkansa, ne voivat ohjelman kasvaessa olla täysin vääriä. Myös jatkokehityksen kannalta tehdyt oletukset ovat ongelmallisia, koska muut ohjelmoijat eivät välttämättä tiedä, mitä oletuksia aikaisemmin on tehty. Puolustautuvan ohjelmoinnin periaatteiden mukaisesti ohjelman yksittäisistä osista pyritään tekemään mahdollisimman itseään suojelevia eli niissä on tarvittava virheentarkastus ja -raportointi. Näin on mahdollista välttää ohjelman suunnitellunvastainen käytös myös sellaisissa tilanteissa, joissa ei ole otettu tiettyä käyttäytymistä huomioon. (Goodliffe, 2007.) Käytännön tasolla se tarkoittaa sitä, että kaikki mahdollinen vuorovaikutus sovelluksen ja käyttäjän välillä tarkastetaan virheiden varalta ja sovellus osaa raportoida myös hyvin epätodennäköiset virheet.

Herää kuitenkin kysymys, kuinka on mahdollista tarkastaa kaikki mahdolliset virheet, jos ei ole mahdollista tehdä oletuksia etukäteen? Vastaus tähän on yksinkertainen eli ei voikaan – ainakaan perinteisessä mielessä. Virheentarkistus tulisikin nähdä enemmän sallittujen toimintojen sallimisena

(eng. white list) mahdollisuuksien mukaan. Mikäli pyritään ainoastaan estämään jotakin käytöstä (eng. black list), vaarana on, ettei kaikkia toimintamalleja oteta huomioon. Sallittuina toimintoina tarkoitetaan esimerkiksi, jos käyttäjän syöttämän tiedon pitää olla päivämäärä (PP.KK.VVV), muita tai muussa muodossa kirjoitettuja arvoja ei sallita.

Tietoturvan kannalta virheraportointi on kuitenkin ongelmallista, sillä vaikka se auttaa käyttäjiä ratkaisemaan syntyneet ongelmat, se antaa myös vihjeitä järjestelmän väärinkäyttäjille. Sen vuoksi on hyvin tärkeää arvioida tapauskohtaisesti virheilmoitusten informaatioarvo käyttäjille ja väärinkäyttäjille. Tämän takia monet virheilmoitukset kytkettiin pois päältä julkaisun yhteydessä tai niiden sisältö muutettiin yleisluontoisemmaksi esimerkiksi järjestelmään kirjautumisen epäonnistuminen väärän käyttäjänimen tai salasanan vuoksi ei ilmoita kumpi syötetyistä tiedoista oli väärin, vaan käyttäjää pelkästään kehotetaan syöttämään oikea käyttäjänimi ja salasana.

4.8. Itsedokumentoituva lähdekoodi

Jatkokehityksen kannalta dokumentaatio on hyvin tärkeä osa mitä tahansa sovellusprojektia ja se voidaan toteuttaa monella eri tavalla, mutta yhteistä kaikella sovelluskehittäjille suunnatussa dokumentaatiossa on selventää sovelluksen nykyistä toimintatapaa mahdollisimman selkeästi ja yksityiskohtaisesti. Kuitenkin ainoa dokumentti, joka selventää ohjelman lähdekoodin täydellisesti ja oikealla tavalla on lähdekoodi itse (Goodliffe, 2007). Se ei kuitenkaan vielä tarkoita, että lähdekoodi olisi automaattisesti paras mahdollinen kuvaus, mutta sovelluskehityksessä tulisi pyrkiä tekemään mahdollisimman helppolukuista lähdekoodia. Miten tähän päämäärään voidaan sitten päästä? Tässä projektissa pyrittiin tuottamaan itsedokumentoitavaa lähdekoodia kolmella tavalla:

- jakamalla sovellus loogisiksi osiksi
- antamalla sovelluksen osille kuvaavia nimiä
- kommentoimalla osien toimintaa

Yksinkertaisimmillaan sovelluksen jako loogisiksi osiksi tarkoittaa selvästi samaan ryhmään kuuluvien elementtien erottamista omiksi tiedostoiksi. Esimerkiksi toteutetussa termipankkisovelluksessa `searchFunctions.php` sisältää hakutoiminnot ja `authFunctions.php` auktorisointitoiminnot. Myös tiedoston sisällä jako vielä pienempiin osiin on mahdollinen ja esimerkiksi tässä projektissa selvästi toisiinsa kytköksissä olevat funktiot ja muut elementit pyrittiin pitämään mahdollisimman lähellä toisiaan eli loogisessa järjestyksessä.

Elementtien sisältöä kuvataan antamalla eri osille niiden toimintaa kuvaavia nimiä. Toteutettavan termipankkisovelluksen kannalta erityisen tärkeässä roolissa nimitysten suhteen ovat ennen kaikkea

muuttujat ja funktiot (Listaus 4-7). Tällä tavoin on helpompi lukea lähdekoodia ja tunnistaa sen toimintoja, mikä puolestaan vähentää perinteisen dokumentaation tarvetta.

```
// Muuttaa syötetyn päivämäärän DD.MM.YYYY -> YYYY-MM-DD (RFC 3339/ISO 8601)
// reformatDate([string])
function reformatDate($dateString) {

   $newdate = strftime("%Y-%m-%d", strtotime($dateString));
   return $newdate;

}

$insertedDate = $_GET['date'];
$dateDBformat = reformatDate($insertedDate);
```

Listaus 4-7. PHP: Sovelluksen osien kommentointi ja funktioiden ja muuttujien kuvaavat nimet

Pelkillä kuvaavilla nimillä ei kuitenkaan ole mahdollista kuvata täysin eri osien toimintoja, joten dokumentaatiolta ei voi välttyä. Toimintojen kommentointi voidaan nähdä tässä vaihtoehtona erilliselle dokumentaatiolle. Esimerkiksi Goodliffen mukaan kommentointia tulisi kuitenkin välttää mielellään kokonaan mahdollisimman selkeällä lähdekoodilla (kommentoinnin tarpeen vähentäminen) sen vuoksi, koska kaikki ylimääräiset lisäykset tekevät koodista raskaslukuisempaa. Olen osittain samaa mieltä, mutta silti hyvä kommentointi on usein elintärkeä sovelluksen toiminnan kuvaamisessa, varsinkin silloin kun funktioiden sisällä on useita viittauksia muihin funktioihin. Vaikka projektin toteutuksessa ns. turhaa kommentointia pyrittiin välttämään Goodliffen ohjeen mukaisesti, suurin osa funktioiden toiminnasta on silti kommentoituna (Listaus 4-7). Lähdekoodin logiikka ei välttämättä ole itsestäänselvyys jatkokehitykseen osallistuville ja sen vuoksi kommenttien puuttuminen olisi mahdollisesti aiheuttanut ongelmatilanteita jatkokehityksessä.

4.8.1. Semanttinen merkintätapa

Käyttöliittymän kohdalla itsedokumentoituvalla lähdekoodilla on myös toinen tarkoitus – yhdenmukaisen esitysmuodon varmistaminen selainkohtaisten eroavaisuuksien vuoksi. Semanttisella merkintätavalla (eng. semantic markup) tarkoitetaan www-ohjelmoinnissa ennen kaikkea ulkoasun ja sisällön erottamista toisistaan. Käytännössä tämä tarkoittaa sitä, että lähdekoodi kuvastaa sen sisältämiä elementtejä ja elementtien järjestys on looginen (Listaus 4-8) samantapaisesti kuten lähdekoodinkin kohdalla.

```
<h1>Otsikko</h1>

<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Phasellus libero
felis, iaculis vitae, elementum id, laoreet sit amet, elit.</p>

<p>Cras iaculis aliquet augue. Quisque lacus velit, tempus at, faucibus eget,
lacinia id, libero.</p>
```

Listaus 4-8. HTML: 1-tason otsikko ja kaksi kappaletta

```
<font size="14px" color="#000"><b>Otsikko</b></font>

<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Phasellus libero
felis, iaculis vitae, elementum id, laoreet sit amet, elit.<br />Cras iaculis
aliquet augue. Quisque lacus velit, tempus at, faucibus eget, lacinia id,
libero.</p>
```

Listaus 4-9. HTML: Epäsemanttinen merkintätapa, otsikon näköinen elementti ja pakotettu rivinvaihto

Ulkoasultaan samaan lopputulokseen voi päästä myös toisella tavalla (Listaus 4-9), miksi siis pitäisi noudattaa semanttista merkintätapaa? Siihen on olemassa useita syitä, joista tärkeimpinä pidän seuraavia:

- Ohjelmoijan näkökulmasta semanttisesti kirjoitettua lähdekoodia on selvästi helpompi lukea ja ymmärtää, koska jokainen sisältöelementti merkitsee jotakin ja rakenne on looginen.
- Päivittäminen on helpompaa, koska elementteihin voidaan viitata esimerkiksi CSS-tyylimäärittelyillä, jolloin muutokset vaikuttavat kaikkiin elementteihin.
- Kone tunnistaa sisällön paremmin ja esimerkiksi näin on mahdollista rakentaa ohjelmamoduuleita, jotka muokkaavat sivuston tekstiä vaikkapa tiedostoon vientiä varten.
- Sisällön elementit ovat oikeannäköiset ja oikeassa järjestyksessä myös sellaisilla laitteilla ja ohjelmilla, joilla ei ole ulkoasumääritysten tukea.

Kaikille mahdollisille elementeille ei kuitenkaan ole olemassa semanttista tagia ja myös olemassa olevat eivät välttämättä kerro elementin varsinaisesta sisällöstä, kuten esimerkiksi otsikko-elementit `<H1>`–`<H6>` kertovat vain, että kyseessä on tason 1-6 otsikko ja `<P>` kertoo vain sen, että kyseessä on tekstikappale. Tähän puutteeseen on olemassa useita ratkaisuja (mm. XML, jossa luodaan sisältöä vastaavia tageja: <author>, <book> jne.), joista äärilaitana ovat semanttisen Webin (eng. Semantic Web) ratkaisut. W3C:n XHTML 1.0 ja CSS2.1 spesifikaatioiden (W3C: XHTML 1.0 The Extensible HyperText Markup Language, 2002 ja CSS2.1 Specification, 2009) mukaisesti on kuitenkin myös mahdollista käyttää `class` ja `id` attribuutteja, joiden tarkoitus on kuvata ja ryhmitellä elementtejä (Listaus 4-10).

```
<h1 id="faq" class="helpItems">Otsikko</h1>

<p class="firstParagraph helpItems">Lorem ipsum dolor sit amet, consectetuer
adipiscing elit. Phasellus libero felis, iaculis vitae, elementum id, laoreet
sit amet, elit.</p>

<p class="secondParagraph helpItems">Cras iaculis aliquet augue. Quisque lacus
velit, tempus at, faucibus eget, lacinia id, libero.</p>
```

Listaus 4-10. ID ja CLASS -attribuutit lisäämässä lähdekoodin semanttisuutta

W3C:n suositusten mukaisesti `class` attribuutti määrittelee elementin kuulumisen yhteen tai useampaan ryhmään ja `id` attribuutti on yksilöllinen elementin nimi, joten se voi esiintyä vain yhdellä elementillä kerrallaan. `id` attribuutilla on myös erikoistehtävä internet-selainten toiminnan kannalta, sillä `id` attribuutin avulla voidaan luoda kirjamerkkejä, jolloin esimerkiksi `http://www.sivusto.fi/sivu.html#faq` -osoitteeseen siirryttäessä selain pyrkii automaattisesti etsimään sivulta elementtiä, jonka id on ”faq” ja vierittämään sivu siihen kohtaan, mikä korostaa yksilöllisten `id` -attribuuttien tärkeyttä.

Sen lisäksi että lähdekoodista tulee semanttisempaa `class` ja `id` attribuutteja käytetään laajasti myös viittaamaan elementteihin ja elementtiryhmiin esimerkiksi CSS-määrittelyissä ja JavaScriptissä, minkä vuoksi ne tarjoavat joustavan tavan ohjelman osien toteuttamiselle eri ohjelmointikielillä.

Räikein esimerkki epäsemanttisesta merkintätavasta on taulukoiden käyttö täysin vastoin alkuperäistä tarkoitusta. W3C määrittelee taulukot käytettäväksi taulukkomuotoisen tiedon (Taulukko 4-1) esittämiseen ja erikseen mainitaan, ettei taulukoita tulisi käyttää puhtaasti ulkoasun määrittelyyn (W3C: HTML 4.01 Specification, 1999). Taulukoiden väärinkäytön ongelma on todellinen, sillä niiden käyttöön ulkoasun muotoiluun opastetaan edelleen osassa arvostetuista teoksista (mm. HTML & XHTML: The Definitive Guide: *”While tables are useful for the general display of tabular data, they also serve an important role in managing document layout.”,* Kennedy & Musciano, 2006) ja taulukoita käytetään laajasti verkkosivustojen kehityksessä. Esimerkiksi kaikki tämän projektin puitteissa tutkitut termipankit (TEPA-termipankki, MultiTerm Online, WebTerm, I-Term jne.) käyttivät taulukoita myös puhtaasti ulkoasun määrittelyssä.

Taulukon otsikko	
Ensimmäisen sarakkeen otsikko	**Toisen sarakkeen otsikko**
Ensimmäisen rivin ensimmäinen solu	Ensimmäisen rivin toinen solu
Toisen rivin ensimmäinen solu	Toisen rivin toinen solu
Taulukon alatunniste	

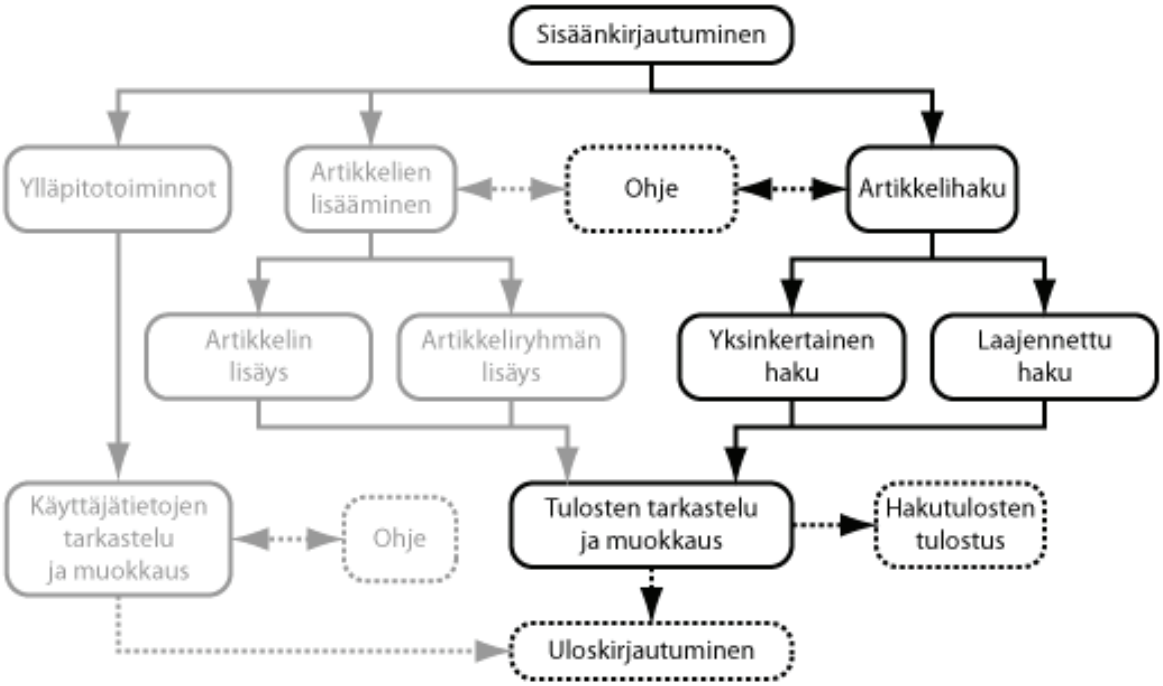
Taulukko 4-1. Taulukkomuotoisen tiedon esitys

Ulkoasumääritykset tulisi kuitenkin hoitaa muilla keinoilla esimerkiksi CSS-tyylimääritysten kautta ja tarvittavat elementtien ryhmittelyt `div` ja `span` tageilla. W3C määritysten mukaan `div` ja `span` elementit yhdessä `id` ja `class` attribuuttien kanssa tarjoavat yleisen mekanismin dokumentin rakenteen luomiseen. `span` määrittelee elementin olevan tekstitason (eng. inline) elementti ja `div` taas lohkotason (eng. block) elementti. Muita esitystavan määrityksiä ei näihin elementteihin sisälly, vaan ne määritellään esimerkiksi CSS-määritysten avulla. (W3C: HTML 4.01 Specification, 1999.) Termipankin käyttöliittymä pyrittiin pitämään mahdollisuuksien mukaan täysin taulukkovapaana ulkoasumääritysten suhteen.

