Markus Manninen

Enumeraatio- ja kielityökalujen kehittäminen PlanMill-toiminnanohjausjärjestelmässä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

5.5.2014

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Markus Manninen Enumeraatio- ja kielityökalujen kehittäminen PlanMill-toiminnanohjausjärjestelmässä 31 sivua + 1 liitettä 5.5.2014
Tutkinto	insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Senior consultant/Manager Marjukka Niinioja Lehtori Simo Silander

Työn tarkoituksena oli parantaa PlanMill-toiminnanohjausjärjestelmässä käytettävien enumeraatioarvojen ja kielitermien räätälöintiin käytettyjä työkaluja. Tarkoituksena oli lisätä työkaluihin uusia ominaisuuksia, jotka helpottaisivat räätälöintiprosessia. Merkittävimpänä uudistuksena järjestelmään lisättiin tuki linkitetystä tietokannasta haettujen kielitermien ja enumeraatioiden näyttämiselle käyttöliittymässä sekä näiden sisällyttäminen järjestelmän sisäiseen hakutoiminnallisuuteen.

Tämä projekti oli osa suurempaa kokonaisuutta, jolla tähdätään järjestelmän helpompaan räätälöintiin. Tässä projektissa kehitettyjen ominaisuuksien avulla on tarkoitus tehostaa räätälöintiprosessia ja myöhemmin mahdollistaa muidenkin kuin kehittäjien tehdä yksinkertaisia muokkauksia enumeraatioihin ja kielitermeihin.

Projektin toteutus venyi hieman alkuperäisestä suunnitelmasta, minkä seurauksena osa kehitetyistä ominaisuuksista jätettiin julkaisematta. Merkittävimmät ominaisuudet saatiin kuitenkin toimimaan halutusti, ja niistä on ollut merkittävää hyötyä räätälöintien tekemisessä.

Avainsanat	PlanMill, räätälöinti, enumeraatiot, lokalisaatio

Author(s) Title Number of Pages Date	Markus Manninen Improvement of Enumeration and Language Tools in PlanMill PSA System 31 pages + 1 appendices 5 May 2014
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Marjukka Niinioja, Senior consultant/Manager Simo Silander, Senior Lecturer

The goal of this study was to develop tools for managing enumerations and language strings in PlanMill PSA. This was done by adding new features to existing tools so that they are easier to use and would ease the customization process. Most notable new feature was showing of shared enumerations and languages from linked database in the system's UI and also enabling systems internal search to support these shared resources.

This project was part of bigger strategy that aims for improvement of customization of the system. Features that were developed in this project were aimed to make customization process more efficient and also make it possible to non-developers to make simple customizations to the system by themselves.

Implementation of the project got slightly delayed from original plan and that caused some features to be left out from the release. However most important features were released successfully and they have been great help when doing customizations to PlanMill.

Keywords	Planmill, customization, enumeration, localization

Sisällys

Liitteet

Liite 1. Osa muokatusta PMLanguage-luokasta

Lyhenteet

CRM	Customer Relationship Management eli asiakkuudenhallinta.
HRM	Human Resource Management eli henkilöstöhallinta.
Java	Java on Sun Microsystemsin kehittämä laaja teknologiaperhe ja ohjelmistoalusta, johon kuuluu muun muassa laitteistoriippumaton oliopohjainen ohjelmointikieli sekä ajoaikainen ympäristö virtuaalikoneineen ja luokkakirjastoineen.
JavaScript	JavaScript on alun perin Netscape Communications Corporationin kehittämä pääasiassa Web-ympäristössä käytettävä komentosarjakieli. JavaScriptin tärkein sovellus on mahdollisuus lisätä Web-sivuille dynaamista toiminnallisuutta. JavaScriptiä ei tule sekoittaa Javaan.
JSP	JavaServer Pages on Sun Microsystemsin kehittämä Javaan perustuva menetelmä luoda HTML- ja XML-muotoisia websivuja.
Multitenanttisuus	Toteutusmalli, jossa yksi sovellusinstanssi palvelee useita asiakkaita samanaikaisesti.
PM	Project Management eli projektinhallinta.
PSA	PSA on lyhenne sanoista "Professional Services Automation", ja se tarkoittaa asiantuntijayritysten toiminnanohjausta. Perinteisestä toiminnanohjauksesta se eroaa kevyemmän rakenteensa puolesta.
SaaS	Software As A Service. Ohjelmisto palveluna.
SSL	Secure Socket Layer. Salausprotokolla.

1 Johdanto

Tämän projektin tilasi PlanMill Oy, joka on tämänhetkinen työnantajani.

Yksinkertaisen ja yksikielisen sovelluksen tekstiosuudet, kuten valikot tai painikkeiden tekstit, voivat usein olla kovakoodattuna sovelluksen lähdekoodiin. Kun sovelluksen koko ja monimutkaisuus kasvavat tai sitä on tarkoitus tarjota usealle eri kielelle, kovakoodattujen tekstien ylläpito käy hyvin työlääksi. Tämän takia useissa sovelluksissa on päädytty käyttämään erillistä sanastoa, josta tekstit haetaan valitun kielen perusteella. Itse kielikäännösten lisäksi järjestelmissä tarvitaan paljon enumeraatioita, eli listoja, joiden avulla yhdistetään numeroarvo siihen liittyvään tekstiarvoon. Enumeraatioita tarvitaan, sillä osa tiedosta on tehokkainta tallentaa numeromuodossa, mutta silti se halutaan esittää selkokielisenä käyttöliittymässä. Myös enumeraatioiden tulee tukea monikielisyyttä, joten ne liittyvät siltä osin läheisesti muihin kielikomponentteihin.

Näistä syistä myös PlanMill-toiminnanohjausjärjestelmässä käytetään sanastoa ja enumeraatioita. Muita syitä, jotka tekevät erillisen sanaston ja enumeraatioiden käytön välttämättömiksi järjestelmässä, on mm. se, että PlanMill tarjoaa mahdollisuuden räätälöidä järjestelmää asiakkaan toivomalla tavalla. Asiakkaat voivat esimerkiksi haluta uusia ominaisuuksia järjestelmään tai sitten vain yhtenäistää järjestelmässä käytetyt termit ko. yrityksen kulttuurin mukaisiksi. Tällaisten muutosten ylläpito olisi käytännössä mahdotonta ilman kielten ja enumeraatioiden hallintatyökaluja.

Projekti oli osa laajempaa projektia, jonka tarkoitus on kehittää hallintatyökaluja ja mahdollistaa myöhemmin asiakkaiden itse tehdä joitain yksinkertaisia räätälöintejä. Tämän projektin tarkoitus oli parantaa järjestelmän kielten ja enumeraatioiden hallintatyökaluja ja sitä kautta tehostaa räätälöintiprosessia ja helpottaa ylläpitoa. Tarkoituksena oli lisätä järjestelmän yleinen haku (global search) tukemaan kielitermien sekä enumeraatioiden hakua sekä paikallisesta tietokannasta että instanssien yhteisestä tietokannasta. Tämän lisäksi myös kielitermien ja enumeraatioiden hallintakäyttöliittymää oli tarkoitus parantaa.

Työn sisältö koostuu aluksi lyhyestä PlanMill Oy:n kuvauksesta. Seuraavaksi työssä käydään läpi PlanMill-sovelluksen arkkitehtuuria erityisesti niiltä osin, jotka olivat projektin kannalta keskeisiä. Tämän jälkeen työssä on projektin toteutus: suunnittelu, määrittely ja käytännön toteutus. Lopuksi käydään läpi julkaisuprosessi.

2 PlanMill

PlanMill Oy on vuonna 2001 perustettu Pk-yritys, joka toimittaa samannimistä SaaS-tyyppistä toiminnanohjaussovellusta. SaaS eli software as a service tarkoittaa jakelumallia, jossa sovelluksesta ei makseta perinteisen lisenssimallin mukaisesti vaan käytön laajuuden perusteella. SaaS-ohjelmistoissa ei ole erillisiä tuotantoympäristöjä vaan ohjelma, ja kaikki data sijaitsee keskitetysti ohjelmiston tarjoajan pilvessä. Ohjelmistoa käytetään sopivan asiakasohjelman kuten WWW-selaimen avulla. (Järvi ym. 2011: 8-9.)

Asiakkaita PlanMillillä on 25 eri maassa ja päivittäisiä käyttäjiä on yli 20 000. Asiakkaat ovat pääasiallisesti palveluyrityksiä tai yritysten IT- tai R&D-osastoja.

2.1 PlanMill-sovellus

PlanMill-sovellus on toiminnanohjausjärjestelmä (PSA), joka sisältää työkaluja mm. asiakkuudenhallintaan (CRM), henkilöstöhallintaan (HRM) ja projektinhallintaan (PM). Näiden ominaisuuksien lisäksi PlanMillillä voi hallita mm. tuotteita, tuntikirjauksia sekä laskutusta. Sovellus on multitenanttinen, eli yksi sovellusinstanssi palvelee useita asiakkaita. Sovellusta voi käyttää lähes millä tahansa nykyaikaisella WWW-selaimella. Sovellusta tarjotaan sekä pilvipalveluna että asiakkaan ylläpitämänä paikallisena on-premise-asennuksena. Selvästi suurin osa asiakkaista valitsee pilvipalvelun.

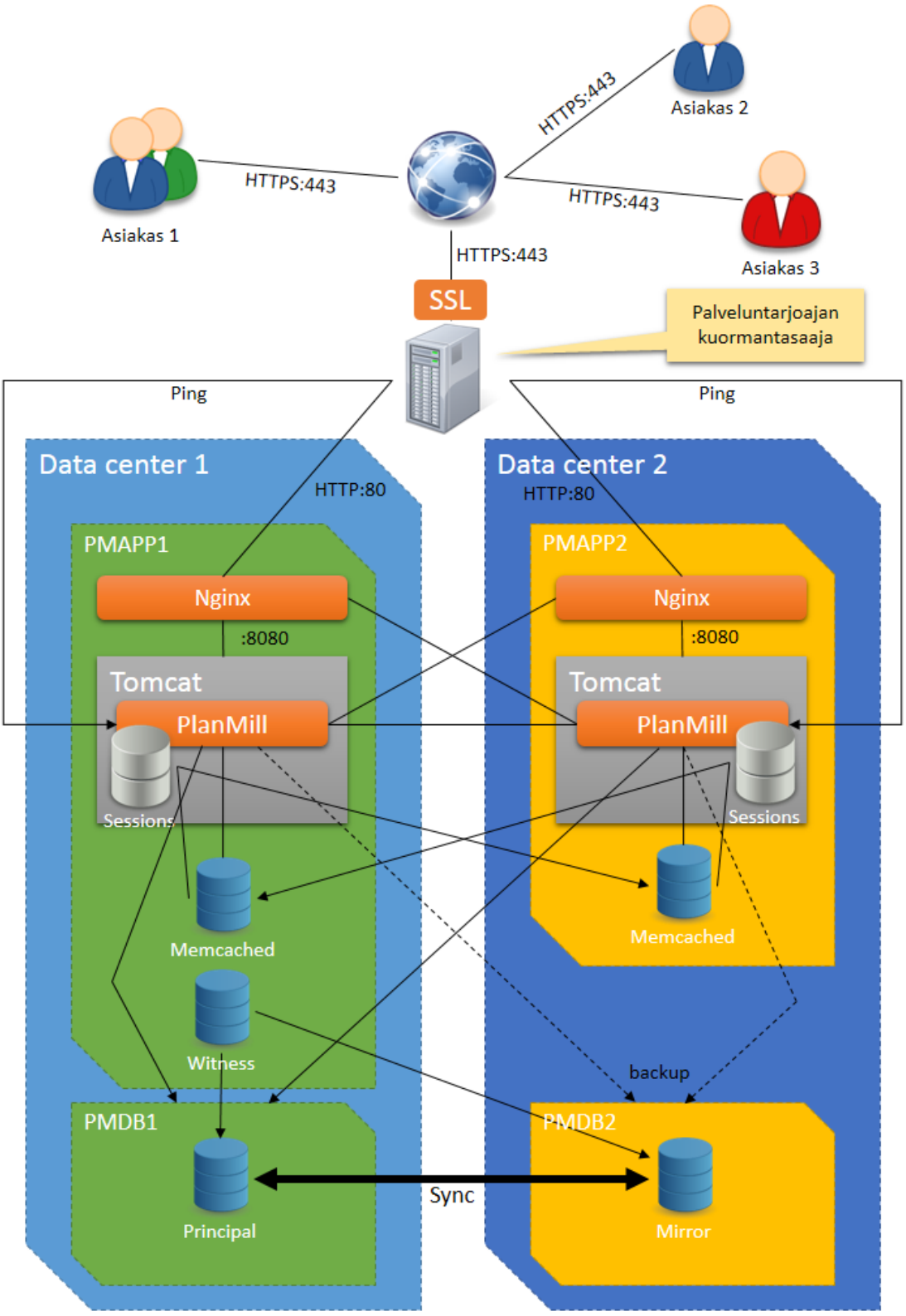
2.2 Järjestelmän arkkitehtuuri

PlanMill-sovellus, kuten monet muutkin WWW-pohjaiset sovellukset, voidaan jakaa karkeasti kolmeen osaan: taustajärjestelmään (backend), edustajärjestelmään (frontend) sekä tietokantaosaan. PlanMillin taustajärjestelmä on koodattu Javalla ja sitä ajetaan Apache Software Foundationin tarjoamalla avoimen lähdekoodin Tomcat-sovelluspalvelimella. Edustajärjestelmä perustuu dynaamisesti XSL-tyylipohjien avulla luotaviin XHTML-sivuihin ja selaimessa ajettavaan JavaScript-koodiin, jonka avulla saadaan lisättyä dynaamisia toimintoja, kuten AJAX-kutsuja, muuten staattisille XHTML-sivuille. Keskustelu edusta- ja taustajärjestelmän välillä käydään JSP-sivujen välityksellä. Näiden lisäksi sovelluksen taustalla on tietokanta, johon kaikki käyttäjien tuottama data sekä so-

velluksen toimintaan vaikuttavat parametrit tallennetaan. Jokaisella asiakkaalla on luonnollisesti oma tietokantansa, ja tämän lisäksi on ns. jaettu tietokanta, johon on tallennettu järjestelmän oletusasetukset. Tietokantapalvelimena toimii Microsoftin SQL Server 2008 R2, ja sen kanssa käytetään Transact-SQL-kieltä (T-SQL), joka on Microsoftin ja Sybasen kehittämä laajennos tavalliseen SQL-kieleen. (Transact-SQL, 2014.)

