Suunnitteludokumentti

Kohahdus

Helsinki 20.11.2006

Ohjelmistotuotantoprojekti

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Kurssi

581260 Ohjelmistotuotantoprojekti (6 ov)

Projektiryhmä

Taro Morimoto, Projektipäällikkö
Tuomas Palmanto, Vaatimusmäärittelyvastaava
Mikko Kinnunen, Suunnitteluvastaava
Markus Kivilä, Koodivastaava
Jari Inkinen, Testausvastaava
Paula Kuosmanen, Dokumenttivastaava

Asiakas

Teemu Kerola

Johtoryhmä

Sanna Keskioja

Kotisivu

`http://www.cs.helsinki.fi/group/kohahdus`

Versiohistoria

Versio	Päiväys	Tehdyt muutokset
1.0	12.10.2006	Ensimmäinen versio
1.1	19.10.2006	Pikkuvirheitä ja sanamuotoja korjattu
2.0	15.11.2006	Toisen iteraation dokumentti FTR:ää varten
2.1	20.11.2006	Lopullinen dokumentti

Sisältö

1 Johdanto

TitoTrainer on järjestelmä automaattisesti tarkastettavien TTK-91-konekielen harjoitustehtävien luomiseen ja ratkaisemiseen. Järjestelmä on tarkoitettu käytettäväksi opetuksen tukena, opetettaessa Tietokoneen toiminta -kurssia. Tietojenkäsittelytieteen opettajat voivat tehdä järjestelmään uusia tehtäviä ja määritellä kuinka ne tarkastetaan automaattisesti. Tietokoneen toiminta -kurssin opiskelijat ja kurssin tehtävistä kiinnostuneet itseopiskelijat voivat ratkaista tehtäviä ja saada palautetta niiden onnistumisesta.

Painopiste projektissa on opettajan ja opiskelijan käyttöliittymillä. Käyttöliittymistä tehdään mahdollisimman selkeät ja helppokäyttöiset. Opettajan käyttöliittymän avulla määritellään tehtävät parametreineen ja kuinka opiskelijan ratkaisun oikeellisuus tarkistetaan, sekä nähdään statistiikkatietoja opiskelijoiden suorituksista. Opiskelijan käyttöliittymän avulla opiskelija valitsee tehtäviä, syöttää niiden ratkaisut ja saa palautetta vastauksensa oikeellisuudesta.

Projekti käyttää valmiina olevaa Titokone-simulaattoria harjoitustehtävien ratkaisemiseen. Projektin rakennetaan olemassa olevaan eAssari-kehykseen, joka sopii geneeriseen automaattisesti tarkistettavien tehtävien määrittelyyn ja toteutukseen. eAssarin keskeneräisyydestä johtuen kehystä tullaan muokkaamaan tuotteella asetettujen vaatimusten täyttämiseksi.

Järjestelmä toteutetaan Java-kielellä ja järjestelmän käyttö vaatii, että selain tukee JavaScript-kieltä ja CSS-tyylitiedostoja. Järjestelmä toimii vuoden 2006 yleisimmillä Firefox ja Internet Explorer-selainversioilla.

2 Sanasto

TTK91=Auvo Häkkisen kehittämä ohjelmointikieli, joka läheisesti muistuttaa symbolista konekieltä.

KOKSI=Auvo Häkkisen kirjoittama konekielisimulaattori, joka toteuttaa TTK-91-kielen.

Kohahdus=Syksyn 2006 ohjelmistotuotantoprojekti (tämä projekti)

TitoTrainer=Projektimme tuotos

Järjestelmä=TitoTrainer

Ohjelma=Opiskelijan kirjoittama TTK91-ohjelma, eli vastaus johonkin tehtävään

eAssari=Tietokantapohjainen ympäristö ohjelmallisesti tarkastettavien harjoitus- ja koetehtävien suorittamiseen

Titokone=Koski-nimisen Ohjelmistotuotantoprojektiryhmän vuonna 2004 rakentama järjestelmä konekielisten ohjelmien kääntämiseen ja suorittamiseen.

Koski=Vuoden 2004 Ohjelmistotuotantoprojekti joka rakensi konekielen simulaattorin ja debug-ympäristön, eli Titokoneen

Koskelo=Vuoden 2004 Ohjelmistotuotantoprojekti, joka integroi Titokoneen ja eAssarikehyksen yhteen. Ratkaisusta ei tullut kuitenkaan käyttökelpoista, eikä sitä ole otettu käyttöön.

Kriteeri=Sääntö jonka mukaan tehtävän oikeellisuus tarkistetaan. Kriteereitä voi olla monta yhdelle tehtävälle.

Aihepiiri=Tehtävälle täytyy määritellä aihepiiri, johon tehtävä kuuluu.

AJAX=Tekniikka vuorovaikutteisten verkkosovellusten luomiseen. Tekniikka vaihtaa pieniä määriä dataa palvelimen kanssa taustalla, niin ettei koko verkkosivua tarvitse ladata uudelleen joka kerta käyttäjän tehdessä muutoksen.

3 Toiminnot

Tämä luku kuvaa järjestelmän HTML-sivut ja sen, mitä toimintoja ne tarjoavät käyttäjälle.

3.1 Rekisteröityminen

Rekisteröintisivulla opiskelija rekisteröityy järjestelmän käyttäjäksi. Rekisteröinti vaatii seuraavat tiedot: etunimi, sukunimi, käyttökieli, kurssi, sähköpostiosoite, opiskelijanumero tai henkilötunnus, sekä haluttu käyttäjätunnus ja salasana

Täyttää vaatimukset: 4

3.2 Käyttäjätietojen muokkaaminen

Käyttäjätietosivulla sisäänkirjautunut käyttäjä voi muokata rekisteröintitietojaan, poislukien tunnus joka on valittu rekisteröidyttäessä. Käyttäjätietojen muokkaamisen ei tarvitse toimia opettajien tunnusten kanssa, mutta odotettavissa on, että opettajatunnusten tukeminen ei aiheuta minkäänlaista lisätyötä.

Täyttää vaatimukset: 6,8

3.3 Kirjautuminen

Kirjautumissivulla käyttäjä syöttää tunnuksensa, salasanansa ja pudotusvalikoista kurssin nimen ja käyttökielen. Kurssivalinta määrittää, että minkä kurssin suorituksiin opiskelijan tekemät vastaukset tallennetaan. Opettajan ollessa kyseessä kurssivalinnalla ei ole merkitystä. Jos kirjautuminen onnistuu, näytetään tehtävälistaus. Jos kirjautuminen epäonnistuu, käyttäjä palautetaan kirjautumissivulle.

Täyttää vaatimukset: 5,2

3.4 Kurssien ja tehtävien valinta (opettaja)

Kirjautumisen jälkeen opettajalle näytetään kurssilistaus josta opettaja voi siirtyä kurssin statistiikka-sivulle. Kurssilistauksen alla on myös lomake uuden kurssin luomista varten.

Samalla sivulla näytetään myös tehtävälista. Listassa näytetään tehtävän kieli, aihepiiri ja nimi, sekä viimeinen muokkauspäivä ja muokkaaja. Listattuja tehtäviä voi muokata ja poistaa, sekä käynnistää uuden tehtävän luomisen.

Täyttää vaatimukset: 18,21

3.5 Tehtävän määrittely (opettaja)

Tehtävänmäärittelysivulla opettaja voi luoda uuden tehtävän tai muokata olemassa olevaa tehtävää. Tehtävälle määritellään nimi, aihepiiri, tehtävänanto, tarkastustyyppi (malliratkaisu/kiinteät arvot), tehtävätyyppi (täyttötehtävä/ohjelmointitehtävä), sekä tarkastuskriteerit ja kriteerien palautteet.

Täyttää vaatimukset: 9,10,12,13,14,15,16,17,20,30-43,45-51,53,54

3.6 Kurssin statistiikka (opettaja)

Kurssin statistiikkasivulla opettajalle näytetään lista kaikkien kurssin tehtäviä suorittaneiden opiskelijoiden suorituksista. Listassa näytetään opiskelijan nimi, opiskelijanumero tai henkilötunnus, sekä onnistuneesti ratkaistujen tehtävien määrä. Lista on helposti tulostettavassa muodossa.

Listassa opiskelijan nimi toimii linkkinä ko. opiskelijan käyttäjätietosivulle.

Täyttää vaatimukset: 23

3.7 Opiskelijan tiedot (opettaja)

Käyttäjän statistiikkasivulla opettajalle näytetään opiskelijan tarkemmat tiedot. Henkilötiedoista näytetään salasanaa lukuunottamatta kaikki rekisteröintitiedot. Sivu sisältää myös napin, jota käyttämällä opettaja voi poistaa käyttäjän järjestelmästä.

Täyttää vaatimukset: 22

3.8 Opiskelijan haku (opettaja)

Hakusivulla opettaja voi etsiä nimen perustella käyttäjiä, ja siirtyä käyttäjätietosivulle tarkastelemaan käyttäjän tarkempia tietoja.