5. TERMIPANKIN TOTEUTUS

Vaikka projekti toteutettiin iteratiivisesti (tarkemmin: *4.6. Projektin iteratiivinen toteutustapa*), se ei tarkoita, ettei projektin yleisluontoista määrittelyä ja suunnittelua toteutettu lainkaan. Todellisuudessa projektin toteutusta edelsi melko laaja määrittelyvaihe, jonka aikana hahmottuivat ne toiminnallisuudet, joita toimiva termipankkisovellus vähimmillään vaatii. Nämä osiot määriteltiin käyttäjäprofiilien ja erilaisten käyttöskenaarioiden avulla. Myöhemmin yksittäisten osien toteutusvaiheessa näitä käyttöskenaarioita laajennettiin vastaamaan toteutettavaan osaan kohdistuvia yksityiskohtaisia tarpeita. Toimintojen määrittelyn yhteydessä tehtiin kaavio (Kuva 5-1), jonka perusteella sovelluksen myöhempi kehitys tapahtui. Kaikki kaaviossa määritellyt toiminnot pitää löytyä verkossa toimivasta termipankista, jotta se olisi toimiva ja käyttäjäystävällinen sovellus. Kaikkia toimintoja ei kuitenkaan toteutettu (kaaviossa toteutumattomat osat väritetty harmaalla värillä) resurssien puutteen vuoksi ja taas toisaalta osa toissijaisista toiminnoista toteutettiin (eivät näy kaaviossa), koska ne todettiin toteutuksen yhteydessä sopivan luontevasti termipankin toimintaan.

Toimintojen määrittely ja sitä kautta kaavio (Kuva 5-1) vastaa kysymykseen: Mitä käyttäjä tekee kun hän avaa hänen tarpeita täyttävän termipankkisovelluksen? Kaaviossa esitetyt asiat eivät siis ole ohjelmaosia, vaan keskittyvät enemmän toimintaan ja toiminnasta aiheutuvien tarpeiden määrittelyyn. Kaavion harmaalla värillä merkityt osat ovat itse sovelluksessa toteutuksen ulkopuolelle jääneet osat ja katkoviivalla on merkitty ne toiminnot, jotka eivät välttämättä toteudu. Toiminnoilla, jotka eivät välttämättä toteudu tarkoitetaan sellaisia kriittisiä toimintoja, joita käyttäjät eivät tarvitse jokaisella käyttökerralla tai eivät syystä taikka toisesta käytä, jolloin toimintojen toteutuminen on otettava huomioon muilla tavoin.



Kuva 5-1. Kaavio sovelluksen toiminnallisista osista (projektissa toteutetut osiot merkitty mustalla värillä)

5.1. Sovelluksen runko

Moduulirakenteisena verkkosovelluksena, sovellus tarvitsee rungon, joka tarjoaa rajapinnan sovelluksen moduuleille toistensa kanssa toimimiseen. Rungon pääasiallinen tarkoitus on vähentää ylimääräistä ohjelmointikoodia ja tehdä moduuleista itsenäisempiä, jolloin yksittäisen moduulin lisääminen tai poistaminen ei vaikuta teknisesti muuhun järjestelmään. Runko asettaa pitkälti sovelluksen jatkokehityksen rajoitteet ja hyvin suunniteltu runko mahdollistaa myös sellaisten moduuleiden lisäämisen, joita ei alun perin ole kehityksessä otettu huomioon. Jossain vaiheessa tulee kuitenkin vastaan raja, jolloin varsinaisen rungon muutoksilta ei voi välttyä. Tämän vuoksi sovelluksen runko on pyritty pitämään mahdollisimman pienenä ja yksinkertaisena siirtämällä osan runkoon suunnitelluista osista moduuleiksi, niin että mahdolliset tulevat muutokset olisi mahdollista toteuttaa mahdollisimman pienillä runkomuutoksilla.

Moduuleiden toteutus on toteutettu teknisellä tasolla pitkälti funktioiden avulla. Tällä tavoin yksittäinen moduuli yleensä koostuu pääfunktiosta, jolla moduuli kutsutaan ja useammasta alifunktiosta, joita kutsutaan moduulin sisältä. Tällainen lähestymistapa johtuu funktioiden luonteesta, sillä yksittäisen funktion tarkoitus on tuottaa annetuista lähtötiedoista vain yksi tulos, näin ollen funktion määrittely ja sen merkitys on hyvin lähellä matemaattisen funktion käsitettä.

Joitakin moduuleiden alifunktioita on myös tarkoitus käyttää muissa moduuleissa, jolloin eri ohjelmaosat jakavat keskenään joitakin osia. Tällaisella ratkaisulla pyrittiin vähentämään lähdekoodia ja yhdenmukaistamaan saatavia tuloksia, vaikka lähestymistapa hieman

monimutkaistaakin sovelluksen toimintaa. Lähdekoodin lukua on kuitenkin tällaisissa tapauksissa pyritty helpottamaan kommentoinnin avulla (tarkemmin: *4.8. Itsedokumentoituva lähdekoodi*).

5.1.1. Hakemistorakenne

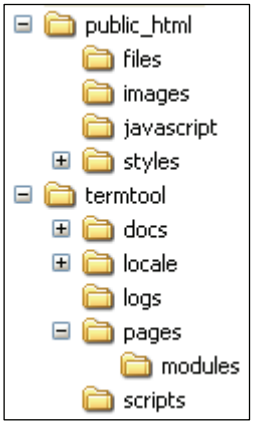
Hakemistorakenteen määritteleminen projektin suunnittelun alkuvaiheessa helpottaa kokonaisuuden hahmottamista ja jäsentää projektin loogisiksi osioiksi. Katsomalla hyvin suunniteltua hakemistorakennetta voi usein myös suoraan päätellä tuntematta projektia minkä tyylisestä sovelluksesta on kyse, mitä pääosioita siinä on ja mitä toiminnallisia vaatimuksia sovellukselle on asetettu. Oikein suunniteltu hakemistorakenne on myös yksi tehokkaimmista tavoista parantaa verkkoresurssin tietoturvaa, koska näin on mahdollista käyttää hyväkseen palvelimen omia tietoturvarakenteita ja lukuoikeuksia. Koska www-palvelimen juurihakemistossa (eng. webroot) olevat tiedostot ovat suoraan käyttäjien saatavilla, kaikki osat, jotka eivät vaadi käyttäjien suoraa pääsyä, pyrittiin sijoittamaan www-palvelimen juurihakemiston ulkopuolelle. Näin käyttäjiltä evättiin pääsy ohjelman kriittisiin komponentteihin, kuten tietokannan salasanoihin jne.

Hyvin suunniteltu hakemistorakenne helpottaa merkittävästi projektin toteutusta sekä sen myöhempää ylläpitoa ja kehitystä. Toteutusvaiheessa on kuitenkin yleensä hyvin vaikea vaikuttaa hakemistorakenteeseen sovelluksen sisäisten viittausten vuoksi. Tällainen ongelma on mahdollista ratkaista määrittelemällä muuttujien (eng. variable) avulla hakemistopolut yhdestä tiedostosta (Listaus 5-1), jolloin myöhemmin sovelluksessa käytetään hakemistoihin viittaamisen yhteydessä näitä muuttujia.

```
…
define('DS', DIRECTORY_SEPARATOR);
define('ROOT', dirname(dirname(__FILE__)));
           define('TERMTOOL', ROOT . DS . 'termtool' . DS);
                      define('PAGES', TERMTOOL . 'pages' . DS);
…
```

Listaus 5-1. Viittaukset kansioihin yhdestä paikasta muuttujien avulla

Yllä olevassa koodiesimerkissä (Listaus 5-1) määritellään juurihakemisto (eng. root) suhteessa kyseisen koodin sisältävään tiedoston kansioon (`../../`), ohjelmakansio (`../../termtool/`) ja ohjelmakansion yksi alikansioista (`../../termtool/pages/`). Näiden muuttujien avulla on mahdollista myöhemmin viitata yksinkertaisesti kirjoittamalla määritelty muuttuja (`PAGES . sivu.php` olisi sama kuin `../../termtool/pages/sivu.php`). Hakemistorakenne tulisi kuitenkin suunnitella mahdollisuuksien mukaan heti alussa toimivaksi. Projektin lopulliseksi hakemistorakenteeksi muovautui:



- **`/public_html`** -kansio on projektin webroot ja siihen on sijoitettu kaikki käyttäjälle näkyvät osat. Näihin tiedostoihin käyttäjillä on suora pääsy palvelimen ulkopuolelta ja sen vuoksi tietoturvasyistä kaikki tiedostot jotka eivät vaadi suoraa pääsyä tulisi sijoittaa muualle. Tässä kansiossa on myös hakemistorakenteen määrittelytiedosto (`rakenne.php`) ja etusivun käynnistysmoduuli (`index.php`).
- **`/public_html/files`** -kansioon on sijoitettu mahdolliset käyttäjien ladattavaksi tarkoitetut tiedostot.
- **`/public_html/images`** -kansioon on sijoitettu kaikki sovelluksessa käytetyt kuvat kuten ikonit, logot, maiden liput jne.
- **`/public_html/javascript`** -kansioon on sijoitettu kaikki käyttäjien tietokoneilla suoritettavat javascript -komentosarjat. Nämä komentosarjat huolehtivat mm. sovelluksen dynaamisista elementeistä.
- **`/public_html/styles`** -kansioon on sijoitettu kaikki sovelluksen käyttämät tyylimäärittelyt. Tyylimäärittelyt määräävät sovelluksen ja tulosteiden ulkoasun.
- **`/termtool`** -kansio on projektin varsinaisen sovelluksen juurikansio. Tämän kansion sisältöön käyttäjillä ei ole pääsyä palvelimen ulkopuolelta. Tästä kansiossa löytyy myös sovelluksen asetukset (`configuration.php`).
- **`/termtool/docs`** -kansio on varattu erilaiselle ylläpitäjille tarkoitetulle dokumentaatiolle.
- **`/termtool/locale`** -kansiossa on sovelluksen lokalisointitiedostot.
- **`/termtool/logs`** -kansio on varattu erilaisille sovelluksen tuottamille lokitiedostoille.
- **`/termtool/pages`** -kansiossa on erilliset internetsivut jotka ladataan käyttäjän valinnan mukaan etusivun käynnistysmoduuliin (`index.php`).
- **`/termtool/pages/modules`** -kansiossa on erillisten internetsivujen yhteiset osat (eng. shared elements).
- **`/termtool/scripts`** -kansiossa on sovelluksen ajettavat aliohjelmat, jotka ajetaan taustalla eivätkä ne näy suoranaisesti käyttäjille.

5.1.2. Sivuohjaus

Sivunohjaus on tärkeä periaatteellinen päätös sovelluksen rungon arkkitehtuurissa, sillä se vaikuttaa merkittävästi sovelluksen myöhempään kehitykseen. Sivuohjaus on mahdollista toteuttaa monella eri tavalla ja näillä kaikilla tavoilla on omat hyvät ja huonot puolensa ja käyttötarkoituksensa.

Käytetyimmiksi toteutustavoiksi voidaan erottaa kolme:

- Erilliset sivut (`/sivu1.php`, `/sivu2.php`, `/sivu3.php` jne.).
- Dynaaminen sivunohjaus AJAX-komponentin avulla (`/sivu.php` voi saada eri sisältöjä).
- Sivunohjaus palvelupyyntöjen avulla GET-metodia käyttäen, jossa resurssi määritellään URI:n avulla (`/sivu.php?haettuSivu=sivu2`).

Yleisin ja helpoin tapa toteuttaa sivunohjaus on erilliset sivut. Tällaisessa ratkaisussa yhteiset osat, kuten ylä- ja alatunnisteet, valikot jne. haetaan jokaiselle erilliselle sivuille muista tiedostoista. Suurin ongelma tällaisessa ratkaisussa on se, että uusien yhteisten osien kehityksen yhteydessä jokaisen sivun viittaukset pitää päivittää erikseen.

Dynaaminen sivuohjaus tapahtuu sivun sisällä yleensä JavaScript-avusteisesti, jolloin sivun sisältö vaihtuu päivittämättä sivua. Tällaisella toteutustavalla on käyttökohteensa ja varsinkin sivun yksittäisten osien latauksessa, tekniikka on saanut paljon suosiota. Suurimpana ongelmakohtana on kuitenkin selaimen sisäänrakennetun ”siirry sivu taakse/eteenpäin” -toiminnon tukeminen, koska sivujen välillä ei selaimen näkökulmasta siirrytä. Käyttäjäystävällisen toteutuksen kannalta tällainen toteutustapa on useimmissa tapauksissa huono, mutta joissakin tapauksissa tukea sivun sisällön automaattiselle uudelleentuottamiselle ei vaadita.

Viimeaikoina paljon suosiota saanut sivunohjaus palvelupyyntöjen avulla oli valittu tämän sovelluksen toimintatavaksi. Tekniikalla on useita hyviä puolia ja suurin ero erillisiin sivuihin nähden on toimintalogiikassa, sillä yhteiset osat kuten ylä- ja alatunnisteet, valikot jne. ovat sovelluksen rungossa ja erilliset sivut ladataan sovelluksen runkoon, jolloin uuteen yhteiseen elementtiin viittauksen luonti tai vanhan poisto vaikuttaa koko järjestelmään.

Ratkaisulla on myös huonoja puolia, joista tärkein on tietoturvan varmistaminen, koska palvelupyyntöjä voidaan käyttää suunnitellun vastaisesti muokkaamalla niitä. Sen vuoksi palvelupyynnöt, tuli ne sitten sivuohjauksesta, lomakkeesta taikka jostain muusta elementistä, tulisi aina tarkastaa ennen prosessointia.

Toteutettavan termipankin sivuohjaus perustuu kokonaisuudessaan palvelupyyntöjen käsittelyyn, jolloin melkeinpä mikä tahansa sisältö voidaan tuottaa uudestaan suoraan sivulinkistä. Tällä tavoin on mm. mahdollista tallentaa hakutulokset ym. keskeneräinen työ esimerkiksi suosikkikansioon ja jatkaa työtä myöhemmin tai lähettää linkkinä toiselle käyttäjälle, jolloin linkkiä käyttäen käyttäjä pääsee kirjautumisen jälkeen suoraan tarkastelemaan haluttua sisältöä. Tässä toiminnossa on pakko painottaa auktorisoinnin tarvetta eli suorat linkit sisältöön ovat usein tietoturvauhka ja sen vuoksi kirjautumisjärjestelmän on oltava sellainen, ettei sitä voida ohittaa millään palvelukäskyllä.

Sivuohjauksen toteutus oli yksi syy siihen, että käyttäjätasot tarkastetaan sivun jokaisella latauskerralla (tarkemmin: *5.3. Auktorisointi*).

5.2. Käyttöliittymä

Käyttöliittymä (eng. UI; User Interface) on paljon laajempi käsite kuin pelkästään sovelluksen ulkoasu, vaikka ulkoasu onkin hyvin tärkeä osa-alue käyttöliittymän toteutuksessa. Sen vuoksi käyttöliittymän suunnittelu tulisi toteuttaa pitkin sovelluskehitysprosessia eikä erillisenä toimintona, jossa liimataan käytettävyys tuotteeseen. Käyttöliittymäsuunnittelu tulisi sijoittaa projektin alkuun, ennen toteutuksen suunnittelua, jotta käyttöliittymäratkaisuja ja järjestelmän soveltuvuutta käyttötarkoituksiinsa voitaisiin testata niin varhaisessa vaiheessa, että testitulosten vaatimat suuretkin muutokset olisivat vielä helposti toteutettavissa. (Sinkkonen, 2006.)

Verkkosovelluksen käyttöliittymä koostuu toiminnan mahdollistavista elementeistä (HTML-koodista), kuten esimerkiksi lomakkeesta, joilla tiedot syötetään tai valikkopainikkeista, joilla siirrytään sivulta toiselle ja toiminnallisten elementtien ulkoasumäärityksistä, joilla varmistetaan helppokäyttöinen ja mielekäs käyttökokemus. Vaikka molemmat osa-alueet ovat hyvin vahvasti riippuvaisia toisistaan ja niistä sovellusmoduuleista, joiden käyttöä käyttöliittymän osa mahdollistaa, käyttöliittymä pyrittiin pitämään erillisenä toteutuksena, koska hyvän ohjelmointitavan mukaisesti prosessointi ja ulkoasuelementit on pidettävä mahdollisuuksien mukaan erillään selkeän ohjelma-arkkitehtuurin säilyttämiseksi (tarkemmin: *4.8.1. Semanttinen merkintätapa*).

Peruskäyttöliittymä toteutettiin sovelluksen rungon toteutuksen yhteydessä ja myöhemmin sitä laajennettiin vastaamaan moduulien tarpeita moduulien toteutuksen yhteydessä. Projektin käyttöliittymän toteutus voidaan jakaa karkeasti kahteen vaiheeseen:

- Toiminnallisten osien suunnittelu ja toteutus, jossa varmistetaan käyttöliittymän looginen ja joustava toiminta varsinaisen sovelluksen kanssa. Tämä vaihe voidaan käsittää rajapinnan luomisena sovelluksen ja käyttäjän välille.
- Ulkoasusuunnittelu ja -määritykset, jossa varmistetaan käyttöliittymän miellyttävyys ja helppokäyttöisyys.