PlanMill-sovellus on hajautettu tällä hetkellä kahdelle sovelluspalvelimelle (PMAPP1 ja PMAPP2) ja kahdelle tietokantapalvelimelle (PMDB1 ja PMDB2) (Kuva 1). Sovelluspalvelinten välistä kuormaa tasataan käyttämällä kuormantasaajaa, joka ohjaa sisään tulevan liikenteen sovelluspalvelimille senhetkisen tilanteen mukaan. Hajauttaminen mahdollistaa mm. PlanMillin tai itse palvelinohjelmistojen päivittämisen ilman, että se näkyy käyttäjille. Päivitys voidaan tehdä siten, että kaikki liikenne ohjataan siksi aikaa toiselle palvelimelle, kun toista päivitetään, ja sen jälkeen sama toistetaan toisen palvelimen kanssa vastaavasti. Hajauttaminen parantaa myös järjestelmän vikasietoisuutta, sillä vika toisella sovelluspalvelimella ei kaada koko sovellusta.

Useiden samanaikaisten käyttäjien aiheuttamaa kuormaa pyritään helpottamaan hajautuksen lisäksi käyttämällä Nginx-välityspalvelinta. Nginx välittää käyttäjiltä tulevat pyynnöt sovelluspalvelimelle, joka palauttaa ohjelmalogiikan mukaan saadun tuloksen. Erityistä hyötyä Nginx:stä on staattisen sisällön, kuten kuvien, jakamisesta käyttäjille. (Nginx, 2014.)



Kuva 1. Järjestelmän arkkitehtuurikaavio

3 Kansainvälistäminen ja lokalisointi

Kansainvälistämisellä tarkoitetaan prosessia, jossa ohjelmisto muutetaan tukemaan mm. useita kieliä ja muita maa- tai kulttuurikohtaisia käytäntöjä, kuten

- kansainvälisiä merkistöjä
- päivämäärä ja aika formaatteja
- valuuttoja

Lisäksi kansainvälistämisellä tulisi saavuttaa mahdollisuus lokalisoida järjestelmä. Tämä voi esimerkiksi tarkoittaa sitä, että käännettävät komponentit on erotettu sovelluslogiikasta, jolloin käännöksiä voidaan laatia koskematta lähdekoodiin. (Esselink, 2000: 25.) Tätä varten on kehitetty useita eri menetelmiä, kuten Javan Resource Bundlet, joiden avulla käännettävät kohteet voidaan eristää omaan resurssitiedostoon (Internationalization, 2014).

Erityisiä haasteita aiheuttavat mm. merkistöihin perustuvat kielet sekä kielet, joiden lukusuunta on oikealta vasemmalle. Vaikka nykyään yleisti käytössä oleva UTF-8-merkistökoodaus tukee edellä mainittujen kielien merkistöä. Tällaisten kielten tukeminen vaatii usein muutoksia käyttöliittymään. Esimerkiksi tekstin erilaisen tilavaatimuksen takia joidenkin käyttöliittymäkomponenttien kokoa voidaan joutua muuttamaan. Erityisesti aasialaiset merkistöt ovat tilavaatimukseltaan hyvin poikkeavia latinalaiseen merkistöön verrattuna.

3.1 Lokalisaatio PlanMillissä

PlanMillillä on käyttäjiä mm. Itävallassa ja Brasiliassa, joten kansainvälistämiselle on ollut tarvetta jo pidemmän aikaa. Toistaiseksi PlanMill on käännetty kokonaan suomeksi, englanniksi ja saksaksi, mutta tuki muillekin vasemmalta oikealle luettaville kielille voidaan lisätä tarpeen vaatiessa. Järjestelmä käyttää merkistönä UTF-8:aa, joka kattaa lähes kaikki käytössä olevat merkistöt (UTF-8, 2014). Myös useiden eri valuuttojen tuki on toteutettu järjestelmään ja sitä on käsitelty aiemmassa opinnäytetyössä (Niinioja, 2012).

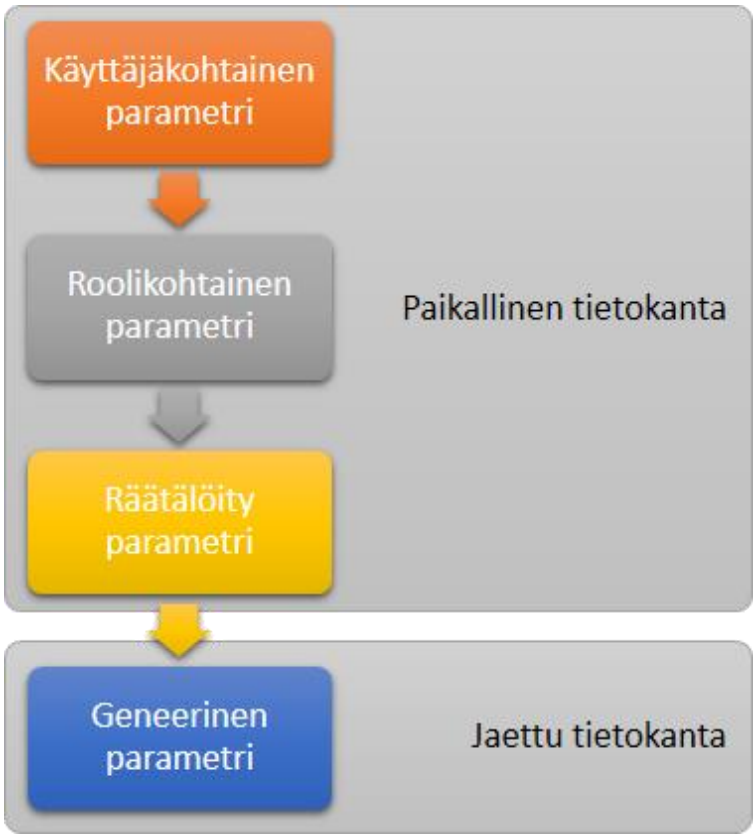
Kielituki on toteutettu siten, että käännettävät kohteet sijaitsevat tietokannassa, josta ne ladataan sovelluksen muistiin ja sieltä tarpeen vaatiessa käyttöliittymäelementteihin. Tähän ratkaisuun on päädytty, koska tällöin kielitermien päivittäminen lennosta on helppoa.

4 Enumeraatioiden ja kielitermien käyttö järjestelmässä

4.1 Parametrit

PlanMill-järjestelmä on hyvin pitkälle parametroitu, mikä tarkoittaa sitä, että suuri osa toiminnallisuudesta pohjautuu erilaisiin parametreihin. Parametreilla voidaan vaikuttaa esimerkiksi siihen, miltä jokin lomake näyttää käyttöliittymässä ja millaisia tietoja sillä voidaan tallentaa, tai vaikkapa siihen, miten jollekin listanäkymälle ladataan tietoa tietokannasta. Tällä hetkellä järjestelmän toimintaan vaikuttavia parametreja on yli 30 000 ja sen lisäksi vielä asiakaskohtaiset räätälöidyt parametrit.

Parametroinnin tarkoituksena on mahdollistaa toiminnan muokkaaminen ohjelman ajon aikana. Tämä on tarpeellista, sillä järjestelmän alasajoa pyritään välttämään viimeiseen saakka, vaikkakin se on mahdollista tehdä ilman, että siitä on haittaa käyttäjille. Toinen merkittävä tarve parametroinnille on asiakaskohtaisten räätälöintien mahdollistaminen. Tämä taas liittyy siihen, että kaikki asiakasinstanssit käyttävät käytännössä samaa taustajärjestelmää. Näin ollen itse taustajärjestelmään ei voida helposti tehdä asiakaskohtaisia räätälöintejä, vaan ne on toteutettava parametreilla. Parametrit voidaankin jakaa karkeasti kahteen ryhmään: geneerisiin ja räätälöityihin. Geneeriset parametrit ovat kaikille instansseille yhteisiä, ja ne ladataan instanssien yhteisestä jaetusta tietokannasta, kun taas räätälöidyt parametrit ladataan instanssikohtaisesta tietokannasta. Parametrit voivat olla instanssin sisällä vielä rooli- tai käyttäjäkohtaisia. Kuvassa 2 on esitetty parametrihierarkia, jossa näkyy parametrien latausjärjestys: Ensimmäisenä yritetään löytää paikallisesta tietokannasta käyttäjäkohtainen parametri. Jos tätä ei löydy siirrytään alaspäin roolikohtaiseen parametriin ja sitten instanssikohtaiseen räätälöityyn parametriin. Jos paikallisesta tietokannasta ei löydy parametria, haetaan sitä instanssien yhteisestä jaetusta tietokannasta. Jos parametri löytyy useasta paikasta, hierarkiassa ylinnä oleva parametri korvaa muut.



Kuva 2. Parametrihierarkia

4.2 Enumeraatiot

Tässä työssä enumeraatioilla tarkoitetaan järjestelmän parametria, joka koostuu avain-arvopareista, joita hyödynnetään tietokantaan tallennettujen numeroarvojen yhdistämisessä selkokielisiin vastineisiinsa. Osa arvoista kannattaa tallentaa tietokantaan numeroarvoina tekstiarvojen sijaan, sillä näin pystytään säästämään huomattavasti tilaa. Esimerkiksi arvon ”1 Very Urgent” tallentamiseen T-SQL-tietokantaan vaihtuvan pituisena tekstinä (nvarchar) vaatisi 28 tavua, kun vastaavasti pienen kokonaisluvun (smallint) tallentaminen vaatisi vain 2 tavua (int, bigint, smallint, and tinyint (Transact-SQL), 2014). Kymmenien tuhansien tietueiden tietokannassa tällä on luonnollisesti huomattava merkitys.

PlanMill-järjestelmässä käytetyt enumeraatiot ovat kuvan 3 kaltaisia. Avainarvoparit on eroteltu toisistaan puolipisteellä, ja avain ja arvo kaksoispisteellä. Arvot ovat kaarisulkeissa, sillä ne ovat viitteitä kielitermeihin.

Kuva 3. Enumeraatio PlanMill-järjestelmässä

Kuvassa esiintyvällä enumeraatiolla yhdistetään tietokantaan tallennettu prioriteetin arvo (1-4) vastaavaan kielitermiin, jonka arvo ladataan käyttöliittymän kieltä vastaavaksi.

