LAPPEENRANNAN TEKNILLINEN YLIOPISTO

TEKNILLISTALOUDELLINEN TIEDEKUNTA

TIETOTEKNIIKAN OSASTO

TIETOVARASTON LATAUSPROSESSIN KEHITTÄMISEN OSITTAINEN AUTOMATISOINTI MICROSOFT SQL SERVER 2008 -YMPÄRISTÖSSÄ

Työ on salainen 8.6.2014 asti.

Työn tarkastajat ovat professori Kari Smolander ja diplomi-insinööri Arto Arffman.

Työn ohjaaja on diplomi-insinööri Arto Arffman.

Jouko Rouhiainen

jouko.rouhiainen@gmail.com

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
Teknillistaloudellinen tiedekunta
Tietotekniikan osasto

Jouko Rouhiainen

Tietovaraston latausprosessin kehittämisen osittainen automatisointi Microsoft SQL Server 2008 -ympäristössä

Diplomityö

2012

77 sivua, 13 kuvaa ja 5 taulukkoa

Tarkastajat: Professori Kari Smolander
Diplomi-insinööri Arto Arffman

Hakusanat: SSIS, SQL Server, ETL

Keywords: SSIS, SQL Server, ETL

Tässä työssä tutkitaan tietovaraston latausprosessin kehittämisen nopeuttamista Microsoft *SQL Server 2008* -ympäristössä. Työn teoriaosuudet on tarkoitettu tukemaan sekä työn tutkimus- että käytännönosia. Aiheeseen liittyviä tutkimuksia käytiin läpi parhaiden latausprosessin kehittämiseen kuluvaa aikaa vähentävien tapojen selvittämiseksi. Nykytutkimus keskittyy valmistajasta riippumattomien mallien kehittämiseen ja valmistajakohtaisen latausprosessin luomiseen näiden mallien pohjalta. Yleinen konsensus parhaan mallin suhteen kuitenkin puuttuu.

Aiheeseen liittyvien tutkimusten pohjalta esitetään arkkitehtuuri, joka saattaisi tulevaisuudessa vähentää latausprosessin kehittämiseen kuluvaa aikaa huomattavasti. Tästä arkkitehtuurista luotiin yksinkertaistettu versio sekä siihen pohjautuva sovellus nopeuttamaan latausprosessin kehittämistä Microsoftin ETL-työkalulla.

ABSTRACT

Lappeenranta University of Technology
Faculty of Technology Management
Department of Information Technology

Jouko Rouhiainen

Implementing Data Warehouse ETL in Microsoft SQL Server 2008 Environment Semi-Automatically

Master's thesis

2012

77 pages, 13 figures and 5 tables

Examiners: Professor Kari Smolander
M. Sc. Arto Arffman

Keywords: SSIS, SQL Server, ETL

This thesis deals with expediting the implementation of data warehouse ETL in Microsoft SQL Server 2008 environment. The theoretical parts of this thesis are there to support the thesis' research and practical parts. Related research on the subject was explored in order to determine the best methods for decreasing ETL development time. Related research currently focuses on developing vendor-independent models for modeling ETL processes and generating vendor-specific ETL code from these models. However, there is no consensus among different research groups as to what kind of a model would be the best.

Based on research of related works, a proposal for an architecture is presented. The architecture could be used to considerably decrease ETL development time in the future. A simplified version of the architecture and an application based on it were created in this work to expedite Microsoft ETL development.

ALKUSANAT

Tämä diplomityö on tehty Aureolis Oy:lle.

Haluan kiittää työni tarkastajaa Kari Smolanderia työni eri versioiden hyvästä kritiikistä. Lisäksi haluan kiittää työni ohjaajaa Arto Arffmania avusta ja hyvistä ideoista matkan varrella.

Erityiskiitokset haluan esittää tyttöystävälleni Tainalle. Ilman hänen tukeaan työn tekeminen olisi ollut huomattavasti raskaampaa, ja ilman hänen erinomaista kielentarkistustaan tämän työn esitysasu olisi ollut huomattavasti huonompi. Kotiväkeni ansaitsee myös kiitoksensa kannustamisesta työn tekoon.

Espoossa 13.5.2012

Jouko Rouhiainen

SISÄLLYSLUETTELO

TERMI- JA LYHENNELUETTELO

ADO.NET	ActiveX Data Objects for .NET
API	Application Programming Interface, ohjelmointirajapinta
Backus–Naur-muoto	Usein käytetty ohjelmointikielien kieliopin esitysmuoto
BI	Business Intelligence
BIDS	Business Intelligence Development Studio
BPEL	ks. WS-BPEL
BPMN	Business Process Model and Notation
C#	Microsoftin luoma olio-ohjelmointikieli
CLR	Common Language Runtime
CRM	Customer Relationship Management
Data Mining	ks. Tiedonlouhinta
Dimensio	Näkökulma tietovaraston tietoon
DSS	Decision Support System
DTS	Data Transformation Studio
ER-malli	Entity-Relationship-malli, abstrakti ja käsitteellinen tiedon ilmaisu
ETL	Extract, Transform, Load
Fakta	Tietovaraston mittaritieto
FTP	File Transfer Protocol
IDE	Integrated Development Environment, Integroitu ohjelmointiympäristö
M2T	Model to Text

MDA	Model-Driven Architecture
MDD	Model-Driven Development
Microsoft Excel	Microsoftin taulukkolaskentaohjelma
Microsoft SQL Server	Microsoftin kehittämä relaationaalinen tietokantapalvelin
MOF	Meta-Object Facility
.NET	Microsoftin kehittämä ohjelmistokehys
OLAP	Online Analytical Processing
OLE DB	Object Linking and Embedding, Database
OMG	Object Management Group
OMT	Object-Modeling Technique
OOAD	Object-Oriented Analysis and Design
OOSE	Object-Oriented Software Engineering
PIM	Platform-Independent Model
PSM	Platform-Specific Model
QVT	Query/View/Transformation
SCD	Slowly Changing Dimension
SQL	Structured Query Language
SSAS	SQL Server Analysis Services
SSIS	SQL Server Integration Services
SSRS	SQL Server Reporting Services
Staging	Erillinen työtietokanta tiedon jalostusta varten
Surrogaatti	Merkityksetön ja uniikki kokonaisluku

TDL	Transformation Description Language
Tiedonlouhinta	Prosessi, jossa etsitään toistuvia kuvioita datasta
Tietovarasto	Tiedon keskittämiseen ja historiointiin erikoistunut tietokanta
UML	Unified Modeling Language
WS-BPEL	Web Services Business Process Execution Language
XML	Extensible Markup Language

1 JOHDANTO

Tietovarastoja kehitetään vastaamaan yritysten lisääntyvää tarvetta saada ajankohtaista tietoa yritysten johdon päätösten tueksi. ETL (*Extract, Transform, Load*) on tietovaraston latausprosessi, joka on tärkein osa tietovarastointia. ETL-prosessin kehittäminen on yleensä myös haastavin ja työläin osa tietovaraston kehittämisessä. Tästä syystä viime aikoina on alettu kiinnittää huomiota siihen, kuinka latausprosessin kehittämistä voisi nopeuttaa. Perinteisesti ETL-prosessin kehittämisen kaltaista ohjelmointityötä on helpotettu integroitujen ohjelmointiympäristöjen avulla (engl. *Integrated Development Environment*, IDE), mutta nämä ympäristöt ovat valmistajakohtaisia eivätkä toimi eri valmistajien tuotteiden kanssa. Sen takia ETL:n mallintaminen valmistajista riippumattomalla tavalla ja valmistajakohtaisen ohjelmakoodin generoiminen mallien pohjalta onkin herättänyt kiinnostusta.

Tämä diplomityö keskittyy latausprosessin kehittämisen nopeuttamiseen Microsoft SQL Server 2008 -ympäristössä. Microsoft SQL Server 2008 -alustan ETL-väline on *SQL Server Integration Services* (SSIS).

1.1 Työn tausta

Tämä diplomityö sai alkunsa ideasta lyhentää sitä aikaa, joka latausprosessin kehittäjällä menee eri projekteissa toistuvien yksinkertaisten, mutta huomattavan työläiden, tehtävien suorittamiseen. Latausprosessit ovat usein samankaltaisia eri projekteissa. Tätä varten on mahdollista tehdä valmiita latausprosessipohjia eri projekteissa käytettäväksi, mutta ne eivät sinänsä nopeuta kehittämistä huomattavasti, sillä edellä mainittuja työläitä osuuksia ei ole mahdollista sisällyttää latauspohjiin. Pohjista onkin apua lähinnä kehitysprosessin standardoinnissa.

1.1.1 Yritysesittely

Työn kohdeyritys, Aureolis Oy, on vuonna 2001 perustettu Business Intelligenceen ja tietovarastointiin keskittyvä tietotekniikka-alan yritys. ”Aureolis on erikoistunut rakentamaan asiakaskohtaisia informaatiojärjestelmiä, joiden avulla eri tietojärjestelmien tiedot kootaan yhteen tietokantaan, puhdistetaan ja jalostetaan käyttäjien kannalta helppolukuiseen muotoon” (Aureolis 2010). Henkilöstöä yrityksessä on yli 40 ja sen pääkonttori on Espoossa.

Työ toteutetaan Prima-raportointi-projektin rinnalla hyödyntäen siitä ja muista projekteista saatuja kokemuksia. Prima-raportointi-projektissa tehdään uusi Microsoft-teknologiaan perustuva tietovarastointi- ja raportointiratkaisu myytäväksi eri asiakkaille. Tavoitteena on, että vastaavanlaisia ratkaisuja edelleen kehitettyinä pystyttäisiin jatkossa kehittämään huomattavasti nopeammin. Lisäksi myös muissa Microsoft-teknologiaa käyttävissä projekteissa pyritään nopeuttamaan latausprosessin kehittämistä.

1.2 Työn tavoitteet ja rajaus

Työn tavoitteena on luoda sovellus, joka lukee latausprosessin määrityksen määritystiedostosta ja tekee sen pohjalta latausprosessille automaattisesti rungon, jonka pohjalta muodostetaan SSIS-paketti (*SQL Server Integration Services*). SSIS-paketti kuvaa latausprosessin työnkulun ja logiikan *SQL Server* -ympäristössä. Koska latausprosessin työläimmät ja eri latauspaketeissa usein toistuvat rakenteet sisältyvät juuri runkoon, sen luomisen automatisointi nopeuttaa latausprosessin kehittämistä merkittävästi. Työ on rajattu käsittämään latausprosessin rungon teko *staging*-, dimensio- ja faktatyyppisille tauluille, jotka ovat moniulotteisessa tietovarastossa käytettyjä taulutyyppejä.

1.3 Tutkimusmenetelmät

Tämän työn tutkimuskysymys on: *miten lyhentää tietovaraston latausprosessin kehittämiseen kuluvaa aikaa ohjelmallisesti?* Tutkimusmenetelmänä työn teoreettisessa osuudessa käytetään kvalitatiivista eli laadullista tutkimusmenetelmää. Aineistona tutkimuksessa käytetään ETL-prosessin mallintamista ja automaattista generoimista käsitteleviä tieteellisiä artikkeleita, joita esitellään kolmannessa luvussa ETL:n esittelyn yhteydessä. Tutkimuksessa on tarkoitus selvittää, miten olemassa olevassa tutkimustiedossa on lyhennetty tietovaraston latausprosessin kehittämiseen kuluvaa aikaa. Lisäksi hyödynnetään suunnittelututkimusmenetelmää, ja pyrkimyksenä on hyödyntää kvalitatiivisessa tutkimuksessa saatua tietoa kahdella tavalla. Ensin tieteellisistä artikkeleista saaduista tiedoista poimitaan toimivimmat ehdotukset, joiden pohjalta kehitetään optimaalinen ratkaisu ETL-prosessin kehittämiseen kuluvan ajan lyhentämiseksi. Sen jälkeen tästä ratkaisusta mietitään diplomityön rajaukseen sopiva käytännön toteutus.

1.4 Työn rakenne

Ensin työssä luodaan yleiskatsaus tietovarastoihin luvussa kaksi. Luvussa kolme käydään läpi käsitettä ETL (*Extract, Transform, Load*) ja perehdytään ETL:n mallintamiseen ja automaattiseen generointiin liittyvään tutkimustietoon. Tutkimustiedon pohjalta esitetään ratkaisuvaihtoehto ETL-prosessin kehittämisen nopeuttamiseksi. Neljännessä luvussa perehdytään Microsoftin ETL-työkaluun *SQL Server Integration Services* (SSIS). Viidennessä luvussa esitellään SSIS:n ohjelmointirajapinnan mahdollistamaa latauspaketin generointia C#-koodilla. Kuudennessa luvussa käydään läpi latauspaketin määritysdokumenttipohjaa. Seitsemännessä luvussa tarkastellaan määritysdokumenteista saadun mallin yhdistämistä latauspaketin generoinnin mahdollistavaan ohjelmointirajapintaan. Luvussa kahdeksan pohditaan tuloksia ja jatkokehitysmahdollisuuksia. Yhdeksännessä ja viimeisessä luvussa esitellään työn johtopäätökset.

2 YLEISKATSAUS TIETOVARASTOIHIN

Tietovarastot ovat tiedon keskittämiseen ja yleensä myös historiointiin erikoistuneita tietokantoja. Niihin haetaan tietoa yhdestä tai useammasta tietolähteestä, jotka voivat olla tiedostoja tai tietokantoja, käyttämällä latausprosessia. Latausprosessista käytetään nimitystä ETL-prosessi (*Extract, Transform, Load*). Tietovarastossa olevaa tietoa hyödynnetään raportoinnissa ja analysoinnissa. Perinteisistä tietokannoista tietovarastot eroavat käyttötarkoitukseltaan ja siten, että niiden sisältämä tieto on erittäin harvoin reaaliaikaista. Tietovarastoihin tallennettu tieto on myös yleensä tallennettu eri tavalla kuin lähdejärjestelmissä, jotta se soveltuu paremmin tietovarastointiin. (Rainardi 2007, s. 1; Malinowski & Zimányi, 2008, s. 2.)