Käyttöliittymän toiminnallisten osien suunnittelu- ja toteutus tapahtui varsinaisen moduulin suunnitteluvaiheessa, koska näin kokonaisuuden hahmottaminen oli helpompaa ja vältyttiin suuremmilta muutoksilta sovelluksen lähdekoodissa. Ulkoasusuunnittelu ja -määritys sijoittuivat taas moduulin toteutuksen loppuvaiheeseen, jolloin sovellusmoduulin toimintaa hienosäädettiin, eikä lähdekoodiin tullut enää merkittäviä muutoksia. Jokaisesta käyttöliittymän osasta toteutettiin

suunnitteluvaiheessa useita eri versiota (prototyyppejä), joita vertaamalla toteutuksessa syntyi lopullinen ulkoasu (LIITE 1: TERMTOOL – termipankki).

Sovelluksen ulkoasu ei ole pelkästään esteettinen yksityiskohta, vaan se vaikuttaa merkittävästi käytettävyyteen, asiasisältöjen ymmärrettävyyteen ja resurssin käytön mielekkyyteen (esimerkiksi punainen teksti vihreällä taustalla rasittaa silmiä ja on vaikealukuista ellei jopa mahdotonta värisokeille ihmisille). Sen vuoksi helppokäyttöinen ja intuitiivinen käyttöliittymä oli projektin tärkeimpiä tavoitteita ja ulkoasun suunnitteluun oli varattu reilusti aikaa. Ulkoasusuunnittelun tärkeimpiä tavoitteita ovat (Lynch, 2009):

- luoda selkeä visuaalinen hierarkia, jotta ensisilmäyksellä selviäisi mikä on tärkeä ja mikä toissijaista
- määritellä sivun funktionaaliset alueet
- ryhmitellä toisiinsa liittyvät elementit sisällön rakenteen selkeyttämiseksi

Meillä kaikilla on kokemuksiemme kautta muodostunut käsitys siitä, miltä jonkun tietyn sovelluksen tai internetsivuston tulisi näyttää ja mitä toiminnallisuuksia siitä tulisi löytyä. Jokainen vähänkin internetiä käyttänyt henkilö osaa heti kertoa usein jo pelkästään ulkonäön perusteella onko kyseessä esimerkiksi uutis-, urheiluväline-, galleria-, verkkokauppa- vai hakukonesivusto ja mitä toimintoja kyseiseltä sivustolta löytyy. Muiden palvelujen kautta syntyneitä ns. standardeja tulisi noudattaa intuitiivisen toiminnan varmistamiseksi. Nämä ns. standardit varmistavat, että käyttäjät tietävät mitä toimintoja odottaa, miltä ne toiminnot näyttävät, mistä löytyvät ja miten toimivat. (Nielsen, 2004.) Mikäli tästä säännöstä poiketaan, siihen on oltava erittäin hyvin perusteltavissa oleva syy, sillä kaikki poikkeamat totutusta nostavat oppimiskynnystä, aiheuttavat järjestelmän väärinkäyttöä ja aiheuttavat käyttäjissä ristiriitoja lisäten resurssin käytön epämiellyttävyyttä.

5.2.1. Erilaiset työpöydät

Kaikkia käyttäjien tarpeita ja mieltymyksiä ei kuitenkaan ole mahdollista tyydyttää parhaimmallakaan käyttöliittymällä ja eri käyttötarkoituksiin sopii tietyt toteutustavat paremmin kuin toiset. Järjestelmän testauksesta puuttui myös laajemmat käytettävyystestaukset, joihin olisi osallistunut järjestelmän todellisia käyttäjiä, joten olisi hyvin rohkeata väittää käyttöliittymätoteutusta täydelliseksi. Edellä mainituista seikoista johtuen käyttöliittymän toteutus suunniteltiin helposti muokattavaksi kokonaisuudeksi. Tästä syntyi ajatus erilaisista työpöydistä, joista käyttäjät voisivat valita vapaasti mieleisensä ja käyttötarkoitukseensa sopivan käyttöliittymän ulkoasun eli työpöydän (eng. work space).

Eri työpöytien ulkoasumääritykset toteutetaan suunnitelman mukaisesti CSS 2.1 tyylimäärittelyillä erillisiin tyylitiedostoihin ja sovelluksen erillinen työpöydän valintaan tarkoitettu moduuli lataa käyttäjän määrittelemän tyylitiedoston, jonka mukaisesti sovelluksen ulkoasu määräytyy. Tyylitiedostojen määrää ei ole millään tavoin rajoitettu, eivätkä ne vaikuta muuhun kuin käyttöliittymän ulkoasuun, joten uusien työpöytien luonti on yksinkertaista. Käyttäjän tekemät valinnat tallentuvat evästeiden avulla päätekohtaisiksi asetuksiksi ja näin ollen asetuksia ei tarvitse muuttaa jokaisella käyttökerralla. Toteutus on päätekohtainen, eikä käyttäjäkohtainen sen vuoksi, koska näin mahdollistetaan myös kirjautumattomille käyttäjille eri työpöytien käytön.

5.3. Auktorisointi

Internetsovelluksissa tärkeäksi kysymykseksi muodostuu se, kuinka laajasti anonyymiä toimintaa sallitaan. Tutkimuskäytössä termipankin luotettavuus on tärkeä, koska muiden käyttäjien tekemien/muokkaamien artikkelien oikeellisuuden tarkastaminen on työlästä. Termipankkia täydentävillä henkilöillä on oltava selvillä vähintäänkin terminologisen työn perusperiaatteet, koska muussa tapauksessa termipankki täyttyy helposti roska-artikkeleista ja menettää käyttöarvonsa. Tällainen ongelma on tiedostettu myös muissa olemassa olevissa projekteissa, mistä kielii mm. se, että kaikki luotettavat termipankkiprojektit ovat suljettuja projekteja. Terminologisen työn omalaatuisuudesta johtuen en näe mahdollisena täysin avointa toteutustapaa termipankkiympäristössä (vertaa: Wikipedia) siinä muodossa kuin avoimet projektit tällä hetkellä esiintyvät. Myös yhteistyön rakentaminen on ongelmallinen aidosti avoimissa projekteissa sillä, vaikka avoimet projektit tarjoavatkin laajat yhteistyönmahdollisuudet, ne eivät mielestäni mahdollista tarpeeksi tiivistä ja systemaattista yhteistyötä eri osallistujien välille, koska roolijakoa ei niissä ole mahdollista määritellä tarpeeksi selvästi. Termipankin sisällön uskottavuuden säilyttäminen on yksi suurimmista haasteista termipankin kasvaessa, jopa silloin kun tekijät ovat ammattilaisia.

Toteutettava termipankkisovellus päätettiin pitää kokonaan suljettuna projektina, mikä tarkoittaa käytännössä sitä, että sovelluksen käyttö alkaa aina sisään kirjautumisella ja loppuu uloskirjautumiseen. Uloskirjautuminen merkittiin toiminnallisten osien kaavioon (Kuva 5-1) katkoviivalla, koska vaikka toiminto on pakollinen, usein käyttäjät eivät kirjaudu ulos järjestelmästä suunnitellulla tavalla, vaan esimerkiksi siirtyvät suoraan toiselle sivulle tai sulkevat selaimen. Tällainen toiminta on epätoivottavaa, mutta harmillisen yleinen ja sen vuoksi se on otettava huomioon toteutuksessa. Tässä projektissa ongelma on otettu huomioon, sillä että uloskirjautuminen tapahtuu automaattisesti, mikäli mitään toimintaa ei ole havaittavissa tietyn ajan kuluessa. Tällainen toimintatapa voi katkaista käyttäjän toiminnan kesken työskentelyn, mutta

sivuohjauksella (tarkemmin: *5.1.2. Sivuohjaus*) pyritään varmistamaan toiminnan jatkumisen siitä pisteestä, missä toiminta oli ennen automaattista uloskirjautumista.

Sisään kirjautumisen pakollisuuden lisäksi käyttöön otettiin eritasoiset pääsytasot eri käyttäjäryhmille. Pääsytasoilla on mahdollista tarjota eri käyttäjille eri toimintoja ja taas vastaavasti rajata niitä toisilta pois. Pääsytaso ja sitä kautta sisään kirjautumisen status tarkastetaan jokaisella sivun latauskerralla. Tällä tavoin pyritään välttämään tilanteet (tietoturva-aukot), joissa auktorisointi olisi mahdollista kiertää. Eri pääsytasojen määrä rajattiin neljään:

- **0 – Ei kirjautunut käyttäjä:** Ei oikeutta hakea, lisätä, poistaa tai muokata artikkeleja eikä muuttaa käyttäjätietoja.
- **1 – Kirjautunut käyttäjä:** Oikeus hakea artikkeleja ja muuttaa itseään koskevia käyttäjätietoja, muttei oikeutta lisätä, poistaa tai muokata artikkeleja eikä muuttaa toisten käyttäjätietoja.
- **2 – Tutkija:** Oikeus hakea lisätä, poistaa tai muokata artikkeleja ja muuttaa itseään koskevia tietoja, muttei oikeutta muuttaa toisten käyttäjätietoja.
- **3 – Ylläpitäjä:** Kaikki oikeudet. Mm. oikeus muuttaa toisten käyttäjätietoja tai lisätä/poistaa käyttäjiä.

Termipankin kasvaessa esimerkiksi hakutoiminnon salliminen avoimeen käyttöön olisi täysin perusteltua ja pääsytasovaatimusten kautta on pyritty ottamaan huomioon sovelluksen toimintojen mahdollistaminen tulevaisuudessa ilman kirjautumista.

5.4. Ylläpitotoiminnot

Ylläpitotoiminnot ovat tärkeä osa minkä tahansa verkkosovelluksen hallintaa, mikäli se vaatii rekisteröitymistä. Ylläpitotoiminnot jäivät varsinaisen toteutuksen ulkopuolelle, mutta ne on otettu huomioon suunnittelussa.

Ylläpitotoiminnoksi lasketaan käyttäjätilien hallinta sen kaikissa muodoissa eli rekisteröityminen, oman käyttäjätilin tietojen muokkaus, oman käyttäjätilin poisto järjestelmästä, muiden käyttäjätilien muokkaus ja uusien käyttäjien lisääminen tai vanhojen poisto järjestelmästä. Yksittäisille käyttäjille saatavilla olevat ylläpitotoiminnot vaihtelevat kuitenkin pääsytason mukaisesti.

Tällä hetkellä järjestelmän käyttäjäksi rekisteröityminen tapahtuu kirjallisella lomakkeella, joka toimitetaan ylläpitäjälle, joka puolestaan lisää manuaalisesti käyttäjätiedot järjestelmään. Kirjallisen lomakkeen tulostaminen, täyttäminen ja toimittaminen ylläpitäjille on työlästä ja tällainen toimintatapa on hidas, kuormittaa ylläpitäjiä ja mahdollinen vain silloin kun järjestelmässä on vain

muutama käyttäjä. Käyttäjämäärän kasvaessa rekisteröityminen tulisi hoitaa automaattisesti sovelluksen rekisteröintilomakkeella, jolloin ylläpitäjän tehtäväksi jää ainoastaan hyväksyä tai hylätä hakemus. Ehdotettu automaattinen rekisteröintilomake on kuitenkin ristiriidassa Tampereen yliopiston palvelimien ylläpitoa koskevia määräyksiä, joissa vaaditaan kirjallinen allekirjoitettu hakemus. Siten ehdotettu rekisteröinti palvelun kautta on nykyisellään mahdotonta toteuttaa. Pitää kuitenkin myös muistaa, että rekisteröintilomake alentaa merkittävästi kynnystä rekisteröitymiseen palvelun käyttäjäksi ja uskon että tulevaisuudessa todentamiskeinojen parantuessa asiaan tulee muutos.

Koska sovellus on tarkoitettu tutkimuskäyttöön ja lähteen luotettavuus on hyvin olennainen osa tällaista tietokantaa, uusien käyttäjien rekisteröityminen, rekisteröintilomaketapauksessa, tulisi rajoittaa tai vaihtoehtoisesti artikkelien lisääminen/muokkaaminen rajoittaa vain ”luotettaville” käyttäjille. Rekisteröityminen on mahdollista rajoitta useilla eri keinoilla, joista mielestäni toimivimmat ovat rekisteröitymisen salliminen vain tietyistä sähköpostiosoitteista (esimerkiksi vain @uta.fi -pääteisistä sähköpostiosoitteista) tai ylläpitäjän erillinen hyväksyminen, jolloin uudet rekisteröinnit näkyisivät ylläpitäjän ylläpitotoiminnoissa.

Käyttäjien kannalta on tärkeä nähdä sovelluksesta rekisteröitymisen yhteydessä syötetyt itseään koskevat tiedot ja myös päästä muokkaamaan osaa niistä. Rekisteröitymisen yhteydessä syötetyissä tiedoissa voi esiintyä virheitä tai elämäntilanteiden muuttuessa tiedot voi muuttua, sillä esimerkiksi sukunimi voi muuttua naimisiinmenon yhteydessä, yhteydenpitoon käytetty sähköpostiosoite voi muuttua erinäköisistä syistä hyvinkin usein jne. Jo pelkästään tietoturvasyistä salasanat pitäisi vaihtaa säännöllisesti. Mikäli käytössä on tunnuksen vanheneminen, myös tällaiset päivämäärät on hyvä olla esillä tilinhallinnassa, sillä näin työnteko ei keskeydy tilin jäädyttämiseen, mikäli käyttäjä osaa ottaa yhteyttä ylläpitoon ennen kuin tili vanhentuu.

Käyttäjien ei kuitenkaan tulisi päästä näkemään toisten käyttäjien henkilökohtaisia tietoja, eikä varsinkaan muokkaamaan niitä. Ylläpitäjien työtä taas tehostaa merkittävästi hyvin suunniteltu tilien hallinta, jolloin käyttäjätietoja, pääsytasoja ja uusien käyttäjien lisäämistä ja vanhojen poistamista voidaan toteuttaa tehokkaasti. Ylläpitäjien hyvät ylläpitotyökalut heijastuvat myös suoraan käyttäjiin, jolloin heitä koskevia asioita voidaan prosessoida nopeasti.

5.5. Artikkelien lisäys tietokantaan

Artikkelien lisäys jäi varsinaisen toteutuksen ulkopuolelle, koska artikkelien lisäys oli toteutettu sovellusta edeltävässä toteutuksessa, mutta artikkelien lisäys on otettu huomioon suunnittelussa. Artikkelien lisääminen jakaantuu kahteen erityyppiseen toimintoon, joita eri käyttäjät mahdollisesti tarvitsevat eri tilanteissa:

- yksittäisen artikkelin lisääminen tietokantaan
- ennalta kerätyn artikkeliryhmän lisääminen tietokantaan

Yksittäisten artikkelien lisääminen on luontevinta toteuttaa lomakkeen kautta sovelluksen sisältä, jolloin jokainen artikkelin tietue syötetään erikseen. Tällainen toimintatapa on useimmille käyttäjille tuttu muista sovelluksista ja internetsivustoista, sillä se muistuttaa pitkälti rekisteröintilomaketta. Näin ollen tällainen toteutus ei nosta merkittävästi oppimiskynnystä. Huonona puolena lomaketavassa on sen hitaus, varsinkin artikkeliryhmän ollessa suuri. Artikkeliryhmien syöttäminen tietokantaan olisi luontevinta toteuttaa tiedostosta viennillä. Tiedostosta viennin toteutus ei ole ongelmaton ja melko haasteellinen toteuttaa käyttäjäystävällisesti, koska se vaatii etukäteen tiettyjen sääntöjen mukaan muotoillun tiedoston, josta sovelluksen elementit osaisivat tunnistaa oikeat kentät ja sijoittaa ne oikeisiin tietueisiin. Sen lisäksi, että käyttäjät joutuisivat opettelemaan jonkun tietyn muotoilun artikkelikokoelmille, teknisesti tällaisen tekoälyn luominen on vaativa tehtävä.

Artikkelien lisäys tietokantaan luo käyttäjille käytön aikana uuden tarpeen: syötettyjen artikkelien tarkastelu. Varsinkin silloin kun artikkelijoukkoa syötetään käsin, voi käyttäjä mm. toiminnan monotoonisuden vuoksi sekoittaa helposti kohdan missä on menossa, tallennusvaiheessa voi tulla jokin häiriö tai vaikkapa painike, jolla toteutetaan tallennus, ei jostain syystä painaudu. On hyvin turhauduttavaa huomata, että syötetyistä artikkeleista puuttuu jokin artikkeli vasta työn loppuvaiheessa, jolloin käyttäjä joutuu käymään läpi kaikki syötetyt tiedot uudestaan löytääkseen puutteet. Usein tällainen läpikäynti ei edes ole mahdollinen, koska usein artikkelien sisältöjä etsitään useista eri lähteistä.

Ongelmaa helpottaa mikäli syötetyt artikkelit ilmestyvät artikkelien syöttösivulle listaksi, josta voi tarvittaessa muokata yksittäisiä artikkeleja. Tällaisen listan luomisessa on tärkeää, että artikkelin tallentamisen jälkeen se haetaan tietokannasta, eikä vain tulosteta sivulle syöttölomakkeesta, sillä näin tapahtuu luonnollinen tarkastus, että artikkeli on todella tallentunut tietokantaan ja sitä on mahdollista tarkastella juuri sellaisenaan kuin se on tietokannassa.