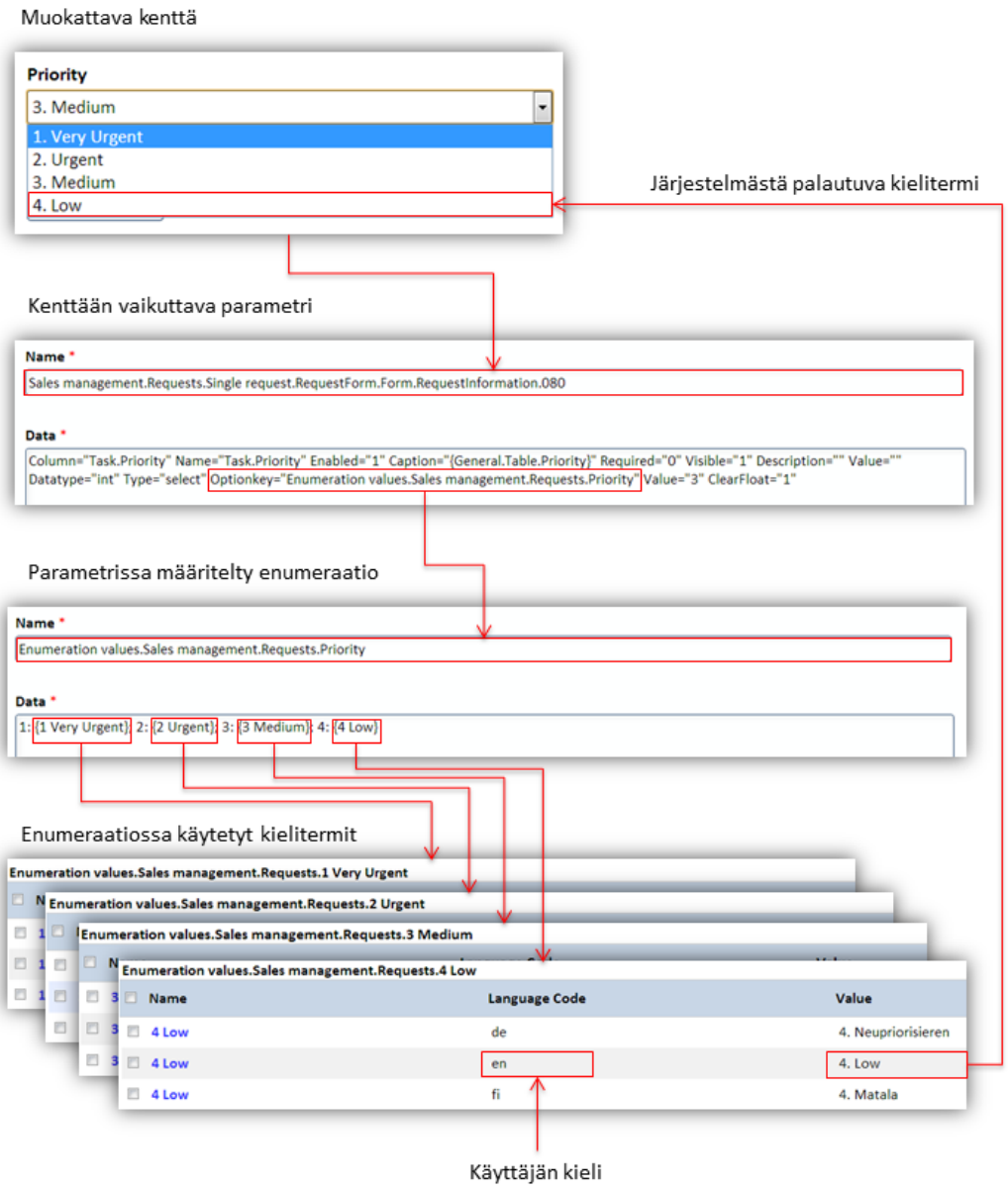
Enumeraatiot, kuten suuri osa muistakin PlanMillin toiminnoista, on parametrisoitu, eli niitä voi muokata helposti asiakaskohtaisesti ilman, että järjestelmää täytyy ajaa alas. Enumeraatioiden kohdalla tämä on välttämätöntä, sillä järjestelmään voidaan lisätä uusia arvoja tai poistaa niitä, jolloin arvoon liittyvää enumeraatiota täytyy muokata vastaavasti. Tällaisia muutoksia tehdään yleensä asiakkaan toiveesta, kun asiakas haluaa muokata järjestelmää vastaamaan omia prosessejaan esimerkiksi poistamalla arvoja, joita asiakas ei tarvitse, tai lisäämällä uuden arvon, jota muut eivät käytä.

4.3 Kielitermit

Kielitermillä tarkoitetaan tässä työssä järjestelmään tallennettua, yhdestä tai useammasta käännöksestä koostuvaa kokonaisuutta. Kielitermit ovat perustavanlaatuinen osa järjestelmän lokalisointia, sillä kaikki käyttöliittymässä esiintyvä sanasto täytyy voida kääntää käyttäjän valitseman kielen mukaiseksi. Ilman kielitermejä käännösten hallinta olisi käytännössä mahdotonta termistön suuren määrän takia. Tällä hetkellä PlanMill-järjestelmän käyttöliittymä on käännetty kokonaan kolmelle kielelle: suomeksi, englanniksi ja saksaksi, ja käytössä olevia kielitermejä on yli 6000. Näiden lisäksi on vielä erikseen laskutuksessa käytettäviä kielitermejä, jotka on muokattu vastaamaan asiakkaan kotimaan tai alueen lainsäädäntöä.

Kielitermeihin viitataan aina niiden nimellä aaltosulkeissa: joko absoluuttisella tai suhteellisella polulla. Kuvan 4 esimerkissä käytetään suhteellista polkua, sillä sitä käyttävän enumeraation polun alkuosa (Enumeration values.Sales management.Request) on identtinen kielitermin kanssa. Tällöin järjestelmä pyrkii etsimään kielitermiä ensisijaisesti tämän polun päästä. Kuten parametrien kanssa, kielitermejä haetaan ensisijaisesti pai-

kallisesta tietokannasta, ja jos tämä haku epäonnistuu, sitä haetaan jaetusta tietokannasta. Lisäksi kielitermistä pyritään ensisijaisesti hakemaan käyttäjän kielen mukainen versio, mutta jos tätä ei löydy, käytetään instanssin oletuskielistä vastinetta. Jos tätäkään ei löydy, näytetään käyttöliittymässä puuttuvan kielitermin nimi aaltosuluissa (eli siinä muodossa, jossa se on parametrissa). Kielitermit poikkeavat parametreista mm. siten, että niitä ei voi asettaa käyttäjä- tai roolikohtaiseksi, sillä tällaiselle toiminnallisuudelle ei ole ollut tarvetta.



Kuva 4. Parametrien, enumeraatioiden ja kielitermien välinen yhteys

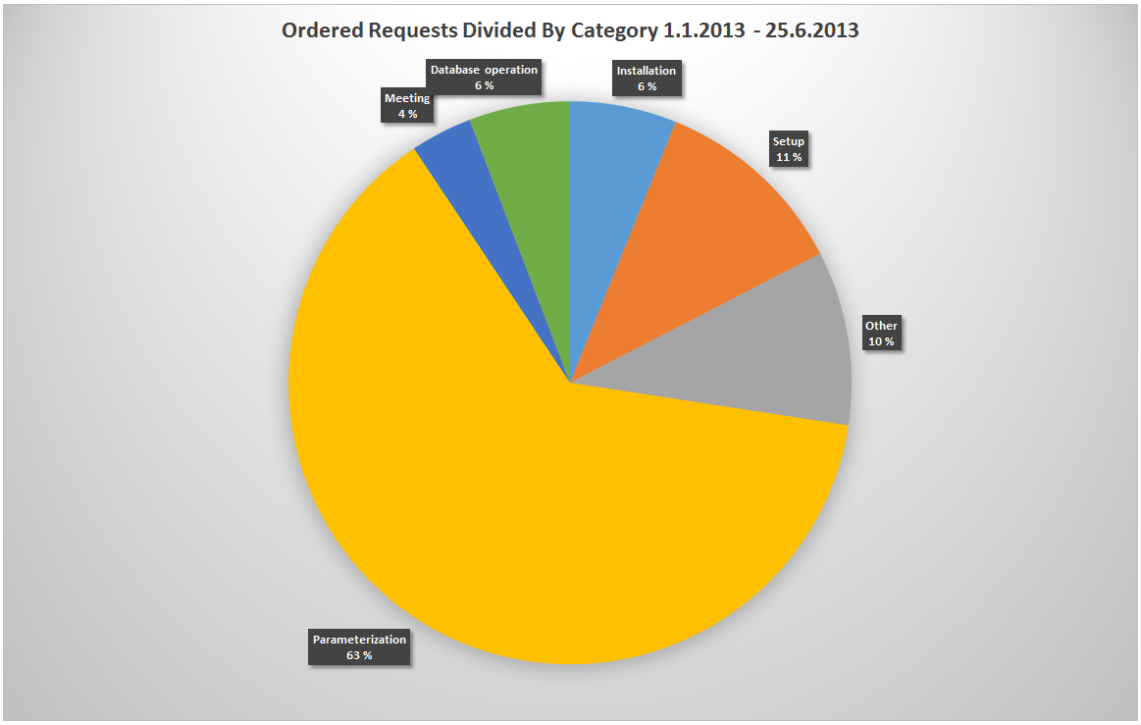
5 Projektin toteutus

Kehitysprojekti koostui seuraavista osista: Esitutkimuksesta, määrittelystä, suunnittelusta, toteutusta, testausta ja julkaisusta. Prosessi noudatti hyvin pitkälle normaalia yrityksessä käytettyä kehitysprosessia.

5.1 Esitutkimus

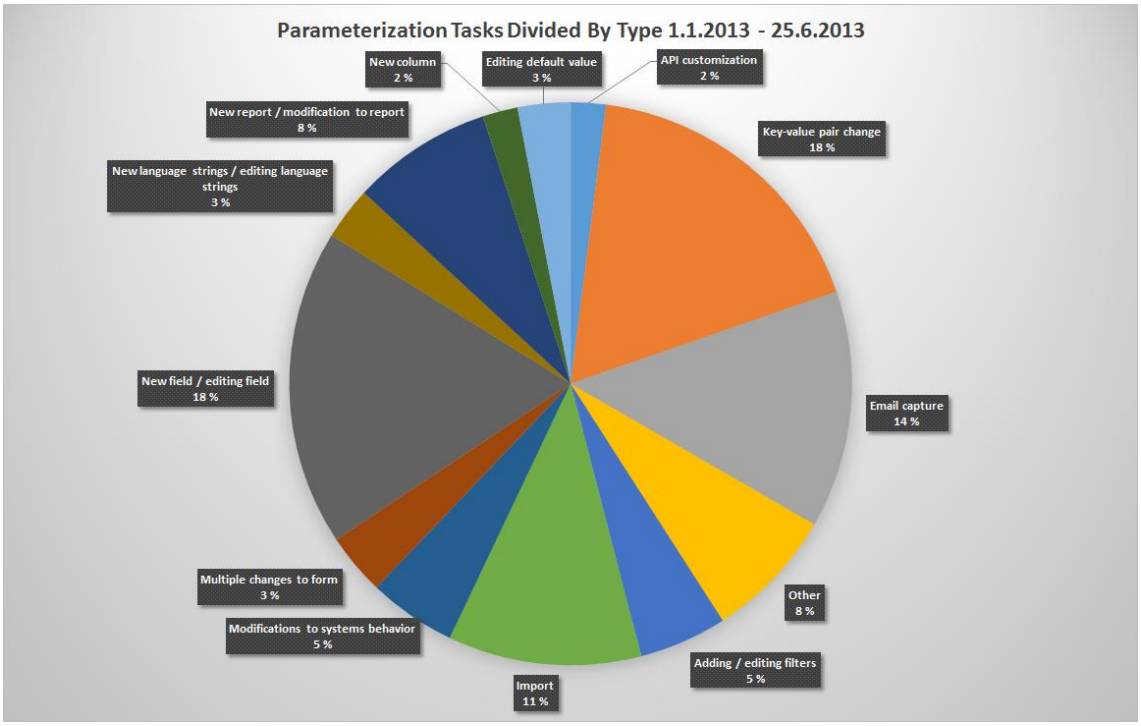
Projekti itsessään oli osa suurempaa kokonaisuutta, jonka tarkoitus on parantaa PlanMill-järjestelmän räätälöintiä ja ylläpitoa siten, että asiakas voisi joissain tilanteissa tehdä itse tarvittavat toimenpiteet. Aluksi nämä parannukset on kuitenkin suunnattu PlanMillin sisäiseen käyttöön, jolloin asiakastuki ja konsultit voisivat helpommin tehdä joitain yksinkertaisia räätälöinti- ja ylläpitotehtäviä.

Ennen varsinaisen projektin aloitusta teimme toisen projektiin osallistuvan insinööriopiskelijan kanssa tutkimusta asiakkaiden tilaamista töistä, jotka oli raportoitu vuoden 2013 ensimmäisellä puoliskolla. Vaikka otanta oli suhteellisen suppea, voitiin syntynyttä raporttia silti pitää suuntaa-antavana. Töiden analysointi aloitettiin jakamalla ne kategorioihin, kuten kuvasta 5 ilmenee. Kategorioihin jakaminen ei ollut aivan yksiselitteistä, sillä kategorioista haluttiin saada riittävän kuvaavia, mutta toisaalta kategorioiden määrä haluttiin pitää kohtuullisen pienenä. Näin ollen jako tapahtui siten, että kategoria valittiin sen perusteella, minkä tyyppistä työtä kyseinen tilaus sisälsi. Esimerkiksi joidenkin raporttien räätälöinti voi vaatia parametrien muokkauksen lisäksi myös SQL-proseduurien muokkausta, mutta usein parametrointi on suuremmassa osassa, jos kyse on yksinkertaisesta muutoksesta. Kuvan 5 perusteella voidaan todeta, että suurin osa asiakkaiden tilaamista töistä oli tyypiltään parametrointia, mikä tarkoittaa sitä, että suurin osa tilatuista töistä on sellaisia, jotka voidaan toteuttaa (ainakin suurimmaksi osaksi) parametreja muokkaamalla.