Yksinkertaisimmillaan tietovarasto koostuu latausprosessista ja moniulotteisesta tietokannasta (Rainardi 2007, s. 4). Moniulotteisen tietokannan tietomalli on suunniteltu siten, että se käyttää moniulotteisia rakenteita tiedon organisointiin ja tietojen välisten suhteiden esittämiseen (O'Brien & Marakas 2009, s. 177). Tällainen tietovarasto on kuitenkin erittäin rajoittunut käytettävyydeltään, jos raportointisovellukset mielletään osaksi tietovarastoa, koska tällöin tiedon saanti ulos tietovarastosta on hankalaa. Normaalisti tietovarastosta tehdään muun muassa raportointia joko kyseiseen tietovarastoon räätälöidyllä sovelluksella tai valmisohjelmistolla. Valmisohjelmistoja käytettäessä tietovarasto pohjautuu yleensä saman yrityksen teknologiaan kuin sovellukset, joilla sitä hyödynnetään.

Perinteiset operatiivisessa käytössä olevat tietokantarakenteet eivät sovellu hyvin sellaiseen raportointiin ja analysointiin, johon tietovarastoja käytetään. Tähän on kaksi pääsyytä: tietovarastossa usein käytetty moniulotteinen taulurakenne sopii paremmin analyysiin kuin perinteisesti operatiivisissa tietokannoissa käytetyt taulurakenteet, ja tietovarastossa tieto on keskitetty useasta tietolähteestä (Rainardi 2007, s. 2). Näiden lisäksi tietovarastoon tehdyt kyselyt ovat usein melko raskaita, ja siten ne aiheuttaisivat turhaa kuormitusta operatiiviseen kantaan.

2.1 Tietovarastojen tausta ja lähestymistavat

Nykyään tietovarastojen perustietokantasuunnittelu voidaan jakaa kolmeen eri ryhmään: Bill Inmonin relaationaaliseen malliin, Ralph Kimballin moniulotteiseen mal-

liin ja hybridimalleihin, kuten Dan Linstedtin *Data Vault* (Inmon 2005, s. 357; Kimball & Caserta 2004, s. 27; Linstedt 2002). Inmonin lähestymistavassa tietovarasto rakennetaan normalisoituun tietokantaan, kun taas Kimballin lähestymistavassa se rakennetaan moniulotteiseen kantaan (Rainardi 2007, s. 16). Normalisointi on peruutettavissa oleva prosessi, jossa relaatiotietokannan rakennetta muutetaan siten, että relaatioilla on yksinkertaisempi ja säännöllisempi rakenne, jolloin vähennetään tiedon päällekkäisyyttä ja riippuvuussuhteita (Codd 1971, s. 11). Rainardi (2007, s. 16–17) huomauttaa, että jos tiedon lataa normalisoituun kantaan, niin hänen mielestään se joudutaan silti lataamaan moniulotteiseen kantaan kyselyjä ja analysointia varten, koska moniulotteinen tietokanta soveltuu siihen paremmin. Näin on etenkin silloin, jos tietovarastoa halutaan käyttää pohjana OLAP-kuutioille (*Online Analytical Processing*), koska ne vaativat moniulotteisen tietorakenteen käyttöä. OLAP-kuutiot ovat kyselyiden tehokkuuteen sekä analyysin monimuotoisuuteen erikoistuneita järjestelmiä (Kimball & Caserta 2004, s. 247–248). Tietojen keskittäminen useista lähteistä onnistuu kuitenkin paremmin, jos ne viedään normalisoituun kantaan (Rainardi 2007, s. 17).

Dan Linstedtin *Data Vault* yhdistää kolmannen normaalimuodon (3NF) tähtimalliin. *Data Vault* pyrkii ratkaisemaan kolmannesta normaalimuodosta muodostuvia ongelmia, joita ovat muun muassa kyselyjen ja porautumisen hankaluudet. Lisäksi sen tavoitteena on lieventää perinteisten moniulotteisten tietovarastojen ongelmia, kuten sitä, että faktataulujen rakennetta ei voi muuttaa tietovaraston käyttöönoton jälkeen. (Linstedt 2002.) Eri normaalimuodot muodostuvat normalisoinnin seurauksena. Tähtimalli on yksinkertaisin tietovarastomalli.

Tässä työssä keskitytään Ralph Kimballin moniulotteiseen malliin pohjautuvaan ETL-prosessiin, koska Kimballin malli on erittäin tunnettu ja yleisesti käytössä. Lisäksi tämän työn käytännön osuus toteutetaan Microsoftin työkaluilla, ja sekä Microsoft että Microsoftin toteutusvälineoppaat ohjaavat suunnittelijoita epäsuorasti moniulotteisen mallin käyttöön. Esimerkiksi Knight et al. (2008) ja Veerman et al. (2009) keskittyvät pelkästään moniulotteiseen malliin teoksissaan.

2.2 Moniulotteiset tietovarastot

Moniulotteisessa taulurakenteessa on kahdentyyppisiä tauluja: dimensiotauluja ja faktatauluja. Dimensiotaulut tarjoavat kontekstin faktatauluille ja siten myös kaikille mittareille tietovarastossa (Kimball & Caserta 2004, s. 161). Dimensiotaulujen sisältö on yleensä tekstimuotoista ja jaettuna useisiin kenttiin (Hovi 1997, s. 73). Dimensiotaulut kuvaavat eri näkökulmien, kuten erilaisten asioiden, esineiden ja henkilöiden, ominaisuuksia ja tietoja. Tällaisia ovat esimerkiksi asiakas- tai tuotetiedot. Jokaista näkökulmaa kohden on oma dimensiotaulu tai useista dimensiotauluista muodostuva hierarkia, joka sisältää kyseisen näkökulman dimensiotiedon. Vaikka dimensiotaulut ovat yleensä paljon pienempiä kuin faktataulut, ne ovat moniulotteisen tietovaraston ydin, koska ne tarjoavat tuloväylän tietoon (Kimball & Caserta 2004, s. 161).

Faktataulut sisältävät yrityksen mittaritiedot. Faktataulujen suhde mittareihin on yksinkertainen: jos mittaritieto on olemassa, se on mallinnettavissa faktataulun rivinä, ja vastaavasti faktataulun rivi on mittari. (Kimball & Caserta 2004, s. 209.) Faktataulut koostuvat dimensiotauluihin linkittävistä avaimista ja mittaritiedoista, kuten myynti- ja asiakasmääristä.

Dimensiotaulut tulisi rakentaa minimaalisella määrällä komponentteja. Primääriavaimena toimii yksi sarake, joka sisältää merkityksettömän ja yksilöllisen kokonaisluvun. Tätä kokonaislukuavainta kutsutaan surrogaatiksi (*surrogate*). Dimensiotaulujen tulee sisältää yksi tai useampi sarake, joista muodostuu dimension luonnollinen avain. Luonnollinen avain pohjautuu yhteen tai useampaan lähtötaulun sarakkeeseen, kun kyseessä ei ole ETL-prosessin kokonaan generoima dimensio. Lähtötaulun sarake voi olla esimerkiksi henkilöstötunnus. Jos dimensiotaulussa pidetään historiatietoa muutoksista, yksi luonnollinen avain voi viitata useaan surrogaattiavaimeen, joista yksi on kerrallaan voimassa. Primääriavaimen ja luonnollisen avaimen lisäksi dimensioista löytyy kuvaavia attribuutteja, kuten nimiä ja osoitteita. (Kimball & Caserta 2004, s. 162–163.)

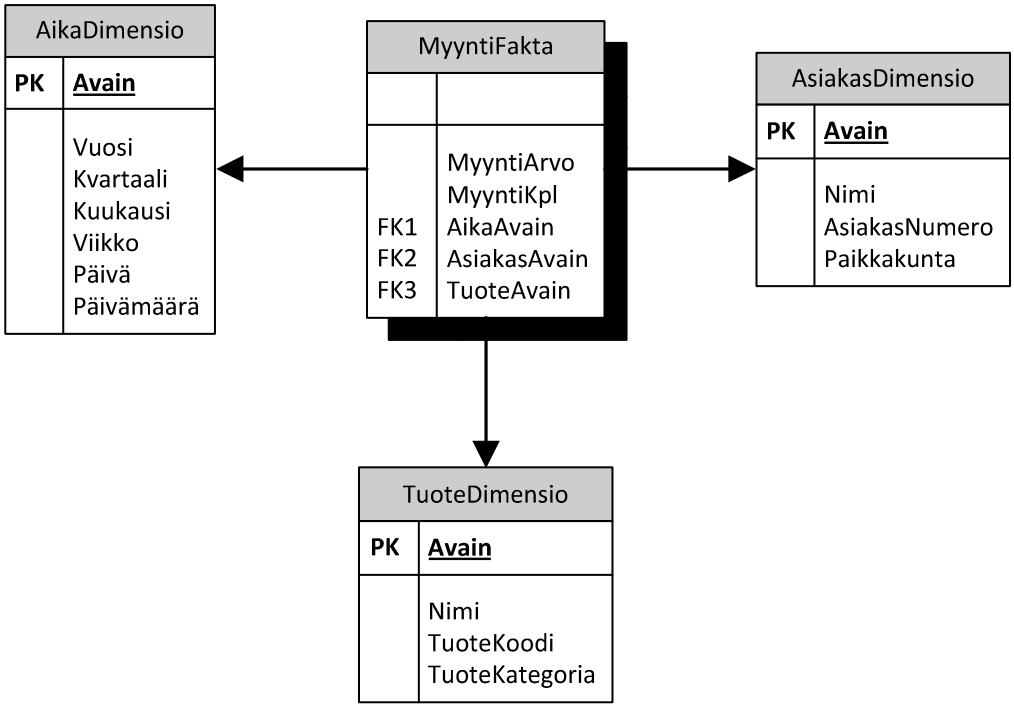
Tietovaraston ETL-prosessi on vastuussa surrogaattiavaimien luomisesta ja kirjoittamisesta tauluun. Surrogaattiavaimien käyttöön on kaksi pääsyytä: suorituskyky ja faktataulujen koon pienentäminen. Muun muassa raportointia varten dimensiotaulut

tulee yhdistää faktatauluihin, jotta eri mittareille saadaan järkeviä selitteitä. SQL:n *join*-operaatiot ovat nopeampia tehtynä kokonaisluvuille kuin esimerkiksi merkkimuotoiselle tiedolle, joten surrogaattiavaimien käyttö parantaa tietovaraston suorituskykyä. Faktatauluihin sijoitetut luonnolliset avaimet puolestaan vievät yleensä moninkertaisesti tilaa verrattuna surrogaattiavaimiin, joten käyttämällä surrogaattiavaimia faktataulujen koot pienenevät huomattavasti, mikä myös tehostaa suorituskykyä. (Kimball & Caserta 2004, s. 162, 164–165.)

Tietovarastomalli kertoo sen, miten tieto on organisoitu tietovaraston tauluihin. Tietovarastomalleja on kolme: tähtimalli, lumihiutalemalli ja galaksimalli. Näistä kahta ensimmäistä käytetään yksinkertaisemmissa tietovarastoissa ja galaksimallia monimutkaisissa tietovarastoissa.

2.2.1 Tähtimalli

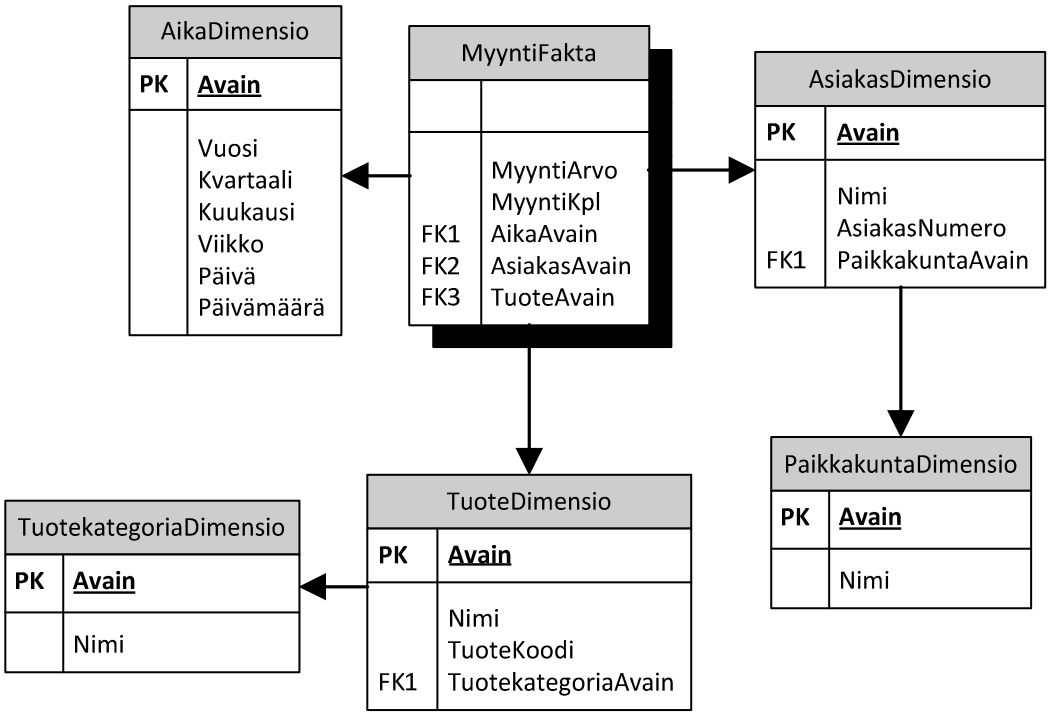
Tähtimallissa jokaista faktataulun dimensioavainta kohden on yksi dimensiotaulu, joka sisältää kaiken dimensiotiedon. Dimensiot eivät ole normalisoituja ja ne voivat siten sisältää päällekkäistä informaatiota varsinkin, kun käytössä on hierarkioita (Malinowski & Zimányi, 2008, s. 50). Kuva 1 on esimerkki tähtimallista. Siinä on yksi faktataulu, johon on yhdistetty kolme dimensiotaulua. Jokainen dimensiotaulu kuvassa on hierarkkinen: esimerkiksi aikadimensiossa hierarkian ylin taso on vuosi ja alin taso päivä. Kuvassa dimensiotaulujen primääriavaimet (PK, *Primary Key*) yhdistyvät faktataulun vierasavaimiin (FK, *Foreign Key*).