Artikkelin tallennus ja syötettyjen artikkelien listan luonti on hyvä toteuttaa dynaamisesti, ilman sivun uudelleenlatausta, Tässä toiminnassa ei vaadita tukea selaimen ”siirry sivu taakse/eteenpäin”-toiminnoille, sillä tallennettu artikkeli ei poistu pelkästään siirtymällä edelliselle sivulle. Dynaamisesti toteutettuna toiminto vähentää palvelinkuormaa, koska suurin osa työstä hoidetaan käyttäjän koneella ja antaa paremman ohjauksen esim. lomakkeiden sisällön hallintaan (lomakkeet eivät tyhjene oletusarvoisesti kuten sivun uudelleenlatauksen jälkeen). Sivun uudelleenlataus puhdistaa taas puolestaan työpöydän jo syötettyjen artikkelien listasta ja lomakkeiden sisällöstä ja näin käyttäjä voi aloittaa uuden artikkelien syöttötyön.

5.6. Artikkelihaku tietokannasta

Artikkelien haku voidaan jakaa kahteen erityyppiseen toimintoon, joita eri käyttäjät mahdollisesti tarvitsevat eri tilanteissa:

- yksinkertainen haku
- laajennettu haku

Yksinkertainen haku (Kuva 5-2) mahdollistaa nopean ja yksinkertaisen artikkeliryhmän haun ennalta määriteltyjen ehtojen mukaisesti, joihin käyttäjillä on vain rajalliset vaikutusmahdollisuudet. Tarkoituksena oli toteuttaa tämä osio mahdollisimman helppokäyttöiseksi.

Kuva 5-2. Termipankin yksinkertainen haku

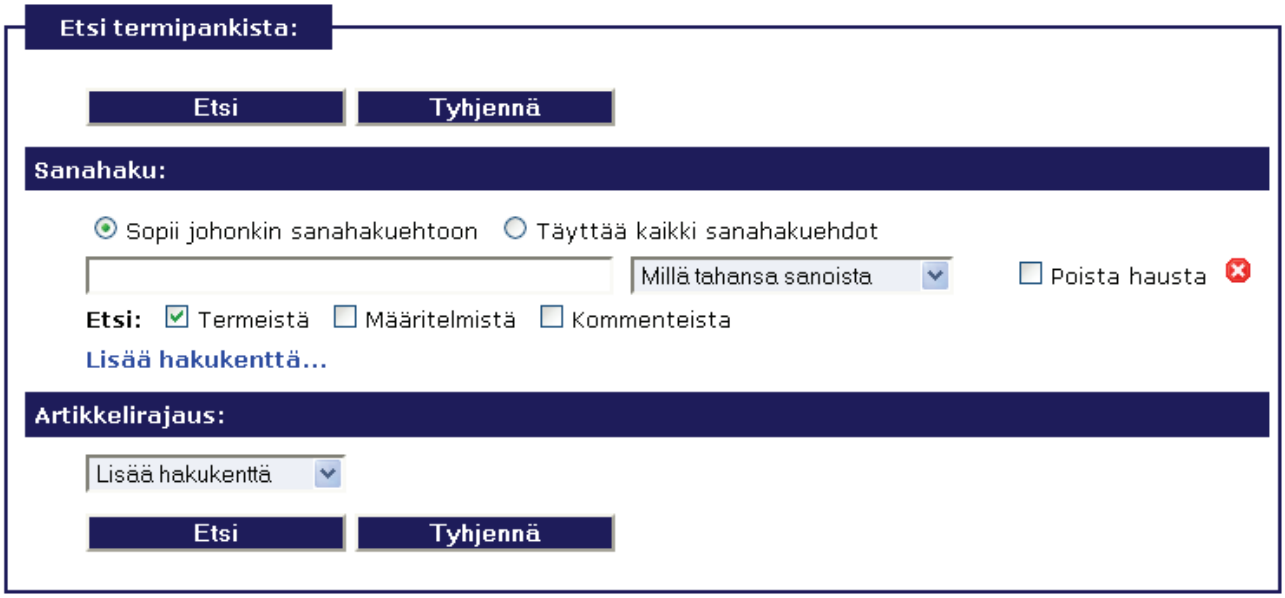
Yksinkertaisen haun avulla on mahdollista hakea vain kolmesta eri kentästä: 1. termeistä, 2. määritelmistä ja 3. kommenteista ja se toimii sanakirjantyylisesti eli sen käyttöfunktio on löytää yksittäisiä termiartikkeleja. Hakuehtojen suhde rakentuu `OR` operaattorin avulla eli vain yhden ehdon on täytyttävä. Tällainen haku tuottaa siis mahdollisimman laajan hakujoukon. Esimerkiksi Google käyttää hauissaan aina `AND` operaattoria mahdollisimman tarkkojen hakutulosten tuottamiseen ja operaattorin vaihto `AND` operaattoriksi termipankin kasvaessa isommaksi olisi täysin perusteltua.

Varsinkin tutkimuskäytössä käyttäjiä usein kiinnostavat yksittäisten termien sijan tarkat ehdot täyttävät artikkeliryhmät. Tällaista tarvetta varten järjestelmään suunniteltiin laajennettu haku. Laajennettu haku mahdollistaa hakujen rajaamisen monien eri ehtojen avulla ja näitä ehtoja on mahdollista yhdistää monipuolisesti. Sovelluskehityksen kannalta ongelmaksi muodostuu se, että

sovelluskehittäjä ei voi ennalta tietää, millaisia rajauksia käyttäjät tulevat tarvitsemaan ja kuinka laajasti. Hakumoduulin toteutusvaiheessa arvioitiin useita erilaisia käyttöskenaarioita, joiden avulla pystyttiin rajaamaan työkalun mahdolliset käyttötavat. Käyttöskenaarioiden kautta muodostui erittäin suuri hakuvaihtoehtojen määrä, koska rajausehtoja on mahdollista yhdistää vapaasti keskenään. Laajennetussa haussa päätettiin jättää käyttäjille mahdollisimman suuri vapaus määritellä itsenäisesti hakujen rajausehdot, mikä johti laajennettujen hakutyökalujen yleiseen ongelmaan: hakulomakkeen paisumiseen. Havainnollistavana esimerkkinä toimii monipuolinen, mutta hyvin raskaslukuinen Googlen laajennettu haku (Kuva 5-3).

Kuva 5-3. Laajennetun haun yleinen ongelma: Lomakkeen pituus ja raskaslukuisuus.
(http://www.google.fi/advanced_search)

Yleensä käyttäjä ei tarvitse kerralla muutamaa hakuehtoa enempää, vaikka nämä ehdot voivatkin vaihdella kerrasta toiseen. Googlen esimerkin mukaisessa kiinteässä lomakkeessa, joka on laajalti käytössä myös useimmissa termipankkisovelluksissa, jotka mahdollistavat laajennetun haun, on myös toinen ongelma, siinä ei voi yhdistää samoja hakuehtoja toisiinsa. Termipankkikäytössä tällainen tekninen rajoitus vähentää merkittävästi työkalun käyttöarvoa käyttäjien kannalta, sillä esimerkiksi kahden eri käyttäjän tekemät artikkelit olivat käyttöskenaarioissa mahdollinen haku. Ongelma päätettiin ratkaista dynaamisella hakulomakkeella (Kuva 5-4).



Kuva 5-4. Termipankin laajennetun haun perusnäkymä

Dynaamisen ja samalla käyttäjäystävällisen hakulomakkeen luominen on hyvin haasteellista, sillä ohjelmistokehittäjän mahdollisuus vaikuttaa lomakkeen sisältöön katoaa. Luotujen käyttöskenaarioiden avulla pyrittiin poistamaan sellaisten hakuyhdistelmien mahdollisuus, jotka eivät ole mielekkäitä tai loogisia.

Laajennettu haku jaettiin kahteen osaan: 1. sanahakuun ja 2. artikkelirajaukseen. Vaikka jako onkin osittain keinotekoinen, tällaisen jaon tarkoitus oli selkeyttää kenttien suhdetta toisiinsa, koska niitä ei voitu pitää samoina tai antaa kokonaan käyttäjän määriteltäväksi. Sanahaku sisältää vapaan sanahaun ja on vain laajennettu versio yksinkertaisesta hausta. Artikkelirajaus sisältää taas rajauksen luokan, kielen, valmiusasteen, tekijän, luonti- ja muokkauspäivämäärän mukaan. Kaikki edelliset jakaantuvat vielä pienempiin osiin, esimerkiksi rajaus käyttäjän mukaan voidaan määritellä tarkemmin artikkelin ”luojaksi tai muokkaajaksi”, ”luojaksi” tai ”muokkaajaksi” ja päiväykset määritellä tarkemmin seuraavalla tavalla: ”yhtä suuri kuin (=)”, ”pienempi tai yhtä suuri kuin (<=)” tai ”suurempi tai yhtä suuri kuin (>=)”.

5.6.1. Hakusanojen ja -kenttien suhde toisiinsa nähden

Hakusanojen ja -kenttien suhde toisiinsa nähden määräytyy loogisten operaattoreiden avulla (tarkemmin: *4.4.2. Kyselyehtojen määrittely*). Yksittäisen sanahakukentän sanojen ja/tai sanaliittojen suhteen toisiinsa nähden käyttäjä voi määritellä erikseen, kuten myös sanahakukenttien suhteen toisiinsa nähden. Sanahakukenttien suhde toisiinsa nähden on silti aina sama eli jos käyttäjä määrittelee käytettäväksi `OR` operaattorin (”Sopii johonkin sanahakuehtoon”), kaikki sanahakukentät saavat toisiinsa nähden `OR` operaattorin. Tällainen rajoitus johtuu siitä, että jos käytössä on useampi kenttä ja niiden suhde toisiinsa on eri, ei voida tietää tämän suhteen

rakentumista eli esimerkiksi `hakukenttä1 OR hakukenttä2 AND hakukenttä3` voi olla kumpi tahansa seuraavista:

- `(hakukenttä1 OR hakukenttä2) AND hakukenttä3`
- `hakukenttä1 OR (hakukenttä2 AND hakukenttä3)`

Artikkelirajauskenttien suhde toisiinsa on taas ennalta määrätty, eikä käyttäjä voi vaikuttaa siihen millään tavoin. Tämä johtuu siitä, että loogisten hakujen aikaansaamiseksi näillä kentillä ei voi olla kuin yksi suhde toisiin rajauskenttiin. Esimerkiksi artikkelilla ei voi olla kahta tekijää tai luontipäivämäärää, jolloin haku: `tekijä = X AND tekijä = Y` olisi täysin järjetön. Toisaalta myös haku: `tekijä = X OR päivämäärä = 01.01.09` olisi melko epälooginen, koska näillä kentillä ei ole minkäänlaista keskinäistä suhdetta. Artikkelirajauskenttien suhde toisiinsa ei ole yksiselitteinen ja riippuu kentän sisällöstä. Samojen rajausehtojen suhde toisiinsa määräytyy `OR` operaattorin avulla (`tekijä = X OR tekijä = Y`) ja eri rajausehtojen suhde määräytyy `AND` operaattorin avulla (`luokka = X AND tekijä = Y`). Huomioitavaa on, että eriävät tarkemmat rajausehdot lasketaan eri rajausehdoiksi eli näin ollen suhde on `OR` (`luoja (luoja) = X OR luoja (muokkaaja) = Y`).

Sanahakukenttien ja artikkelirajauskenttien välinen suhde on aina poissulkeva (`AND`) eli sanahakukenttien ehtojen on täytyttävä ja artikkelirajauskenttien ehtojen on täytyttävä samanaikaisesti. Tällainen rajoitus puolestaan johtuu siitä, että esimerkiksi samanaikainen haku hakusanalla tai (`OR`) tekijällä ei olisi millään tavoin looginen, sillä se tuottaisi kaikki artikkelit joissa esiintyy hakusana tai joiden tekijä on X.

Sekä sanahaussa että artikkelirajoituksissa on myös mahdollista asettaa käänteinen haku (`NOT`) eli ”hakutulos ei sisällä” hakukentässä olevaa arvoa. Mahdollisuus käänteisiin hakuihin on hyvin tärkeä ja se on tehokas rajauskeino muiden hakuehtojen kanssa käytettynä, sillä se mahdollistaa haut, joista poistetaan yksi tekijä, kuten esimerkiksi artikkelin haku kaikista muista luokista paitsi ei luokasta X. Samaan lopputulokseen voi päästä asettamalla tarpeeksi ”normaaleja” hakuehtoja (hae luokasta X, luokasta Y jne), mutta tällainen ratkaisu on selvästi käyttäjäystävällisempi ja selkeämpi.

Käänteisiin hakuihin pätee samat periaatteet kuin normaaleissakin hakuehtojen suhteissa eli esimerkiksi sanahaussa `NOT` operaattori saa käyttäjän määrittämän lisämäärityksen `OR` (”Sopii johonkin sanahakuehtoon”) tai `AND` (”Täyttää kaikki sanahakuehdot”). Käänteinen haku on kuitenkin siinä mielessä poikkeuksellinen, koska se katsotaan olevan eriävä rajausehto verrattuna sen ”normaaliin” muotoon eli suhde muihin kenttiin on seuraavanlainen:

- Molemmat käänteisiä: `NOT tekijä = X OR NOT tekijä = Y`
- Vain toinen on käänteinen: `tekijä = X AND NOT tekijä = Y`

5.6.2. Tietojen esitys

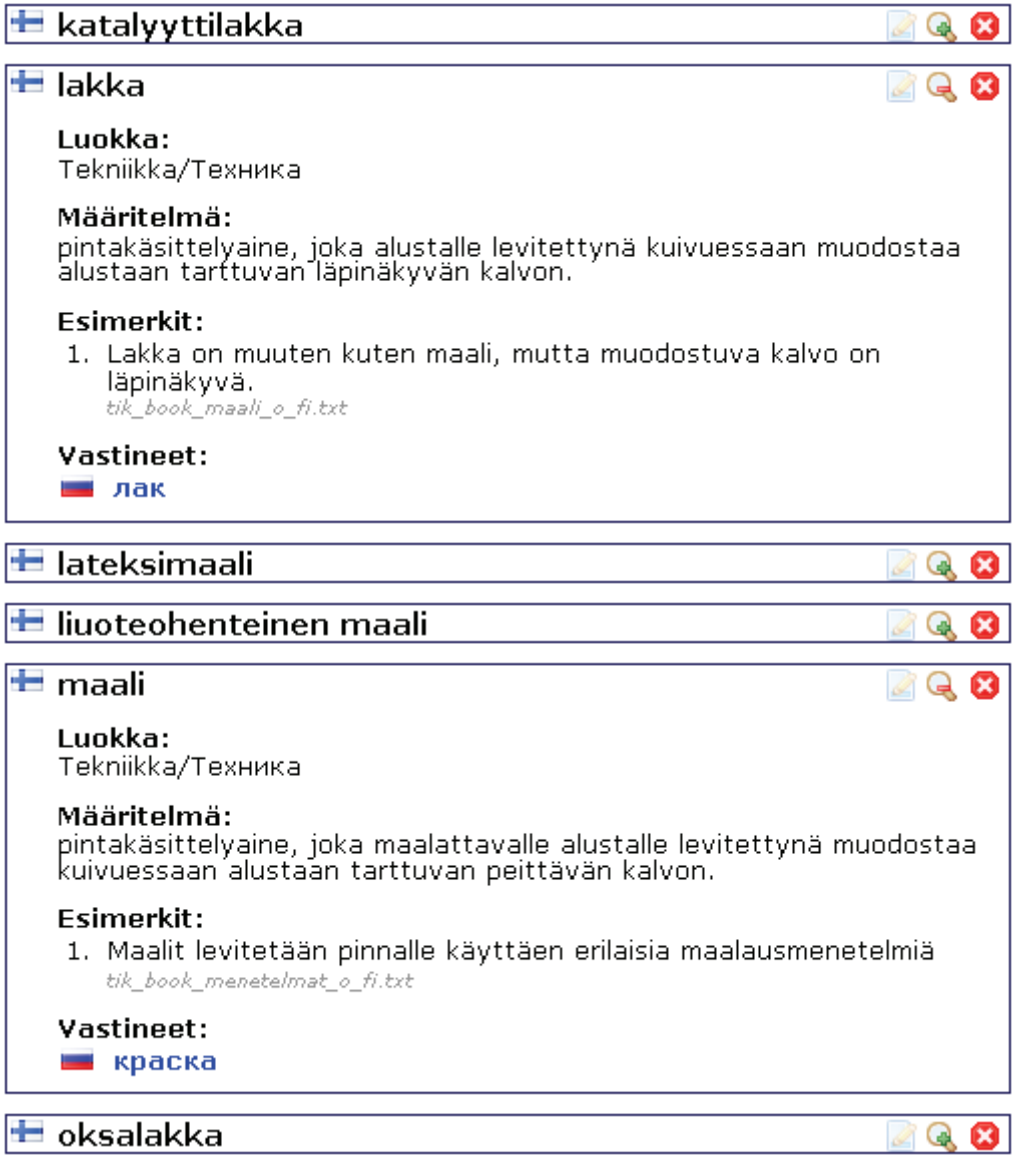
Taulukoita käytetään taulukkomuotoisen tiedon esittämiseen, mutta näennäisesti taulukkomuotoinen tieto ei aina ole sitä. Termipankin tietokannan sisältö voidaan kuvata taulukkomuotoisesti, mutta silloin esityksen pitäisi olla taulukossa 5-1. esitetyssä muodossa.