Kuva 5. Tilattujen töiden suhteellinen määrä kategorioittain

Tämän jälkeen paramterisaatiokategoriassa olevat tilaukset otettiin tarkempaa tarkasteluun ja sen perusteella saatiin kuvan 6 jakauma. Määrällisesti suurimmiksi osioiksi muodostui Avain-arvo parien muokkaus, Email capture ja Uuden kentän luonti tai muokkaus. Näistä Avain-arvo parien muokkaus on puhtaasti enumeraatioiden käsittelyä, kun taas Email capture eli sähköpostien sisäänlukuun liittyvät tehtävät ja kenttien luominen tai muokkaaminen on toteutettu suurimmaksi osaksi muita parametreja muokkaamalla. Puhtaasti kielitermien muokkaamista vaativat tehtävät olivat selvässä vähemmistössä 3 %:n osuudellaan, mutta kielitermien luomista tai muokkaamista esiintyy usein mm. uusien kenttien luonnin sekä enumeraatioiden muokkauksen yhteydessä, joten todellisuudessa tämän tyyppistä työtä on enemmän kuin kaavio antaa ymmärtää.



Kuva 6. Parametrointikategorian tehtävien suhteellinen määrä tyypeittäin

Tällaisia tehtäviä arvioitaessa omien ja muiden räätälöintejä toteuttaneiden henkilöiden kokemusten perusteella suurin pullonkaula oli se, että jaettuja resursseja kuten parametreja (enumeraatiot mukaan lukien) tai kielitermejä ei voitu käsitellä suoraan asiakasinstansseissa vaan niitä joutui tarkastelemaan instanssien yhteisessä jaettuja resursseja varten luodussa instanssissa. Käytännössä jos jostain vakioparametrista tai kielitermistä halusi tehdä instanssikohtaisen muokatun version, täytyi ensin mennä jaettuun instanssiin hakemaan siitä vakioversio ja kopioida se käsittelyssä olevaan instanssiin ja sen jälkeen muokata halutulla tavalla. Sama ongelma oli myös paikallisella palvelimella sijaitsevilla kehitysinstansseilla, joilla myös on jaettuja resursseja varten oma tietokanta ja instanssi.

5.2 Määrittely

Esitutkimuksen perusteella aloitettiin kaksi projektia, joista toinen keskittyy parametrien hallinnan parantamiseen ja toinen enumeraatio- ja kielityökalujen kehittämiseen. Tässä työssä käsitellään jälkimmäistä projektia, vaikka nämä projektit joiltain osin olivat päällekkäisiä. Projektien välillä tehtiin myös jonkin verran yhteistyötä ja toteutukseen saatiin apua projektin ulkopuolisilta kokeneimmilta kehittäjiltä.

Tässä työssä käsiteltävän projektin tavoitteeksi päätettiin ensisijaisesti enumeraatio- ja kielityökaluja siten, että asiakasinstansseissa pystyy näkemään myös jaetut enumeraatiot ja kielitermit, sekä mahdollisuus tehdä niistä paikalliset kopiot muokkausta varten. Lisäksi kielitermeille oli tarkoitus lisätä hakutoiminto, jonka avulla voidaan löytää mm. kielitermit, joilta puuttuu jonkinkielisiä käännöksiä. Nämä ominaisuudet tuli voida rajoittaa vain tehokäyttäjille ja ylläpitäjille. Hyväksymiskriteerinä oli myös se, että asiakasinstansseista käsin ei voi muokata tai poistaa jaettuja resursseja. Tämä on tärkeää, sillä jaetut resurssit vaikuttavat kaikkiin instansseihin, jolloin harkitsemattomat muutokset voivat aiheuttaa vakavia seurauksia.

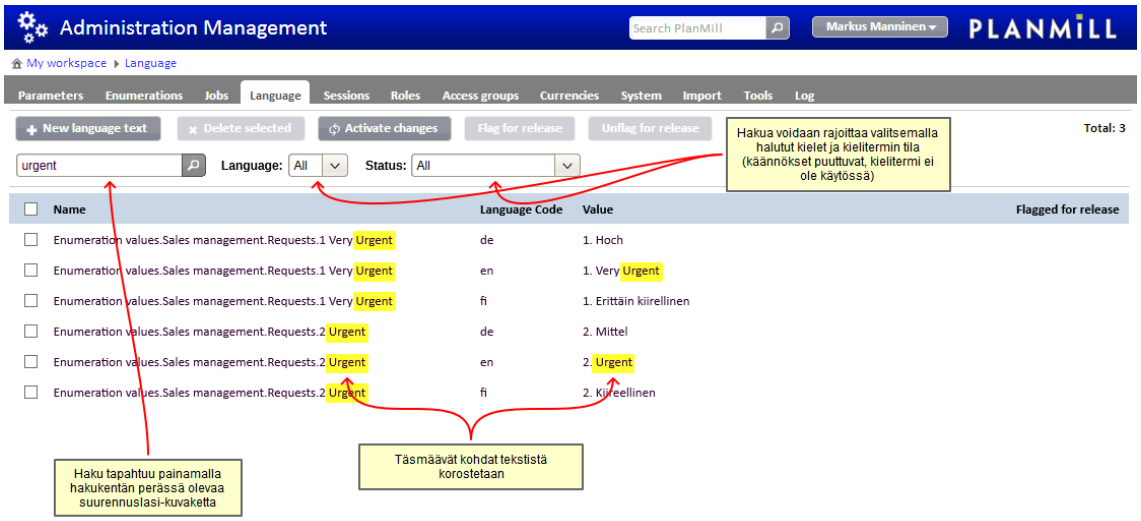
Toisena tavoitteena projektissa oli tarkoitus parantaa työkalujen käytettävyyttä muokkaamalla enumeraatioiden ja kielitermien luonti-/muokkauslomakkeita. Etenkin kielitermien luonti ja muokkaus kaipasi kehitystä, sillä tämä tapahtui toistaiseksi yksi käännös kerrallaan, jolloin jokainen erikielinen käännös jouduttiin luomaan erillisellä lomakkeella.

5.3 Suunnittelu

Suunnittelu aloitettiin luomalla luonnoksia uusista ominaisuuksista, joiden pohjalta käytyjen keskusteluiden perusteella voitiin lyödä lukkoon kehitettävät ominaisuudet. Luonnosten suunnittelussa pääpainona oli se, että uudet ominaisuudet ovat käytettävyydeltään yhteneväisiä muun järjestelmän kanssa. Myös se, että projektia varten oli varattu suhteellisen lyhyt ajanjakso, piti pitää mielessä suunnitelmia laadittaessa. Luonnokset tehtiin avoimen lähdekoodin Pencil (http://pencil.evolus.vn/) sovelluksella ja pohjina käytettiin PlanMillin sen hetkistä ulkoasua.

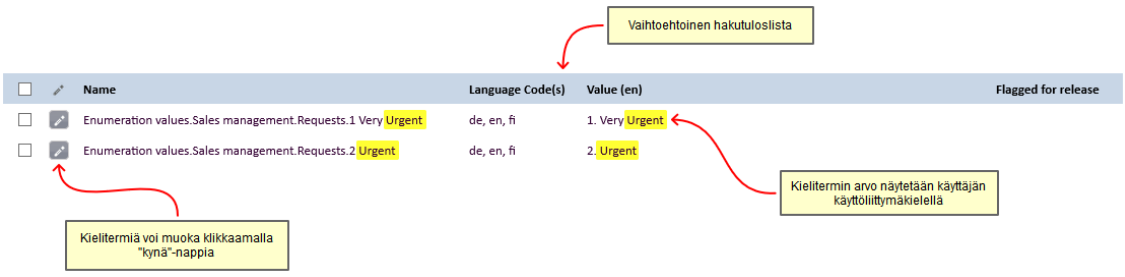
5.3.1 Kielitermien haku

Kielitermien hakunäkymän luonnoksessa (Kuva 7) pyrittiin siihen, että hakunäkymä olisi vastaavanlainen kuin muissakin moduuleissa, joissa näytettävää listaa voidaan rajoittaa vapaasti kirjoitettavien sanojen avulla ja lisäksi valintatyyppisillä suodattimilla. Normaalista listanäkymästä poiketen haun oli tarkoitus näyttää tyhjä joukko, jos tekstihakukenttä oli tyhjä. Suodattimiksi valittiin aluksi kieli ja kielitermin tila, jotka helpottaisivat puutteellisten kielitermien löytämistä. Hakutuloksissa oli tarkoitus käyttää samanlaista korostusta kuin muissakin listanäkymissä, eli tekstihaussa esiintyvät sanat korostetaan hakutuloksissa. Kielitermien nimet toimisivat linkkinä ko. kielitermin muokkaus näkymään.



Kuva 7. Kielitermien hakunäkymän luonnos

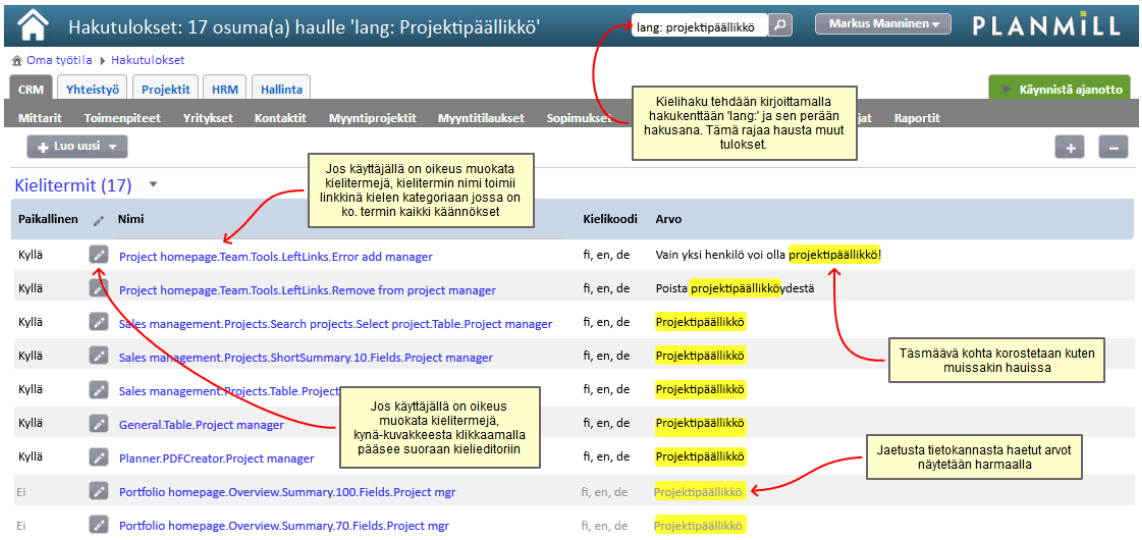
Vaihtoehtoisessa luonnoksessa (Kuva 8) hakutulokset ryhmiteltäisiin siten, että jokainen kielitermi näytetään vain kerran ja mahdolliset käännökset listataan kielikoodit-sarakkeeseen. Lisäksi jokaisen kielitermin kohdalla on muokkausnappi, jota klikkaamalla päästäisiin muokkausnäkymään, jossa voitaisiin muokata kaikkia käännöksiä yhdellä kertaa. Kielitermin arvo näytettäisiin käyttäjänkielisenä tai sen puuttuessa oletuskielisenä. Tässä tapauksessa ratkaisemattomaksi jäi se, miten sellaiset kielitermit tulisi näyttää, joissa haku osuu johonkin käännökseen, joka ei ole kieleltään sama kuin käyttäjän kieli.