Kuva 1. Esimerkki tähtimallista.

2.2.2 Lumihiutalemalli

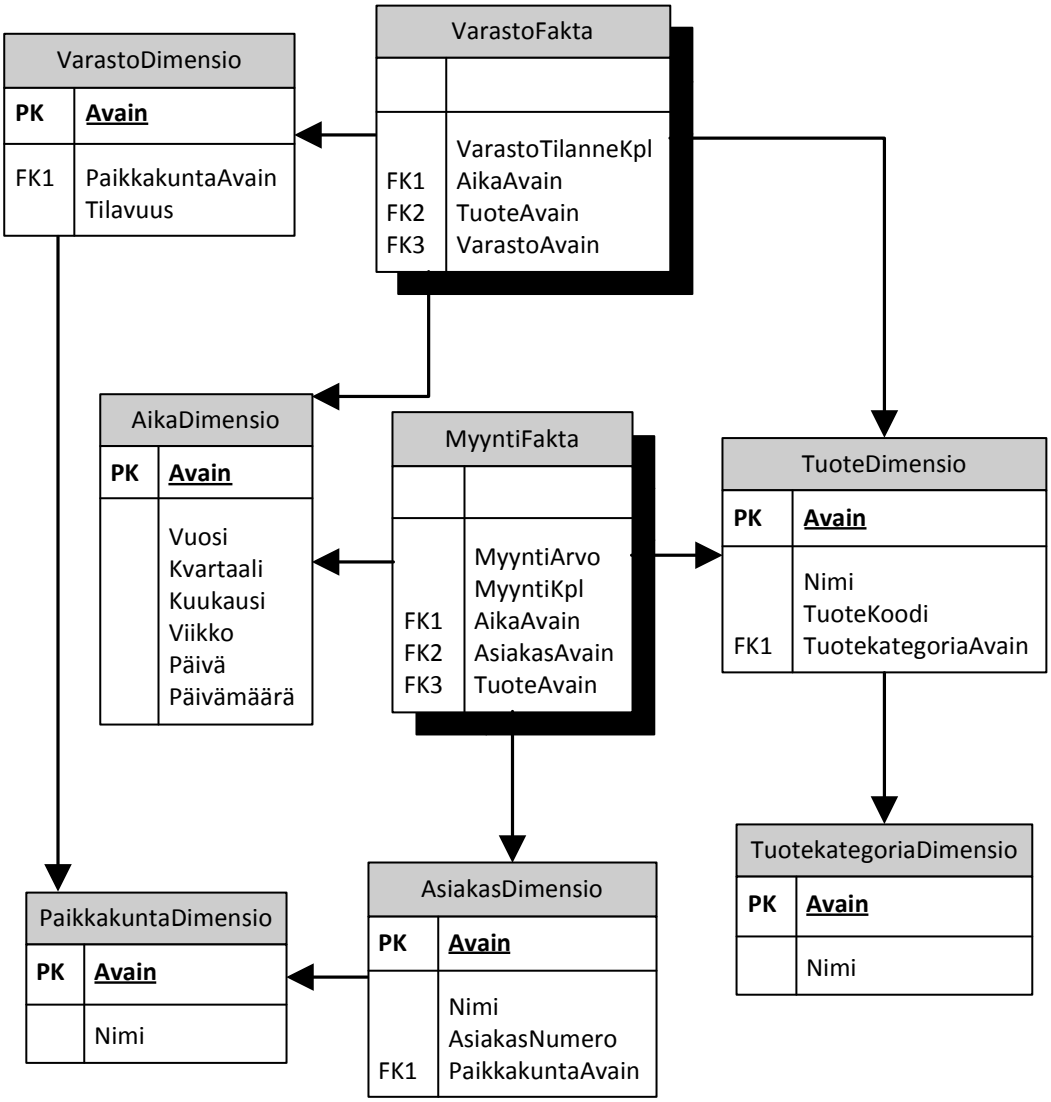
Lumihiutalemallissa faktataulun kutakin dimensioavainta kohden on yksi dimensiotaulu, josta on yhteyksiä muihin dimensiotauluihin, jotka sisältävät osan dimensiotiedosta. Dimensiotaulut ovat siis normalisoituja. Normalisoiduista dimensioista muodostuu luontevasti hierarkioita. Yhdistämällä tähtimalli ja lumihiutalemalli dimensiot voidaan rakentaa normalisoituina tai ei-normalisoituina tarpeen mukaan. Tätä yhdistelmää voidaan kutsua tähtihiutalemalliksi (Malinowski & Zimányi 2008, s. 50). Kuva 2 on esimerkki lumihiutalemallista. Tähtimalliin verrattuna asiakasdimensio ja tuotedimensio ovat normalisoituja, ja niistä on yhteys hierarkian ylemmän tason dimensiotauluun.



Kuva 2. Esimerkki lumihiutalemallista.

2.2.3 Galaksimalli eli tähdistömalli

Galaksimallissa on useita faktatauluja, jotka jakavat dimensiotauluja keskenään. Dimensiotaulut voivat olla normalisoituja tai ei-normalisoituja (Malinowski & Zimányi 2008, s. 50). Kuva 3 on esimerkki galaksimallista. Se on muuten samanlainen kuin kuvan 2 lumihiutalemalli, paitsi että siinä on lisäksi toinen faktataulu ja yksi uusi dimensiotaulu.



Kuva 3. Esimerkki galaksimallista.

2.3 Tietovarastojen käyttökohteet

Tietovarastoja käytetään useimmiten *Business Intelligenceen* (BI), asiakkuudenhallintaan (*Customer Relationship Management*, CRM) ja tiedonlouhintaan (*Data Mining*) (Rainardi 2007, s. 17). Muita käyttökohteita ovat raportointi ja tiedon keskittäminen. Eri käyttötarkoitukset eivät ole toisiaan poissulkevia, ja usein samaa tieto-

varastoa voidaan käyttää eri tarkoituksiin. Lisäksi eri käyttötarkoitukset voivat hyödyntää toisiaan ja sisältyä toisiinsa: esimerkiksi asiakkuudenhallintajärjestelmä voi käyttää BI-järjestelmää, joka puolestaan voi toimittaa raportteja.

2.3.1 Business Intelligence

Rainardin (2007, s. 17) mukaan monet järjestelmätoimittajat käyttävät tietovarastoinnista termiä *Business Intelligence*. Hän jatkaa todeten, että tämä johtuu siitä, että *Business Intelligencessä* keskitytään siihen, miten tietovarastointi voi edistää liiketoimintaa. *Business Intelligencen* tarkoituksena on siis auttaa yrityksiä ymmärtämään liiketoimintaansa paremmin ja siten tekemään parempia päätöksiä ja samalla parantaa liiketoiminnan tehokkuutta (Rainardi 2007, s. 17). Malinowski ja Zimányi (2008, s. 412) mainitsevat, että *Business Intelligencea* käytetään joskus synonyymina päätöksenteon tukijärjestelmälle (*Decision Support System*, DSS). *Business Intelligence* -järjestelmät ovat kuitenkin enemmänkin liiketoimintaan erikoistuneita päätöksenteon tukijärjestelmiä, joita käytetään myös muuten kuin vain liiketoiminnan apuna.

Joissain yhteyksissä *Business Intelligencestä* on käytetty suomenkielisiä vastineita, kuten *yritystiedon rikastus*, *analyyttinen tiedon hallinta*, *tiedon hallinnan prosessi* ja *liiketoimintatiedon hallinta*, mutta mikään näistä ei ole vakiintunut yleiseen käyttöön kirjallisuudessa tai alan sanastossa (Hovi et al. 2009, s.78).

2.3.2 Asiakkuudenhallinta

Asiakkuudenhallinta on joukko toimia, joita organisaatio tekee hallitakseen ja tehdäkseen analyysiä asiakkaistaan, ylläpitääkseen ja hankkiakseen asiakassuhteita sekä markkinoidakseen ja tuottaakseen uusia tuotteita ja palveluita asiakkailleen (Rainardi 2007, s. 14). CRM-järjestelmät koostuvat sovelluksista, jotka hoitavat asiakkuudenhallintatoimia. Asiakkuudenhallinnassa tietovarastoja voidaan hyödyntää parhaiten muun muassa kampanjanhallinnassa, asiakaspalvelussa ja asiakasanalyysissä (Rainardi 2007, s. 18).

2.3.3 Tiedonlouhinta eli Data Mining

Tiedonlouhinta on prosessi, jolla etsitään toistuvia kuvioita datasta ja ennustetaan datan tulevaa käyttäytymistä kuvioiden perusteella (Rainardi 2007, s. 19). Tarkoituksena on löytää sellaista informaatiota, jota ei pystyisi löytämään suoraan dataa tutki-

malla tai jonka löytäminen olisi liian työlästä. Tiedonlouhintaa voidaan käyttää monenlaisten tietolähteiden kanssa, mutta tietovarastot soveltuvat siihen parhaiten, sillä:

- niiden tieto on valmiiksi siistitty
- ne ovat jäsenneltyjä
- ne sisältävät metatietoa, joka kuvaa itse dataa
- niissä oleva tieto on keskitetty useasta lähteestä
- niiden sisältämä tieto on vakaata ja staattista
- niissä käytetty moniulotteinen tietorakenne sopii hyvin eri tiedonlouhintatehtäviin (Rainardi 2007, s. 19).

2.4 Yhteenveto tietovarastoista

Tässä luvussa esiteltiin tietovarastoja, jotka ovat tiedon historiointiin ja keskittämiseen erikoistuneita tietokantoja, joiden tietomallina käytetään usein moniulotteista tietomallia. Niitä käytetään muun muassa tiedon analysointiin ja raportointiin yritysten päätöksenteon tukena.

3 EXTRACT, TRANSFORM, LOAD

Extract, Transform, Load, suomennettuna poiminta, muokkaus ja lataus, on prosessi, joka hakee tiedot lähteistä, muokkaa ne haluttuun muotoon ja kirjoittaa ne tietovarastoon (Hovi et al. 2009, s. 48). ETL-prosessi on keskeinen osa tietovarastointia, vaikkei se ole mitenkään näkyvissä tietovaraston loppukäyttäjille tai edes tietovaraston pohjalta luotujen sovellusten kehittäjille. ETL-prosessi toimii tietovaraston perustana: se on vastuussa tiedon lataamisesta lähdejärjestelmistä, tiedon laadusta ja eheydestä, eri tietolähteistä tulevan datan oikeanlaisesta yhdistämisestä, ja datan tuottamisesta loppukäyttäjille ja sovelluskehittäjille sopivassa muodossa (Kimball & Caserta 2004, s. xxi). ETL:n suunnittelu ja kehittäminen ovat työläimpiä vaiheita tietovaraston rakentamisessa: niihin kuluu eri arvioiden mukaan yleensä 50–80 % tietovarastointiprojektiin käytetystä työpanoksesta (Veerman et al. 2009, s. 13; Hovi et al. 2009, s.48; Kimball & Caserta 2004, s. xxi).

ETL-prosessi toteutetaan joko ETL-välineellä tai ohjelmoimalla. Ohjelmoidessa voidaan käyttää perinteisiä sovelluskehitykseen tarkoitettuja ohjelmointikieliä tai tietokantakieliä, joilla ETL-prosessi voidaan toteuttaa käyttämällä esimerkiksi niiden tarjoamia tallennettuja proseduureja (*stored procedure*). ETL-prosessin toteuttaminen ohjelmoimalla on kuitenkin työlästä, ja ilman hyvää dokumentointia prosessin ylläpitäminen voi olla hankalaa, jos tekijä vaihtuu. ETL-välineet ovat sovelluskehittimien kaltaisia, ja ne tarjoavat yleensä graafisen suunnitteluympäristön tietojen siirtymisien ja muunnosten suunnitteluun. Moni tietokantavalmistaja tarjoaa ETL-työkalun ilmaiseksi tietokantajärjestelmänsä mukana. (Hovi et al. 2009, s. 53–54, 60.)

Nissen (2003) on vertaillut ETL-työkalujen ja käsin ohjelmoitujen ETL-prosessien etuja, ja Kimball & Caserta (2004) ovat täydentäneet Nissenin (2003) listaa. He luettelevat muun muassa seuraavat ETL-työkalujen edut:

- ETL-prosessin kehittäminen on yksinkertaisempaa, nopeampaa ja halvempaa. Työkalu maksaa itsensä takaisin isoissa tai monimutkaisissa projekteissa.
- ETL-välineitä osaavat käyttää muutkin kuin ohjelmoijat.
- Monet ETL-työkalut saavat prosessin metatiedot tietokannoista ja pakottavat yhtenevään metatiedon käyttöön.

- ETL-välineissä on valmiit yhteyskomponentit, jotka tukevat useimpia tietojärjestelmiä.
- Useimmat ETL-välineet suoriutuvat hyvin jopa suurista datamääristä.
- ETL-työkaluun voidaan lisätä toiminnallisuutta käsin ohjelmoiduilla moduuleilla.

Käsin ohjelmoitujen ETL-prosessien etuja ovat muiden muassa:

- Testausprosessi voidaan automatisoida muun muassa automaattisia yksikkötestaustyökaluja hyödyntämällä.
- Olio-ohjelmointitekniikoita voidaan hyödyntää tekemään transformaatioista yhteneviä.
- Metatietoa voidaan hallita suoremmin kuin työkalupohjaisissa järjestelmissä.
- Käsin ohjelmointi tarjoaa rajatonta joustavuutta. (Nissen 2003, Kimballin & Casertan 2004, s. 10–12 mukaan.)