3.9 Tehtävien valinta (opiskelija)

Kirjautumisen jälkeen opiskelijalle näytetään tehtävälista, josta opiskelija voi valita tehtävän ratkaistavaksi. Listassa näytetään tehtävän kieli, aihepiiri ja nimi, sekä tieto onko opiskelija yrittänyt jo tehtävän suoritusta, yritysten määrä ja onko yritys onnistunut.

Täyttää vaatimukset: 19(=25),29

3.10 Tehtävään vastaaminen (opiskelija)

Opiskelijan ratkaistavaksi valitsemasta tehtävästä näytetään nimi ja tehtävänanto, mahdollinen valmis koodi mikäli kyseessä on täyttötehtävä, sekä tekstialueet johon opiskelija

kirjoittaa ohjelman ja sille annettavat syötteet. Lisäksi on linkki jolla pääsee takaisin tehtävälistaukseen.

Kun opiskelija on kirjoittanut vastauksensa, hän painaa "Suorita-nappia joka vie takaisin tehtävävastaussivulle. Sivulla näytetään samat tiedot kuin alussa, mutta lisäksi näytetään palautetta ratkaisun onnistumisesta/epäonnistumisesta, sekä Titokoneen suorituksen lopputila.

Täyttää vaatimukset: 9,10,26,27,28,52

3.11 Tehtävien aihepiiri (opettaja)

Tehtäville voidaan määritellä aihepiiri (category). Tämä yksittäiseen taskiin liittyvä tieto tallennetaan task tauluun taskmetadata-kenttään xml-muodossa muiden attribuuttien mukana. Aihepiirilista tallennetaan attributevalues-tauluun objecttype arvolla "Q". Tehtävän luomisessa aihepiiri valitaan aihepiirilistasta, joka on HTML-sivulla esitettynä pudotusvalikkona. Pudotusvalikon viimeinen valinta on uuden aihepiirin luonti, jonka valitsemalla kutsutaan AJAX-tyyppisesti uutta JSP-sivua, joka luo uuden aihepiirin kantaan (käyttäen DBHandleria). Tämä uusi aihepiiri lisätään myös aukiolevan tehtävänluontisivun aihepiirilistapudotusvalikkoon.

Aihepiirien hallinta on toteutettu erillisellä JSP-sivulla, jossa on lista aihepiireistä. Näitä aihepiirejä voi poistaa, muokata ja lisätä painamalla vastaavia HTML-linkkejä.

Täyttää vaatimukset: 53, 54, 55, 56

3.12 Monikielisyys

Opiskelijan käyttöliittymä toteutetaan suomen- ja englanninkielisenä. Opiskelija valitsee rekisteröitymisen yhteydessä järjestelmän oletuskielen. Kirjautuessaan järjestelmään opiskelija voi halutessaan vaihtaa kielen toiseksi.

Opettajan käyttöliittymä toteutetaan ainoastaan englanniksi.

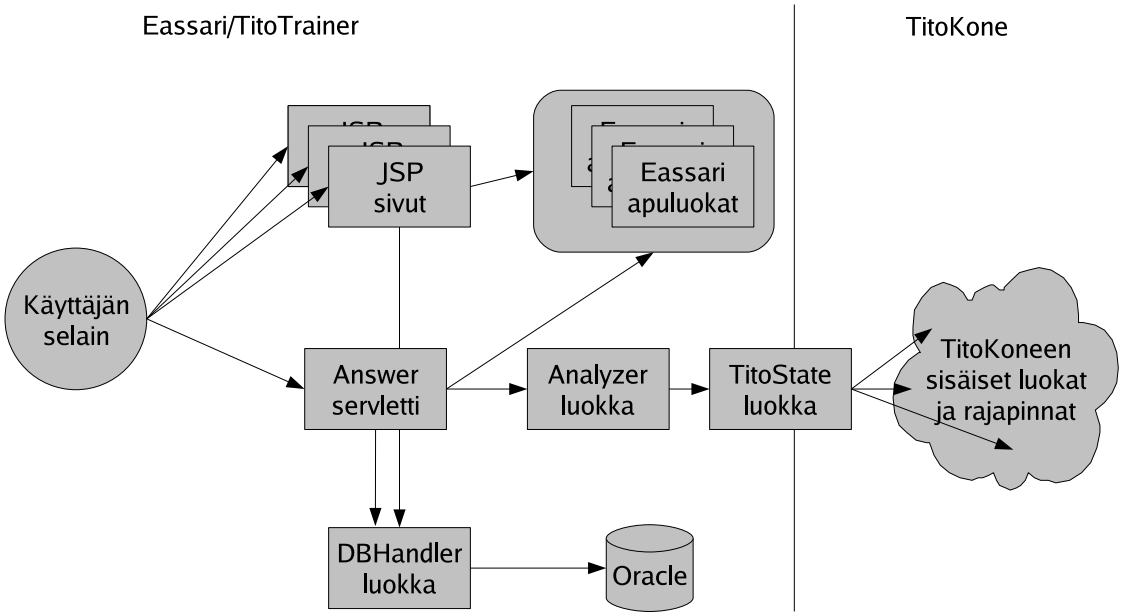
Täyttää vaatimukset: 57,58

4 Arkkitehtuurisuunnitelma

Järjestelmä on jaettu kahteen osajärjestelmään, Titokone ja eAssari/TitoTrainer. Ideaalitapaus olisi ollut kolme osajärjestelmää, jossa eAssari on TitoTraineria kutsuva sovelluskehys. Tämä malli ei kuitenkaan ole mahdollinen, johtuen eAssarin keskeneräisyydestä. Lopputuotteelle asettujen vaatimusten täyttäminen vaatii huomattavia lisäyksiä eAssariin, mm. puuttuvat kurssien ja käyttäjien hallinta täytyy toteuttaa. Kireän aikataulun vuoksi eAssaria ei yritetä säilyttää geneerisenä kehyksenä, vaan muokkaukset tehdään vain Kohahdus projektin tarpeita ajatellen. Käytännössä eAssarista tehdään siis ns. code fork, eli synnytetään täysin TitoTrainer-spesifinen eAssari-versio, jonka ei taata olevan yhteensopiva geneerisen eAssarin kanssa.

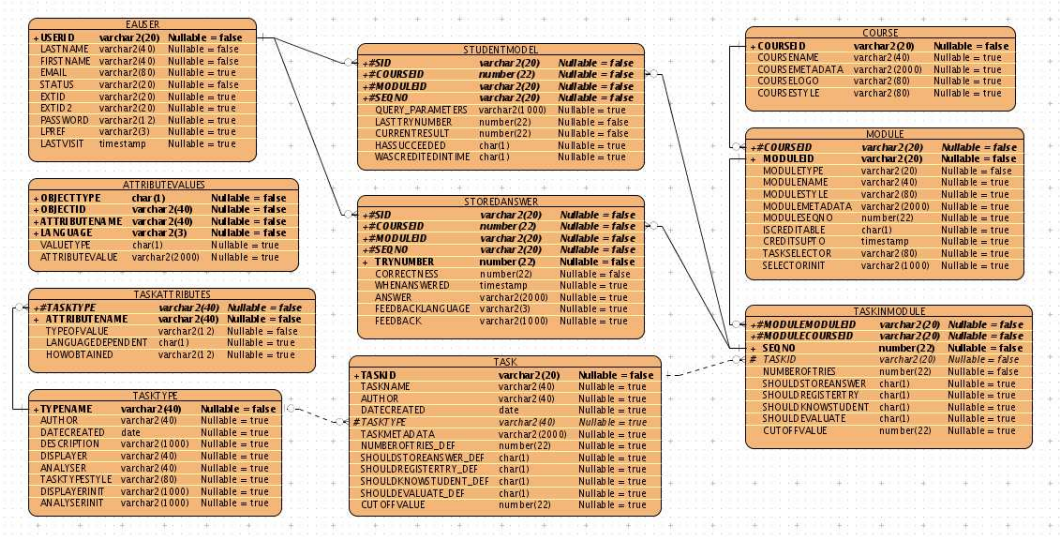
Ainoa osuus eAssarista jota pyritään käyttämään sen alkuperäisen arkkitehtuurin mukaan, on tietokanta. Tietokannan rakennetta ei muuteta, mutta kaikkia tauluja ei käytetä (kts. tämän dokumentin luku Tietokanta).

TitoKonetta käytetään erikseen luotavan TitoState luokan kautta, jotta yhteys TitoKoneeseen saadaan mahdollimman kevyeksi. TitoState käyttää suoraan TitoKoneen konkreettisia luokkia, joita myös tarpeen tullen muokataan kaikkien suoritustietojen saamiseksi. Koska muokkauksien tarvetta ja laajuutta on vaikea arvioida etukäteen, niiden yksityiskohdat jätetään täysin TitoState luokan implementoijan vastuulle.