Termi	Luokka	Määritelmä	Esimerkit	Vastineet
Lakka	Tekniikka	pintakäsittelyaine, joka alustalle levitettynä… kuivuessaan muodostaa alustaan tarttuvan läpinäkyvän kalvon.	1. Lakka on muuten kuten maali, mutta muodostuva kalvo on läpinäkyvä. tik_book_maali_o_fi.txt	ru лак
Maali	Tekniikka	pintakäsittelyaine, joka maalattavalle alustalle levitettynä muodostaa kuivuessaan alustaan tarttuvan peittävän kalvon.	2. Maalit levitetään pinnalle käyttäen erilaisia maalausmenetelmiä tik book menetelmat o fi.txt	ru краска
…	…	…	…	…

Taulukko 5-1. Esimerkki termipankin tietokannan taulukkomuotoisesta esityksestä

Tällainen esitystapa ei kuitenkaan sovellu mielestäni artikkelien esitykseen, koska se ei ole käyttäjäystävällinen. Mikäli artikkelit pitäisi esittää käyttäjäystävällisesti taulukkomuodossa, jokaiselle artikkelille pitäisi luoda oma taulukko. Tällainen lähestymistapa on kuitenkin toteutuksen kannalta ongelmallinen, koska silloin joudutaan tuottamaan paljon ylimääräistä koodia ja tällaisen toteutuksen lähdekoodin semanttisuus on kiistanalainen. Myös aikaisemmin esitetty (*4.8.1. Semanttinen merkintätapa*) ulkoasun muutosten hankaluus taulukoita käytettäessä on otettava huomioon jo senkin takia, koska artikkelien sisältöön kohdistuu useita dynaamisia toimintoja. Termipankin hakutulosten esitys on toteutettu täysin ilman taulukoita `div` ja `span` elementeillä ja ulkoisilla CSS-tyylimäärittelyillä (Kuva 5-5).

Jotta samaan tulokseen olisi mahdollista päästä taulukoiden avulla, pitäisi luopua kokonaan semanttisen määrittelyn periaatteista, koska peräkkäin seuraavat rivit olisivat eriasteisessa suhteessa toisiinsa. Myös viittaaminen artikkeleihin ja artikkeliosiin esimerkiksi dynaamisen sisällön tuottamisen yhteydessä vaikeutuu ja rajoittuu merkittävästi, koska esimerkiksi yhden sarakkeen poistaminen tai piilottaminen vaatii muutoksia koko taulukon rakenteeseen. Taulukoiden ulkoasua ei myöskään ole mahdollista muokata niin monipuolisesti kuin muita elementtejä, niiden tarkoin määritellyn muodon vuoksi, mikä puolestaan rajoittaa erilaisten työpöytien kehittämisen periaatetta.



Kuva 5-5. Artikkelien esitys termipankkisovelluksessa

Toteutuksen esitystapa ei kuitenkaan sellaisenaan mahdollista yleiskuvan luomista hakujoukosta, mikä on käytettävyyden näkökulmasta erittäin ongelmallista. Sanakirjatyyppisissä toteutuksissa tällaista ongelmaa ei ole, koska hakutulos tuottaa termilistan. Projektin termipankissa ongelma on ratkaistu dynaamisten toimintojen kautta niin, että hakutulos tuottaa termilistan, joiden näkymän voi laajentaa artikkelinäkymään yksitellen tai kaikki artikkelit kerrallaan (Kuva 5-5). Samalla tavoin hakujoukosta voidaan poistaa ne artikkelit, joita ei hakutuloksiin haluta.

5.7. Tietojen vienti termipankista

Terminologisessa työssä termipankkisovelluksen käytön lopullinen päämäärä on siirtää tutkimusaineisto järjestelmästä keinolla taikka toisella painettuun muotoon ns. paperille tai muuhun käyttöön järjestelmän ulkopuolelle. Tietojen vienti järjestelmän ulkopuolelle tulisi aina toteuttaa tarkoitukseen sopivalla tavalla halutun lopputuloksen saavuttamiseksi eli esimerkiksi, jos

tarkoituksena on muokata tietoja taulukkolaskentaohjelmassa, tietojen vienti tulisi tapahtua siihen soveltuvassa formaatissa. Tarkoitukseen sopimattomien työkalujen käyttäminen johtaa vähintäänkin ylimääräiseen työhön (kuten uudelleenmuotoiluun), mutta pahimmassa tapauksessa tuottaa täysin vääränlaisen tuloksen.

Yksittäisten artikkelien kohdalla tiedon järjestelmästä vienti on mahdollista kopioimalla tarvittava tieto suoraan sivulta ja muotoilemalla se sopivaan muotoon, mutta laajemman artikkelijoukon kohdalla tällainen toteutustapa on hyvin hidas. Mikäli tietoja on tarkoitus muokata järjestelmästä viennin jälkeen, toteutus tulisi tehdä tiedostoon vientinä, mikä vaatii omanlaisen moduulin. Projektin aikana tiedostoon vientiä ei toteutettu useasta syystä:

- Termipankkisovelluksen on tarkoitus tuottaa valmis tuote, mikä ei vaadi muokkausta järjestelmän ulkopuolella.
- Artikkelien esitystapa pyrittiin standardisoimaan mahdollisimman pitkälle yhdenmukaisen esityksen varmistamiseksi, koska standardimuotoisen esityksen vertailu muihin vastaaviin töihin on sujuvampaa ja luotettavampaa.
- Tiedostoon viennin moduuli olisi hyvin monimutkainen kokonaisuus, jota ei ole mahdollista toteuttaa käyttäjäystävällisesti osaamillani ohjelmointikielillä.

Tietojen vienti termipankista toteutettiin tulostuksen kautta (LIITE 4: Termipankista tulostettu artikkelisivu). Se ei kuitenkaan tarkoita sitä, että tulostaminen rajoittuisi vain paperille tulostamiseen, sillä tulostaminen on mahdollista myös tiedostoon esim. PDF-tulostimella. Tällainen lähestymistapa täyttää myös täysin asetetut vaatimukset esitysmuodon standardisoimiselle.

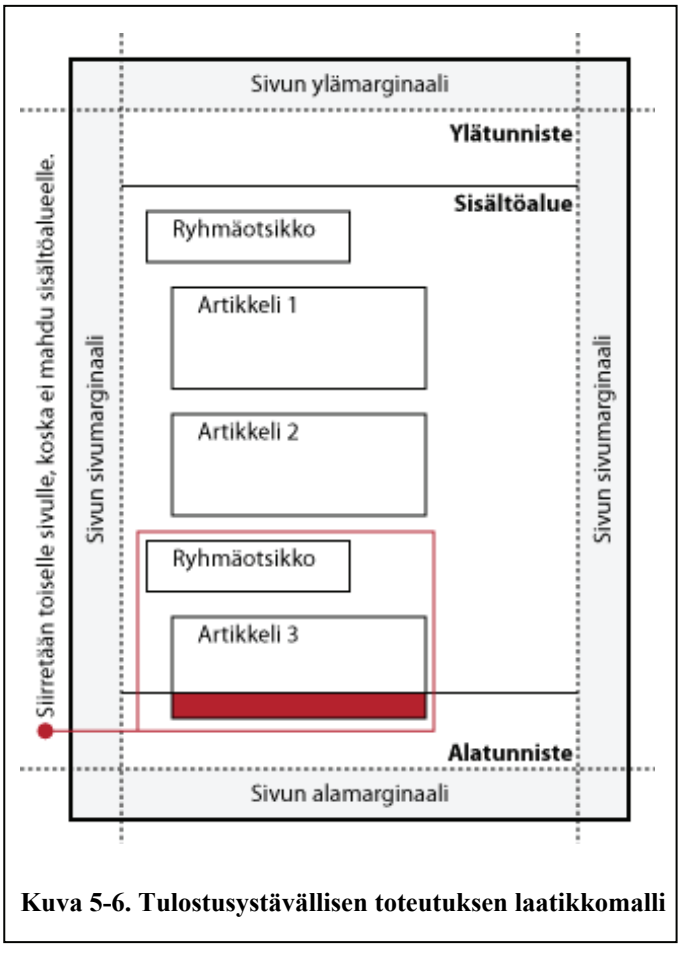
Järjestelmän suunnittelussa on siis otettava huomioon myös painettu media ja sen asettamat erityistarpeet. Päällepäin voisi luulla, ettei internetin ja painetun median välillä ole merkittävää eroa, mutta todellisuudessa internet eroaa mediana merkittävästi painetusta mediasta. Esimerkiksi monia korostuskeinoja kuten alleviivausta, lihavointia, kursivointia ja taustakuvia käytetään aivan toisiin tarkoituksiin tai toisella tavalla näissä medioissa. Internetsivuilla on myös usein toiminnallisia ja visuaalisia elementtejä, kuten linkkejä, painonappeja, logoja ja bannereita. Suurimmat erot muodostuvat kuitenkin siitä, että internet on interaktiivinen ja painettu media ei.

Sovelluskehittäjät ovat tottuneet muokkaamaan käyttöliittymän ulkoasua, mutta muiden medioiden huomioiminen harvoin otetaan mukaan suunnitteluun, ja mikäli otetaankin, sovelluskehittäjien ajattelu usein rajoittuu vain dokumentin tuottamiseen toisessa muodossa. Tällainen erillisten samansisältöisten dokumenttien ylläpitäminen ja päivittäminen on hyvin hankalaa. (Meyer, 2002.) Ulkoasun suunnittelu painetulle medialle on mahdollista toteuttaa CSS2-spesifikaatioiden (W3C:

CSS2.1 Specification, 2009) mukaisesti käyttäjäystävällisesti käyttämällä eri medioille tarkoitettuja tyylimäärityksiä samalla tavoin kuten käyttöliittymän suunnittelussakin, joten toteutustapa oli luonnollinen jatke projektin muulle sovelluskehitykselle.

Vaikka tarpeelliset CSS-määrittelyt olivatkin CSS2-spesifikaatioiden mukaisia, yleisimmissä selaimissa tulostusta koskevien tyylimäärittelyjen tuki havaittiin puutteelliseksi (mm. marginaalit: `@page`, sivunvaihdon esto: `page-break-inside: avoid`). Näin ollen alkuperäinen suunnitelma tulostuksesta suoraan sivulta ilman ”tulostusystävällinen versio” -painiketta osoittautui toimimattomaksi ja jouduttiin turvautumaan melko radikaaleihin ratkaisuihin tavoitteiden saavuttamiseksi, koska asia huomattiin vasta toteutuksen loppuvaiheessa.

Täysin CSS-ohjattujen ylä- ja alatunnisteiden luominen ei ole tällä hetkellä mahdollista ja sen vuoksi osittaisen ratkaisun ongelmaan tarjoavat taulukot (Wilkinson, 2004). Verkkosivun esitys taulukointiin kolmeen osaan (ylätunniste, sisältö, alatunniste), mikä on epäsemanttinen ratkaisu ja on vahvasti ristiriidassa alkuperäisen suunnitelman (toteutus ilman taulukoita) kanssa. Tällä tavoin saatiin kuitenkin käyttöön taulukoiden `thead` ja `tfoot` -määritykset, jotka ovat laajasti tuettuja suosituimmissa selaimissa ja sitä kautta mahdollistavat ylä- ja alatunnisteiden määrittelyn.



Kuva 5-6. Tulostusystävällisen toteutuksen laatikkomalli

Suurin ongelma oli kuitenkin se, että sivun vaihdon esto ei ole mahdollinen nykyisissä suosituimmissa selaimissa (Wilkinson, 2003). Yksittäiset artikkelit taas haluttiin pitää kokonaisuudessaan yhdellä sivulla ja niiden osien siirtyminen sivunvaihdon yhteydessä toiselle sivulle estää. Tällainen ratkaisu on erittäin tärkeä esityksen selkeyden säilyttämiseksi ja sen vuoksi jouduttiin luomaan javaScript-käsittelijä sivun tulostusystävällisen version luomista varten. JavaScript-moduuli laskee elementtien mitat ja määrittelee siten elementtien sopivuuden sivulle laatikkomallin (eng. box model) avulla (Kuva 5-6). Toteutuksessa sidotaan myös mahdolliset artikkelien ryhmäotsikot seuraavaan artikkeliin, jolloin sivunvaihdon yhteydessä artikkelia mahdollisesti edeltävä

otsikko siirtyy mukana. Ratkaisu ei ole täydellinen ja sen toteutus on hyvin kiinteä, mikä tarkoittaa sitä, että tulostaminen käyttäen ”tulostusystävällistä ratkaisua” on mahdollista ilman muotoiluongelmia vain ennalta määritellylle paperikoolle. Luotu laatikkomalli vastaa toimintaperiaatteelta CSS ja HTML/XHTML -spesifikaatioissa käytettäviä laatikkomalleja, vaikka sen toiminta on yksinkertaistettu kyseiseen tarkoitukseen sopivaksi.

Kaikkia suunnitelman mukaisia toimintoja ei kuitenkaan ollut mahdollista toteuttaa erikoisratkaisuillakaan. Tärkein näistä toiminnoista on marginaalien määrittely. Koska sovelluskehittäjillä ei ole mahdollisuutta vaikuttaa tulostettavan dokumentin marginaaleihin, se mahdollistaa vain oletusmarginaaleilla tulostamisen. Alkuperäisen suunnitelman mukaiset toiminnallisuudet ovat kuitenkin olemassa ja selaintuen parantuessa erikoisratkaisut on mahdollista ottaa kokonaan pois käyttöliittymästä.

5.8. Ohjeet

Sovelluskehityksessä tulisi aina pyrkiä sellaiseen intuitiiviseen ja helppokäyttöiseen toteutukseen, mikä ei tarvitse ohjeita, koska ohjeiden lukeminen vie käyttäjien aikaa varsinaiselta toiminnalta. Olisi kuitenkin itsepetosta luulla, ettei ohjeita tarvita, jo pelkästään sen vuoksi, koska ihmiset ajattelevat ja käsittävät ympärillä tapahtuvia asioita erilailla. Myös käyttäjien tietotaito vaihtelee ja sen vuoksi sovelluksen liiallinen yksinkertaistaminen on edistyneimpien käyttäjien aliarviointia. Ohjeet ovat siis sovelluksen kannalta tärkeä aputoiminto, jota ei ihannetapauksessa tarvita. Sovelluksen käyttöohjeiden tarkoituksena on:

- tarjota ratkaisuja käyttäjän ongelmiin
- varmistaa sovelluksen oikeanlaisen käytön
- tarjota käyttäjille mahdollisuuden sovelluksen monipuolisempaan käyttöön

Loppukäyttäjille suunnatut ohjeet voidaan toteuttaa pääsääntöisesti kahdella tavalla:

Toimintojen yhteyteen liitetyt ohjeet tarjoavat nopeasti mahdollisia ratkaisuja käyttäjän ongelmiin ja siten auttavat hyvin suunniteltuina sovelluksen käytössä. Yhtenä ratkaisuna tällaiseen toteutukseen voisi olla toiminnon yhteydessä oleva ohje-painike, josta avautuu vinkkikehys (eng. tooltip) ja välitön virheraportointi toimintoa suorittaessa. Tällaisen toteutuksen huonot puolet syntyvät sen vahvuuksista, sillä tiivismuotoinen sisältö ei mahdollista toiminnon laajempaa kuvausta, eikä siinä ole mahdollista ottaa huomioon laajempaa kokonaisuutta. Myös se päätös siitä, mitä ohjeita toimintojen yhteyteen annetaan, on hyvin vastuullinen, sillä tässä usein joudutaan tekemään kompromisseja. Pelkästään vinkkikehysten varaan jätetyt ohjeet ovat myös hajanaisia, jolloin käyttäjän on vaikea löytää ratkaisua mahdollisiin laajempiin ongelmiin. Toteutustavan

käyttöfunktio on siis minusta hyvin rajoittunut ja ohjeiden jättäminen pelkästään vinkkikehysten varaan on harvoin perusteltua.

Erillinen ohjekirja ei puolestaan tarjoa nopeaa ongelmanratkaisua ja vaatii käyttäjältä tietoista siirtymistä ohjeisiin, mutta toisaalta tällainen lähestymistapa poistaa ne ongelmat joihin törmätään toimintojen yhteyteen liitetyissä ohjeissa. Jatkuvasti muuttuvassa sovellusympäristössä yhtenäisen ohjetiedoston päivittäminen on myös paljon helpompaa, kuin hajautettujen toimintosidonnaisten ohjeiden ajan tasalla pitäminen.

Termipankin toteutuksessa ei toteutettu laajempaa toimintojen liitettyä ohjeistusta, mutta väärin toteutetuista toiminnoista käyttäjille annetaan mahdollisuuksien mukaan välitön palaute virheraportoinnin kautta, jolloin käyttäjä voi korjata toimintaansa oikeaksi ja jatkaa työskentelyä mahdollisimman sujuvasti. Pääasiallisesti ohjeet toteutettiin kuitenkin omalle sivulle käyttöohjekirjatyyppisesti, koska sovelluksen toiminnallisia ominaisuuksia on toistaiseksi melko vähän ja toimintojen yhteyteen liitettyjen ohjeiden tarve katsottiin tässä vaiheessa pieneksi.