3.1 Poiminta

ETL-prosessin ensimmäinen vaihe, poiminta, voidaan jakaa kahteen perustoteutusmenetelmään: työntöön ja vetoon. Työntömenetelmässä lähdejärjestelmän puolella ladattavat tiedot kirjoitetaan rajapintatiedostoihin, joista ne sitten poimitaan muokattavaksi ja ladattavaksi tietovarastoon. Myös lähdejärjestelmän puolella voidaan tehdä joitakin muokkausoperaatioita. Vetomenetelmässä lähdejärjestelmän tietokantaan otetaan suora yhteys ja sieltä poimitaan haluttu tieto muokkausta ja latausta varten. Käytännössä toteutusmenetelmiä on useampia: esimerkiksi työntömenetelmää voidaan muokata siten, että tiedostojen sijaan käytetään välitietokantaa, johon tiedot viedään poimittavaksi, tai vetomenetelmää voidaan yksinkertaistaa luomalla näkymiä lähdejärjestelmään. Toteutusmenetelmän valintaan vaikuttavat muun muassa lähdejärjestelmän rakenne ja sen valmistaja. Poiminta tehdään usein erilliseen työtietokantaan (*staging area*, *staging*-kanta) tai -tauluihin, jotta tiedon käsittelyä ei tarvitse tehdä verkon yli. (Hovi et al. 2009, s. 48, 50–53.)

Hovi et al. (2009) luettelevat työntömenetelmän eduiksi seuraavat ominaisuudet:

- ETL-prosessi ei lue tietoja väärään aikaan, koska lähdejärjestelmä kertoo, milloin tiedot ovat valmiita ladattaviksi.

- Monimutkaisten ja vaikeaselkoisten tietokantarakenteiden tapauksessa on hyvä, että lähdejärjestelmän asiantuntijat vastaavat tiedon poiminnasta lähtötauluista.
- Tiedostot muodostavat selkeän rajapinnan.
- Rajapinnan tiedostot voivat auttaa virhetilanteiden selvittämisessä.
- Lähdejärjestelmä voidaan vaihtaa ongelmitta, kun uudesta järjestelmästä tilataan samat tiedot. (Hovi et al. 2009, s. 51.)

Työntömenetelmän etuna on lisäksi se, että sitä käytettäessä lähdejärjestelmä voi olla myös sellainen, josta olisi hankala hakea tiedot suoraan esimerkiksi järjestelmien yhteensopivuusongelmista johtuen. Työntömenetelmän huonoiksi puoliksi Hovi et al. (2009) mainitsevat tietokantaa hankalamman tietojen lukemisen ja vetomenetelmää useampien vaiheiden aiheuttamat mahdolliset lisäkustannukset (Hovi et al. 2009, s. 51).

Vetomenetelmän etuja ovat Hovin et al. (2009) mukaan työntömenetelmää yksinkertaisempi sekä mahdollisesti joustavampi ja nopeampi toteutus. Ongelmiksi he luettelevat seuraavat huonot puolet:

- On olemassa keskeneräisten tietojen lukemisen mahdollisuus.
- Ei samanlaista selkeätä rajapintaa kuin työnnössä.
- ETL-kehittäjät eivät tunne lähdejärjestelmää, joten on mahdollisuus väärinkäsityksiin.
- Tiedon uudelleenlataukset voivat tuottaa eri tietoa lähdejärjestelmän muutosten vuoksi.
- Lähdejärjestelmän vaihtuminen aiheuttaa muutoksia ETL-prosessiin.
- Lähdejärjestelmän rakenteen tulkitseminen voi olla vaikeaa. (Hovi et al. 2009, s. 52.)

3.2 Muokkaus ja lataus

ETL-prosessin toinen vaihe, muokkaus, käsittää tiedon muokkaamisen tietovarastoon sopivaksi. Muokkauksessa tehdyt operaatiot voidaan jakaa kahteen tyyppiin: tarkastukseen ja tiedon muuntamiseen. Tarkastuksessa tiedoista muun muassa etsitään virheellisiä rivejä tai arvoja sekä tehdään muoto- ja raja-arvotarkistuksia. Virheelliset rivit voidaan hylätä, viedä erilliseen virhetauluun tai kirjoittaa tietovarastoon virheel-

lisiksi merkittyinä. Tiedon muuntamisessa on tarkoituksena muokata lähdejärjestelmien tiedot tietovaraston rakenteeseen sopiviksi. Erilaisia muunnoksia ovat muun muassa tietojen yhdistelyt, yksikkömuunnokset, summaukset ja surrogaattiavaimien muodostamiset. (Hovi et al. 2009, s. 56–57.)

Lataus on ETL-prosessin viimeinen vaihe ja siinä ladataan muokatut tiedot tietovarastoon. Latausvaiheessa faktataulujen rivit lisätään yleensä olemassa olevien rivien perään, ja dimensiotauluihin menevillä riveillä joko korvataan vanhat tiedot tai kirjoitetaan uusi rivi ja historioidaan vanha tieto. Täysin uudet rivit lisätään muiden perään uutena. Faktataulujen latauksessa tulee huolehtia viite-eheyksistä dimensiotauluihin, jotta tauluun ei mene sellaisia rivejä, joissa on viittauksia, jotka eivät yhdisty mihinkään dimensiotaulun riviin. (Hovi et al. 2009, s. 58; Kimball & Caserta 2004, s. 212.)

3.3 Moniulotteisen tietovaraston latausprosessi

Moniulotteiset tietomallit ovat yleisimmät loppukäyttäjän kyselyitä ja analyysia varten käytetyt tietorakenteet. Ne ovat yksinkertaisia luoda, vakaita muuttuvien ympäristöjen piirissä, loppukäyttäjien intuitiivisesti ymmärrettävissä sekä nopein tietorakenne relaatiotietokannan kyselyihin. Lisäksi ne ovat OLAP-kuutioiden perustana, sillä käytännössä OLAP-kuutiot ovat vain tehokkaita moniulotteisia malleja toteutettuna erityisohjelmistolla. (Kimball & Caserta 2004, s. 45.) Näistä syistä johtuen tässä työssä keskitytään moniulotteisen tietomallin ETL-prosessiin. Moniulotteisen tietovaraston latausprosessi jakautuu kahteen osaan: dimensiolataukseen ja faktalataukseen. Näiden lisäksi usein käytettyyn *staging*-kantaan tehty lataus on oma prosessinsa.

Staging-latauksen perusrakenne on useimmiten erittäin yksinkertainen sen suoraviivaisuudesta johtuen. *Staging*-latauksessa ei yleensä suoriteta tarkistuksia eikä muunnoksia tiedoille, vaan sen tarkoitus on vain hakea tiedot lähdejärjestelmästä *staging*-kantaan, josta käsin varsinainen tiedon käsittely suoritetaan. Joitakin yksinkertaisia operaatioita, kuten tyyppimuunnoksia, saatetaan tehdä. Syinä *staging*-kannan käyttöön on yleensä, että näin lähdejärjestelmää kuormitetaan mahdollisimman vähän, ja verkon yli tehdyt operaatiot hidastaisivat turhaan prosessia. Lisäksi *staging*-alue tarjoaa palautuspisteen, jolloin tietoja ei tarvitse hakea uudelleen lähteistä, jos jokin

transformaatio epäonnistuu, sekä mahdollisuuden koordinoida latauksia eri lähteistä, joista data on ladattavissa eri aikaan (Inmon 2005, s. 168; Kimball & Caserta 2004, s. 30). *Staging*-latauksessa yleensä otetaan vain tarvittavat sarakkeet lähtötauluista ja haettavaa tietoa voidaan rajata esimerkiksi päivämäärän mukaan. Tämä voidaan mieltää ETL-prosessin ensimmäistä vaihetta, poimintaa, vastaavaksi.

Staging-latauksen jälkeen suoritetaan dimensiotaulujen lataus. Tämä tulee tehdä ennen faktataulujen latausta, koska muuten faktataulujen kenttien avainten korvaaminen dimensiotauluista saaduilla surrogaattiavaimilla ei onnistu. Dimensiolatauksessa erotetaan *staging*-tauluista dimensiotiedot omiin tauluihinsa. Kaikkia dimensiotauluja ei luoda lähtötaulujen pohjalta, vaan osa niistä generoidaan kokonaan latauksessa tai ETL-prosessin ulkopuolella: tällainen dimensiotaulu on esimerkiksi päivämäärädimensiotaulu (Kimball & Caserta 2004, s. 172–173).

Yksi tärkeä osa dimensiolatausta on surrogaattiavaimien luonti. Surrogaattiavaimet voidaan jättää tietokantajärjestelmän hoidettavaksi, jolloin kohdetaulussa on määritelty yksi sarake identiteettisarakkeeksi, joka luo jokaiselle lisätylle riville yksilöllisen avaimen. Toinen vaihtoehto on surrogaattiavainten generoiminen ETL-prosessissa. Surrogaattiavaimien generoimisen jättäminen tietokantajärjestelmän hoidettavaksi yksinkertaistaa taulun latausta, mutta aiheuttaa helposti sen, että testi- ja tuotantokantojen surrogaattiavaimet eivät vastaa toisiaan, mikä voi haitata testaamista. (Kimball & Caserta 2004, s. 164.) Lisäksi joissain tapauksissa taulun lataus saattaa monimutkaistua, jos esimerkiksi samalla ladataan jokin dimensiotauluun linkittyvä taulu. Tällöin pitää ratkaista se, miten avaimet saadaan toiseen tauluun, kun ei tiedetä, mikä kunkin rivin avain tulee olemaan ensimmäisessä taulussa.

Oma osansa dimensiolatauksessa on muuttuneiden tietojen hallinta. Muuttuneita tietoja hallitaan niin sanotuilla hitaasti muuttuvilla dimensioilla (*Slowly Changing Dimension*, SCD). SCD:n kolme perusmenetelmää ovat tyypin 1, 2 ja 3 hitaasti muuttuvat dimensiot. Tyypin 1 SCD on yksinkertainen vanhentuneiden tietojen ylikirjoitus, eikä tietoja historioida mitenkään. Tyypin 2 hitaasti muuttuva dimensio kirjoittaa aina uuden rivin tauluun, kun jokin tieto muuttuu, ja vanha tieto merkitään vanhentuneeksi. Vanhentumisen merkitsemisen voi tehdä yksinkertaisella tosi-/epätosi-sarakkeella, mutta tällöin menetetään tieto voimassaoloajoista. Toinen tapa on käyt-

tää sarakkeita, joihin kirjoitetaan voimassaolon alku- ja loppupäivämäärät. Tyypin 3 SCD:ssä käytetään vanhentuneille arvoille omia sarakkeita. Uusi arvo kirjoitetaan voimassaolevaan sarakkeeseen, ja vanhentunut tieto kirjoitetaan sille määrättyyn sarakkeeseen. Vanhentuneille tiedoille voi olla varattuna useita sarakkeita, jolloin tiedot siirretään aina yhtä saraketta eteenpäin. Dimensiotaulu voi myös käyttää niin sanottua hybridi-SCD-rakennetta, jolloin se sisältää eri SCD-tyypin sarakkeita. Näin sellaisten tietojen päälle, joita ei tarvitse historioida, voidaan kirjoittaa ja muut tiedot voidaan historioida tyypin 2 tai 3 mukaan. (Kimball & Caserta 2004, s. 183–194.)

Faktalataus voidaan suorittaa, kun dimensiolataus on valmis. Faktataulujen latausprosessin rakenne muodostuu pääasiassa dimensiotaulujen perusavaimia vastaavien sarakkeiden arvojen korvaamisesta surrogaattiavaimilla. Luonnollisten avainten korvaamisessa surrogaattiavaimilla tulee huolehtia viite-eheydestä. Jos jokin faktalatauksessa esiin tuleva luonnollinen avain puuttuu dimensioista, ei kyseistä riviä voida kirjoittaa faktatauluun ilman, että viite-eheys rikkoontuu. Monesti tietokantajärjestelmä pakottaa noudattamaan dimensio- ja faktataulujen välisiä viite-eheyksiä, jolloin virheellistä riviä ei edes voisi kirjoittaa tauluun. Tällaiset virheelliset rivit käsitellään tietovaraston määrittelyssä sovitulla tavalla. (Kimball & Caserta 2004, s. 212–215.)

Numeeriset tiedot, kuten kappalemäärät ja hinnat, voidaan viedä sellaisenaan faktatauluun, tai niille voidaan tehdä erilaisia tarkistuksia ja muunnoksia tarpeen mukaan. Uudet rivit sijoitetaan muiden rivien perään, ja vanhentuneet ja väärät tiedot yleensä päivitetään tai poistetaan ja ladataan uudelleen tauluun. Väärien tietojen poistaminen on todennäköisimmin paras vaihtoehto, koska tietojen päivittäminen on raskas operaatio varsinkin silloin, jos faktataulu on iso. Jos tietovaraston käyttötarkoitus sallii, on pienen faktataulun tapauksessa yksinkertaisinta tyhjentää faktataulu kokonaan ja ladata kaikki tieto uudelleen. (Kimball & Caserta 2004, s. 228–231.)

3.4 ETL-prosessin ajaminen

ETL-prosessi ajastetaan alkamaan yleensä ilta- ja yöaikaan, jotta se häiritsisi mahdollisimman vähän muita järjestelmiä ja niiden käyttäjiä. Koska ETL-prosessin aikana tietovaraston tieto ei ole tehokkaasti hyödynnettävissä ETL:n raskauden ja tiedon puutteiden takia, pyritään lataus saamaan valmiiksi yön aikana. Näin ollen latauksen tuottama data on käytettävissä raportointia ja analysointia varten seuraavana päivänä

(Hovi et al. 2009, s. 48, 58). Tiedon määrästä ja sille tehtävästä käsittelystä riippuen latausprosessi voi kestää minuuteista tunteihin. Yleisimmät keinot latauksen nopeuttamiseksi ovat prosessin optimointi ja laitteiston tehon lisäys. Jos mikään muu ei auta, niin on myös mahdollista jakaa ETL osiin niin, että joka yö ladataan tärkeimmät ja tuoreempaa tietoa vaativat osuudet, ja viikonloppuisin ladataan koko data.