Kuva 1: Arkkitehtuuri

5 Tietokanta



Kuva 2: Tietokantakaavio

Tietokannan rakenteen määrää eAssari kehys. Rakenne on kuvattu kuvassa 1. Ajanpuutteesta johtuen module-tietokantataulua ei tulla käyttämään sen mahdollistamassa laajuudessa. Sen sijaan että kursseille luotaisiin useita moduuleja, luodaan jokaiselle kurssille vain yksi moduuli ja lisätään kaikki tehtävät sinne. Näin vältetään tarve erilliselle moduulienhallinta toiminnolle ja käyttöliittymälle.

Tietokantaa käytetään aina DBHandler-nimisen singleton-luokan kautta, joka piilottaa tietokantarakenteen yksityiskohdat. SQL-lauseita tulee siis vain DBHandler luokkaan eikä muualle koodiin. DBHandler-luokasta voidaan luoda instanssi staattisen DBHandler.getInstance() -metodin avulla. Tämä tarkoittaa, että ko. luokkaa voidaan käyttää virtuaalikoneen sisällä mistä vaan.

5.1 Tasktype

Tasktype on eAssari-kehyksen tarvitsema taulu, joka määrittelee tehtävätyypin käyttämät Analyzer ja Displayer luokat. Tauluun olisi mahdollista tallentaa myös muuta tehtävätyyppikohtaista tietoa, mutta tälle ei ole näkyvissä tarvetta.

5.2 Task

Task määrittelee tehtävän. Task-tauluun tallennetaan kriteerietä lukuunottamatta tehtävän kaikki tiedot. Tunniste, nimi ja muokkaustiedoille on olemassa valmiit sarakkeet, muut tiedot tallennetaan XML-muodossa taskmetadata sarakakkeeseen.

Task taulua on TitoTrainer kehityksen aikana muutettu kasvattamalla taskmetadata-kentän kokoa.

5.3 Course

Määrittelee kurssin nimen ja tunnisteen. Tähän tauluun viitataan useista muista tauluista, jotta esimerkikisi opiskelijoiden suoritukset voidaan kirjata tietylle kurssille.

5.4 Module

EAssarin tehtäviä ei liitetä suoraan kursseihin, vaan moduuleihin ja moduulit liitetään kursseihin. Kurssiin voi liittää useita moduuleja, mutta yhtä moduulia ei voi lisätä useaan kurssiin. TitoTrainer ei käytä moduuleja siinä laajuudessa kuin tietokannan rakenne mahdollistaisi, vaan toteutuksen ja käyttöliittymien yksinkertaistamiseksi jokaiselle kurssille luodaan tarkalleen yksi moduuli johon lisätään kaikki tehtävät.

5.5 Taskinmodule

Yhdistää tehtävän kurssiin ja moduuliin. Normaalitapauksessa siis tehtävä lisättäisiin jonkin kurssin johonkin moduulin luomalla rivi taskinmodule-tauluun.

TitoTrainerin tapauksessa on määritelty, että kaikki tehtävät kuuluvat kaikkiin kursseihin, joten kun opettaja luo tehtävän, TitoTrainer luo automaattisesti yhtä monta taskinmodule riviä kuin on kursseja. Samoin kun luodaan uusi kurssi, sille luodaan automaattisesti moduuli ja kaikki olemassa olevat tehtävät lisätään ko. moduuliin. Näin ollen Taskinmodule rivien määrä on aina Task rivien määrä * Course rivien määrä.

5.6 Attributevalues

Yleiskäyttöinen varasto jonne voi tallentaa mitä tahansa lisätietoa koskien mitä tahansa muuta taulua. Tänne tallennetaan tehtävien kriteerit ja aihepiirilista.

5.7 EAuser

Taulu kaikkien käyttäjätietojen tallentamiseen. Sekä oppilaat ja opettajat tallennetaan tänne, erona näillä on vain status-sarakkeen arvo.

5.8 Storedanswer

Kun opiskelija vastaa tehtävään, vastaus sekä tiedot sen oikeellisuudesta tallennetaan tähän tauluun.

5.9 Studentmodel

Jokaista tehtävän vastausyritystä kohden luodaan storedanswer-tauluun yksi rivi. Studentmodel-taluun tallennetaan yksi rivi per tehtävä, vaikka vastausyrityksiä olisi useampia. Rivillle tallennetaan tieto siitä, onko tehtävä suoritettu onnistuneesti ja kuinka monta yritystä tehtävään on käytetty. Nämä tiedot olisi mahdollista päätellä SQL:n koostefunktioden avulla storedanswer-taulusta, mutta ilmeisesti tehokkuuden parantamiksi on luotu erillinen koostetaulu.

5.10 Taskattributes

Task-attributes on eAssarissa tarkoitettu kuvaamaan tehtävien attribuuttien rakennetta. eAssari ei kuitenkaan missään vaiheessa itse hyödynnä tätä taulua, eikä TitoTrainerkään sitä tarvitse, joten taulu jää täysin käyttämättä.

5.11 Pluginparameters

Tämän taulun tarkoitus on jäänyt täysin hämärän peittoon, eAssari ei missään kohtaa koodissaan viittaa mihinkään plugineihin. TitoTrainerillä ei myöskään ole tälle taululle mitään käyttöä, joten se jää täysin käyttämättä.

5.12 Pluginparamattributes

Kuten pluginparameters, jää käyttämättä.

6 Monikielisyys

Järjestelmän monikielisyys toteutetaan helposti ylläpidettävien xml-tiedostojen avulla. Järjestelmää käynnistettäessä ladataan automaattisesti kaikki kieliriipuvaiset resurssit, jolloin ne ovat jsp-sivujen käytettävissä yksinkertaisen rajapinnan kautta. Toteutuksessa pyritään mahdollistamaan uusien kielivaihtoehtojen lisääminen vähällä vaivalla.

6.1 xml-tiedostot

Järjestelmä käyttää yksittäistä xml-tiedostoa yhden jsp-sivun sisältämän kieliriippuvaisen datan tallentamiseen. Sivukohtaisen tiedoston rakenne on seuraava:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment></comment>
<entry key="kentänNimi_FI">kentän sisältö suomen kielellä</entry>
<entry key="kentänNimi_EN">field content in english</entry>
</properties>
```

Kielikohtainen data lisätään <entry> tagien avulla. Key arvoksi asetetaan resurssin nimi ja suffiksiksi kieli (_FI, _EN). JSP sivulla näkyvä sisältö kirjoitetaan tagien väliin.

Sivukohtaisten tiedostojen hallinnointi keskitetään xml-muotoiseen konfiguraatiotiedostoon. Tässä tiedostossa määritellään järjestelmää käynnistettäessä ladattavat resurssit. Konfiguraatiotiedoston rakenne on seuraava:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment></comment>
<entry key="jsp-sivu1">/tiedosto/polku/tdsto_nimi.xml</entry>
<entry key="jsp-sivu2">/tiedosto/polku/tdsto_nimi.xml</entry>
</properties>
```

6.2 Monikielisyyden hallinnointi

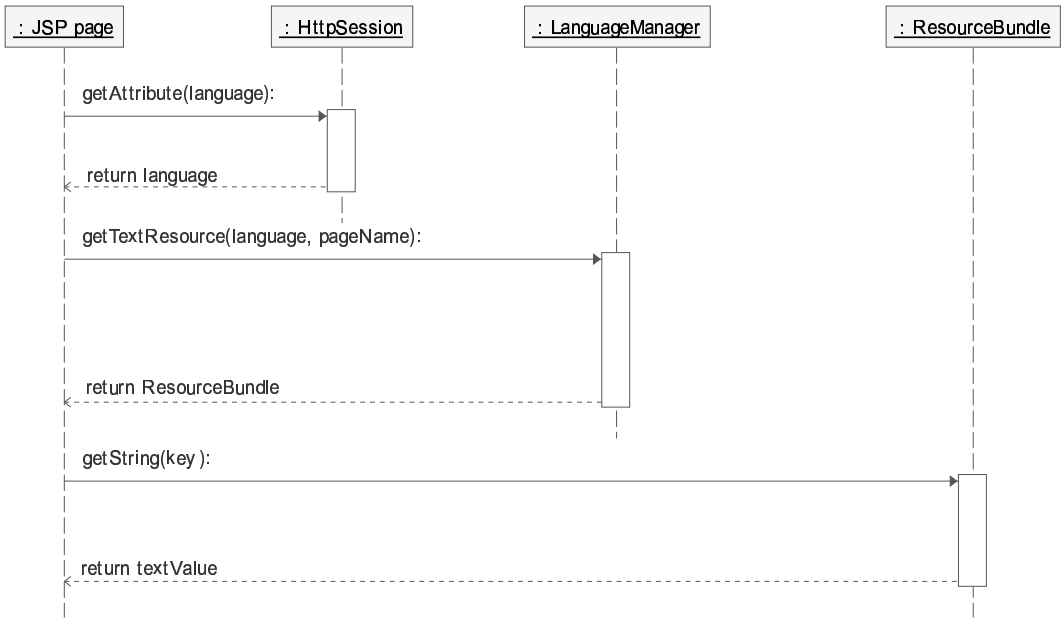
Kielipakettien hallinnointiin luodaan luokka LanguageManager. Luokka lukee järjestelmän käynnistymisen yhteydessä ylläkuvatun xml-konfiguraatiotiedoston ja alustaa siinä määritellyt kielipaketit. Lisäksi luokka tarjoaa rajapinnan kielipakettien hakuun.