Ohjeet kirjoitettiin aina käyttäjän kannalta tärkeiden moduulien valmistumisen yhteydessä ja julkaisun yhteydessä uudet ohjeet päivitettiin olemassa oleviin vastaamaan sovelluksen sen hetkistä toiminnallista tilaa. Tällä tavoin oli mahdollista pitää ohjeet ajan tasalla pienillä yksittäisillä muutoksilla verrattuna koko käyttöohjeen tekoon kerralla.

Ohjeiden toteutustavat eivät sulje toisiaan, vaan päinvastoin täydentävät toisiaan ja on perusteltua varsinkin vaikeampien toimintojen kohdalla tarjota välittömiä ohjeita ikään kuin muistinvirkistykseksi. Sen vuoksi tulevaisuudessa toimintojen kasvaessa olisi luontevaa tarjota käyttäjille molemmat ohjevaihtoehdot samanaikaisesti erillisten moduulien kautta.

5.9. Lokalisointi

Sovelluksen kieli määrittelee pitkälti potentiaalisen käyttäjäkunnan, sillä käyttäjälle vieraalla kielellä toteutettua sovellusta on hyvin vaikea käyttää. Yhtenä vaihtoehtona on toteuttaa sovellus yhdellä valtakielistä (LIITE 2: Internetin käytetyimmät kielet), jonka osaamisprosentti olisi ennakoidusta käyttäjäryhmästä suurin. Projektin termipankin tulevien käyttäjien arvioitiin osaavaan melko hyvin englantia, joten se olisi luonnollinen valinta sovelluksen kieleksi kattaen 29,1 % kaikista internetkäyttäjistä. Tällöin loput 70,9 % internetin käyttäjistä jäisi kuitenkin automaattisesti järjestelmän ulkopuolelle.

Monikielisellä käyttöliittymällä voidaan toteuttaa käyttäjien tarkempaa rajausta käyttöliittymän kielen perusteella tarjoamalla senkielisiä käyttöliittymiä, joita käyttäjät tarvitsevat. Sovelluksen

lokalisointi on mahdollista toteuttaa eri tavoin, mutta lokalisoinnin tärkein perusperiaate on Microsoftin sovelluskehitysohjeiden mukaisesti jakaa sovellus kahteen lohkoon: lähdekoodilohkoon ja sisältölohkoon. Tällöin lokalisointi voidaan kohdistaa vain sisältöön puuttumatta lähdekoodiin. Tällainen lähestymistapa mahdollistaa myös sisällön keskittämisen yhteen paikkaan. (Esselink, 2000.) Valittu toimintatapa, jossa lokalisoidaan vain sisältö, ei puutu sovelluksen ulkoasuelementteihin, joten erikieliset osiot ovat esitysasultaan identtisiä keskenään. Tällainen lähestymistapa on rajoittava, mutta laajempi ns. täydellinen lokalisointi on resurssivaatimuksiltaan aivan toista luokkaa ja yleensä tarkoittaa kokonaan uuden projektin perustamista (vertaa: esim. eri maiden McDonalds -sivut: http://www.mcdonalds.com, http://www.mcdonalds.fi jne.).

Lokalisointiprosessi jakaantuu aina kahteen osioon: sovelluskehitykseen, jossa luodaan lokalisointimekanismit ja mahdollistetaan lokalisointitiedostojen luonti ja varsinaiseen lokalisointiin, jossa käyttöliittymä käännetään toiselle kielelle. Projektin lokalisointiratkaisuksi valittiin GNU työkaluihin kuuluva Gettext. Gettext on muodostunut standardiksi avoimen lähdekoodin sovelluskehityksessä ja sitä käytetään hyvin laajasti mm. linux-ohjelmien monikielisten tekstiosien hallintaan. Gettext on myös PHP:n sisäänrakennettu lisäosa, joten sen valinta lokalisointiratkaisuksi oli luonnollinen, koska se ei vaadi erityisohjelmia.

Gettext mahdollistaa lokalisointimekanismin yksinkertaisen toteuttamisen sovellukseen ja mikäli lokalisointimekanismi otetaan huomioon sovelluksen suunnittelun alkuvaiheessa, sen toteuttaminen ei vaadi suuria resurssiuhrauksia. Käytännössä mekanismi on toteutettu niin, että lähdekoodissa on vain viittaukset tekstiosiin (Listaus 5-2) ja erillinen moduuli, jolla voidaan ladata haluttu erillinen lokalisointitiedosto.

```
<label for="style"><?PHP echo _("Valitse sivun ulkoasu"); ?>:</label>
```

Listaus 5-2. Lokalisointimekanismin toteuttaminen Gettext työkalun avulla

Esimerkin (Listaus 5-2) mukaisesti `_("")` on Gettext-komento `ja Valitse sivun ulkoasu` on yksilöllinen viittaus tekstiosaan, eikä itse teksti, vaikka se siltä näyttäisikin. Todellisuudessa viittaus voisi olla vaikkapa `_("styleselect")` tai `_("string10293")` ja silti se tuottaisi saman lopputuloksen, koska teksti ladataan lokalisointitiedostosta. Viittaukset muistuttavat kohdan tekstiä kahdesta syystä:

- itsedokumentoinnin vuoksi, sillä näin sovelluskehittäjien on helpompi lukea lähdekoodia
- varmistuksena, sillä mikäli lokalisointitiedostoa ei ole jostain syystä mahdollista ladata, käyttöliittymään ladataan oletusteksti, joka on viittauksen sisältö.

Lokalisointimekanismi ei kuitenkaan voi rajoittua pelkästään tekstiin, sillä lokalisointi ei ole kääntämistä. Sovelluksissa on usein monia osia, joiden esitysasu on kulttuurisidonnainen. Tällaisia osia ovat päivämäärien, lukujen yms. esitys, värit ja kuvat. Lokalisoinnin kannalta erityistä huomiota tulisi kiinnittää kuviin ja väreihin, koska niitä on vaikeampi hallita kuin tekstiä. Järjestelmässä on pyrittävä käyttämään vain yleisluontoisia kuvia, jotka sopivat eri kulttuuriympäristöihin ja välttämään tekstiä sisältävien kuvien käyttöä. Värien kohdalla pitää taas välttää merkityksiä sisältäviä tai kantaaottavia (puolue, valtion lippu jne.) väriyhdistelmiä. (Esselink, 2000.) Mikäli kieli- tai maakohtaisia kuvia on pakko käyttää, ne tulisi merkitä järjestelmässä selvästi ja yhdenmukaisesti viittaamaan johonkin kieleen, jolloin erillisen moduulin kautta on mahdollista toteuttaa lokalisoidun kuvan lataus (Listaus 5-3). Tässä esimerkissä `<?PHP echo IMAGES . $triggers['locale'] . '_header.png'; ?>` tuottaa `public_html/images/fi_FI_header.png` linkin lokalisoituun kuvaan. Näin ollen voidaan luoda esim. `ru_RU_header.png`, `en_US_header.png` jne. kuvia jotka käyttävät standardinmukaisia lokalisointilyhenteitä.

```
<img class="localeHeader" title="<?PHP echo _("Tervetuloa"); ?>" alt="<?PHP echo
_("Tervetuloa"); ?>" height="40" width="100" src="<?PHP echo IMAGES .
$triggers['locale'] . '_header.png'; ?>" />
```

Listaus 5-3. Esimerkki lokalisoidun kuvan lataamisesta

Lokalisointimekanismin ollessa kunnossa, varsinaisen lokalisoinnin voi hoitaa muutkin kuin sovelluskehittäjät, koska varsinainen tekstien kääntäminen tapahtuu kokonaan irrallaan järjestelmästä. Antamalla kääntäjille jonkun muunkielisen valmiin lokalisointipaketin kansiosta `/termtool/locale` he voivat luoda uusia kielipaketteja järjestelmään, jotka myöhemmin integroidaan sovelluskehittäjien toimesta. Kielipaketin rakenne on tarkoin määritelty ja sisältää aina kaksi tiedostoa:

- `<kieli_MURRE>/LC_MESSAGES/messages.mo`
- `<kieli_MURRE>/LC_MESSAGES/messages.po`

`kieli_MURRE` on kielen yksilöivä kansion nimi, jonka avulla on mahdollista erottaa mm. `en_US` (Pohjois-Amerikan englanti), `en_GB` (Iso-Britannian englanti), `en_CA` (Kanadan englanti) jne.

Kielipakettien tekemiseen on olemassa useita apuohjelmia ja projektin aikana lokalisointitiedoston luomiseen käytettiin poEdit (http://www.poedit.net/) apuohjelmaa sen sisältämien ominaisuuksien vuoksi. Ohjelmassa on sisäänrakennettu yksinkertainen käännösmuisti ja se osaa seurata projektikansiossa tapahtuneita muutoksia, jolloin muutokset sovelluksessa heijastuvat

automaattisesti myös kielipaketteihin. Tällä tavoin kielipaketteja on helpompi päivittää. Ohjelma osaa huolehtia myös Gettextin vaatimasta tarkasta tiedoston rakenteesta, jolloin kääntäjä voi keskittyä olennaiseen eli kääntämiseen, minkä vuoksi kielipakettien kääntäminen ei vaadi syvällistä tietoteknistä osaamista.

Sisällön ja lähdekoodin erottaminen ei kuitenkaan onnistunut täysin Gettextin rajoitusten ja HTML-tiedoston rakentumisen vuoksi. Esimerkiksi yksittäisen tekstiobjektin osia joudutaan usein korostamaan lihavoinnilla, kursiivilla, värillä jne. ja koska tällaisen toiminnan aikaansaamiseksi tarvitaan HTML-tageja, ne jouduttiin upottamaan joskus tekstiin. Tällaista toimintaa pyrittiin välttämään, mutta usein ei ole mielekästä katkasta lausetta useampaan osaan, sillä kääntäjän on vaikea kääntää sellaista tekstiobjektia, johtuen mm. kielten toisistaan eroavista lauserakenteista ja tekstiobjektin sisällön hahmottamisen vaikeudesta. Esimerkiksi tuottaakseen tekstin: ”Kaikki **suomenkieliset** artikkelit, joiden *valmiusaste* on **0-2**” vaaditaan seuraavanlainen tekstiobjekti: `Kaikki <strong>suomenkieliset</strong> artikkelit, joiden <em>valmiusaste</em> on <strong>0-2</strong>`, joka välittyy sellaisenaan kielipakettiin. Tämän vuoksi kääntäjien on tiedettävä joitakin perusasioita HTML-merkintätavasta, mutta toisaalta se mahdollistaa yksittäisten elementtien tarkemman kielikohtaisen muokkauksen.

6. LOPUKSI

Katson projektin onnistuneeksi, sillä projektille asetetut tavoitteet joustavan sovellusrungon ja monipuolisten hakuominaisuuksien toteuttamiselle saavutettiin täysin ja osittain jopa ylitettiin. Vähäisen käyttäjäpalautteen perusteella, myös käytettävyyden näkökulmasta sovellus on mielestäni osoittautunut helppokäyttöiseksi sen monipuolisuuden siitä kärsimättä. Projektin ainoa epäonnistunut toiminto on mielestäni tietojen järjestelmästä vienti ja se vaatii jatkokehitystä, mikäli selaintuki ei parane lähitulevaisuudessa.

Projektille asetetut aikataulut eivät kuitenkaan pitäneet ja ylittyivät monenkertaisesti. Tämä johtui pääosin toteutuksen vaativuuden aliarvioinnista projektin alkuvaiheessa ja sovelluksen muutoksista, joita jouduttiin tekemään projektin aikana. Liiallinen optimismi ja kokemattomuus vastaavien järjestelmien toteutuksesta myös näkyivät projektin rajaamisen vaikeudessa projektin alkupuolella, mistä johtuen tehtiin paljon ”turhaa” työtä. Toisaalta tästä ns. turhasta työstä oli myös paljon hyötyä projektin toteutuksessa ja jatkokehityksen määrittelyssä, sillä kuten Thomas A. Edison oli sanonut: ”*En ole epäonnistunut. Olen vain löytänyt 10000 keinoa, jotka eivät toimi.*”.

Resurssien puutteen vuoksi termipankin toiminnallisen kaavion (Kuva 5-1) mukainen toteutus jäi kuitenkin kesken. Keskenjääminen ei kuitenkaan tarkoita toimimatonta tuotetta, vaan termipankki toimii ja sitä käytetään aktiivisesti terminologisen työn apuvälineenä. Keskenjääminen tulisikin käsittää enemmän sovelluksen jatkokehityksen tarpeena.

Jatkokehityksen suurin tarve kohdistuu toiminnallisen kaavion osille, jotka jäivät toteutuksen ulkopuolelle, koska nämä osat voidaan laskea kriittisiksi kokonaisuuden kannalta. Laajin jatkokehityskokonaisuus on siis teknisissä osissa, eikä rajoitu pelkästään toiminnalliseen kaavioon. Yksi isoimmista kokonaisuuksista, jota ei suunniteltu projektin puitteissa, on käsitekaavioiden integrointi järjestelmään, mikä on hyvin haastava ja varmasti aikaisemmin toteuttamaton kokonaisuus siinä muodossa kuin se on esitetty tässä työssä. Sovelluksen toiminnan optimointi erilaisten testien kautta ja käyttäjien tarpeiden mukainen lisätoimintojen kehitys mm. käyttäjäpalautteen perusteella on sovelluskehityksessä osa tuotteen luonnollista elinkaarta. Ei voida ajatella, ettei sovellus tarvitsisi ylläpitoa ja kehitystä jatkuvasti muuttuvassa sovelluskehityksen maailmassa, vaan sitä tulisi soveltaa ja parantaa vastaamaan uusia kehitysmenetelmiä ja suosituksia, sillä jatkuva kehitys ja ylläpito pidentävät tuotteen elinkaarta merkittävästi.

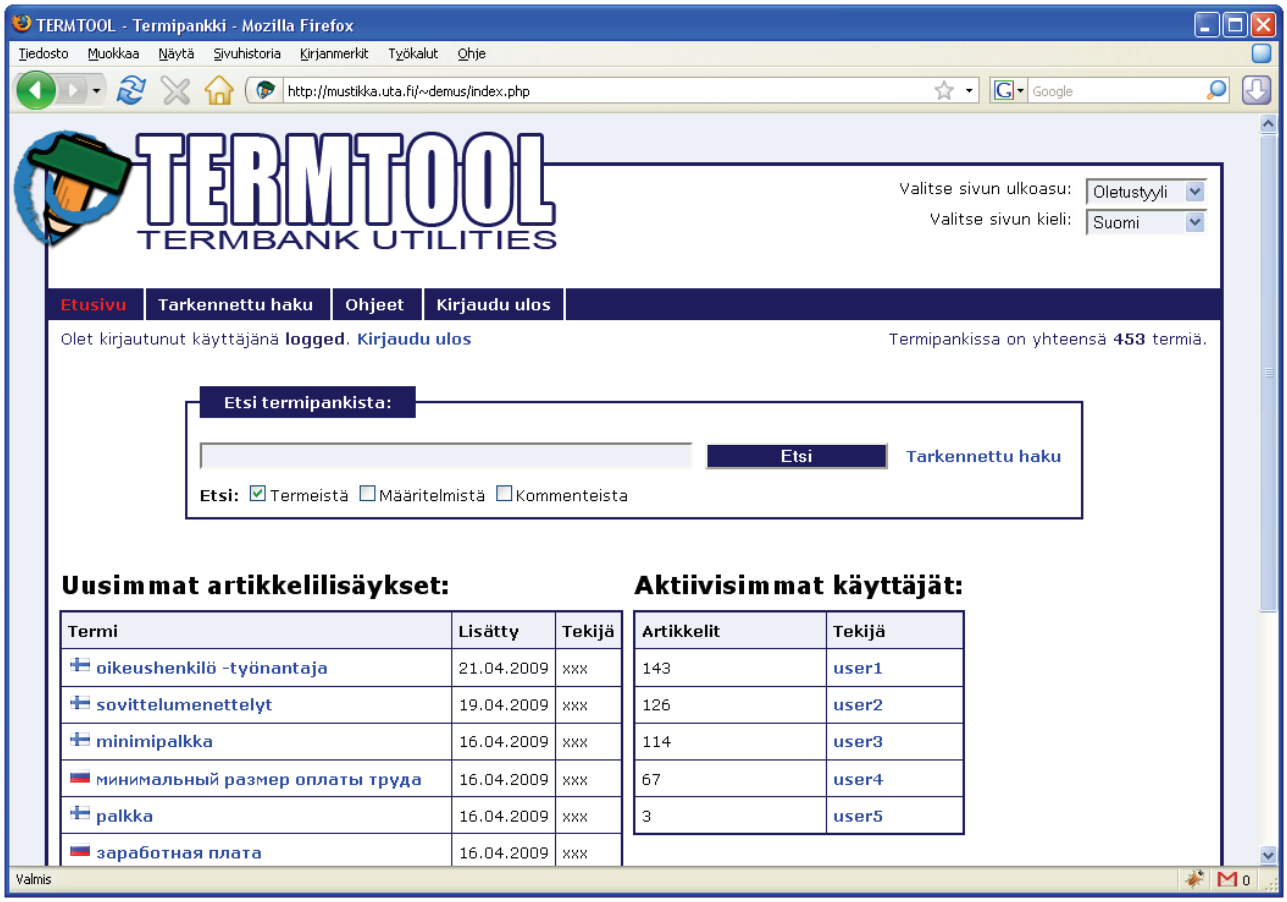
Tarvetta on myös laajemmalle käytettävyystestaukselle ja esimerkiksi erikielisten lokalisointipakettien luomiselle, joten jatkokehityksen tarve ei rajoitu pelkästään teknisiin osiin. Myös sovelluksen toimintalogiikan optimointi käyttäjäystävällisemmäksi tutkimalla käyttäjien käyttäytymistä olisi hyvin tärkeää sovelluksen toimintojen lisääntyessä. Kaikessa jatkokehityksessä pitää kuitenkin huomioida termipankin pääkäyttötarkoitus, joka on terminologinen työ ja siten asettaa useita omanlaisia vaatimuksia sovelluksen toiminnalle. Tämä ei tietenkään sulje pois muunlaisten moduulien kehitystä, mutta suotavaa on, että toimintojen määrityksiin ja muutospäätöksiin osallistuisi myös terminologeja tai vähintäänkin hyvät perustiedot terminologisesta työstä omaavia henkilöitä.