3.5 ETL-prosessin kuvaaminen ja mallintaminen

ETL-prosessin kuvaamiseen vaaditaan vähintään tiedot lähde- ja kohdejärjestelmistä sekä tarkat kuvaukset datan käsittelysäännöistä. Yksinkertaisimmillaan kuvaukset voidaan kirjata ylös taulukkolaskentaohjelman taulukkoon. (Hovi et al. 2009, s. 55.) Useat välineet tarjoavat ympäristön ETL:n suunnittelulle. Jokainen ETL-väline käyttää omaa kuvausmalliaan ETL-prosessista, mikä on johtanut siihen, että työtavat ja parhaat käytännöt eroavat työkalusta toiseen. Lisäksi ETL-välineiden muodostamat mallit ovat liian tarkkoja yleiseen ETL:n mallinnukseen ja pakottavat siten suunnittelijan ottamaan toteutuksen huomioon ETL-suunnitteluprosessin alusta alkaen. Tämä johtuu siitä, että ETL:lle ei ole standardoitua kuvausmallia. (Akkaoui & Zimányi 2009, s. 41, 44.)

ETL:n mallintamista ovat tutkineet muun muassa Vassiliadis et al. (2002) ja Trujillo & Luján-Mora (2003). ETL:n mallintamiseen ehdotetut tavat voidaan jakaa kahteen pääkategoriaan. Näistä ensimmäiseen kuuluu olemassa olevan kuvauskielen käyttäminen mallintamisessa ja toiseen varta vasten ETL:ää varten suunnitellun itse kehitetyn kuvauskielen käyttö. Ensimmäiseen kategoriaan sisältyvät muun muassa ETL:n UML-mallinnus (*Unified Modeling Language*) ja BPMN-mallinnus (*Business Process Model and Notation*).

3.5.1 ETL:n mallintaminen UML:llä

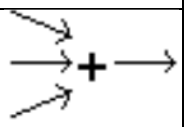
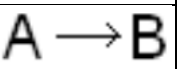
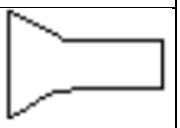
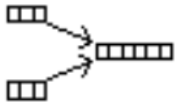
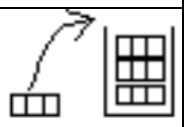
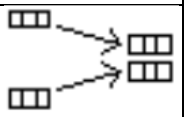
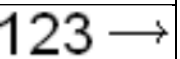
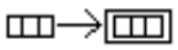
UML on visuaalinen kuvauskieli, joka on tarkoitettu ohjelmistojärjestelmien osien määrittämiseen, rakentamiseen ja dokumentointiin. Se on lähtöisin kolmesta eri oliomallinnuskielestä: *Booch* eli *Object-Oriented Analysis and Design* (OOAD), *Object-Modeling Technique* (OMT) ja *Object-Oriented Software Engineering* (OOSE). *Object Management Group* (OMG) julkaisi UML:n ensimmäisen kerran marraskuussa 1997. (ISO/IEC 19501-1:2012, s. 8.) Mallin esittämiseen UML tarjoaa neljätoista erilaista graafista kaaviota: aktiviteetti-, luokka-, kommunikointi-, komponentti-,

kooste-, sijoittelu-, kokoava vuorovaikutus-, olio-, pakkaus-, profiili-, tila-, sekvenssi-, ajoitus- ja käyttötapauskaavion. (ISO/IEC 19501-2:2012, s. 700–701).

ETL:n UML-mallinnusta ovat esittäneet Trujillo & Luján-Mora (2003) ja Muñoz et al. (2008). Trujillo & Luján-Mora (2003) esittävät UML-pohjaisen käsitemallin ETL-prosessin suunnittelua varten. Syy siihen, miksi he ovat valinneet UML:n mallikseen, on kaksijakoinen: he ovat aiemmissa töissään (Trujillo et al. 2001; Luján-Mora et al. 2002a; Luján-Mora et al. 2002b) esittäneet UML-pohjaista moniulotteisen tietovaraston mallia, joka sopii yhteen heidän ETL-prosessin käsitemallinsa kanssa, ja toisaalta UML:n asema standardina minimoi kehittäjien vaivan opetella uusia kuvioita ETL-prosessin mallintamiseksi. Heidän mukaansa tämä lähestymistapa lyhentää tietovaraston kehittämiseen kuluvaa aikaa, helpottaa datasäilöjen ja tietovaraston hallinnointia sekä antaa suunnittelijan suorittaa riippuvuusanalyysia, kuten tietolähteiden muutoksien vaikutuksia tietovarastossa. (Trujillo & Luján-Mora 2003, s. 308.) Toisaalta ETL-prosessien suunnittelijat ovat yleensä ei-teknisiä henkilöitä, joilla ei ole ohjelmointitaustaa (Akkaoui & Zimányi 2009, s. 42). Näin ollen, vaikka UML on ohjelmistotaustaisille ihmisille tuttu, niin UML:n asema standardina ei kuitenkaan ole kovin iso etu ETL:n kuvaamisessa, koska se on kuitenkin monelle ETL-prosessien tekijälle vieras.

Trujillo & Luján-Mora (2003) käyttävät vain UML:n luokkakaaviota kuvaamaan ETL-prosesseja, koska esimerkiksi ajonaikaista kommunikointia ETL-prosessien välillä ei tarvitse kuvata. Heidän mallissaan koko ETL-prosessi muodostuu UML-paketeista, jotka mahdollistavat prosessin suunnittelun jakamisen omiksi loogisiksi yksiköikseen. Eri ETL-mekanismit esitetään omilla stereotyypitetyillä luokillaan, joita varten he ovat kehittäneet omat kuvakkeensa (taulukko 1). ETL-mekanismit yhdistetään toisiinsa UML-riippuvuussuhteilla, jotka ovat kaavioissa nuolellisia katkoviivoja. UML-kommentteja käytetään mekanismin toiminnallisuuden selittämiseen ja lähde- ja kohdeattribuuttien yhdistämiseen toisiinsa. (Trujillo & Luján-Mora 2003, s. 310.)

Taulukko 1. ETL-mekanismit ja kuvakkeet (Trujillo & Luján-Mora 2003, s. 311).

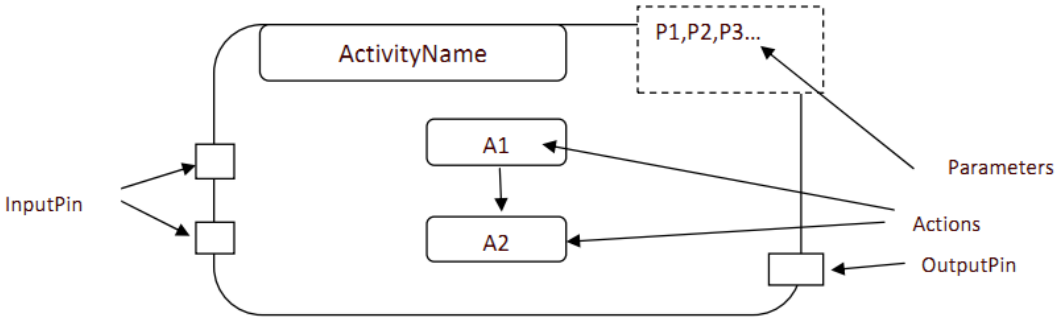
ETL-mekanismi	**Kuvaus**	**Kuvake**
Aggregation (Aggregaatio)	Aggregoi dataa jonkin kriteerin perusteella.	
Conversion (Konversio)	Muuttaa datan tyyppiä ja formaattia, tai johtaa uutta dataa olemassa olevasta datasta.	
Filter (Suodatin)	Suodattaa ja todentaa dataa.	
Incorrect (Epäkelpo)	Uudelleenohjaa epäkelpoa dataa.	
Join (Liitos)	Liittää kaksi tietolähdettä toisiinsa ennaltamääritettyjen attribuuttien perusteella.	
Loader (Lataaja)	Lataa datan ETL-prosessin kohteeseen.	
Log (Loki)	Kirjoittaa lokiin ETL-mekanismin tapahtumat.	
Merge (Yhdistäminen)	Yhdistää kaksi tai useampia tietolähteitä, joilla on yhteensopivat attribuutit.	
Surrogate (Surrogaatti)	Generoi yksilölliset surrogaattiavaimet.	
Wrapper (Päällys)	Muuntaa alkuperäisen datalähteen taulumuotoiseksi datalähteeksi.	

Etuna Trujillon & Luján-Moran (2003) mallissa on se, että se mahdollistaa monimutkaisen ETL-prosessin pilkkomisen joukoksi yksinkertaisia prosesseja. Heikkoutena siinä kuitenkin Muñozin et al. (2008) mukaan on se, että eduista huolimatta Trujillon & Luján-Moran (2003) malli mallintaa staattisia rakenteita eikä mahdollista ETL-prosessin dynaamisten puolien tai käyttäytymisen kuvaamista. (Muñoz et al.

2008, s. 46.) Toisaalta herää kysymys siitä, mihin käyttötarkoitukseen mallia halutaan käyttää. Jos mallia halutaan käyttää kuvaamaan vain ETL-prosessin osat, kuten tietovarastointiprojektin alkuvaiheissa, jossa tarvitaan nopeaa dokumentaatiota järjestelmistä ja niiden välisistä suhteista, niin esimerkiksi käyttäytymistä ei tarvitse huomioida (Vassiliadis et al. 2002, s. 16). Suurin ongelma Trujillon & Luján-Moran (2003) mallissa ETL-prosessin automaattisen generoimisen kannalta on se, että se ei tarjoa tapaa muodostaa ETL-prosessia automaattisesti mallin pohjalta.

Toisin kuin Trujillo & Luján-Mora (2003), Muñoz et al. (2008) käyttävät mallissaan UML:n aktiviteettikaaviota luokkakaavion sijaan. UML:n aktiviteettikaaviot kuvaavat käyttäytymistä, ja niitä käytetään järjestelmän dynaamisten osien kuvaamiseen. Muñoz et al. (2008) suunnittelivat aktiviteettikaavioon mallinnuselementtejä, jotka vastaavat ETL-prosessin toimintoja. Ne mahdollistavat prosessin käyttäytymisen mallintamisen ja ETL-prosessin suunnittelun yhdistämisen tietovarastomalliin. Muñoz et al. (2008) eivät mainitse tutkimuksessaan, onko heidän käytössään oleva tietovarastomalli myös UML-pohjainen, mutta näin voi olettaa, sillä he nojaavat Trujillon & Luján-Moran (2003) esittämään ETL-malliin lähestymistavassaan. Etuna verrattuna Trujillon & Luján-Moran (2003) esittämään malliin Muñoz et al. (2008) mainitsevat ETL-prosessin dynaamisten ominaisuuksien ja prosessin käyttäytymisen arvioinnin mahdollisuuden (Muñoz et al. 2008, s. 45–46.)

ETL-mekanismit Muñoz et al. (2008) lainaavat suoraan Trujillon & Luján-Moran (2003) mallista, mutta niiden esittämistavat eroavat toisistaan kaavioiden erojen vuoksi. Siinä missä Trujillon & Luján-Moran (2003) mallissa yksi luokka vastaa yhtä ETL-mekanismia, Muñozin et al. (2008) mallissa yksi aktiviteettikaavio vastaa yhtä ETL-mekanismia, ja lisäksi ETL-mekanismit on jaettu pienempiin loogisiin osiin kuvan 4 osoittamalla tavalla. Kuvassa *InputPin*-laatikot ovat ETL-komponentin sisään tulevaa dataa eri lähteistä tai komponenteista. Data kulkee näistä *Actions*-laatikoiden A1 ja A2 kautta, joissa sitä käsitellään tarkistuksin ja muunnoksin. Data johdetaan ulos komponentista *OutputPin*-laatikon kautta. Parametrit (kuvassa *Parameters*) kuvaavat komponentissa käytettäviä datan sarakkeita, mutta voivat kuvata myös komponentissa tarvittavia ominaisuuksia. (Muñoz et al. 2008, s. 47–48.)



Kuva 4. Pohja ETL-prosessin määrittämiseksi käsitemallitasolla (Muñoz et al. 2008, s. 48).

Muñozin et al. (2008) esittämän mallin etuna on Trujillon & Luján-Moran (2003) mallin etujen lisäksi se, että se mahdollistaa ETL-prosessin järjestyksen ja käyttäytymisen kuvaamisen. Toisaalta malli on monimutkaisempi kuin Trujillon & Luján-Moran (2003) malli, joten se ei sovellu niin hyvin ETL-prosessin osien nopeaan dokumentointiin. Muñozin et al. (2008) mallia on kuitenkin hyödynnetty ETL-prosessin muodostamisessa automaattisesti mallin pohjalta myöhemmässä tutkimuksessa Muñoz et al. (2009).

3.5.2 ETL:n mallintaminen BPMN:llä

BPMN-mallinnuksen (*Business Process Model and Notation*) tarkoituksena on tarjota helposti ymmärrettävissä oleva graafinen esitystapa liiketoiminnan prosesseista liiketoiminnan eri käyttäjäryhmille. Se tarjoaa tavan formalisoida liiketoiminnan käyttäjien suosiman vuokaavioesitystavan mahdollistamalla liiketoiminnan prosessin visualisoinnin yhdistämisen sopivaan ajoformaattiin, kuten WS-BPEL-kieleen (*Web Services Business Process Execution Language*). (OMG 2011, s. 21.) WS-BPEL:stä voi käyttää myös lyhyempää lyhennettä BPEL. Liiketoiminnan prosessit ovat järjestettyjen tehtävien ryhmiä, jotka kuvaavat, miten työ tehdään organisaatiossa. Niitä esitetään usein XML-pohjaisella ajokielellä, kuten BPEL. BPMN on tarkoitettu ratkaisuksi BPEL:n ongelmiin, joita ovat muun muassa BPEL-työkalujen graafisten notaatioiden monimutkaisuus ja liika tarkkuus liiketoiminnan käyttäjien tarpeisiin, sekä eri työkalujen notaatioiden erilaisuuden aiheuttamat kommunikaatiovaikeudet eri organisaatioiden välillä. (Akkaoui & Zimányi 2009, s. 43, 46.)