Yksittäisen kielipaketin toteuttaa luokka TitoBundle, joka periytyy java.util.ResourceBundle luokasta. ResourceBundle luokkaa ei tosin käytetä täysin API:n mukaisesti, koska kielipakettien lataaminen hoidetaan sen sijasta LanguageManager luokan avulla. TitoBundle

luokka säilöö yksittäisen xml-tiedoston sisällön hajautustauluun java.util.Properties luokan avulla.

6.3 Monikielisyyden käyttö jsp-sivulta

Kielivaihtoehtojen lisääminen jsp-sivuille hoituu seuraavasti. Sivua ladattaessa tarkistetaan sessiosta haluttu kieli ja pyydetään LanguageManager luokalta kielipakettia getTextResource metodilla. Parametreina välitetään kieli ja jsp-sivun nimi. Metodin palauttaman kielipaketin metodilla getString voidaan hakea tarvittava kieliriippuvainen data.



Kuva 3: Kielipaketin käyttö jsp-sivulta

7 Luokat

7.1 Criterion

public abstract class Criterion

Base class for all criterion types. Each task has a fixed number of criteria for analyzing the student's solution. Criterion may be used to pass/fail the solution, evaluate the quality of the solution, or do both/neither.

Different types of criteria are implemented as Criterion-subclasses. Task composer used for creating new tasks uses mostly the interfaces defined here, but also may use some methods specifc to the subclass. Task analyzer component used for checking the student's solution uses only this abstract interface. The analyzer is not aware of the details of the subclasses.

This class and all sub-classes shall provide following quarantees:

- **getters always return Strings, regardless of the field type**
- **getters never return null, but they may return empty strings**
- **setters always take Strings, regarless of the field type. Invalid strings, (eg. non-numeric string for numeric field), empty strings, and null values are acceptable and will either clear the field or set the field to some default value.**
- **neither getter nor setters will ever throw exceptions**
- **Criterion objects are never in an invalid state. This is done by setting and validating mandatory fields in the constructor. However, Criterion object deserialized from the database are not subject to validation, they are assumed to be always valid.**

```
Criterion (abstract)
+- InstructionCriterion (abstract)
|  +- ForbiddenInstructionsCriterion
|  +- RequiredInstructionsCriterion
+- MeasuredCriterion (abstract)
|  +- CodeSizeCriterion
|  +- DataAreaSizeCriterion
|  +- DataReferencesCriterion
|  +- ExecutetionStepsCriterion
|  +- MemReferencesCriterion
|  +- StackSizeCriterion
+- ScreenOutputCriterion
+- VariableCriterion (abstract)
   +- RegisterCriterion
   +- SymbolCriterion
```

For sake of brevity, Criterion subclasses are not described in this document.

`protected static final long UNDEFINED = Long.MIN_VALUE;`
Special case for signaling undefined numeric value. Criteria that deal with numeric types (such as RegisterCriterium) need all 32-bits of integer, but we also need a way to represent undefined (ie. null) values. Best-but-still-ugly solution is to use longs for all numeric types and use a special signal value for null.

`protected Criterion()`
Empty constructor for deserialization

`protected Criterion(String id, boolean usesSecretInput)`
Constructor to initialize mandatory data members of Criterion.
`@param id` Identifier that can used to distinguish the different criteria of one task
`@param usesSecretInput` true if criterion is to be used in conjunction to secret input

`public String getId()`
Return the identifier of this Criterion

`public abstract String getName(ResourceBundle languageBundle)`
Return human-readable name of this criterion
`@param languageBundle` Bundle containing language-dependant parts of crierion names

`public boolean isSecretInputCriterion()`
Return true if this criterion is to be used with secret input

`public String getFailureFeedback()`
Return feedback string used solution attempts that do not pass the acceptance test

`public String getAcceptanceFeedback()`
Return feedback string used for solution attempts that pass the acceptance test but not the quality test

`public String getHighQualityFeedback()`
Return feedback string used for for solution attempts that pass pass the acceptance test, and also the quality test if one is defined.

`public String setFailureFeedback()`
Set failure feedback

`public String setAcceptanceFeedback()`
Set acceptance feedback

`public String setHighQualityFeedback()`
Set high quality feedback

`public abstract boolean hasAcceptanceTest(boolean usingModelAnswer)`
Return true if this criterion has test for evaluating failure/success of the student's answer. Return of false means this criterion should NOT be used to test passesAcceptanceTest(..)
`@param usingModelAnswer` True if the inspection method for this task is model-answer

`public abstract boolean passesAcceptanceTest(TitoState studentAnswer, TitoState modelAnswer)`
Return true if student's solution meets the passing requirement of this Criterion. The behaviour of this method is undefined if called despite a false return from hasAcceptanceTest(..)
`@param studentAnswer` end state of TitoKone for student's answer
`@param modelAnswer` end state of TitoKone for teacher's answer, or null

`public abstract String getAcceptanceTestValue()`
Return the value the student's answer will be compared to

`public abstract void setAcceptanceTestValue(String test)`
Set the value the student's answer will be compared to

`public boolean hasQualityTest(boolean usingModelAnswer)`
Return true if this criterion has test for evaluating quality of the student's answer. Criterion class provides a default implementation that always returns false

`public boolean passesQualityTest(TitoState studentAnswer, TitoState modelAnswer)`
Return true if student's solution meets the quality requirement of this Criterion. Criterion class provides a default implementation that always returns false

`public String getQualityTestValue()`
Return the value the student's answer will be compared to. Criterion class provides a default implementation that always returns an empty string

`public abstract void setQualityTestValue(String test)`
Set the value the student's answer will be compared to. Criterion class provides a default implementation that is a no-op

`public String serializeToXML()`
Return a serialized copy of this Criterion in XML-format

`protected abstract String serializeSubClass();`
Serialize non-static data-members of Criterion sub-class to XML format. The subclass can freely decide the names of the XML tags. The abstract Criterion class will handle the serialization of its data-members. The serialized string is stored in the eAssari database in a 2000-char field so subclasses should try to keep the tags and data short (without being cryptic).

`protected abstract void initSubClass(String serializedXML);`
Initialize non-static data-members of this Criterion subclass instance using the serialized representation returned by serializeToXML. Data-members of the abstract Criterion class will have already been deserialized when this method is called.

`public static Criterion deserializeFromXML(String xml)`
Instantiate new Criterion object using the serialized form Xml

`protected static String toXML(String tagname, boolean value)`
Serialize boolean value to XML string. Helper function for serializeSubClass()

`protected static String toXML(String tagname, String value)`
Serialize String value to XML string. Helper function for serializeSubClass()

`protected static String toXML(String tagname, long value)`
Serialize integer value to XML string. Helper function for serializeSubClass()

`protected static boolean parseXMLBoolean(String XML, String tagname)`
Deserialize boolean value from XML string. Helper function for initSubClass()

`protected static String parseXMLString(String XML, String tagname)`
Deserialize String value from XML string. Helper function for initSubClass()

`protected static int parseXMLLong(String XML, String tagname)`
Deserialize integer value from XML string. Helper function for initSubClass()

7.2 DBHandler

public class DBHandler

Singleton class used for database interactions. Each public method of DBHandler class encapsulates one database transaction, and thus may cause multiple inserts/updates/removes with one call. The atomicity of the operations is quaranteed by using the transaction model provided by the SQL standard.

`private DBHandler()`
Prevent outside instatiation by giving constructor private scope

`public static DBHandler getInstance()`
Return DBHandler instance

`private boolean init()`
Initialize db connection pool

`protected void release(Connection conn) throws SQLException`
Closes db connection

`protected Connection getConnection () throws SQLException`
Load database driver if not already loaded

`public LinkedList<Course> getCourses() throws SQLException`
Returns all courses

`public synchronized boolean createCourse(Course course) throws SQLException`
Adds new course to database

`private boolean addCourse(Course course) throws SQLException`
Add new course to database. Does not check whether the course already exists in the DB.

`public boolean removeCourse(String courseID) throws SQLException`
Remove course from database (and thus all refering taskinmodules).

`private boolean addModule(String courseID) throws SQLException`
Add a default module to course.

`private boolean addTaskInModule(String courseID, String taskID) throws SQLException`
Adds a taskinmodule entry so that the task is linked with the course.

`public LinkedList<Task> getTasks() throws SQLException`
Return all tasks

`public Task getTask(String taskID)`
Return task identified by taskID

`public synchronized boolean createTask(Task task, List<Criterion> criteria) throws SQLException`
Add new task to task database. The insert will affect all courses. This operation will also create the criteria for the task

`public boolean addTask(Task task) throws SQLException`
Adds a task to DB without any linkage to courses and modules

`public boolean updateTask(Task task, List<Criterion> criteria) throws SQLException`
Update existing task. The update will affect all courses. This operation will also update the criteria of the task.