Jatkokehityksen tulisi tapahtua käyttäjien tarpeet huomioon ottaen ja toteutuksen laatu ensisijaisena tavoitteena. Sovelluksen joustava arkkitehtuuri mahdollistaa omalta osaltaan projektien rajaamisen hyvinkin pieniksi ja siten projektit on mahdollista sovittaa vastaamaan resursseja laadun siitä kärsimättä. Ei koskaan voida liikaa korostaa, että laadukas sovellus kannustaa käyttäjiä käyttämään sitä ja heikkolaatuinen pelottaa käyttäjät pois. Termipankin käyttöarvo muodostuu taas sen sisällön laadusta ja laajuudesta, joita luovat juuri sen käyttäjät.

LÄHTEET

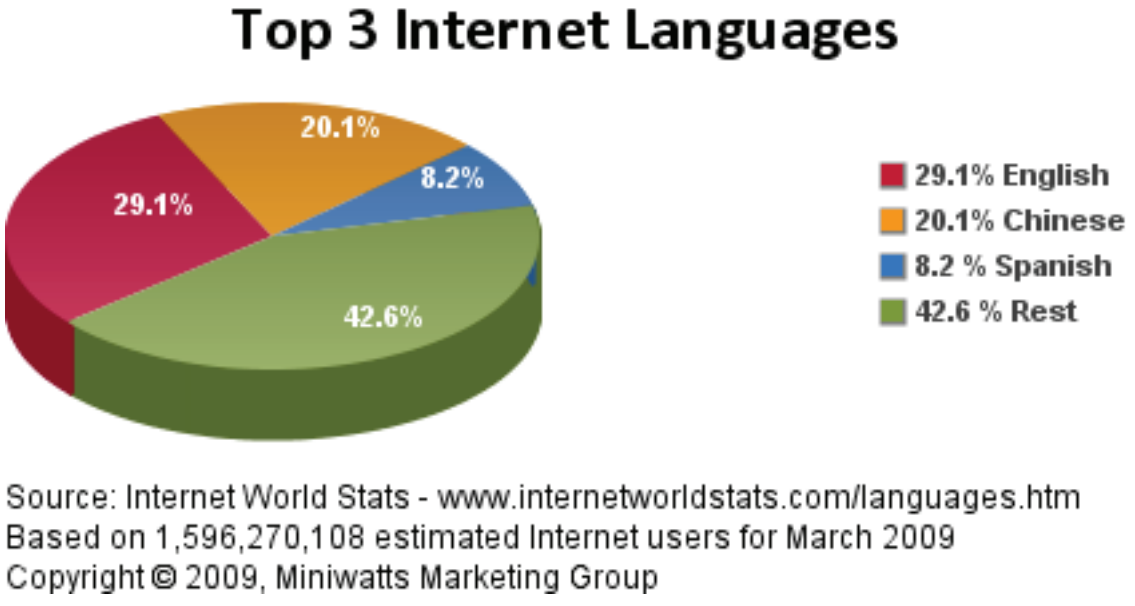
1. Esselink B: The Practical Guide to Localization, 2000, John Benjamins Publishing Co
2. Goodliffe P: Code Craft – The Practice Of Writing Excellent Code, 2007, No Starch Press
3. Honkala J: Wikipedia kaupallisen tietosanakirjan haastajana, Artikkelikokoelma: Wikipedia – monta näkökulmaa avoimeen tietosanakirjaan, Tieteessä tapahtuu 7/2007, Saatavilla: http://ojs.tsv.fi/index.php/tt/article/viewFile/319/282
4. Huttunen J: Ketterän ohjelmistokehitysmenetelmän määrittely, vertailu ja käyttäjäkysely, 2006, Diplomityö, Helsingin teknillinen korkeakoulu, Saatavilla: http://lib.tkk.fi/Dipl/2007/urn007665.pdf
5. Kalliokuusi V, Seppälä K: Vastinetyö sanastoprojektissa, 1999, Toimikunnista termitalkoisiin – 25 vuotta sanastotyön asiantuntemusta, Sanastokeskus TSK
6. Kennedy B, Musciano C: HTML & XHTML: The Definitive Guide, 6th Edition, 2006, O'Reilly
7. Korpela J: Datatekniikka ja viestintä, viim. päivitys 2007, http://www.cs.tut.fi/~jkorpela
8. Kroll P: Transitioning from waterfall to iterative development, 2004, http://www.ibm.com/developerworks/rational/library/4243.html
9. Kudashev I: Proektirovanie perevodčeskih slovarej special'noj leksiki, 2007, Helsingin yliopisto, Saatavilla: http://urn.fi/URN:ISBN:978-952-10-4037-5
10. Lahtonen T, Ekonoja A, Mäntylä J & Heinonen P: Henkilökohtaisen tiedonhallinnan perusteet, 2003, http://appro.mit.jyu.fi/doc/tiedonhallinta/
11. Lynch P, Horton S: Web Style Guide: Basic Design Principles for Creating Web Sites, 3rd edition, 2009, Yale University Press, Saatavilla: http://www.webstyleguide.com/wsg3/index.html
12. Meyer E: CSS Design: Going to Print, 2002, A List Apart, http://www.alistapart.com/articles/goingtoprint/
13. Nielsen J: The Need for Web Design Standards, 2004, http://www.useit.com/alertbox/20040913.html
14. Nielsen J: The Web Usage Paradox: Why Do People Use Something This Bad?, 1998, http://www.useit.com/alertbox/980809.html
15. Sanastokeskus TSK, http://www.tsk.fi
16. Sinkkonen I: Käyttöliittymät ja käytettävyys, 2006, Adage Usability, http://www.adage.fi/julkaisut/arkisto/kayttoliittymat_ja_kaytettavyys.HTML
17. Skinner J: Simplify Ajax development with jQuery, 2007, http://www.ibm.com/developerworks/library/x-ajaxjquery.html
18. Terminologian sanasto TSK 36, 2006, Sanastokeskus TSK, saatavilla: http://www.tsk.fi/tiedostot/pdf/TerminologianSanasto.pdf
19. Vapaa tietosanakirja Wikipedia: http://www.wikipedia.org
20. W3C: Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification, 2009, http://www.w3.org/TR/CSS2/
21. W3C: HTML 4.01 Specification, 1999, http://www.w3.org/TR/1999/REC-html401-19991224/
22. W3C: XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition), 2002, http://www.w3.org/TR/xhtml1/
23. Wilkinson J, Printing Page Headers and Footers, CSS-Discuss Wiki, 2004 (viimeisin muutos 2005), http://css-discuss.incutio.com/?page=PrintingHeaders
24. Wilkinson J, Printing Web documents and CSS, CSS-Discuss Wiki, 2003 (viimeisin muutos 2007), http://css-discuss.incutio.com/?page=PrintStylesheets

LIITE 1: TERMTOOL – termipankki



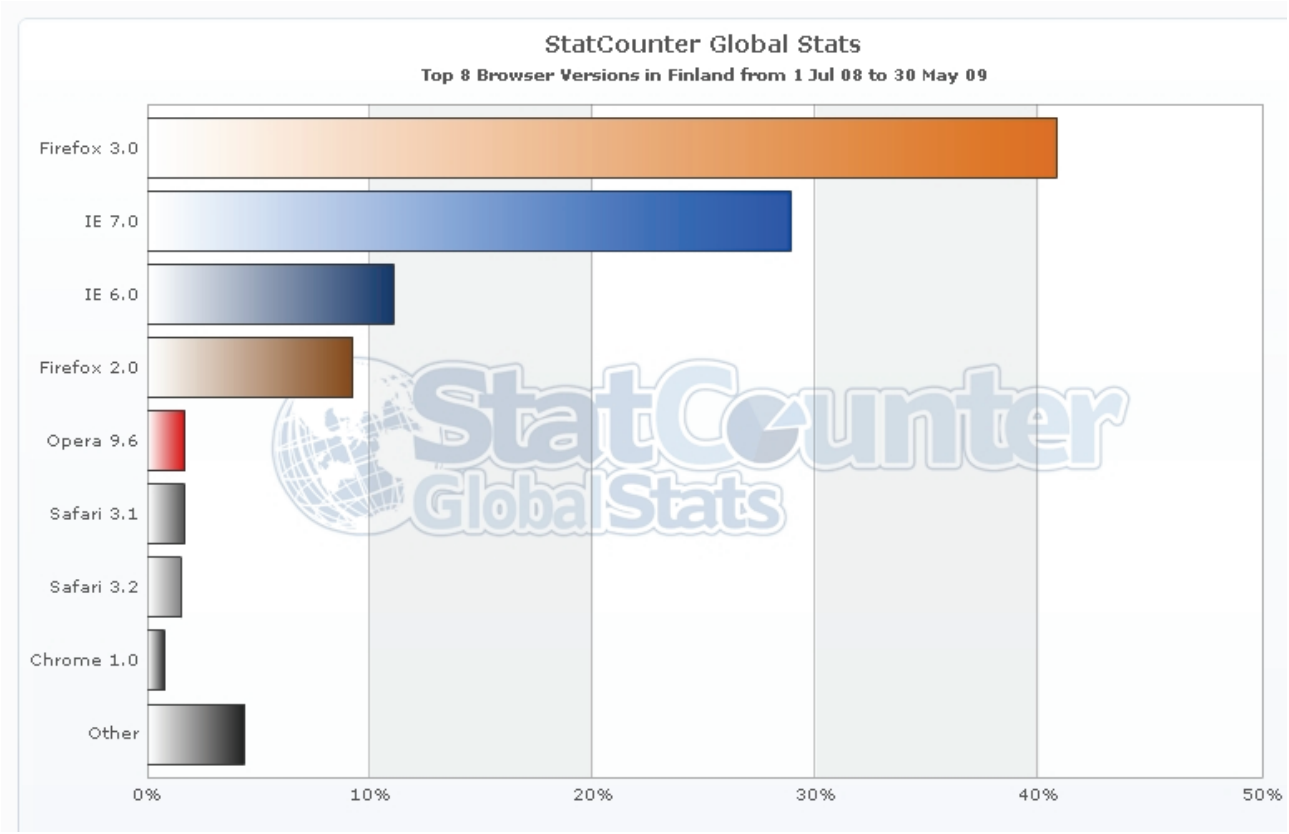
Kuva L1-1: Termipankin aloitussivu sisäänkirjautumisen jälkeen oletusasetuksilla (16.6.2009)

LIITE 2: Internetin käytetyimmät kielet

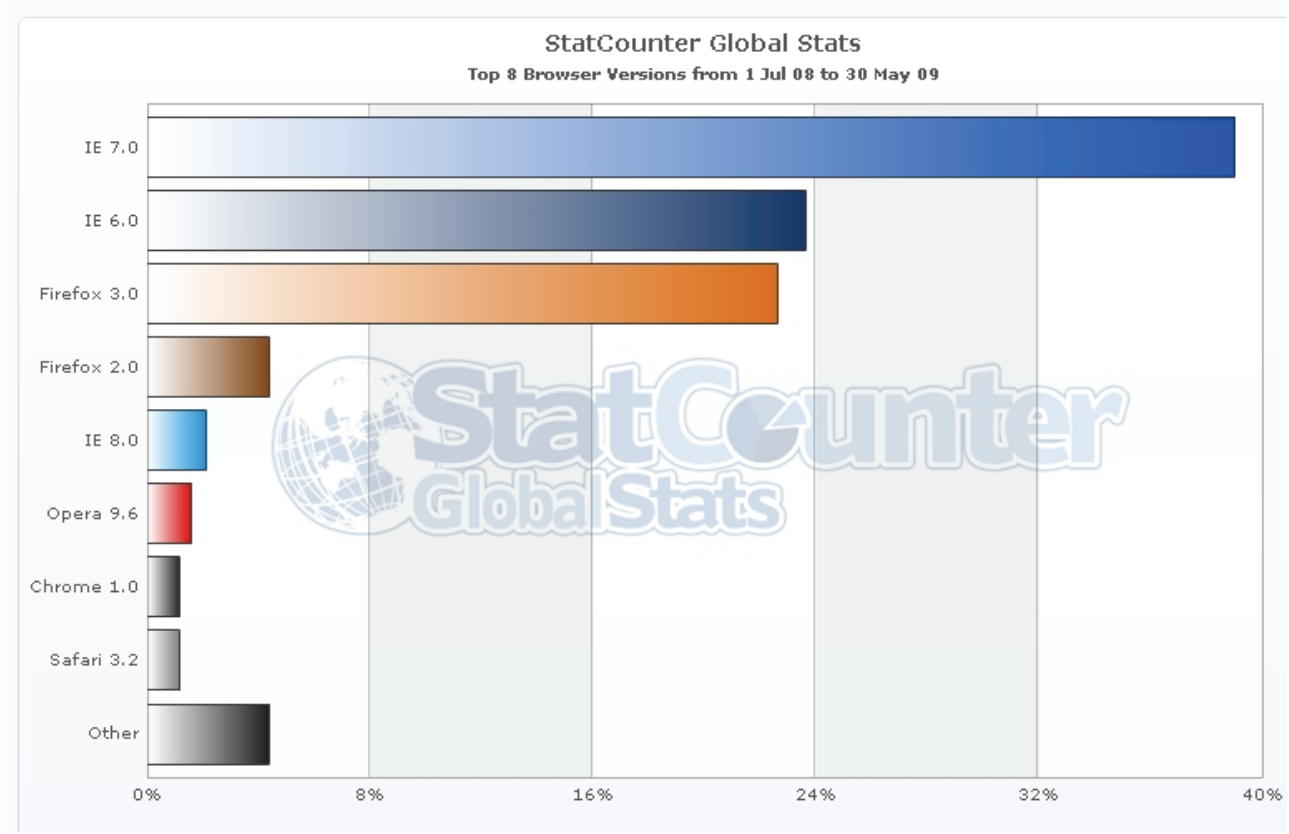


Kuva L2-1: Internetin kolme käytetyintä kieltä 3.2009 (http://www.internetworldstats.com)

LIITE 3: Selainten markkina-asemat Suomessa ja maailmalla



Kuva L3-1: Selainten markkina-asemat Suomessa 1.7.2008–30.5.2009 (http://gs.statcounter.com)



Kuva L3-2: Selainten markkina-asemat maailmalla 1.7.2008–30.5.2009 (http://gs.statcounter.com)

TERMTOOL

Kieli- ja käännöstieteiden laitos, Tampereen Yliopisto

halkeilu

fi **halkeilu**

Luokka:
Tekniikka/Техника

Määritelmä:
alustan muodonmuutokset aiheuttavat maalikaivoon venytystä ja taivutusta. Jos maalikalvo ei ole riittävän elastinen, niin se halkeilee.

Esimerkit:
1. Kun lämpötila vaihtelee, jatkuva laajeneminen ja kutistuminen heikentää maalikalvon joustavuutta ja aiheuttaa sen halkeilua.
   internet

Vastineet:
ru растрескивание

hartsi

fi **akryylihartsi**

Luokka:
Tekniikka/Техника

Määritelmä:
sideainetyyppi, jota käytetään sekä liuoteohenteisissa maaleissa että lateksimaaleissa.

Esimerkit:
1. Akryylimaalin sideaineena on akryylihartsi, jota voidaan modifioida eri monomeereillä, kuten vinyylillä tai styreenillä, maalin ominaisuuksien muuntelemiseksi.
   internet

Vastineet:
ru акриловая смола

fi **alkydihartsi**

Luokka:
Tekniikka/Техника

Määritelmä:
hapettumalla kuivuva sideaine.