ETL:n mallintamista BPMN:llä ovat tutkineet Akkaoui & Zimányi (2009) ja Akkaoui et al. (2011). Akkaoui & Zimányi (2009) esittävät BPMN:ään pohjautuvan

käsitemallin ETL:n mallinnukseen. He luettelevat BPMN:n valinnan syiksi sen tarjoamat työkalut, joilla kuvata liiketoiminnan prosesseja kielellä, jonka voi myöhemmin yhdistää ajokieleen, sekä sen aseman yritysprosessien suunnittelun notaationa. ETL-prosessia voi pitää eräänlaisena liiketoiminnan prosessina, joten BPMN sopii sen mallintamiseen. (Akkaoui & Zimányi 2009, s. 41, 44.)

Akkaouin & Zimányin (2009) käyttämät BPMN-rakenteet, joita heidän mallissaan käytetään ETL-prosessin mallinnukseen, ovat virtaoliot (*flow object*), artefaktit (*artifact*), liitosoliot (*connecting object*) ja uimaradat (*swim lane*). Näistä virtaoliot jaetaan ETL-tehtäviin (*ETL task*), jotka ovat työnkulun rakenteita, ja ohjausolioihin (*control object*), jotka esitetään BPMN:n ohjaustoiminnoilla. ETL-tehtävät ovat yksittäisiä tai osista koostuvia yksiköitä työnkulussa, jonka kuvaamisessa ne ovat keskeisiä. (Akkaoui & Zimányi 2009, s. 44.)

BPMN:llä ETL-tehtävät kuvataan aktiviteeteilla (*activity*) tai aliprosesseilla (*subprocess*). Aktiviteetti on jakamaton tehtävä, ja aliprosessi on kokoelma aktiviteetteja. Tehtävät voivat toimia myös silmukoina, jolloin tehtävää suoritetaan uudelleen, kunnes määritetty ehto täyttyy. ETL-tehtävät jaetaan vielä rivioperaatioihin (*row operation*), rivistöoperaatioihin (*rowset operation*) ja ohjausoperaatioihin (*control operation*). Rivioperaatiot suoritetaan nimensä mukaisesti yhdelle riville kerrallaan ja rivistöoperaatiot taas ryhmälle rivejä. Ohjausoperaatiot hallitsevat muun muassa tiedostoja ja tiedonsiirtoa verkossa. Ohjausoliot pitävät yllä työnkulun järjestystä erillään prosessin läpi kulkevasta datasta. Ohjausolioina käytetään BPMN:n ohjaustoimintoja, joita ovat portit (*gateway*) ja tapahtumat (*event*). Porteilla ohjataan prosessin kulkua ehtojen perusteella: niillä voidaan muun muassa yhdistää tai jakaa tietovirtaa. Tapahtumat puolestaan esittävät työnkulkuun vaikuttavia tapahtumia, jotka jaetaan aloitus-, väli- ja lopetustapahtumiin. Ne voivat olla omillaan tai jonkin aktiviteetin tai aliprosessin yhteydessä työnkulussa. (Akkaoui & Zimányi 2009, s. 44.)

BPMN:n pääartefakteja ovat huomautukset (*annotation*) ja tieto-oliot (*data object*). Akkaouin & Zimányin (2009) mallissa huomautukset on jaettu kahdeksi uudeksi olioksi: datahuomautukseksi ja porttiehdoksi. Datahuomautuksia käytetään tarkentamaan ETL-tehtävän semantiikkaa lisäämällä tehtävän yhteyteen muun muassa sisään- ja ulosmenevän datan tiedot, parametreja ja aloitus- ja lopetusehtoja. Porttieh-

dot määrittävät nimensä mukaisesti portissa käytettävät ehdot. Tieto-olioita käytetään kuvaamaan tehtävien välillä siirtyviä dokumentteja. (Akkaoui & Zimányi 2009, s. 45.)

BPMN sisältää kolme erilaista liitosoliota: järjestys- ja viestivirran (*sequence flow, message flow*) sekä assosiaation (*association*). Järjestysvirta on perusliitin, joka yhdistää oliot toisiinsa. Se määrää tietovirran järjestyksen, ja seuraavaa oliota ei suoriteta, ennen kuin edellinen järjestysvirran siihen yhdistämä olio on suoritettu. Järjestysvirrasta on vielä kaksi erityistapausta, joista ensimmäisessä järjestysvirtaan on liitetty ehto, ja toisessa määritetään oletusvirta, jos on monta jakautuvaa virtaa. Viestivirta kuvaa eri altaiden (*pool*) välillä kulkevia viestejä. Viestivirta on ainoa olio, joka voi yhdistyä altaan ulkopuolelle. Assosiaatiot puolestaan yhdistävät artefaktit virtaolioihin. (Akkaoui & Zimányi 2009, s. 45–46.)

Uimarata on organisointiolio, joka koostuu altaasta ja radoista (*lane*), jotka määräävät prosessin rajat. Yhden työnkulun tulee sisältyä vain yhteen altaaseen, mutta allas voi jakautua useiksi radoiksi, jotka esittävät eri rooleja tai palveluita yrityksessä. Akkaouin & Zimányin (2009) mallissa uimaradat mahdollistavat erilaisten ja monitasoisten ETL-prosessien organisoimisen ja hierarkisoimisen. Uimaradoilla voi organisoida ETL-prosessin teknisen arkkitehtuurin, käyttäjäprofiilin tai liiketoimintayksiköiden mukaan. (Akkaoui & Zimányi 2009, s. 46.)

Akkaoui & Zimányi (2009) käyvät tutkimuksessaan vielä läpi BPMN:n kääntämistä BPEL:iin, koska he ovat valinneet BPEL:in hallitsemaan ETL-ajomoottoria. Jotta BPMN:n voi kääntää BPEL:ksi, suunnittelijan täytyy tarjota vielä lisätietoa prosessista, samoin kuin kääntäessä muita käsitemalleja loogiseen malliin. Jokainen korkean tason tapahtuma tulee tarkentaa yksityiskohtaisiksi aliprosesseiksi, kunnes saadaan toteutettavissa olevia ETL-operaatioita. Tämän jälkeen käännös voidaan suorittaa esimerkiksi automaattisen työkalun avulla. Vaihtoehtoisesti korkean tason mallista voidaan ensin tehdä käännös, jonka jälkeen käännöksestä saatu epätarkka pohja täytetään. (Akkaoui & Zimányi 2009, s. 46.)

Akkaouin & Zimányin (2009) mallilla on samat edut kuin Muñozin et al. (2008) mallilla. Lisäksi sen etuna on se, että siitä on mahdollista muodostaa valmis ETL-prosessi BPEL:n avulla (Akkaoui & Zimányi 2009, s. 46), vaikkakin he hylkäävät

BPEL:n myöhemmässä työssään Akkaoui et al. (2011). He myös mainitsevat, että koska BPMN:ää käytetään liiketoiminnan prosessien kuvaamiseen, loppukäyttäjien ei tarvitse opetella uutta kieltä ETL-prosessien määrittämiseen (Akkaoui & Zimányi 2009, s. 48). Tämä väite perustuu siihen, että yleensä ETL-prosessien kehittäjien taustana on liiketoiminta eikä tietotekniikka (Akkaoui & Zimányi 2009, s. 42). Toisaalta, siinä missä tietotekniikkataustaisille kehittäjille esimerkiksi Trujillon & Luján-Moran (2003) esittämä UML-pohjainen malli on helposti omaksuttavissa, BPMN-pohjainen malli saattaa olla heille vieras. Ongelmana mallissa on huomautusten käyttö, sillä ne saattavat saada malliin aikaiseksi tungosta.

Akkaoui et al. (2011) hyödyntävät aiemmin esittämäänsä mallia Akkaoui & Zimányi (2009). He kuitenkin hylkäävät BPEL:in käytön, koska se ei sovellu riittävän hyvin usean konkreettisen kehittämisalustan tukemiseen. He ovat nimenneet mallinsa BPMN4ETL:ksi ja kertovat parantaneensa mallia mahdollistamalla sen käytön valmistajasta riippumattomana osana heidän alustaansa. (Akkaoui et al. 2011, s. 46–47.) Koska Akkaoui et al. (2011) eivät mainitse tarkemmin mallinsa parannuksia, vaan keskittyvät alustan muihin osiin, käydään heidän esittämäänsä työtä tarkemmin läpi myöhemmin ETL:n automaattisen generoimisen yhteydessä.

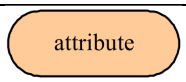
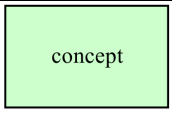
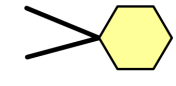
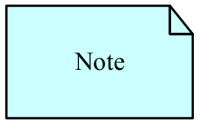
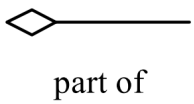
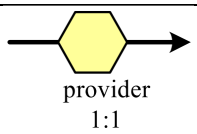
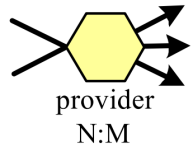
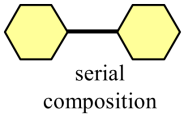
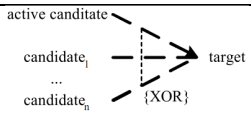
3.5.3 ETL:n mallintaminen itse kehitetyllä kuvauskielellä

ETL:n mallintamista omalla kuvauskielellä ovat tutkineet eniten kreikkalaiset Panos Vassiliadis, Alkis Simitsis ja Spiros Skiadopoulos tutkimuksissaan Vassiliadis et al. (2002) ja Simitsis & Vassiliadis (2003). Vassiliadis et al. (2002) esittävät käsitemallin, jonka painopiste on eri attribuuttien ja käsitteiden välisissä suhteissa sekä tietovaraston latauksen yhteydessä vaadittavissa transformaatioissa. Simitsis & Vassiliadis (2003) havainnollistavat mallin käyttöä. Heidän esittämänsä yleiskäyttöinen metamalli käyttää pientä joukkoa geneerisiä rakenteita, joilla kaikki tapaukset voidaan esittää. Tämä on metamallitaso (*metamodel layer*) heidän arkkitehtuurissaan. Lisäksi käytössä on erikoistumismekanismi, joka mahdollistaa suunnittelijalle erilaisten ETL-toimintojen luomisen periyttämällä ne geneerisemmistä rakenteista samankaltaisesti kuin oliomallinnuksessa. Nämä ETL-rakenteet muodostavat alijoukon laajemmasta metamallitasosta ja kuuluvat mallipohjatasoon (*template layer*). Kaikki suunnittelijan käytössä olevat rakenteet ovat metamallitason mekanismeja, ja mallipohjatasolla olevat mekanismit periytyvät metamallitasolta. Vassiliadis et al. (2002)

painottavat, että heidän mallinsa ei ole prosessi- eikä työnkulkumalli. Syiksi he sanovat, että ETL-prosessin käsitemallissa pääpainona on tietolähteiden dokumentointi ja formalisointi suhteessa tietovarastoon eikä teknisen ratkaisun tarjoaminen prosessin toteutusta varten, ja että ETL:n käsitemallin rakentaminen tapahtuu aikaisessa vaiheessa tietovarastointiprojektia, jolloin tarkan kuvauksen sijaan tarvitaan nopeaa dokumentaatiota tietojärjestelmistä ja niiden välisistä suhteista. (Vassiliadis et al. 2002, s. 15–16.)

Käsitemallinsa graafista esittämistä varten Vassiliadis et al. (2002) ovat kehittäneet joukon kuvakkeita (taulukko 2). Malli ottaa vaikutteita UML:stä ja ER-mallista (*Entity-Relationship*). Esimerkiksi attribuuttien rooli on sama kuin ER-mallissa, huomautukset ovat suoraan UML:stä lainattuja ja osa jotakin -suhde (*part of*) toimii samankaltaisesti kuin UML:ssä. Taulukosta 2 näkee eri mekanismien selitykset, mutta osaa niistä on syytä tarkentaa enemmän. Käsitteet (*Concept*) ovat muun muassa tiedostoja, tauluja lähdekannassa tai tietovarastossa, ja niiden yhteydessä määritetään joukko attribuutteja osa jotakin -suhteella, jota käytetään erityisesti painottamaan sitä, että attribuutit toimivat erillään käsitteistä. ETL-rajoitetta (*ETL Constraint*) käytetään esimerkiksi määrittämään primääriavaimia. Ehdokassuhteella (*Candidate Relationship*) halutaan ilmaista mahdolliset lähteet tietovarastotaulun täyttämiseen, koska tietovarastointiprojektin alkuvaiheissa voi olla useita mahdollisia lähteitä. Jos vain yksi voidaan valita, siitä ilmoitetaan {XOR}-merkinnällä. Valittu ehdokassuhde eli aktiivinen ehdokassuhde (*Active Candidate Relationship*) erotetaan muista nuolella. Koska joitakin transformaatioita tulee tehdä perätysten joukolle attribuutteja, ilmaistaan transformaatioiden sarjasommittelulla, että transformaatioiden välillä kulkee useita attribuutteja. (Vassiliadis et al. 2002, s. 16–19.)

Taulukko 2. Käsitemallin jäsenet (Vassiliadis et al. 2002, s. 16–19).

Jäsen	**Kuvaus**	**Kuvake**
Attribute (Attribuutti)	Yksittäinen tietomoduuli.	
Concept (Käsite)	Lähdetietokannassa tai tietovarastossa esiintyvä kokonaisuus.	
Transformation (Transformaatio)	Abstraktio yhden tehtävän suoritusta esittävästä ohjelmakoodista tai koodin osista.	
ETL Constraint (ETL-rajoite)	Yksittäinen transformaatio, joka suorittaa jonkin rajoitteen rajatulle määrälle attribuutteja.	
Note (Huomautus)	Vapaamuotoinen kommentti.	
Part of Relationship (Osa jotakin -suhde)	Havainnollistaa jonkin komponentin olevan osa toista komponenttia.	
Provider 1:1 Relationship (Tuottaja 1:1 -suhde)	Yhdistää lähdeattribuutin kohdeattribuuttiin jonkin transformaation kautta.	
Provider N:M Relationship (Tuottaja N:M -suhde)	Yhdistää ryhmän lähdeattribuutteja ryhmään kohdeattribuutteja jonkin transformaation kautta.	
Transformation Serial Composition (Transformaatioiden sarjasommittelu)	Yhdistää kaksi transformaatiota johdonmukaisesti.	
Candidate / Active Candidate Relationship (Ehdokassuhde / aktiivinen ehdokassuhde)	Määrittää joukon lähde-ehdokkaita, jotka saattavat päätyä tietovaraston tauluun. Aktiivinen ehdokassuhde on valittu tietovaraston taulun osaksi.	