`private boolean updateTask(Task task) throws SQLException`
Updates a task to DB without modifying any courses or modules

`public boolean removeTask(Task task)`
Remove task from task database (and thus all courses). This will also remove all stored answers of the task.

`private boolean addCriterion(Task t, Criterion c)`
Add criterion c to task. Helper for addTask(..) and updateTask(..)

`private boolean updateCriterion(Task t, Criterion c) throws SQLExcept`
Updates criterion c of task

`private void removeCriteria(Task t)`
Remove criteria form task. Helper for removeTask(..) and updateTask(..)

`public LinkedList<Criterion> getCriteria(Task task) throws SQLExcepti`
Return the criteria of task

`public CriterionMap getCriteriaMap(Task task) throws SQLException`
Return the criteria of a task in a map

`public User[] getUsers(Course course) throws SQLException`
Return all users who have attempted to solve at least one task of Course

`public User getUser(String userID, String password) throws SQLExcepti`
Return user identified by userID and password (login)

`public User getUser(String userID) throws SQLException`
Return user identified by userID

`public boolean createUser(User user)`
Add new user to user database. Does not check weather the user already exists in the DB.

`public boolean updateUser(User user)`
Update existing user to DB with userID. Does not check weather the user exists or not.

`public void removeUser(String userID)`
Removes all student's answers and records from database (studentmodel, storedanswer and eauser).

`public String[][] getAnswerStatistics(String courseID)`
Returns a two dimensional matrix of stored answers (table studentmodel) of tasks and student names.

`public User[] searchStudent(String name)`
Returns a list of students (Users) that matches the param name with first or lastname in the eauser table.

`public LinkedList getStudentAnswers(String userID)`
Returns a list of tasks that student has answered. Each task on the list is a hashtable with columns as entries.

```
public String storeStateAndAnswer(boolean shouldStore, User student,
String courseId, String taskID, String[] answers, boolean accepted,
String lang, Feedback feedback) throws SQLException
```
Stores the answer of the student into database table storedanswer. Also stores the state of the student's answered task into table studentmodel.

7.3 Task

public class Task

Modify existing Task class by adding the following methods

`public Task()`
Creates a new instance of Task

`public Task(String taskID)`
Creates a new instance of Task

`public setTaskID(String taskID)`
Sets ID of task

`public setLanguage(String lang)`
Set the preferred language of this user. The language is either "EN"or "FI

`public void setName(String name)`
Set the name of this task

`public void setAuthor(String name)`
Set "last task modification by"attribute to Name, set last-modification-timestamp to current date and time

`public void setDescription(String desc)`
Set description (tehtävänanto) of this task

`public void setPublicInput(String input)`
Set the public input as String of this task

`public void setSecretInput(String input)`
Set the secret input as String of this task

`public void setModelAnswer(String code)`
Set model answer code

`public void setCategory(String categ)`
Set task category of this task

`public void setTitoTaskType(String type)`
Set task type of this task

`public void setFillInPreCode(String code)`
Set the code that is prepended before student's code in a fill-in task

`public void setFillInPostCode(String code)`
Set the code that is appended to student's code in a fill-in task

`public void setFillInTask(boolean fillIn)`
Set this task as fill-in or create-full-program

`public void setNoOfTries(int noOfTries)`
Sets number of tries for student

`public void setValidateByModel(boolean useModel)`
Set the validation method of this task

`public void setModificationDate()`
Set the date and time this task was last modified

`public void setHasSucceeded(boolean hasSucceeded)`
Set the hasSucceeded true or false depending the student has passed the task

`public void setPassFeedBack(String pass)`
Set return feedback if student passes the task

`public void setFailFeedBack(String fail)`
Set return feedback if student fails the task

`public void setMaximumNumberOfInstructions(int num)`
Endless loop prevention

`public void setMaximumNumberOfInstructions(String num)`
Endless loop prevention

`public String getTaskID()`
Return id of this task

`public String getLanguage()`
Return the language of this task

`public String getName()`
Return the name of this task

`public String getAuthor()`
Return name of the last person who has modified this task

`public String getDescription()`
Return the description (tehtävänanto) of this task

`public String getPublicInput()`
Return the public input as String of this task

`public String getSecretInput()`
Return the secret input as String of this task

`public String getModelAnswer()`
Return code of the model answer provided by teacher

`public String getCategory()`
Return the task category of this task

`public String getTitoTaskType()`
Return the task type of this task

`public String getFillInPreCode()`
Return the code that is prepended before student's code in a fill-in task

`public String getFillInPostCode()`
Return the code that is appended to student's code in a fill-in task

`public int getNoOfTries()`
Return the number of tries student has tried to make this task

`public boolean isFillInTask()`
Return true if this is a fill-in task

`public boolean isProgrammingTask()`
Return true if this is a programming task

`public boolean isValidateByModel()`
Return true if this task is to be validated by comparing results of the student's answer to results of teacher's answer

`public Date getModificationDate()`
Return the date and time this task was last modified

`public boolean isHasSucceeded()`
Return true if student has made the task, false if she/he hasn't

`public String getPassFeedBack()`
Return feedback if student passes the task

`public String getName()`
Return the name of this task

`public String getFailFeedBack()`
Return feedback if student fails the task

`public int getMaximumNumberOfInstructions()`
Returns loop prevention value

`public String getDateAsString()`
Returns String of the date and time

`protected static String parseXMLString(String XML, String tagname)`
Deserialize String value from XML string. Helper function for initSubClass()

`protected static boolean parseXMLBoolean(String XML, String tagname)`
Deserialize boolean value from XML string. Helper function for initSubClass()

`protected static int parseXMLint(String XML, String tagname)`
Deserialize int value from XML string. Helper function for initSubClass() return value or 0

`public int getMaximumNumberOfInstructions()`
Returns loop prevention value

`protected static String toXML(String tagname, int value)`
Serialize long value to XML string. Helper function for serializeSubClass()

`protected static String toXML(String tagname, String value)`
Serialize String value to XML string. Helper function for serializeSubClass()

`public String serializeToXML()`
Return a serialized this Task class into XML-format

`public void deserializeFromXML(String xml)`
Instantiate the Task class's parameters using the serialized form XML. @throws RuntimeException exceptions in the deserialization are caught and rethrown as uncheckced exception.

`public Date getModificationDate()`
Return the date and time this task was last modified

`public String toString()`
Returns String from the task

7.4 User

public class User

Modify existing User class by adding the following methods

```
public static final String STATUS_STUDENT = "student";
public static final String STATUS_TEACHER = "teacher";
public static final String STATUS_ADMIN = "adm";
```

`public User()`
Construct unitialized User object

`public User(String userID)`
Construct unitialized User object with userid

`public void setUserID(String userID)`
Set userID of this user

`public void setLastName(String lastname)`
Set last name of this user

`public void setFirstName(String firstname)`
Set first name of this user

`public void setEmail(String email)`
Set email address of this user

`public void setStatus(String status)`
`throws IllegalArgumentException if Status is not a valid Status string`
Set user status (teacher / student)

`public void setStudentNumber(String studentnum)`
Set student number of of this user

`public void setSocialSecurityNumber(String ssn)`
Set social security number of this user

`public void setPassword(String pass)`
Set password of this user to Pass

`public void setLanguage(String lang)`
Set the preferred language of this user. The language is either "EN"or "FI

`public String getUserID()`
Return userID of this user

`public String getLastName()`
Return last name of this user

`public String getFirstName()`
Return first name of this user

`public String getEmail()`
Return email address of this user

`public String getStatus()`
Return status of this user

`public String getStudentNumber()`
Return student number of this user. This identifier maps to <code>aeuser.extid</code> in the database

`public String getSocialSecurityNumber()`
Return social security number of this user. This identifier maps to <code>aeuser.extid2</code> in the database

`public String getPassword()`
Return plaintext password of this user

`public String getLpref()`
Return the preferred language of this user as String. The language is either "EN"or "FI"

`public boolean isTeacher()`
Return true of this user has the privelidges to add/remove/modify tasks and browse user statistics

`public boolean isStudent()`
Return true of this user has student privelidges

`public boolean isAdmin()`
Return true of this user has admin privelidges

`public boolean isValid()`
Test validity of this user object. The object is considered valid if all data members are set with non-empty values.

`private boolean isEmptyString(String str)`
Return false if str is null or empty string

7.5 Course

public class Course

Simple class for holding basic course information

`public Course(String name, String id)`
Create new Course instance using the specified name and ID

`public String getName()`
Return name of this course

`public void setName(String name)`
Set name of this course

`public String getID()`
Return course ID of this course

`public String toString()`
Return name and id of this course as one String

7.6 TitoAnalyzer

public class TitoAnalyzer

Class for running TitoKone and analyzing student's answer

`public TitoFeedback Analyze(Task task, List<Criterion> criteria, String programCode, String keyboardInput)`
Analyzes student's answercode. Returns feedback from analysis.