Esimerkit:
1. Alkydihartseja käytetään ilmakuivuvissa maaleissa, polttomaaleissa ja happokovetteisissa maaleissa.
   tik_book_kasitteet_o_fi.txt

Vastineet:
ru алкидная смола

РЕЗЮМЕ

Настоящая дипломная работа является частью проекта, задачей которого было усовершенствовать уже существующую терминологическую базу данных и создать для нее поисковый интерфейс.

Создание нового терминологического банка было обусловлено постоянно растущим интересом к терминологическим работам и исследованиям со стороны ученых и тем, что требования, установленные по отношению к программе, было невозможно, по различным причинам, удовлетворить уже существующими продуктами (см. *О других терминологических банках*). Терминологический банк был создан для исследовательских нужд Института современных языков и переводоведения Университета г. Тампере (кафедра перевода русского языка) и таким образом проект является на данный момент закрытым, но возможность открытия проекта пользователям вне университета не исключается и принята во внимание во время проведения проекта.

Терминография

Терминография тесно связана с терминологией, ее задачей является систематический сбор, анализ, описание и представление информации о специальных понятиях и их наименованиях, т.е. о терминах. Основной задачей терминографии является повышение эффективности коммуникации в какой-либо сфере.

Обычно в таком терминологическом процессе участвуют как терминологи, так и специалисты той области, которой касается работа. Таким образом, для достижения качественного результата терминологические проекты обычно проводятся путем совместной работы специалистов разных областей (Sanastokeskus TSK).

Определение терминологического банка

Терминологический банк следует рассматривать как рабочий инструмент наряду с другими вспомогательными инструментами терминологии. Терминологический банк является, по сути, словарём специальной лексики и состоит из терминологических статей. Таким образом статьи не сводятся только к терминам, а включает также определения, примеры употребления, синонимы и т.д.

Терминологический банк сильно отличается от классического "бумажного" словаря. Основные различия можно заметить в поиске нужной информации, но есть разница и в

способах обновления словарных материалов, а также и данных, которые можно использовать в статьях (звук, видео, географические карты и т.д.).

У терминологических банков можно выделить две возможные функции:

- установление отношений между разными понятиями и формирование групп понятий;
- поиск описания, синонима, эквивалента и т.д. для определённого термина или поиск термина на основе каких-либо признаков.

В терминологических исследованиях внимание заостряется именно на первой функции, так как интерес часто сосредоточен не на отдельных терминах, а на их группах. Поэтому основной функцией работы терминологического банка данного проекта стала как раз первая из вышеназванных. Вторая функция часто требуется при выполнении переводов или иных работ с текстом, когда задачей является решить проблему, возникшую из отдельной информации.

О других терминологических банках

До начала проекта на кафедре перевода русского языка уже имелся терминологический банк, в который входили база данных и элементарная программа для простого использования базы данных. Программа включала в себя аутентификацию, ввод данных и ограниченный поиск и таким образом не содержала совсем таких функций как расширенный поиск, вывод отчетов и справки. Систему было невозможно усовершенствовать, таким образом, как это было задумано, и поэтому было принято решение создать новое веб-приложение. Поскольку ресурсы проекта были ограниченны, а предыдущая программа пригодна для использования, было принято решение о развитии по возможности тех элементов, которых в системе нет, оставляя старую программу в использовании наряду с новой.

Во время проведения проекта было проведено исследование рынка терминологических банков и уже существующих продуктов. В ходе исследования было установлено, что имеющимися на рынке продуктами невозможно полностью удовлетворить установленные к создаваемой системе требования по ряду различных причин.

Все проекты были закрытыми, т.е. хотя поиск был открытым для всех пользователей, добавление и усовершенствование статей было ограничено для определённой группы пользователей. Сильное влияние ЕС заметно в количестве и тематике терминологических проектов. В основном проекты были созданы для упорядочивания разноязычной специальной терминологии какой-то отдельной сферы внутри ЕС и таким образом

включённые в проекты языки были в основном языки стран ЕС. Из найденных проектов основными можно отметить следующие: InterActive Terminology for Europe (IATE), EuroTermBank, терминологические проекты NORDTERM и в Финляндии терминологические проекты Sanastokeskus TSK и политематический словарь государственного совета Финляндии VALTER.

Также было найдено несколько программ для создания терминологических банков и управления ими. Среди них близкими по целям и задачам к нашему проекту оказались следующие веб-приложения: I-Term национального центра терминологии Дании (DANTERMcentre), WebTerm фирмы The Star Group и MultiTerm Online фирмы SDL. Основной функцией данных приложений является построение и поддержка специальных словарей для нужд перевода и поэтому их сложно использовать для исследовательских целей. Основная причина заключается в отображении статей в словарном формате и таким образом одновременная работа с группой статей сильно затруднена. Также все эти приложения являются коммерческими и содержат ограничения на количество пользователей, что ограничивает возможности проведения временных проектов.

На основе имеющейся системы и похожих проектов были определены основные задачи проекта и требования по отношению к создаваемой программе. Программа должна была быть легкой и удобной в обращении и максимально облегчить терминологическую работу пользователей.

Основы принципов построения терминологического банка

Программа для работы с терминологической базой данных была создана в виде веб-приложения. Главные преимущества веб-приложений в сравнении с традиционными программами заключаются в возможности обновления и использования приложения без отдельной установки его на компьютер пользователя, а также возможность пользования приложением вне зависимости от рабочего терминала. (Wikipedia.) Веб-приложение устанавливается на сервер, и клиенты могут работать с ним с помощью браузера со своей рабочей станции (Kennedy & Musciano, 2006).

Созданное приложение является довольно сложным продуктом, в котором используется несколько различных языков программирования расширяющие возможности приложения. Система состоит из четырёх автономных уровней (пользователь видит только интерфейс):

- реляционная база данных
- обработка информации на сервере

- обработка информации на клиенте
- интерфейс пользователя

Итеративный метод разработки

Традиционно в разработке программных продуктов грубо выделяется пять этапов (Huttunen, 2006.):

- определение
- проектирование
- конструирование
- тестирование
- ввод в эксплуатацию и поддержка

Согласно данной модели, одна стадия должна быть полностью завершена до перехода к следующей и аналогично возврат назад невозможен. Данный подход, однако, плохо подходит для разработки модульных приложений, каковым является разрабатываемое приложение, поскольку модули работают самостоятельно и таким образом их можно добавлять, изменять и убирать на всём протяжении жизненного цикла приложения.

Решение данной проблемы заключается в итеративном методе проведения проекта, который представлен в гибкой методологии разработки (eng. agile methods). Отдельная итерация включает в себя все стадии развития и итерации повторяются на протяжении всего проекта. Одна итерация обычно ограничивается по времени, и в данном проекте длина одного цикла составляла примерно две недели.

Защитное программирование

Во время написания программы у разработчиков часто возникают различные предположения по отношению работы приложения. Такие всевозможные предположения очень опасны с точки зрения эволюции приложения, поскольку даже если они на определённый момент правильные при росте или развитии программы они часто оказываются неверными или недостаточными. Главная задача защитного программирования (eng. defensive programming) -- избавиться от таких предположений. (Goodliffe, 2007.)

Согласно методу защитного программирования отдельные программные элементы проектируются как можно более автономными и в них внедряется система проверки и вывода ошибок. Таким образом, возможные ошибки можно изолировать в отдельных блоках.

Ошибки проверяются способом "белого списка" (eng. white list), которым разрешается только запланированная работа элемента.

Самодокументирующий исходный код

Документация проекта -- важный аспект с точки зрения его дальнейшей разработки. Поскольку для разработчиков исходный код может полностью описать только он сам (Goodliffe, 2007), в данном проекте документация приложения проведена методом самодокументирующего исходного кода. На практике это означает деление приложения на логические части, семантически прозрачные названия отдельных элементов и комментирование работы отдельных элементов. Хотя Гудлайфф и считает, что комментариев следует избегать путем написания семантически прозрачного кода, я считаю, что хорошие комментарии значительно повышают читаемость кода. Поэтому основная часть функций и прочих элементов содержит комментарии и описание работы.

В конструировании интерфейса у самодокументации есть ещё одна функция: построение единой презентации интерфейса вне зависимости от различий браузеров. Цели можно достичь, отделяя презентацию от информации, т.е. исходный код отображает вид информации, находящейся в нём, и порядок элементов логичен. На практике это означает использования семантических тэгов и добавление описания элементов при помощи атрибутов `id` и `class`.

Реляционная база данных

База данных -- это коллекция информации, у элементов которой есть какие-то отношения между собой (Lahtonen ym., 2003). В проекте используется система управления базами данных PostgreSQL. Эта СУБД основана на модели реляционной базы данных, которая является в настоящее время самой распространённой моделью данных.

В реляционной базе данных информация отображается таблицами (eng. tables). Таблицы состоят из строк (eng. record), а строки из ячеек (eng. field) которых на каждой строке одинаковое количество. (Lahtonen ym., 2003.) Согласно рекомендациям, у каждой строки должен быть уникальный первичный ключ (eng. primary key), который идентифицирует отдельные строки.

Структурой базы данных управляют и добавляемую информацию добавляют, изменяют, уничтожают и запрашивают при помощи запросов (eng. query) на языке SQL (Structured Query Language).

Структура базы данных терминологического банка

В систему терминологического банка входит две базы данных: база данных пользователей, которая используется для аутентификации и терминологическая база данных, в которой находятся все терминологические статьи.

Терминологическая база данных состоит из шести таблиц (terms, examples, spheres, langs, grammar и style), которые делятся на множество ячеек. Таблица terms содержит в себе основную информацию для словарной статьи, а другие таблицы -- классифицирующую информацию для статей. Единственным исключением является таблица examples, в которой хранятся примеры употребления терминов.

Проектирование и создание терминологического банка

Несмотря на итеративный метод разработки, в проекте был довольно длительный процесс предварительного определения и проектирования. Во время этого процесса были установлены те функции, которые должен содержать полноценный терминологический банк.

Хотя все установленные функции и не были реализованы в пределах этого проекта из-за ограниченных ресурсов, они были включены в процесс проектирования и поэтому описываются в данной работе. Таким образом, было возможно определить, например, их сложность. Также, жёсткое ограничение проекта в самом начале включает в себя опасность того, что некоторые детали, влияющие на дальнейшее развитие проекта, не будут приняты во внимание.

Программный каркас

Модульное приложение требует для работы средство осуществления взаимодействия между модулями. Такое средство можно назвать программным каркасом. Основная задача каркаса -- уменьшить количество исходного кода и сделать отдельные модули автономными, таким образом, что добавление или уничтожение отдельного модуля не влияет на систему в целом. Принятые решения при построении каркаса устанавливают границы и возможности будущего развития приложения, и хорошо спланированный каркас делает возможным развитие даже таких модулей, которые не были приняты во внимание в данном проекте.

Интерфейс

Как понятие, интерфейс следует рассматривать в более широком смысле, чем просто как оболочку. Поэтому проектирование интерфейса следует проводить на протяжении всего проекта, а не как отдельный процесс (Sinkkonen, 2006). Основной интерфейс был создан во

время конструирования каркаса и позже его расширяли согласно требованиям отдельных модулей во время конструирования этих модулей.

Интерфейс веб-приложения состоит из элементов для выполнения различных действий (HTML-разметка), таких как формы, кнопки и т.д., и описаний внешнего вида этих элементов (CSS-описания). Хотя обе части сильно зависят друг от друга, их следует разделять при конструировании, эти части также предполагают логическое деление отдельного процесса создания интерфейса. Элементы, позволяющие действия, добавлялись во время создания отдельного модуля, а внешний вид проектировался в конце процесса создания модуля.

Один и тот же интерфейс не всегда подходит для различных задач и представления пользователей о том, что такое хороший интерфейс сильно различаются. Разделяя элементы и их внешний вид, можно создавать различные рабочие пространства (eng. work space). В приложении это предусматривается модулем выбора стиля рабочего пространства.

Аутентификация

Поскольку данный терминологический банк является закрытым проектом, работа с приложением начинается всегда с входа в систему вводом имени пользователя и пароля и заканчивается выходом из системы.

Помимо аутентификации в системе используются уровни допуска. При помощи уровней допуска возможно предлагать различным группам пользователей разные функции, и таким образом в будущем возможно частично открыть терминологический банк для открытого пользования. В ходе проекта было установлено четыре различных уровня: 0 – Не аутентифицированный пользователь ("гость"), 1 – Аутентифицированный пользователь, 2 – Исследователь и 3 – Администратор.

Административные функции

Административные функции -- важная часть любого веб-ресурса, где требуется регистрация, но данные вопросы в рамках данного проекта не решались, часть функций реализована в другом приложении. В данные функции входят все функции, которые подразумевают поддержку и модификацию учётных записей пользователей, т.е. регистрация и добавление/изменение/уничтожение персональной или чужой учётной записи. Допуск к этим функциям соответствует уровню допуска, т.е. допуск к чужим учётным записям должен быть только у администратора (3) и т.д.

Многосторонние административные функции:

- значительно поднимают эффективность работы, как администраторов, так и пользователей, поскольку вопросы, связанные с учётной записью можно таким образом решать быстрее
- сохраняют достоверность информации, предоставляя возможность просматривать и изменять личные данные
- улучшают информационную безопасность, предоставляя возможность менять пароли

Добавление статей в базу данных

Поскольку добавление статей реализовано в предыдущем приложении терминологического банка, данные функции не создавались в рамках данного проекта. Добавление статей в базу данных возможно двумя способами:

- добавление отдельной терминологической статьи
- добавление группы статей

Первую из данных функций практичней всего создать с помощью формы ввода внутри приложения, поскольку формы ввода знакомы многим пользователям по другим веб-приложениям и Интернет-страницам. Таким образом, каждая ячейка статьи заполняется вручную, что очень медленно, особенно когда требуется ввести большое количество статей в сжатые сроки.

Решить данную проблему можно методом импорта информации из файла. Проектирование модуля обработки файлов -- довольно сложная задача. Также такая функция устанавливает определённые требования к формату специального файла со стороны пользователей, что следует принять во внимание при разработке.

Поиск статей

Обычно поиск реализуется в двух вариантах:

- простой поиск,
- расширенный поиск.

Простой поиск обеспечивает быструю и простую работу с терминологическим банком и выводит группу статей согласно заранее заданным параметрам. Простой поиск содержит только поиск строкой символов (слов) и ограничивается тремя полями: термины, толкования и комментарии. Данная функция подходит для поиска отдельных статей и логика работы напоминает работу с электронными словарями.

В исследовательских целях такой метод поиска часто оказывается недостаточным и требуется более широкая возможность ограничения запросов. Для таких нужд был создан расширенный поиск, который позволяет ограничивать запросы разнообразными критериями. Расширенный поиск состоит из специально спроектированной динамической формы, в которой можно добавлять и убирать критерии поиска в любом порядке и количестве. Форма поиска состоит из двух частей: словарного поиска и ограничителей. Деление является немного искусственным, но обязательным, поскольку для сохранения логичности и легкости использования системы некоторые связи между критериями поиска приходится ограничивать.

Экспортирование данных и отчеты

В исследованиях конечной целью при использовании терминологического банка может быть перенос нужной информации из системы в бумажный или иной вид. Вывод информации был реализован в рамках этого проекта методом вывода на печать. Следует заметить, что печать не обязательно означает печать на бумагу – печатать можно также и в файл при помощи PDF-принтера.

Согласно планированию данной функции печать была реализована через отдельное специально сконструированное описание внешнего вида печатного варианта. Данный подход оказался несовершенным из-за неполной поддержки установок для печати со стороны браузеров. Поэтому были проведены некоторые радикальные изменения построения страниц и создан отдельный модуль для создания печатной версии вывода информации. Данное решение не совсем удачно и может считаться «заплаткой» для нефункционирующего процесса, но поскольку данный недостаток был выявлен только в конце проекта, он бы потребовал значительных изменений во всей логике работы приложения. Поддержка CSS в браузерах постоянно улучшается и поэтому, я надеюсь, что «заплатку» можно будет в скором времени убрать полностью, и приложение будет работать без нее.

Справка

В развитии приложений следует всегда производить настолько легкие в обращении приложения, что они не требуют справок. Стремление всё же не означает, что справка не требуется хотя бы из-за того, что все люди рассматривают и понимают окружающий нас мир по-своему.

Справка приложения, созданная в ходе данного проекта, напоминает руководство пользователя, так как все части справки находятся в одном пункте приложения. Таким

образом, пользователям легче найти решения к более широким проблемам, возникшим во время работы. Справка пополнялась и изменялась в ходе ввода в эксплуатацию отдельных модулей. Таким подходом было легче поддерживать справку в соответствии имеющимся функциям.

Заключение

Проект был на мой взгляд удачным, поскольку все поставленные в ходе проекта задачи были выполнены. Единственный недостаток созданной системы заключается в создании отчетов, это потребует к себе внимания на дальнейших стадиях развития приложения.

Из-за ограниченных ресурсов многие функции не были включены в план развития приложения, причем часть из них можно, с точки зрения работы всей системы, считать критичными. Поэтому нужда в дальнейшей разработке терминологического банка обширная и включает в себя помимо технической разработки множество других вопросов, связанных с различными сферами. В данном проекте была принята во внимание с самого начала дальнейшая разработка системы, поэтому задачи последующих проектов можно определять довольно свободно, исходя из ресурсов и желаний разработчиков.