Vassiliadisin et al. (2002) esittämä käsitemalli erottuu selvästi muiden joukosta. Sen lisäksi, että se on suunniteltu alusta alkaen ETL:n mallintamiseen perustumatta mihinkään olemassa olevaan kuvauskieleen, se pyrkii mallintamaan vain ETL-prosessin eri osat ottamatta kantaa työnkulkuun. Tämä on selkeä etu, koska näin mallin täyttäminen on yksinkertaisempaa ja nopeampaa, ja malli on mahdollista täyttää suoraan esimerkiksi asiakastapaamisen aikana, jolloin se toimii osana tapaamisen muistiinpanoja.

Yhtenä ongelmana Vassiliadisin et al. (2002) mallissa on kuitenkin se, että sen kanssa kilpailee vahvasti taulukkolaskentaohjelman taulukko. Taulukko on vielä yksinkertaisempi täyttää ja voi hyvin suunniteltuna sisältää lähes samat asiat yhtä selkeästi kuin Vassiliadisin et al. (2002) malli. Lisäksi hyvin suunnitellun taulukon ymmärtämiseen ei tarvita erilaisten kuvioiden opettelua. Myös huomautusmerkintöjen käyttöön turvautuminen on Vassiliadisin et al. (2002) mallissa ongelma, sillä kaaviot voivat täyttyä helposti niistä. Lisäksi olemassa oleva kirjallisuus ei tarjoa mallille nimeä ja sen yhteyteen kehitetyn ARKTOS II -suunnitteluympäristön tuki on lopetettu (Vassiliadis et al. 2000), joten sen tämänhetkinen tilanne on epäselvä. Vaikka mallin viemistä lähemmäksi varsinaisen ETL-prosessin toteutusta on tutkittu muodostamalla looginen malli käsitemallista (Simitsis 2005), ETL-prosessin generoiminen sen pohjalta ei tällä hetkellä tarjolla olevan tutkimustiedon mukaan ole mahdollista. Trujillo & Luján-Mora (2003, s. 319) huomauttavat, että koska Vassiliadisin et al. (2002) mallissa jokainen attribuutti on oma kuvionsa, ja niitä voi olla jopa satoja, mallista voi tulla huomattavan monimutkainen.

3.6 ETL-prosessin automaattinen generoiminen

ETL-prosessin automaattista generointia ovat tutkineet muun muassa Muñoz et al. (2009) ja Akkaoui et al. (2011). Jotta ETL-prosessi voidaan generoida automaattisesti, pitää olla olemassa jonkinlainen malli, jonka pohjalta generointi tapahtuu. Akkaoui et al. (2011) ovat tutkineet ETL:n automaattista generoimista pohjautuen aiemmin teoksessaan Akkaoui & Zimányi (2009) esittämäänsä BPMN4ETL-malliin. ETL-prosessin kehittämisessä he hyödyntävät MDD-lähestymistapaa (*Model-Driven Development*). MDD on lähestymistapa ohjelmistokehitykseen, missä tehdään kattavia malleja, ennen kuin kirjoitetaan ohjelmakoodia. Heidän alustassaan käytetään

BPMN4ETL-mallia suunnittelemaan valmistajasta riippumaton ETL-prosessi, jonka pohjalta generoidaan valmistajakohtainen toteutus. Heidän arkkitehtuurinsa jakautuu neljään tasoon *Meta-Object Facilities* -metamallinnusarkkitehtuurin (MOF) mukaan: malliesiintymä-, malli-, metamalli- ja metametamallitasoihin (*Model Instance Layer*, M0; *Model Layer*, M1; *Meta-Model Layer*, M2; *Meta-Meta-Model Layer*, M3). Malliesiintymätaso (M0) edustaa fyysistä järjestelmää, jossa ETL:n suunnittelu ja kehitys tapahtuvat. Mallitasolla (M1) ETL-prosessimalli suunnitellaan, ja sopivilla muunnoksilla saadaan aikaiseksi ETL-prosessin ohjelmakoodi, jolloin siirrytään suunnittelusta toteutukseen. Metamallitaso (M2) koostuu suunnitteluvaiheessa BPMN4ETL-metamallista ja toteutusvaiheessa neljännen sukupolven ohjelmointikielestä. Metametamallitaso (M3) vastaa suunnitteluvaiheessa MOF:n metametamallia ja toteutusvaiheessa Backus–Naur-muotoa (BNF, *Backus–Naur Form*). Näistä mallitaso ja metamallitaso kuvaavat ETL:n semantiikkaa. (Akkaoui et al. 2011, s. 47.)

Akkaouin et al. (2011) alustassa valmistajasta riippumaton ETL-malli luodaan käyttäen BPMN4EL-metamallin tarjoamia rakenteita, jotka esiteltiin luvussa 3.5.2. Valmistajakohtainen toteutus tukee Oracle- ja Microsoft-alustoja, ja siinä käytetään hyväksi laajennettua Backus–Naur-muotoa (EBNF, *Extended Backus–Naur Form*) määrittämään kielioppi (*grammar*) Oraclen OMB:ta (*Orable MetaBase*) ja Microsoftin C#:a varten. Kieliopilla rakennetaan valideja ETL-rakenteita. ETL-prosessin koodi generoidaan automaattisesti ETL-mallista OMG:n M2T-standardin (*Model-to-Text*) transformaatioilla. M2T-transformaatiot yhdistävät metamallin kielioppiin M2-tasolla, jotta lopputulos voidaan ajaa M1-tasolla. M2T järjestää transformaatiot ohjelmakoodipohjiin (*template*), joiden tehtävänä on generoida koodia lähdemallin osista. Jokainen ohjelmakoodipohja sisältää yhtä metamallin käsitettä vastaavan ohjelmakoodivastineen. Ohjelmakoodi voi olla staattista, jolloin se ajetaan sellaisenaan, tai dynaamista, jolloin siihen tehdään muutoksia ajon aikana. Ohjelmakoodipohjan sisältämä ohjelmakoodi riippuu kohteena olevasta ETL-työkalusta. (Akkaoui et al. 2011, s. 48–50.)

Koska eri ETL-välineiden ETL-ohjelmakoodi noudattaa samanlaista toimintasuunnitelmaa, Akkaoui et al. (2011) ovat määrittäneet kaavion (*pattern*) abstrakteille ohjelmakoodipohjille. Näistä abstraktioista näkee tyypillisen ohjelmakoodipohjan luomisjärjestyksen. Abstraktit ohjelmakoodipohjat toimivat olio-ohjelmoinnin abstrakti-

en luokkien kaltaisesti, ja niistä voidaan muodostaa ohjelmakoodipohja muokkaamalla niiden ohjelmakoodia. Abstraktien ohjelmakoodipohjien kaavion lisäksi on muita kaavioita, joiden tehtävänä on hallita uutta ETL-ohjelmakoodia. Joitakin ohjelmakoodipohjia ei voi sisällyttää abstraktien ohjelmakoodipohjien kaavioon, mutta ne voidaan lisätä ETL-ohjelmakoodiin toteutuksen aikana. Tämä johtuu siitä, että vaikka eri ETL-työkaluilla on paljon yhteistä, niissä on myös eroavaisuuksia, jotka pakottavat määrittämään jotkin rakenteet eri tavalla. (Akkaoui et al. 2011, s. 50–51.)

Akkaoui et al. (2011) mainitsevat lähestymistapansa eduksi sen, että se antaa suunnittelijan keskittyä ETL:n semantiikkaan huolehtimatta välinekohtaisista toteutuskysymyksistä. Itse mallin edut ovat uudelleenkäytettävyys ja yhteensopivuus. Kun malli muunnetaan ETL-prosessiksi, sen etuja ovat abstraktit ohjelmakoodipohjat transformaatioille, jotka mahdollistavat tuen eri ETL-työkaluille, ohjelmakoodin validointi, tehokkuus ja ohjelmakoodin laadun paraneminen. (Akkaoui et al. 2011, s. 46.) Kaikki Akkaouin et al. (2011) mainitsemat edut ovat valideja, mutta mallin haittapuolena on se, että se pyrkii kuvaamaan ETL-prosessin täydellisenä, joten siinä joudutaan alusta alkaen miettimään toteutuskysymyksiä, eikä se siten sovellu niin hyvin nopeaan mallinnukseen.

Muñoz et al. (2009) puolestaan hyödyntävät MDA-lähestymistapaa (*Model-Driven Architecture*) ETL-prosessin automaattisessa generoimisessa. MDA luottaa perinteiseen ideaan pitää järjestelmän toiminnot erillään alustasta. Tällä tavalla MDA kehottaa luomaan alustasta riippumattoman mallin (PIM, *Platform Independent Model*), joka ei sisällä tietoa alustasta tai toteutusvälineistä. PIM voidaan muuntaa yhdeksi tai useammaksi alustakohtaiseksi malliksi (PSM, *Platform Specific Model*) sisällyttämällä siihen alustan ja toteutusteknologian tiedot. PSM-mallit voidaan ajaa toteutusalustalla, jolloin tuloksena on valmis ohjelmisto. Muñoz et al. (2009) hyödyntävät aiemmassa tutkimuksessaan Muñoz et al. (2008) luotua UML:n aktiviteettikaavioita käyttävää mallia PIM-mallina, josta muodostetaan PSM-mallit käyttäen QVT-kieltä (*Query/View/Transformation*). (Muñoz et al. 2009, s. 34–35.)

Muñoz et al. (2009) keskittyvät työssään alustasta riippumattoman mallin muuntamiseksi alustakohtaiseksi malliksi. Heidän mukaansa huomattavin etu tässä lähestymistavassa on se, että kun PIM-malli on suunniteltu, ETL-prosessin PSM-malli luodaan

automaattisesti, jolloin tuottavuus paranee ja kehityskustannukset ja -aika vähenevät. Lisäksi, jos uusi ETL-teknologia syntyy, uutta PIM-mallia ei tarvitse suunnitella. Kun QVT-kielinen muunnos on saatu valmiiksi, sitä voidaan hyödyntää kaikkien PIM-mallien kanssa erilaisten PSM-mallien luomiseen eri projekteissa. Jokaista ETL-työkalua varten suunnitellaan oma PSM-mallinsa. Muñoz et al. (2009) suosittelevat metamallin luomista jokaista ETL-työkalua varten, koska jokainen PSM-malli riippuu käytettävästä ETL-teknologiasta. (Muñoz et al. 2009, s. 35–36.) Muñozin et al. (2009) lähestymistavassa on samat hyvät ja huonot puolet kuin Akkaouin et al. (2011) lähestymistavassa.

3.7 Päätelmät ETL-prosessin kehittämisen nopeuttamisesta

Kuten Vassiliadis et al. (2002, s. 16) huomauttavat, tietovarastointiprojektin alkuvaiheessa projektin aikataulupaineet vaativat nopeaa dokumentointia tietojärjestelmistä ja niiden välisistä suhteista. Yksi syy käsittelysääntöjen dokumentoinnin helppouden ja nopeuden vaatimukselle on, että näitä sääntöjä saatetaan kerätä jonkin palaverin aikana, jolloin monimutkaisten rakenteiden miettimiselle ei jää aikaa. Näiden projektin alussa saatujen käsittelysääntöjen kuvausten pohjalta voidaan aloittaa ETL-prosessin suunnittelu. Lähimmäksi nopean dokumentoinnin vaatimusta tässä työssä esitellyistä lähestymistavoista ovat päässeet Vassiliadis et al. (2002) ja Trujillo & Luján-Mora (2003). Nämä lähestymistavat eivät kuitenkaan ole yhtä yksinkertaisia kuin taulukkolaskentaohjelman taulukon täyttäminen, eikä niiden pohjalta ole tällä hetkellä mahdollista generoida ETL-prosessia automaattisesti.

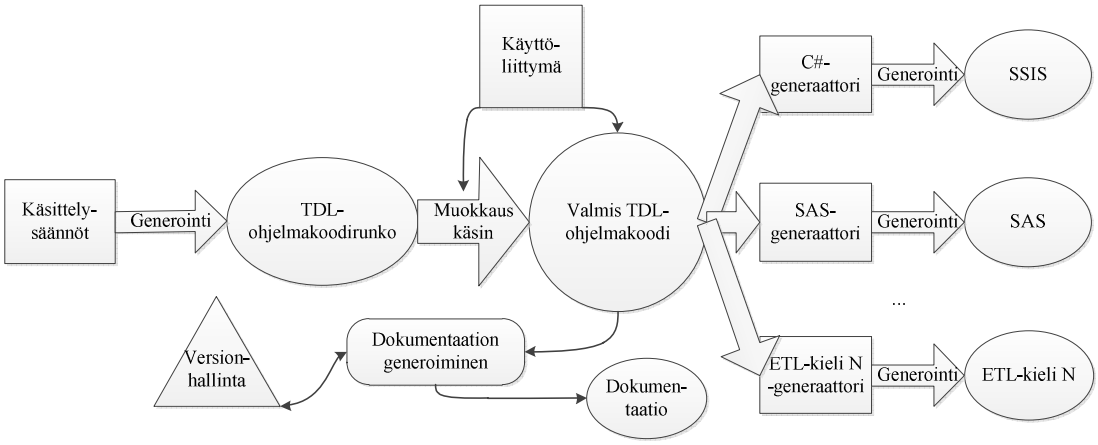
Toisaalta täydellistä ETL-prosessia ei voida generoida automaattisesti pelkkien käsittelysääntöjen pohjalta, sillä siihen vaaditaan tietoa myös ETL-prosessin työnkulusta. Pelkkien käsittelysääntöjen sijaan Muñoz et al. (2009) ja Akkaoui et al. (2011) käyttävät kuvauskieliä, jotka mahdollistavat ETL-prosessin generoimisen automaattisesti. Ne eivät kuitenkaan ole yksinkertaisia eivätkä nopeasti täytettävissä, joten niitä ei voi hyödyntää tietovarastointiprojektin alkuvaiheen dokumentoinnissa. Niiden käyttö vaatii sen, että ensin tietojärjestelmät ja niiden väliset suhteet dokumentoidaan jonnekin projektin alussa, ja kun ollaan valmiita aloittamaan ETL-prosessin suunnittelu, tiedot syötetään malliin. Tästä syntyy päällekkäistä työtä, kun tiedot joudutaan syöt-

tämään kahteen kertaan: ensin esimerkiksi taulukkolaskentaohjelman taulukkoon ja sitten malliin.