7.7 TitoFeedback

public interface TitoFeedback

Interface for feedback data

`public TitoState getTitoState()`
Return the end-state of TitoKone run

`public String getOverallFeedback()`
Return the overall task feedback

`public String getCompileError()`
Return compile-time error message (null if no errors)

`public String getRunError()`
Return run-time error message (null if no errors)

`public boolean wasSuccessful()`
Return true if program was compiled and executed successfully, and all acceptance crietia were met

`public java.util.List<TitoCriterionFeedback> getCriteriaFeedback()`
Return a list criteria feedbacks

7.8 TitoCriterionFeedback

public class TitoCriterionFeedback

Class for criterion feedback information

`public TitoCriterionFeedback(String name, String feedback, boolean success)`
Creates a new instance of TitoCriterionFeedback.

`public String getName()`
Return criterion name

`public String getFeedback()`
Return feedback of this criterion

`public boolean passedAcceptanceTest()`
Return true if this criterion meets passing requirements

7.9 TitoState

public class TitoState

Capsulates the end-state of single run of TitoKone and hides implementation details of TitoKone behind a single class interface.

This class will provide methods for exuting TTK91 programs and querying the resulting TitoKone end state

`public String compile(String sourceCode)`
Compiles given source code. Returns compile time error message, or null if compilation succeeded withour errors. If this method executes successfully (returns null), the program may be run with the run() method.
@param sourceCode TKK91 program source code as single string

`public String execute(String keyboardInput, int maxExecutionSteps)`
Runs previously compiled program. Returns runtime error message, or null if the program

ran to completion without errors.
@param keyboardInput keyboard input for the program, as [\n\r\t\f,.:;] separated list of numbers
@param maxExecutionSteps maximum number instruction to execute (prevent inifinite loops)

`public int getRegister(int registerCode)`
Return contents of register
@param registerCode one of REG_* constants defined in fi.hu.cs.ttk91.TTK91Cpu*

`public int getMemoryLocation(int address)`
Return contents of specified memory address

`public Map getSymbolTable()`
Return symbol table that maps symbol names to symbol value addresses

`public String getScreenOutput()`
Return TitoKone output as String in format "1234, 1234, 1234". The returned string may be empty, but it is never null.

`public int getStackMaxSize()`
Return maximum stack size reached during program execution

`public int getExecutionSteps()`
Return number of executed instructions

`public int getCodeSize()`
Return number of instruction words in the program code

`public int getDataSize()`
Return number of words in program's data-area

`public int getMemoryAccessCount()`
Return number memory references made during program run. This number includes references caused by both data and instruction fetches

`public Set<String> getUsedOpcodes()`
Return used opcodes in Set of Strings

7.10 LanguageManager

public class LanguageManager

`private static HashMap<String, ResourceBundle> bundles`
Stores all initialized ResourceBundles.

`public static synchronized void loadTextResources(String propertiesFi`
Initializes all resources specified in parameter file, which contains filenames of all resources to be loaded in XML format.

`public static ResourceBundle getTextResource(String lang, String page)`
Returns a ResourceBundle object containing all language specific data on a single JSP page.

7.11 TitoBundle

public class TitoBundle extends ResourceBundle

`private Properties xmlData`
Stores data from xml-file

`private String language`
Language of returned data

`public TitoBundle(Properties data, String lang)`
Initalize a new TitoBundle, which returns data from Properties object in a given language

`protected Object handleGetObject(String key)`
Returns an object, which matches the key, from xmlData. It’s preferable to use getString method from parent class instead

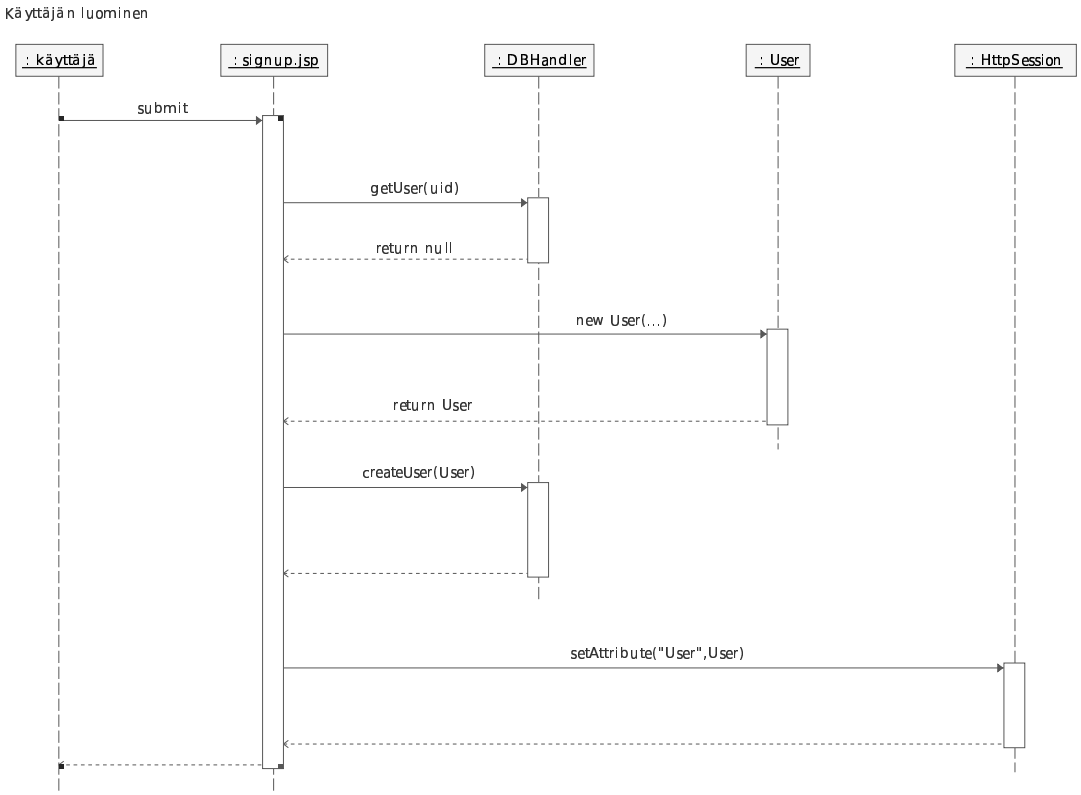
`public Enumeration<String> getKeys()`
Returns all available key values

8 JSP sivujen sekvenssikaaviot

Tämä luku esittelee toteutettavien JSP-sivujen toimintalogiikan. Sekvenssikaaviot esittelevät tärkeimmät eri toiminnoissa käytetyt luokat ja metodit, menemättä kuitenkaan aivan samalle tasolle kuin lopullinen ohjelmakoodi. Myöskään kaikkia toimintoja ei kuvata, esimerkiksi erilaisten listausten näyttäminen katsotaan niin suoraviivaiseksi, että ne onnistuvat ilman erillistä suunnitelmaa.

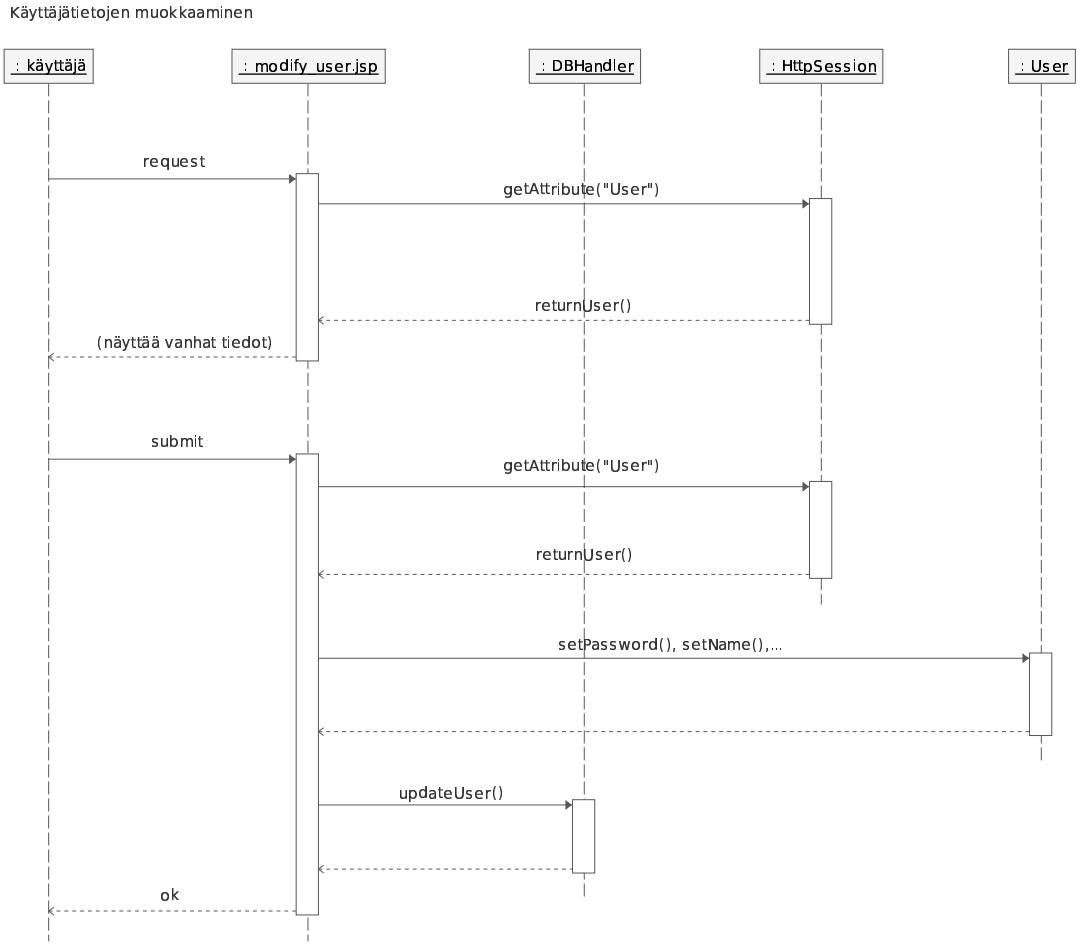
Sekvenssikaavioit esittävät miten TitoTrainer/eAssari käsittelee tapahtuman, jossa käyttäjä on täyttänyt lomakkeen ja lähettänyt sen palvelimen käsiteltäväksi. Vain onnistuneet tapahtumat on kuvattu, esimerkiksi väärän salasanan syöttäminen kirjautumislomakkeeseen katkaisee kuvatun sekvenssin ja palauttaa käyttäjälle virheilmoituksen.