Kuva 8. Vaihtoehtoinen kielitermien hakunäkymän luonnos

Moduulin sisäisen haun lisäksi tarkoitus oli lisätä PlanMillin ”global search”, eli lähes kaikkialla yläpalkissa näkyvä hakutoiminto, tukemaan myös kielitermejä. Toiminto olisi kuitenkin rajoitettu vain pääkäyttäjille ja muille, joilla on tarve päästä tarkastelemaan ja muokkaamaan kielitermejä. Hakutoiminnon haluttiin myös näyttävän jaetusta tietokannasta tulevat kielitermit (Kuva 9). Muilta osin hakutuloslistaus näyttää samanlaiselta kuin muutkin hakutoiminnot. Kielitermiä pääsee muokkaamaan klikkaamalla kynä-ikonia ja

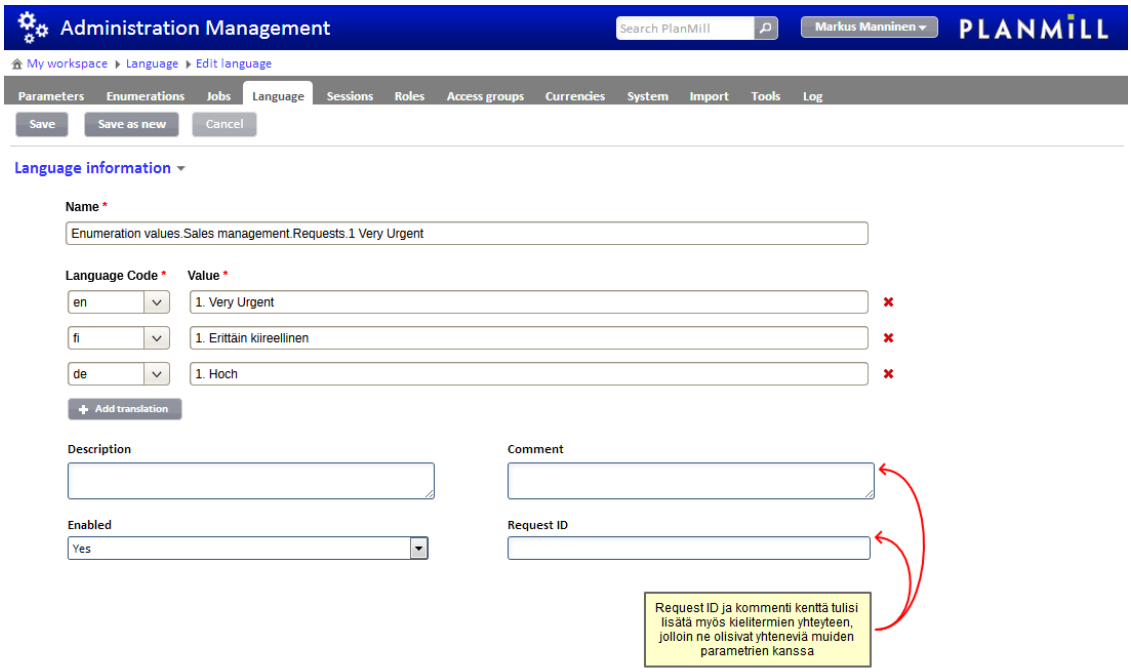
kielitermikategoriaa pääsee tarkastelemaan klikkaamalla kielitermin nimeä. Lisäksi luonnoksen yhteydessä pohdittiin mahdollisuutta lisätä yleiseen hakukenttään tapa hakea vain tietyntyyppisiä objekteja, kuten juuri kielitermejä. Vaihtoehtoisia tapoja olivat mm. hakukenttään lisättävä tarkentava teksti tai valintalista.



Kuva 9. Kielitermien haku yleisellä hakutoiminnolla

5.3.2 Kielitermien muokkaus

Kielitermien muokkausta pyrittiin helpottamaan siten, että aiemmasta poiketen kaikkia käännöksiä voitaisiin muokata samassa muokkausnäkymässä (Kuva 10). Uudessa muokkausnäkymässä käännökset on jaoteltu omille riveilleen ja niitä voi helposti lisätä ja poistaa. Tämän kaltaisia monirivilomakkeita on useassa paikassa järjestelmässä entuudestaan, joten toiminta olisi helposti omaksuttavissa. Kielitermillä tulisi kuitenkin olla vähintään yksi järjestelmän oletuskielinen käännös, mikä pitäisi ottaa huomioon lomakkeen validoinnissa.

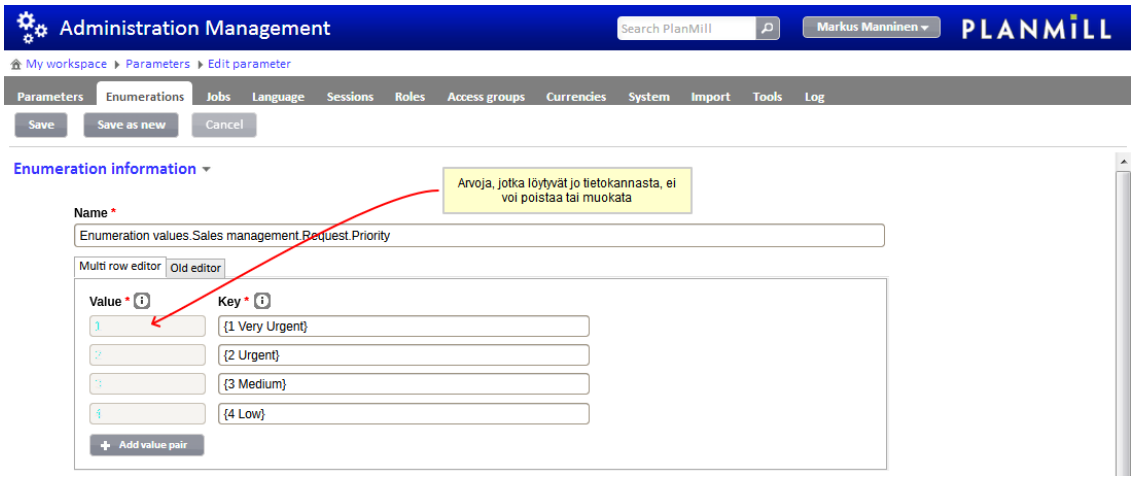


Kuva 10. Kielitermin muokkausnäkymän luonnos

5.3.3 Enumeraatioiden muokkaus

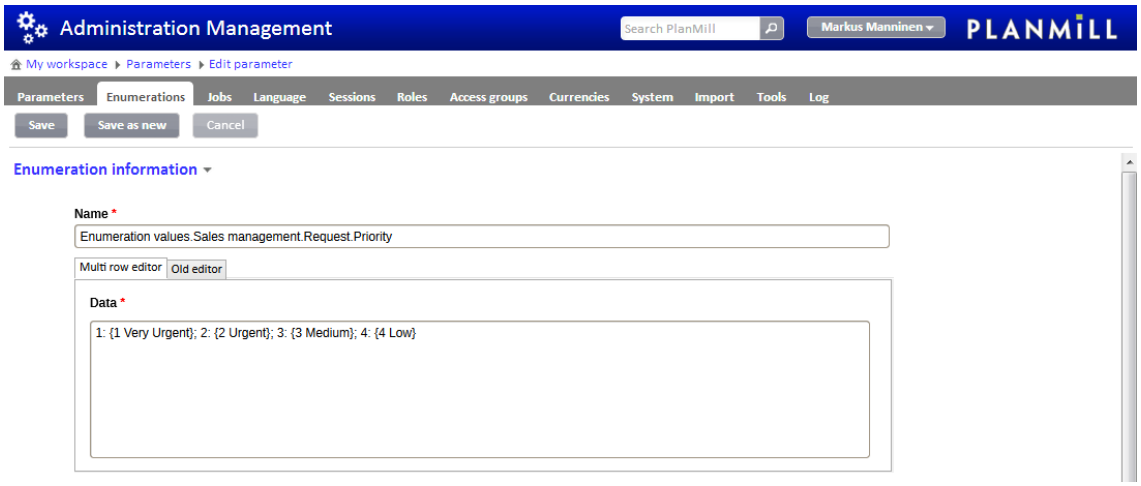
Käytännössä enumeraatiot ovat parametrien osajoukko, ja täten ne käyttävät samaa toimintalogiikkaa. Tästä syystä enumeraatioiden hakutoimintoja ei tässä projektissa käsitellä, sillä niihin vaikuttavat tulokset rinnakkaisprojektista, joka käsittelee parametrien käytön kehitystä.

Tässä projektissa pyrittiin parantamaan enumeraatioiden muokkausta siten, että avain-arvoparit erotellaan omiksi kentikseen omille riveilleen (Kuva 11).



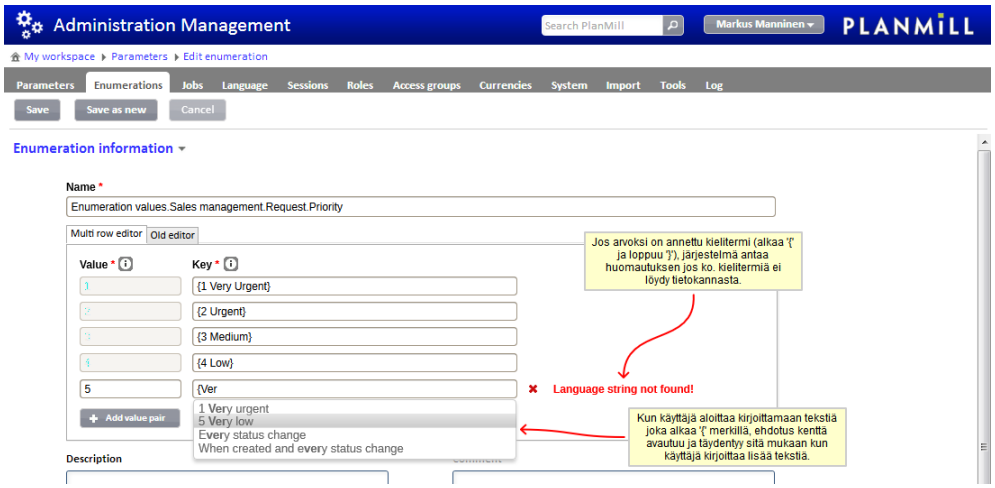
Kuva 11. Enumeraation muokkausnäkymän luonnos

Tämä helpottaa enumeraation luettavuutta aiempaan nähden jossa, avainarvoparit ovat peräkkäin puolipisteellä eroteltuna ja mahdollisesti rivinvaihdolla (kuva 11). Myös virheettömyys paranee, sillä erotinmerkit lisätään automaattisesti tallennettavaan arvoon, jolloin ne eivät pääse unohtumaan. Vanhanaikaista editoria tulisi silti voida käyttää uuden rinnalla, erikoistapausten yhteensopivuuden varmistamiseksi.



Kuva 12. Enumeraation muokkausnäkymän luonnos – vanhan tyyppinen editori

Uudenlainen lomake mahdollistaisi myös muunlaisia parannuksia, kuten mahdollisuuden tarkastaa, onko enumeraation joitain arvoja jo käytössä järjestelmässä ja esimerkiksi täytön avustamisen näyttämällä järjestelmästä löytyviä vaihtoehtoja, kun avaimena käytettään kielitermejä (Kuva 13).



Kuva 13. Enumeraation muokkausnäkymän luonnos - lisätty rivi

5.4 Toteutus

Suunnitteluvaiheen jälkeen pidettiin parametrien hallinnan kehitysprojektin kanssa yhteinen kokous, jossa käsiteltiin tarkemmin työnjakoa ja mahdollisia päällekkäisyyksiä. Kokouksessa päätettiin, että tässä projektissa kehitetään ensisijaisesti kielitermeihin liittyviä asioita, sillä enumeraatioiden kehitys on niin lähellä parametreja.

Toteutuksessa käytettiin hyväksi ketterän kehityksen menetelmiä ja projekti pyrittiin toteuttamaan SCRUM-sprinteissä. Suunnitteluvaiheessa luotujen luonnosten pohjalta käytyiin keskusteluita ja luotiin lista kehitettävistä ominaisuuksista (tuotteen kehitysjono). Kehitysjonosta valittiin toteuttamiskelpoisimmat ja tärkeimmät ominaisuudet sprintin tehtävälistaan. Sprintin pituudeksi asetettiin 2 viikkoa.

Ensimmäiseen sprinttiin valittiin seuraavat ominaisuudet:

- kielitermien lisäys yleiseen hakutoimintoon (global search)
- mahdollisuus näyttää jaetusta tietokannasta haetut kielitermit sekä hakutuloksissa että kielitermimoduulissa
- usean käännöksen muokkaaminen yhdessä näkymässä
- Mahdollisuus kopioida jaettu kielitermi paikalliseksi listanäkymästä.

Enumeraatioiden muokkaustoimintojen kehitys päätettiin tässä vaiheessa jättää kehitysjonoon, sillä niiden kehittäminen olisi aiheuttanut liikaa päällekkäisyyksiä rinnakkaisen parametrien hallinnan kehitysprojektin kanssa. Toisaalta parametriprojektin tuloksena myös enumeraatioita voidaan hakea yleisellä hakutoiminnolla ja jaetusta tietokannasta haetut enumeraatiot voitaisiin näyttää järjestelmässä kuten kielitermitkin.

Ongelmia sprintin tehtävälistan muodostamiseen aiheutti se, että en ollut aiemmin ollut juurikaan mukana kehittämässä järjestelmään mitään tällaista ominaisuutta, joka vaatii muutoksia moneen moduuliin, joten tehtäviin tarvittavan ajan arvioiminen oli hyvin haastavaa. Itse työstä myös suuri osa kului järjestelmän arkkitehtuurin opiskeluun, sillä tarkoituksena oli noudattaa yleisesti muualla järjestelmässä käytettyjä suunnittelumalleja.