8.1 Signup.jsp



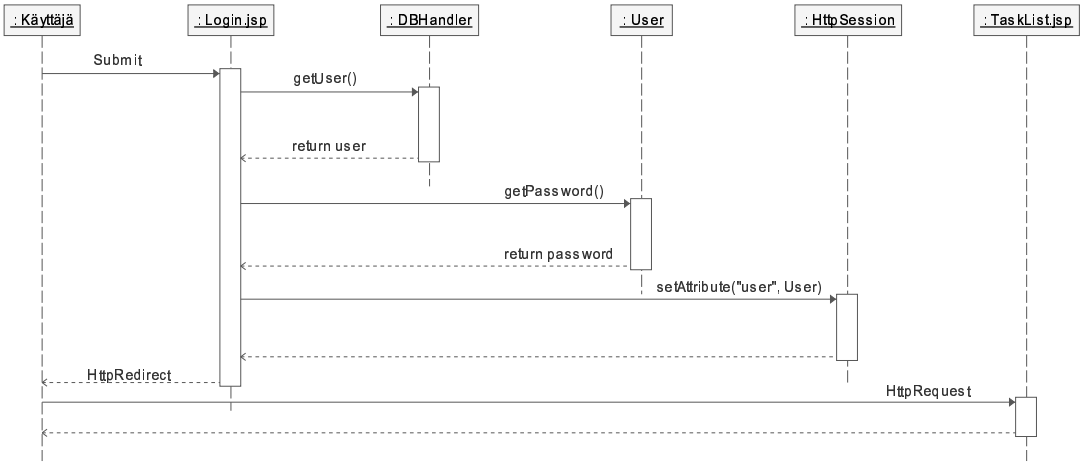
Kuva 4: Käyttäjän rekisteröityminen

8.2 ModifyUser.jsp



Kuva 5: Käyttäjätietojen muokkaaminen (huom kaksi erillistä sivulatausta)

8.3 Login.jsp

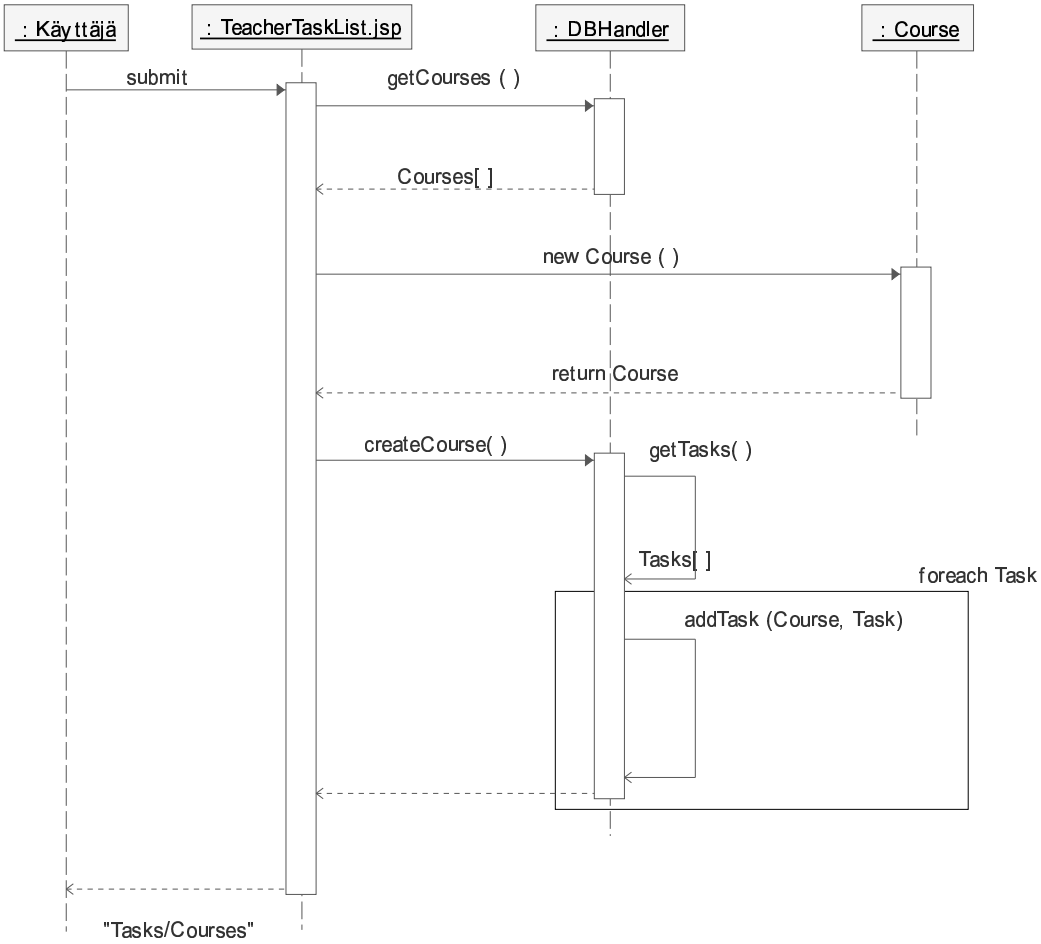


Kuva 6: Kirjautuminen

8.4 StudentTaskList.jsp

Ei kaaviota. Sisältää vain opiskelijan tehtävälistauksen.

8.5 TeacherTaskList.jsp



Kuva 7: Kurssin luominen ja tehtävien lisääminen luotuun kurssiin

8.6 CourseStatistics.jsp

Yksinkertaisen tilastotaulukon toteutus suoraviivaista, sekvenssikaavio ei tarpeen.

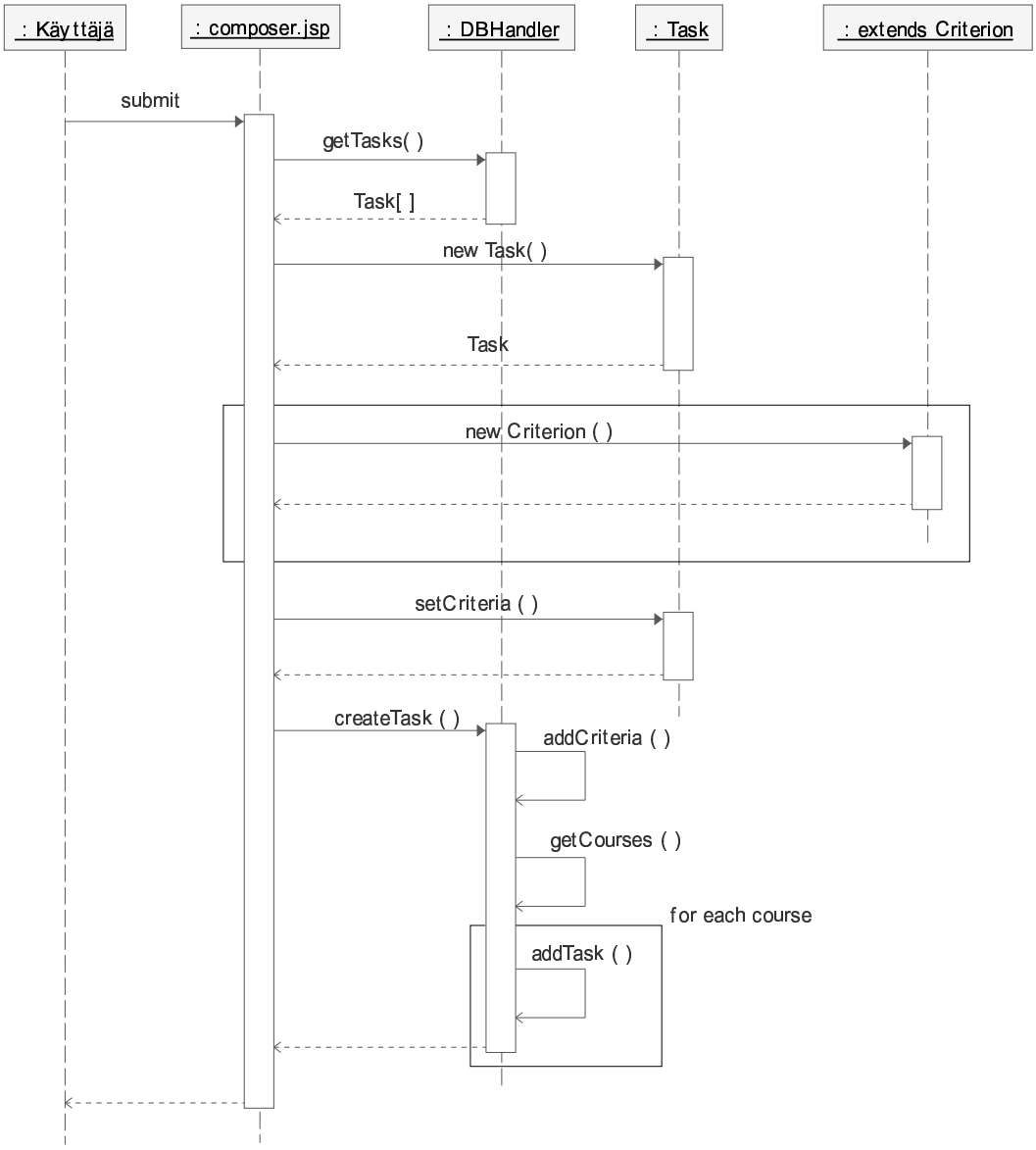
8.7 UserInfo.jsp

Yksinkertaisen käyttäjätietonäkymän sekä tilastotaulukon toteutus suoraviivaista, sekvenssikaavio ei tarpeen.

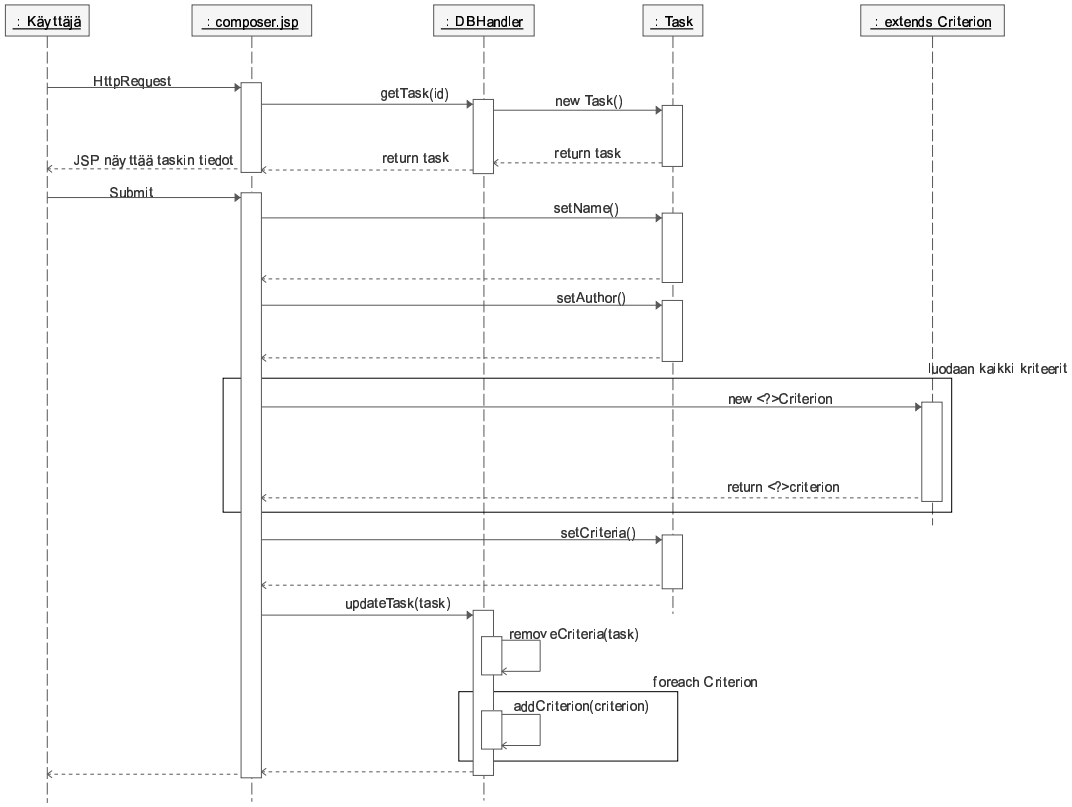
8.8 SearchUsers.jsp

Käyttäjien haku nimen mukaan. Sekvenssikaavio ei tarpeen.

8.9 Composer.jsp



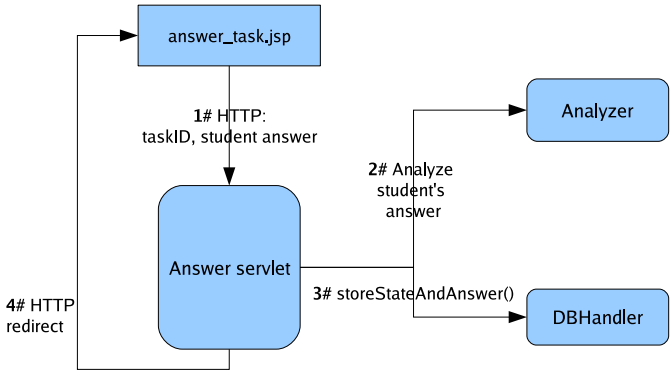
Kuva 8: Tehtävän luominen ja lisääminen kursseille



Kuva 9: Tehtävän muokkaaminen (huom kaksi erillistä sivulatausta)

8.10 AnswerTask.jsp

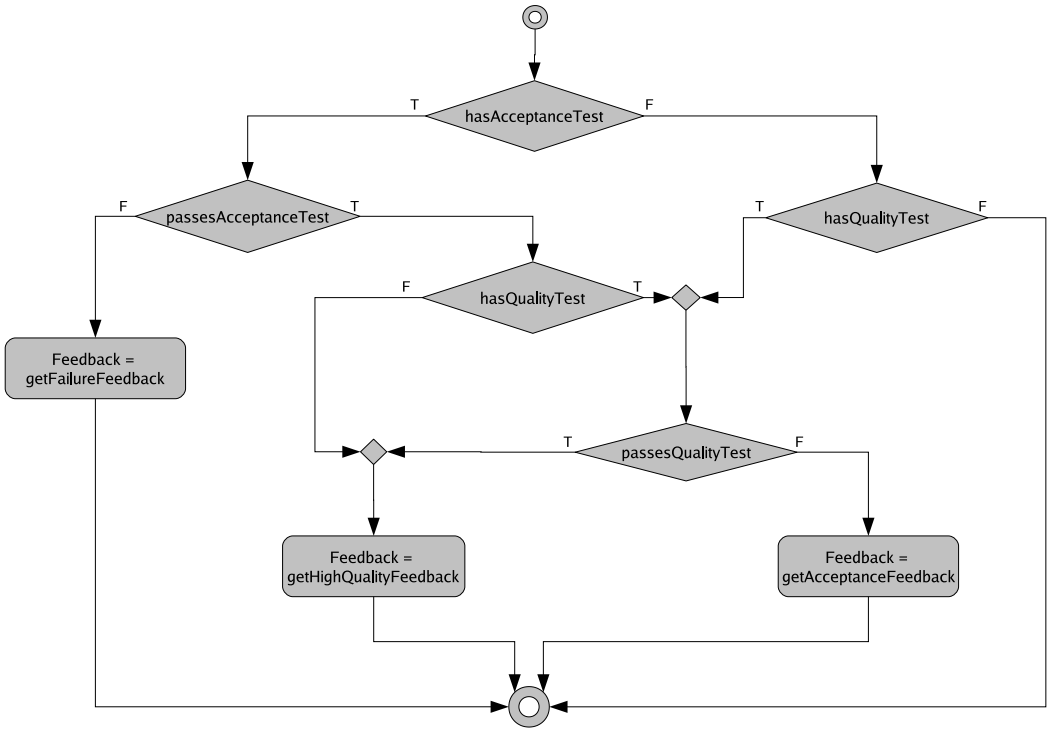
Näyttää tehtävävastauslomakkeen opiskelijalle.



Kuva 10: Tehtävävastauksen toimintavuo

8.11 Answer-servlet

Tarkastaa opiskelijan vastausyrityksen TitoAnalyzer luokan avulla ja tallentaa tuloksen tietokantaan, sekä sessioon jotta AnswerTask.jsp voi näyttää palautetta opiskelijalle.



Kuva 11: Kriteerin tarkistus ja palautteen määräytyminen