Kehitys aloitettiin luomalla uusi haara PlanMill-sovelluksen päähaarasta sekä luomalla testi-instanssi testipalvelimelle. Tällä tavalla kehitettäviä ominaisuuksia pystyttiin kehittämään ja testaamaan päähaarasta riippumattomasti. Molemmat operaatiot on automatisoitu käyttäen jatkuvan integroinnin palvelinta (continuous integration server), Jenkinsiä. Versionhallintaan käytetään Apache Subversionia ja asiakaspäätteenä Tortoise SVN:ää.

5.4.1 Kielitermien haku

Ensimmäisenä tehtävälistalla oli kielitermien lisäys yleiseen hakutoimintoon (global search), minkä toteuttaminen onnistui lähestulkoon pelkästään parametrien avulla. Taustajärjestelmässä oli valmiiksi tuki kahdelle erityyppiselle tavalle etsiä vastaavuuksia tekstimuotoisesta datasta, joka on tallennettu tietokantaan. Ensimmäinen tapa oli loogista operaattoria LIKE käyttävä haku ja toinen oli fulltext-indeksiä käyttävä huomattavasti tehokkaampi haku (http://technet.microsoft.com/en-us/library/ms142571.aspx). Fulltext-hakua käytettäessä tietokantaan tuli luoda vastaava indeksi, jota haku käyttää. Indeksi lisäisi tietokantapalvelimen resurssien kulutusta hieman, mutta nopeuttaisi hakua huomattavasti. Aluksi tätä kokeiltiinkin, mutta tästä luovuttiin sillä fulltext-haku ei kykene hakemaan tekstin osalla kuin tekstin alusta. Tällöin esimerkiksi haettaessa kielitermiä, jonka nimi on ”General.Table.Project manager”, hakusanalla ”Project”, ei vastaavuutta löytyisi, sillä sana Project ei esiinny minkään sanan alussa johtuen parametrien ja kielitermien nimeämiskäytännöstä.

5.4.2 Jaettujen kielitermien näyttäminen käyttöliittymässä

Seuraavaksi tehtäväksi oli valittu jaettujen kielitermien näyttäminen kaikissa instanssseissa. Tähänkin tehtävään oli kaksi lähestymistapaa: ensimmäinen tapa oli hakea kielitermit suoraan sovelluksen muistista ja toinen oli hakea ne tietokantakyselyillä jaetusta tietokannasta. Ensimmäinen tapa olisi tehokkaampi, sillä myös jaetut kielitermit on ladattu instanssin käynnistyessä sovelluksen muistiin, josta niitä on nopea hakea. Ongelmana oli kuitenkin se, että taustajärjestelmässä ei ollut tukea niiden hakuun muistista käyttöliittymään. Toinen ongelma oli epätodennäköiset mutta mahdolliset eheysongelmat, sillä ei voida olla täysin varmoja siitä, että sovelluksen muisti vastaisi jokaisella hetkellä tietokantaa, sillä kielitermejä ei päivitetä muistiin reaaliaikaisesti. Tästä voisi seurata ongelmia, jos jaettuun tietokantaan on tehty muutoksia, mutta instanssissa käsitelläänkin muistista haettua kielitermiä, joka ei vastaa enää tietokannan versiota.

Tässä tapauksessa päädyttiin hakemaan jaetut kielitermit tietokantakyselyillä jaetusta tietokannasta. Tämä oli hieman kokeellinen lähestymistapa, sillä järjestelmässä haut kohdistuvat normaalikäytössä aina instanssin omaan tietokantaan. Jaetusta tietokannasta hakuja tehtäessä ei voida edes olla varmoja, sijaitseeko jaettu tietokanta samalla tietokantapalvelimella, mikä voi hidastaa toimintaa normaalia enemmän.

Rinnakkaisesta tietokannasta haku aiheuttaa omat ongelmansa, kuten se, että eri kielitermeillä saattaa olla sama id, kun ne haetaan jaetusta tietokannasta. Mahdollisuus hakea kielitermejä jaetusta tietokannasta haluttiin myös rajoittaa vain tietyille käyttäjille kuten ylläpitäjille. Myös jaettuun tietokantaan liittyvässä instanssissa haku jaetusta tietokannasta piti estää, sillä instanssilla ei ole viitettä kuin omaan tietokantaansa.

Esimerkkikoodissa 1 on esitetty yksi kielitermien lataamisessa käytetty sql-kysely, jossa on pyritty ratkomaan edellä mainittuja ongelmia. Jaetusta tietokannasta haettujen kielitermien id:t muutetaan miinusmerkkisiksi, jolloin samoja id:itä ei esiinny tulosjoukossa. Taustajärjestelmässä oli jo tuki erilaisille käyttöoikeuksille ja niitä pystyi käyttämään mm. siten, että osa koodista ympäröidään järjestelmän ymmärtämällä erikoiskäsittelyllä. Tällöin, jos syntaksissa määritelty käyttöoikeus puuttuu, ko. koodi poistetaan ennen sen ajoa. Taustajärjestelmään piti tehdä erikoiskäsittely tässä käytettyä ”accessshared”-attribuuttia varten, joka pitää myös huolen siitä, että jos käyttöoikeus puuttuu kokonaan, koodi jätetään ajamatta.

Muita paikanvaraajia (placeholder) ovat mm. {share} ja {where}, joihin taustajärjestelmä täyttää tarvittavat tiedot. Paikanvaraaja {share}:n tilalle järjestelmä hakee ko. instanssin asetuksissa määritellyn osoitteen jaetulle tietokantapalvelimelle jaettujen kielitermien hakua varten. Paikanvaraaja {where}:n tilalle rakennetaan hakuehdot: tässä tapauksessa hakusanat. Näitäkin varten täytyi tehdä muutoksia taustajärjestelmään, sillä normaalista poiketen esimerkiksi {where} saattaa esiintyä kahteen kertaan kyselyssä. Taustajärjestelmä muutettiin laskemaan esiintymiskerrat ja täyttämään paikanvaraajat vastaavasti. Samalla mahdollistettiin tuki käyttää useita samannimisiä paikanvaraajie muissakin tietokantakyselyissä.

```
SELECT TOP 100
    *
FROM (
    SELECT
        *
    FROM (
        SELECT
            'ui-icon-home' AS Icon,
            'ui-icon-pencil' AS Edit,
            Language.Name COLLATE Finnish_Swedish_CI_AS AS Name,
            Language.LanguageCode COLLATE Finnish_Swedish_CI_AS AS LanguageCode,
            Language.Data COLLATE Finnish_Swedish_CI_AS AS Data,
            SUBSTRING(
                (SELECT
                    ',' + Lang1.LanguageCode
                FROM
                    Language Lang1
                WHERE
                    Lang1.Name COLLATE Finnish_Swedish_CI_AS= Language.Name
                ORDER BY
                    Lang1.Name
                FOR XML PATH ('')
                )
            ,2,100) COLLATE Finnish_Swedish_CI_AS AS Translations,
            Language.Id,
            NULL AS SharedStyle,
            1 AS Local
        FROM
            Language
    ) AS Language
    WHERE {where}

[accessshared:'Administration.Languages view shared':'
    UNION

    SELECT
        *
    FROM (
        SELECT
            NULL AS Icon,
            'ui-icon-plusthick' AS Edit,
            Language.Name COLLATE Finnish_Swedish_CI_AS AS Name,
            Language.LanguageCode COLLATE Finnish_Swedish_CI_AS AS LanguageCode,
            Language.Data COLLATE Finnish_Swedish_CI_AS AS Data,
            SUBSTRING(
                (SELECT
                    ',' + Lang1.LanguageCode
                FROM
                    {share}.dbo.Language Lang1
                WHERE
                    Lang1.Name COLLATE Finnish_Swedish_CI_AS = Language.Name
                ORDER BY
                    Lang1.Name
                FOR XML PATH ('')
```

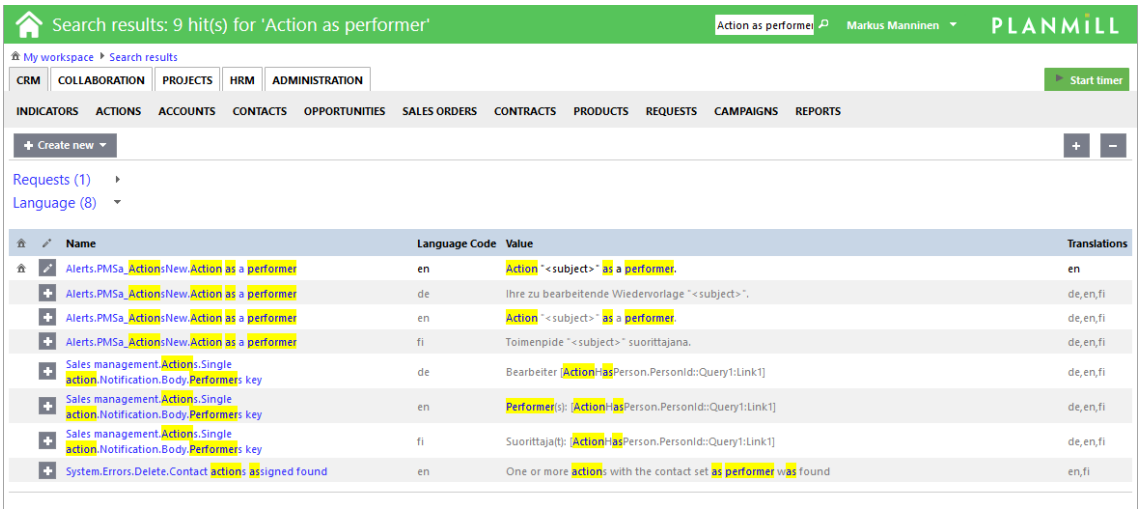
```
                )
            ,2,100) COLLATE Finnish_Swedish_CI_AS AS Translations,
            -Language.Id AS Id,
            N'Color: grey' AS SharedStyle,
            0 AS Local
        FROM
            {share}.dbo.Language
    ) AS Language
    WHERE {where}
']
) AS Language

ORDER BY Language.Local DESC, Language.Name
```

Koodiesimerkki 1. SQL-kysely kielitermien hakemiseen sekä paikallisesta että jaetusta tietokannasta

Koodiesimerkissä on määritelty alussa tulosjoukon maksimikooksi 100 riviä, sillä suurempaa määrää ei ole mielekästä näyttää käyttöliittymässä. Toisaalta muutkin haut palauttavat oletuksena em. määrän rivejä. Jos haettu kohde ei mahdu tulosjoukkoon, tulee hakutermejä tällöin tarkentaa. Lopussa olevassa ORDER BY -lauseessa määritellään järjestys siten, että ensimmäisenä listassa ovat paikalliset (usein muokatut) kielitermit ja sen jälkeen jaetut. Toissijaisena järjestyksenä on kielitermin nimi.

Paikalliset kielitermit erotetaan jaetuista siten, että niiden edessä näkyy paikallisuutta ilmaiseva ”koti”-ikoni ja toisaalta jaettujen kielitermien teksti on harmaa normaalin mustan sijaan (Kuva 14).

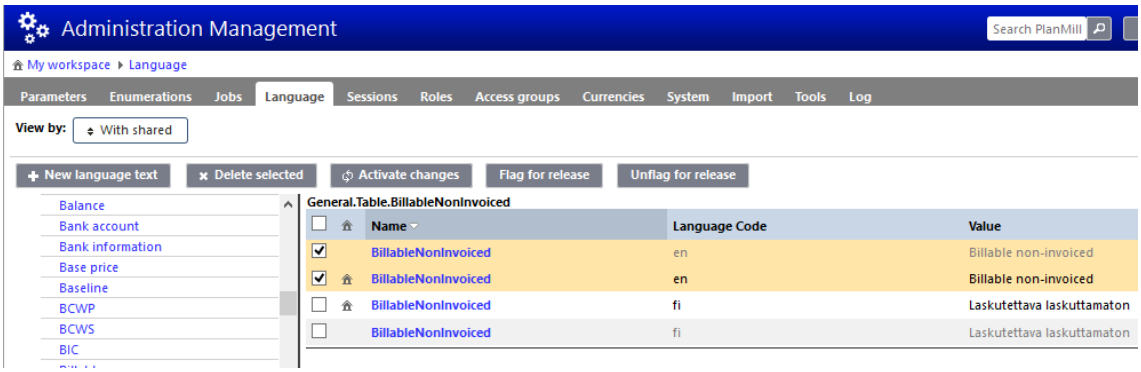
Kyselyssä määritellään aakkostus (collation) erikseen, mitä ei normaalisti tarvitse tehdä, mutta tässä tapauksessa se on tarpeellista, sillä linkitetystä tietokannasta haettaessa ei voida olla varmoja ko. tietokannan aakkostuksesta kaikissa tapauksissa. Kyselyssä käytetään myös hyväksi T-SQL:n tapaa käsitellä tietoa XML-muodossa, minkä avulla useita tulosrivejä voidaan yhdistää yhteen kenttään, kuten käännöksiä ilmaisevien kielikoodien kanssa on tehty.



Kuva 14. Lopullinen versio kielitermien hakutuloksista.

Tuen lisääminen jaettujen kielitermien näyttämiselle lisäsi myös tarpeen päivittää joitain muitakin käyttöliittymän toimintoja, kuten mahdollisuus valita, näytetäänkö myös jaetut kielitermit käyttöliittymässä, sekä rajoituksia kielitermien poistamiseen ja tallennukseen.

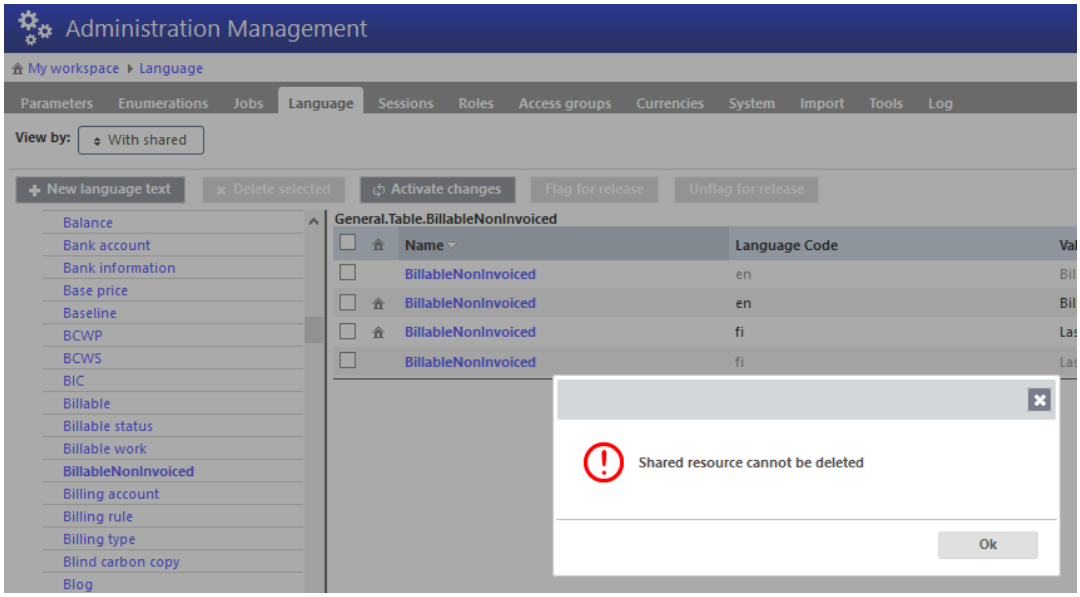
Esimerkiksi jos käyttäjä on valinnut usean kielitermin käännöksen ja joukossa on yksi tai useampi jaettu käännös ja käyttäjä yrittää poistaa nämä käännökset painamalla poistonappia (Kuva 15), järjestelmä estää toiminnon ja näyttää virheviestin (Kuva 16).



Kuva 15. Hallinnointiosion kielitermi näkymä, jossa on valittu kaksi käännöstä

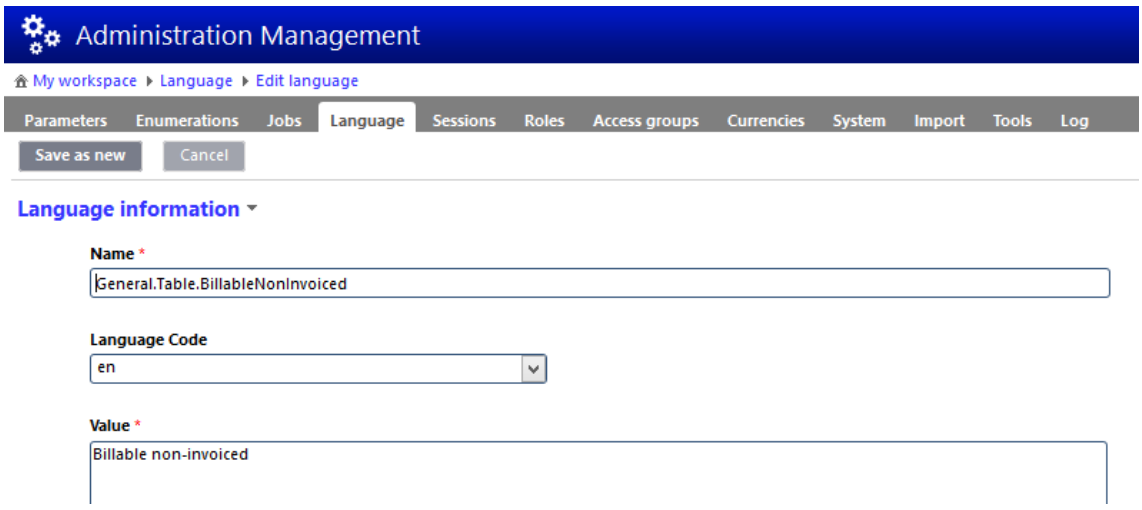
Kuvassa 15 on valittu kaksi kielitermin käännöstä: yksi paikallinen ja yksi jaetusta kannasta haettu. Paikallisen käännöksen edessä on ”koti”-ikoni, ja teksti on normaali musta, harmaan sijaan. Poistaminen tapahtuu painamalla ”Delete selected”-nappia. Muita toimintoja työkalurivillä ovat ”Activate changes”, joka lataa kielitermit uudelleen muistiin tietokannasta, sekä ”Flag for release” ja ”Unflag for release”, jotka asettavat käännökseen

lipun, joka kertoo, että tämä käännös on tarkoitus kopioida kehitysinstanssista julkaisuinstanssiin.



Kuva 16. Virheviesti kielitermin käännöksen poiston yhteydessä

Myös käännöksen muokkauslomaketta muutettiin siten, että jos muokattavaksi valitaan käännös, joka on jaettu, sitä ei voi tallentaa normaalisti aiemman päälle, vaan se täytyy tallentaa uutena, jolloin siitä syntyy uusi kopio paikalliseen tietokantaan (Kuva 17). Tämän jälkeen ko. käännöstä voidaan käsitellä kuten mitä tahansa paikallista käännöstä.



Kuva 17. Yksittäisen jaetun käännöksen muokkauslomake

5.4.3 Usean käännöksen muokkaaminen yhdessä näkymässä

Kun jaetut kielitermit oli saatu näkyviin käyttöliittymässä, alettiin kielitermien muokkausnäkymän kehitys. Tämä oli huomattavasti vaativampi kehityskohde, sillä siinä missä suurin osa aiemmista toiminnoista oli pystytty toteuttamaan suurimmaksi osin parametreja muokkaamalla, piti tämän toiminnon toteuttamiseksi tehdä parametroinnin lisäksi suuria muutoksia taustajärjestelmään.

Järjestelmän muissa osissa on toteutettu usean objektin lataus ja tallennus yhdellä lomakkeella, joten lomakenäkymän parametrit tukivat jo tällaista tominnallisuutta. Usean kielitermin tallennukselle ei kuitenkaan ollut vielä tukea taustajärjestelmässä, joten sellainen piti toteuttaa. Tämä oli toteutettavissa hyvin pitkälti samalla tavalla kuin muissakin järjestelmän osissa, joissa on tarvittu vastaavanlaista toimintaa, mutta sillä erotuksella, että kielitermien käännösten välinen riippuvuus määritellään vain nimen perusteella, joten käännöksillä ei tietokannassa ole suoranaista relaatiota. Toisin sanoen kielitermillä ei ole ylätason objektia, johon käännökset liittyisivät. Tällaista rakenteellista muutosta tietokantaan ei myöskään tässä tapauksessa haluttu toteuttaa, sillä se olisi vaatinut muutoksia käytännössä kaikkialle, jossa kielitermejä käytetään. Projektiin varattu aika ei olisi tähän riittänyt.

Käännösten välinen riippuvuus määriteltiinkin taustajärjestelmässä siten, että lomakkeelle ladataan kielitermit, joilla on sama nimi. Tämä ei ole paras mahdollinen tapa, sillä nimi ei ole aina yhtä yksiselitteinen kuin id, jonka perusteella suurin osa hauista tehdään. Nimellä tehtävä haku katsottiin kuitenkin tässä tapauksessa toimivaksi kompromissiksi. Liitteessä 1 on ote kielitermeihin liittyvän toimintalogiikan sisältävän luokan lisäyksistä.

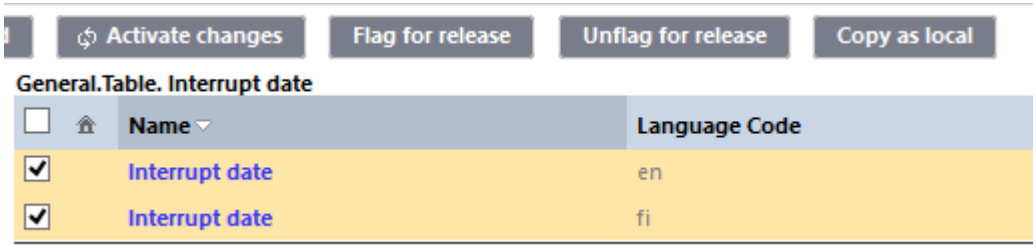
Ongelmia aiheutti linkitetystä tietokannasta haetut kielitermit, sillä niiden käsittelyä ei määritelty määrittelyvaiheessa riittävän tarkasti. Ratkaisematta oli se, miten järjestelmän tulisi toimia sellaisessa tilanteessa, jossa ollaan tallentamassa lomaketta, jossa osa käännöksistä on paikallisia ja osa tulee jaetusta tietokannasta. Toisaalta voidaan kysyä, pitäisikö tällä lomakkeella pystyä käsittelemään kielitermiä, jonka kaikki käännökset ovat peräisin jaetusta tietokannasta.

Projektille varattu aika oli tässä vaiheessa jo melkein käytetty, joten ongelma päätettiin ratkaista siten, että monen käännöksen muokkauslomake näyttää vain ja ainoastaan

paikalliset käännökset. Jaetut käännökset tulisi ensin kopioida paikallisiksi sitä varten tehdyllä työkalulla, joka oli myös tarkoitus toteuttaa.

5.4.4 Työkalu jaettujen käännösten paikallisiksi kopioimista varten

Lopuksi oli tarkoitus vielä toteuttaa yksinkertainen työkalu, jonka avulla voidaan kopioida jaetusta tietokannasta haetut käännökset paikallisiksi. Tämä oli tarkoitus toteuttaa uutena painikkeena käyttöliittymän työkalurivillä, jota painamalla valitut käännökset kopioituvat paikalliseen tietokantaan (Kuva 18).

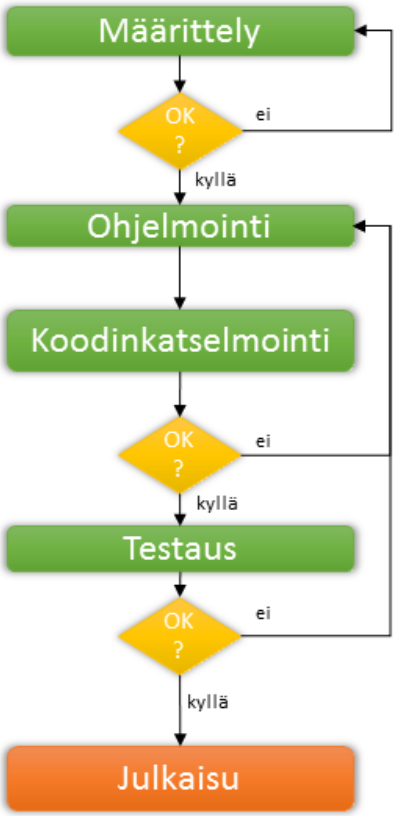


Kuva 18. Luonnos painikkeesta, jolla jaetut käännökset kopioidaan paikallisiksi

Tätä toimintoa käytettäessä kaikkien valittujen käännösten tulee olla jaettuja, jotta turhia duplikaatteja ei syntyisi. Käytännössä tämä toteutettiin hyvin pitkälti samanlaisella logiikalla kuin käännösten poistamistoiminto: ainoa ero oli se, että valitut käännökset tallennetaan paikalliseen tietokantaan sen sijaan, että ne poistettaisiin sieltä.

5.5 Julkaisuprosessi

Tämänkin projektin yhteydessä käytettiin normaalia yrityksessä käytettävää julkaisuprosessia, jossa kaikki uudet ominaisuudet ensin katselmoidaan ja testataan määrittelyvaiheessa määritellyllä tarkkuudella (Kuva 19). Tämän tarkoituksena on pyrkiä löytämään suurin osa ongelmista ennen julkaisua, sillä tässä vaiheessa virheet on helpompi korjata ja toisaalta julkaisun aiheuttamista ongelmista voi olla suurta haittaa asiakkaille.



Kuva 19. Yksinkertaistettu julkaisuprosessi

Tämä projekti päätettiin julkaista yhdessä Parametroinnin kehitysprojektin kanssa, sillä näiden toiminnot liittyivät läheisesti samaan osaan järjestelmässä. Toisaalta näiden projektien toiminnot eivät näy tavallisille käyttäjille mitenkään, joten niiden julkaisu erillään normaalista julkaisuaikataulusta oli perusteltua.

Kuitenkin erinäisten seikkojen vuoksi ensimmäisessä vaiheessa päätettiin julkaista vain kielitermien hakutoiminnot ja jaettujen kielitermien näyttäminen ja käsittely käyttöliittymässä. Monen käännöksen yhtäaikainen muokkaus sekä jaettujen käännösten kopiointi paikallisiksi päätettiin jättää tässä vaiheessa julkaisematta, sillä niiden kanssa oli vielä joitain kohtia määrittelemättä, kuten lomakkeen validointi useaa käännöstä muokatessa. Näitä ongelmia ei ehditty enää korjata julkaisuun mennessä, sillä julkaisuaikataulu oli jo lyöty lukkoon. Toisaalta nämä ominaisuudet olivat selvästi pienemmän prioriteetin parannuksia, joten niiden käyttöön saaminen ei ollut niin tärkeää.

5.5.1 Koodinkatselmointi

Koodinkatselmoinnissa koodi (ohjelmakoodi, sql-proseduurit, parametrit, enumeraatiot ja kielitermit) käydään läpi yleensä kokeneemman ohjelmoijan toimesta. Katselmoinnin tarkoituksena on löytää virheitä ja parannuksia vaativia kohtia, jotka ovat jääneet alkuperäiseltä ohjelmoijalta huomaamatta. Virheiden löytämisen lisäksi katselmointi mahdollistaa kokemattomampien ohjelmoijien kehittyä kokeneemman antaman palautteen avulla.

Jos katselmoinnissa havaitaan puutteita (virheitä tai muuta parannettavaa), palautetaan työ takaisin alkuperäiselle ohjelmoijalle, joka tekee vaadittavat korjaukset. Tämän jälkeen katselmoija tarkastaa, ovatko korjaukset riittäviä.

Tässä työssä katselmointivaiheessa kokeneemmilta ohjelmoijilta saatiin vinkkejä koodin optimoimiseen. Näiden optimointien avulla suorituskykyä saatiin parannettua jonkin verran. Samalla myös suoritettiin joitain testejä tuotantoympäristössä, minkä perusteella pystyttiin arvioimaan sitä, miten raskaita uudet hakuominaisuudet ovat käytännössä ja pystytäänkö niitä edes käyttämään. Testien perusteella haku linkitetystä tietokannasta ei ollut liian raskasta, vaikka tietokannat sijaitsisivat eri palvelimilla.

5.5.2 Testaus

Koodinkatselmoinnin jälkeen uusien ominaisuuksien toiminta testataan. Testauksessa on tarkoitus tarkistaa se, että uudet ominaisuudet toimivat määrittelyn mukaisesti ja että muutokset eivät aiheuta ongelmia muiden toimintojen kanssa. Testauksen suorittaa useimmiten testaustiimin jäsen, joka käy läpi testisuunnitelman vaiheet ja raportoi mahdolliset ongelmat.

Tässä projektissa toiminnallisen testauksen suoritti henkilö, joka ei ollut aiemmin tehnyt paljoa muutoksia kielitermeihin tai parametreihin, mutta joka tunsi parametrien ja kielitermien muokkausprosessin entuudestaan. Näin saatiin testauksen ohella myös palautetta siitä, mitenkä paljon parannusta muutoksista on tällaisia operaatioita tehtäessä. Kielitermien haku mahdollisti muokattavan kielitermin löytämisen nopeammin kuin ennen ja jaettujen kielitermien käsittely ilman tarvetta avata toista instanssia nopeutti kustomointiprosessia huomattavasti.

Samalla myös suoritettiin joitain testejä tuotantoympäristössä, minkä perusteella pystyttiin arvioimaan sitä, miten raskaita uudet hakuominaisuudet ovat käytännössä ja pystytäänkö niitä edes käyttämään. Testien perusteella haku linkitetystä tietokannasta ei ollut liian raskasta, vaikka tietokannat sijaitsisivat eri palvelimilla ja muutenkin uusista ominaisuuksista vaikutti olevan apua räätälöintiprosessissa, kun erillistä instanssia jaettuja kielitermejä varten ei tarvinnut käyttää.

5.5.3 Julkaisu

Kun uudet ominaisuudet ovat läpäisseet testauksen, kehityshaara yhdistetään päähaaraan ja samassa julkaisussa julkaistavien ominaisuuksien kehityshaarat yhdistetään erikseen luotuun julkaisuhaaraan ja samalla kehitysinstansseista kopioidaan parametrit, enumeraatiot, kielitermit ja sql-proseduurit julkaisuinstanssin tietokantaan. Tämän jälkeen julkaisuinstanssissa suoritetaan yksittäisten ominaisuuksien testaus ja niiden lisäksi tehdään myös regressiotestit, jotka pyrkivät kattamaan mahdollisimman suuren osan järjestelmän yleisimmin käytetyistä ja kriittisimmistä toiminnoista, kuten laskutuksen. Tämän tarkoituksena on varmistaa se, että uudet toiminnot eivät aiheuta ongelmia yhdessä muiden uusien ominaisuuksien kanssa.

Tässä projektissa ongelmia aiheutti se, että projekti liittyi läheisesti toiseen projektiin, jonka kanssa osa lähdekoodista oli yhteistä. Molempien projektien kehityshaarat olivat kuitenkin luotu suoraan päähaarasta, mistä seurasi se, että näiden yhdistäminen piti tehdä peräkkäin. Tämä tapahtui siten, että ensin ensimmäinen haara, jossa oli myös kaikki yhteinen koodi, piti katselmoida ja yhdistää päähaaraan ja tämän jälkeen toinen haara päivitettiin päähaarasta, katselmoitiin ja yhdistettiin takaisin päähaaraan. Tämä hidasti jonkin verran julkaisua, ja se olisi voitu välttää siten, että päähaarasta olisi luotu ensin yhteinen kehityshaara, jonka pohjalta luotaisiin lopulliset kehityshaarat.

6 Yhteenveto

Työssä kehitettiin PlanMill-toiminnanohjausjärjestelmässä olevia räätälöintityökaluja kielitermien osalta. Myös enumeraatiotyökalujen kehitystä suunniteltiin, mutta tämä jätettiin pois lopullisesta projektista pienen prioriteettinsa takia. Loppuen lopuksi projektin tuloksena kielitermit saatiin lisättyä järjestelmän yleiseen hakutoimintoon ja ennen kaikkea siten, että myös jaetusta tietokannasta haetut kielitermit ovat mukana.

Projekti lähti käyntiin hieman verkkaisesti alkututkimuksen ja erilaisten kehitysideoiden pohtimisen kanssa. Kun suurin osa luonnoksista oli tehty, saatiin päätettyä todelliset kehityskohteet ja nämä voitiin jakaa pienempiin tehtäviin. Samalla tehtäville myös annettiin jonkinlaiset työmääräarviot, joiden pohjalta pystyttiin valitsemaan sopiva määrä ominaisuuksia toteutettaviksi.

Käytännön toteutus lähti käyntiin nopeasti ja ensimmäiset, parametroinnilla ja sql-lauseilla toteutettavat, ominaisuudet saatiin toimimaan suhteellisen nopeasti. Tähän vaikutti se, että olin ehtinyt perehtyä näihin osa-alueisiin projektia edeltäneen harjoitusjakson aikana. Enemmän haasteita aiheuttivat taustajärjestelmään tehtävät muutokset, sillä tämä osa järjestelmää oli, pintaraapaisua lukuun ottamatta, entuudestaan tuntematon. Vaikka verkkosovelluksen toimintaperiaatteet olivatkin osin hallussa, oli silti uuteen järjestelmään tutustuminen kohtuullisen hankalaa, etenkin kun dokumentaatio oli osittain melko niukkaa. Onneksi järjestelmän hyvin tuntevilta kokeneilta senioriohjelmoijilta sai apua järjestelmän ymmärtämiseen ja toisaalta koko projektin ansiosta taustajärjestelmän toiminnasta oppi todella paljon, mistä on ollut hyötyä myöhemmissä projekteissa.

Parannettavaa oli vielä etenkin työmäärien arvioinnissa, mikä oli toisaalta ymmärrettävää, sillä oltiin tekemässä muutoksia järjestelmään, josta ei ollut vielä kovin hyvää tuntemusta ja toisaalta osa ominaisuuksista oli jollakin asteella kokeellisia. Myös määrityksien tekemisessä olisi pitänyt noudattaa suurempaa tarkkuutta. Näiden seikkojen vuoksi osa ominaisuuksista jouduttiin jättämään pois julkaisusta.

Kokonaisuutena pidin projektia onnistuneena, sillä ne ominaisuudet, jotka julkaistiin, osoittautuivat toimiviksi, ja niistä on ollut paljon hyötyä järjestelmän räätälöintiin liittyvissä tehtävissä. Projektin ohessa saatu oppi itse järjestelmästä, sekä työskentelystä tämän kaltaisissa kehitysprojekteissa oli myös todella arvokasta.

Lähteet

Esselink, Bert. 2000. A Practical Guide to Localization. John Benjamins Publishing Company, Amsterdam. 497 s. ISBN 978-902-72-1956-5.

int, bigint, smallint, and tinyint (Transact-SQL). (2014). Verkkodokumentti. Microsoft TechNet (Sql Server). <http://technet.microsoft.com/en-us/library/ms187745%28v=sql.100%29.aspx>. Luettu 21.4.2014.

Internationalization. (2014). Verkkodokumentti. The Java Tutorials. <http://docs.oracle.com/javase/tutorial/i18n/intro/index.html>. Luettu 21.4.2014.

Järvi, Antero, J. Karttunen, T. Mäkilä & J. Ipatti. 2011. SaaS-käsikirja. Painosalama Oy, Turku. 106 s. ISBN 978-951-29-4557-3

Nginx. (12.4.2014). Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Nginx>. Luettu 21.4.2014.

Niinioja, Olli. 2012. Lokalisointi- ja valuuttaparituen toteutus PlanMill-toiminnanohjausjärjestelmään. Insinöörityö. Metropolia Ammattikorkeakoulu, Helsinki 45 s.

Transact-SQL. (20.3.2014). Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Transact-SQL>. Luettu 21.4.2014.

UTF-8. (20.4.2014). Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/UTF-8>. Luettu 21.4.2014.

Osa muokatusta PMLanguage-luokasta

```
/**
 * Loads single language string into form for editing. If ID is -1
 * empty form is opened.
 * If id is negative it implicates that it is coming from share.
 *
 * @param scont Session container
 * @param data Action parameters
 * @return XML data
 * @throws Exception
 */
private TreeNode loadSingleLanguage(SessionContainer scont,
Map<String, String> data) throws Exception {
    String category = CastUtil.getString(data, PMConst.GUI_CATEGORY);
    CastUtil.getString(data, DBConst.TABLE_LANGUAGE + '.' +
DBConst.LANGUAGE_NAME);
    int id = CastUtil.getInt(data, DBConst.LANGUAGE_ID, -1);
    id = id < -1 ? -id : id; // Negative IDs are from share
    int local = CastUtil.getInt(data, ParamConst.LOCAL, 1);

    LockControl.checkObjectLock(id, scont);

    FormBuilder builder = new FormBuilder(scont, data);
    TreeNode root = builder.build(category);

    if (errorMessages != null && errorMessages.size() > 0) {
        builder.populateError(root, errorMessages, data);
        return root;
    }
    if (id != -1) {
        String sql = ParamControl.getUserParameter(scont,
SQL_LOAD_SINGLE_LANGUAGE_FORM);
        sql = SQLHelper.replacePlaceHolders(scont, sql, null);
        sql = ModuleHelper.getSharedDatabase(scont, sql);
        builder.populate(root, sql, id, local);
    }
    return root;
}

/**
 * Deletes list of languages. If passed data contains IDs from share
 * (any negative IDs) delete query does not get executed
 * and error message is added to session container.
 *
 * @param scont Session container
 * @param data Action parameters
 * @throws Exception
 */
private void deleteLanguage(SessionContainer scont, Map<String,
String> data) throws Exception {
    String sql = ParamControl.getUserParameter(scont,
SQL_DELETE_LANGUAGE);
    String ids = data.get(DBConst.LANGUAGE_ID);

    removeDoNotLoadTop(scont);
```

```
    boolean isSharedFound = false;
    for (String id : ids.split(SystemConst.STRING_DELIMITER)) {
        if (Integer.parseInt(id) < 0) {
            isSharedFound = true;
            break;
        }
    }
    if (isSharedFound) {
        String error = ParamControl.getLangString(scont,
LangConst.LANG_SHARED_CANNOT_BE_DELETED);
        scont.setError(error);
    }
    else {
        ModuleHelper.exec(sql, ids, null);
    }
}
```