Edellä mainituista asioista voidaan päätellä, että optimaalinen tilanne olisi seuraavanlainen. Ensin tiedot syötetään yksinkertaiseen malliin, joka tarjoaa nopean dokumentoinnin. Tähän sopii parhaiten hyvin suunniteltu taulukkolaskentaohjelman taulukko. Seuraavaksi käsittelysääntöjen määrittelydokumentin pohjalta generoidaan automaattisesti monimutkaisemman ETL-prosessin kuvauskielen runko, jossa on määrittelydokumentin pohjalta saadut tiedot. Tämän kuvauskielen tulee olla ETL-prosessin kuvaukseen sopiva ja vastata kaikkiin ETL-prosessin vaatimuksiin sekä tukea olemassa olevia ETL-työkalujen käyttämiä kieliä. Tästä kuvauskielestä voidaan sitten generoida automaattisesti ohjelmakoodimalleja hyödyntäen eri ETL-alustoille valmiit ETL-prosessit. Koska kaikkia tässä työssä esitetyissä tutkimuksissa käytettyjä kuvauskieliä ei ole alusta alkaen suunniteltu ETL-prosessien kuvaamiseen, niissä on useita ongelmia ETL-prosessien kuvaamisen kanssa, kuten tarve käyttää kaaviot täyttäviä kommenttikenttiä latausprosessin kuvaamisessa. Lisäksi niille ei ole esitetty tapaa luoda runkoa automaattisesti käsittelysääntöjen pohjalta. Edellä mainittujen syiden takia paras vaihtoehto olisi luoda uusi kieli, joka on varta vasten ETL-prosesseja varten suunniteltu. Uuden kielen luominen on kuitenkin tämän diplomityön laajuuden ulkopuolella, joten sen luominen jää tulevaisuuteen.

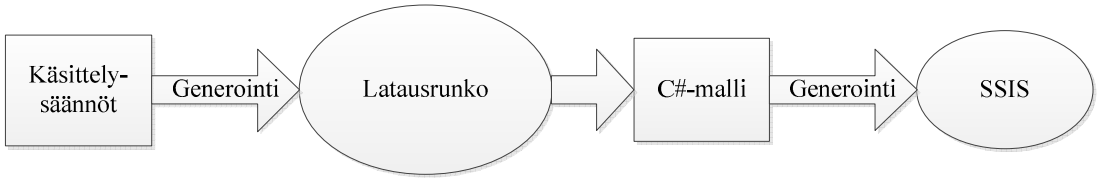
Kuva 5 esittää mahdollista tulevaisuuden tilannetta osittaiseksi ETL-prosessin automaattiseksi generoimiseksi. Käsittelysäännöt määritetään taulukkolaskentaohjelman taulukoissa, joiden pohjalta generoidaan ETL-prosessikielen ohjelmakoodirunko. Kuvassa ETL-prosessikielelle on annettu työnimi TDL, *Transformation Description Language*. Kun runko on saatu valmiiksi, sitä työstetään käsin käyttöliittymän kautta, joka pystyy esittämään TDL-ohjelmakoodin käyttäjälle graafisesti. Tällöin sen visuaalinen esitys on koko ajan kehittäjälle näkyvissä. Kun valmis TDL-ohjelmakoodi on luotu, voidaan sen perusteella generoida valmiit latausohjelmat käyttäen kohdealustalle tarkoitettua generaattoria. Kuvassa SSIS viittaa *SQL Server Integration Services* -pakettiin, SAS (*Statistical Analysis System*) SAS Instituten SAS-ETL-välineen käyttämään ETL-kieleen ja ”ETL Kieli N” siihen, että tarkoituksena on mahdollistaa TDL-kielen kääntäminen useiden eri ETL-välineiden tukemaan kieleen. Versionhal-

lintaa käytetään apuna dokumentaation generoinnissa. Sieltä haetaan muun muassa versiohistoria, muutoksen tekijä ja tekijän kommentit.



Kuva 5. ETL-prosessin automaattisen generoimisen tulevaisuuden tavoitetila.

Koska ETL-prosessikielen luominen joudutaan rajaamaan ulos tästä työstä, käytetään tämän työn käytännön osuudessa kuvaan 5 verrattuna yksinkertaistettua prosessia (kuva 6). Käsittelysääntöjen pohjalta generoidaan *staging*-, dimensio- tai faktalatauksen runko, jonka pohjalta muodostetaan SSIS-latauspaketin runko SSIS:n ohjelmointirajapinnan avulla.



Kuva 6. Työssä saavutettava tilanne.

3.8 Yhteenveto ETL-prosessista

Tässä luvussa esiteltiin ETL-prosessia sekä tutkimuksia sen kuvaamiseksi valmistajasta riippumattomalla tavalla. Lisäksi käsiteltiin ETL-prosessin osittain automaattista generoimista ja tutkimusten pohjalta esitettiin arkkitehtuurimalli ETL-prosessin automaattisen generoimisen tulevaisuuden tavoitetilasta. Täyttä arkkitehtuurimallia ei kuitenkaan toteuteta työn käytännön osuudessa, koska se olisi liian suuri tehtävä diplomityön laajuuteen nähden.

4 SQL SERVER INTEGRATION SERVICES

Tässä työssä käytetty ETL-väline on *SQL Server Integration Services* (SSIS). Ensisijainen syy välineen valintaan on sen käyttö Prima-raportointi-projektissa. Lisäksi sen yleisyys, sen ohjelmointirajapinnan mahdollistama latausprosessin ohjelmallinen käsittely ja sen asema toisena pääasiallisena ETL-toteutusvälineenä kohdeyrityksessä ovat painavia syitä sen käyttöön tässä työssä.

SQL Server Integration Services (SSIS) on osa Microsoftin *SQL er* -tuotepakettia. Se on yksi kolmesta *Microsoft SQL Server Business Intelligence* -alustan tuotteesta. Muita alustan tuotteita ovat *SQL Server Analysis Services* (SSAS) ja *SQL Server Reporting Services* (SSRS). Siinä missä SSIS on alustan ETL-työkalu, SSAS:llä työstetään moniulotteisia kuutioita, ja SSRS:ää puolestaan käytetään raportointiin. Yksinkertaisimmista yritystason ETL-työkaluista poiketen SSIS sisältää graafisen vedä ja pudota -kehitysympäristön, josta löytyy valmiita työkaluja muun muassa tiedonpuhdistukseen. Lisäksi sen toimintaa voi laajentaa kustomoiduilla komponenteilla, jotka on tehty .NET-kielillä, tai käyttämällä sen sisäänrakennettuja skriptauskomponentteja. (Knight et al. 2008, s. 1.)

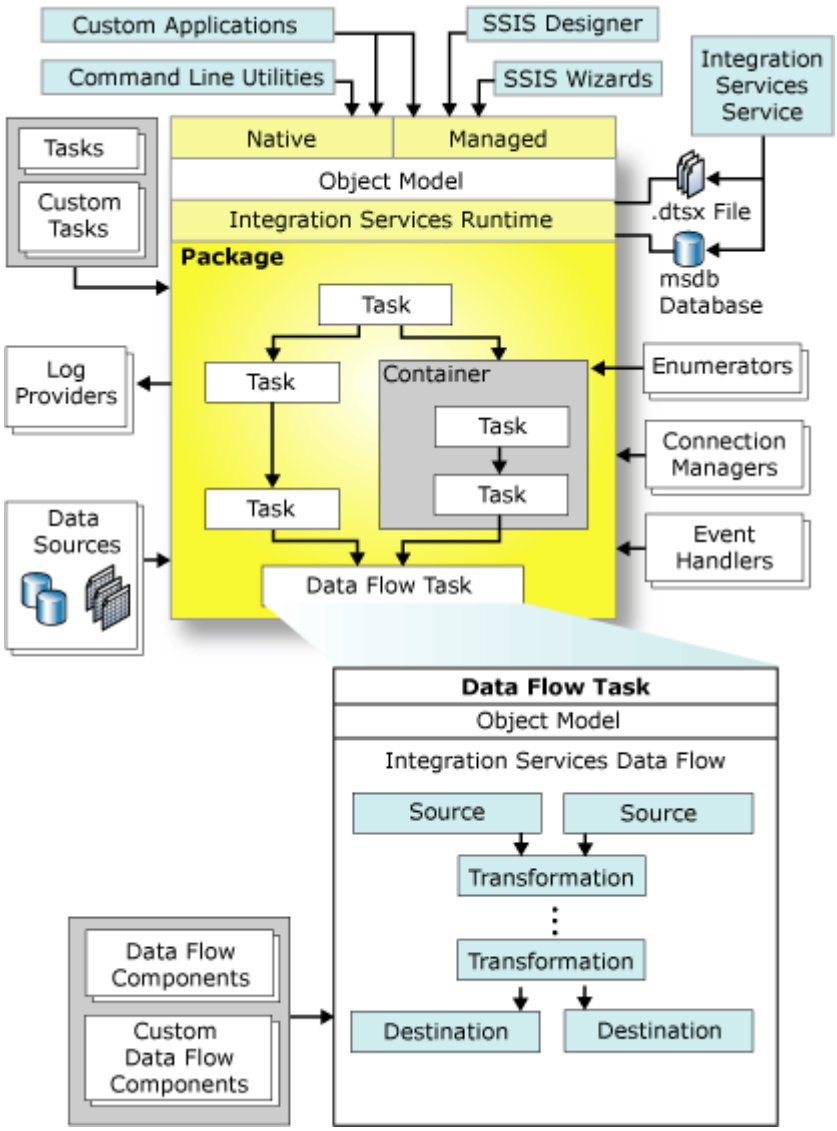
SSIS:n edeltäjä on ollut Microsoftin *SQL Server 7.0* -tietokantajärjestelmästä lähtien *Data Transformation Services* (DTS). DTS:n pääasiallinen tarkoitus oli tuoda data lähes mistä tahansa OLE DB -yhteensopivasta (*Object Linking and Embedding Database*) lähteestä haluttuun kohteeseen, mutta se pystyi myös ajamaan ohjelmia ja skriptejä. DTS:n käyttö yleistyi vuosituhannen vaihteessa varsinkin tietokannan ylläpitäjien keskuudessa. DTS:n käyttö oli kuitenkin monimutkaista, koska kehittäjät joutuivat tekemään monimutkaisia skriptejä saadakseen sen toimimaan halutulla tavalla. Useimmilla tietokannan ylläpitäjillä ei kuitenkaan ollut tarvittavaa skriptausosaamista tai aikaa niiden kirjoittamiseen. (Knight et al. 2008, s. 2.)

SSIS julkaistiin ensimmäisen kerran vuonna 2005 *SQL Server 2005* -tietokantajärjestelmän mukana. Siinä missä DTS oli aliarvostettu ja siihen oli käytetty vain vähän resursseja, SSIS oli Microsoftin tärkeysasteikolla aivan eri luokassa, ja muun muassa käytettävyyteen oli panostettu huomattavasti enemmän kuin DTS:ssä. Vuonna 2008 julkaistiin *SQL Server 2008*, jonka mukana tuli uusi versio

SSIS:stä. Uuteen versioon sisältyi muun muassa tehokkuus- ja skaalautuvuusparannuksia. (Knight et al. 2008, s. 2.) Tästä kaksi vuotta myöhemmin julkaistussa *SQL Server 2008 R2* -järjestelmässä SSIS:ään oli tehty hienovaraisempia muutoksia, kun taas maaliskuussa 2012 julkaistussa *SQL Server 2012* -järjestelmässä uudistukset ovat huomattavampia (Microsoft 2012).

SSIS:n suunnitteluympäristönä toimii normaalisti *Microsoft Visual dio* -kehitysympäristö höystettynä *Business Intelligence* -lisäkomponenteilla. Microsoft on nimennyt tämän kokonaisuuden *SQL Server Business Intelligence Development Studioksi* (BIDS). BIDS on graafinen suunnitteluympäristö, jossa voidaan luoda SSIS-, SSAS- ja SSRS-projekteja.

4.1 Arkkitehtuuri



Kuva 7. SSIS:n arkkitehtuuri (Microsoft 2008a).

Kuva 7 esittää SSIS:n arkkitehtuurin. SSIS-pakettien suunnittelun ja ylläpidon kannalta SSIS:n arkkitehtuurin keskeisimmät osat ovat SSIS-palvelu (*Integration Services Service*), SSIS-ajomoottori (*Integration Services Runtime*), tietovirtatehtävä (*Data Flow Task*) ja sen komponentit (*Data Flow Components*) sekä yhteyshallitsijat (*Connection Managers*) (Knight et al. 2008, s. 6). Nanda (2011, s. 19) on luetellut muuten samat arkkitehtuurin osat, mutta hän ei pidä yhteyshallitsijoita arkkitehtuurille keskeisinä, vaan sen sijaan mainitsee oliomallin (*Object Model*). Yhteyshallitsijat ovatkin sinänsä arkkitehtuurin kannalta pienemmässä osassa, koska ne ovat vastuus-

sa vain yhteyksistä eri tietolähteisiin, kun taas SSIS:n oliomalli mahdollistaa muun muassa pakettien ohjelmallisen käsittelyn.

Integration Services- eli SSIS-palvelu mahdollistaa muun muassa SSIS-pakettien ajamisen ja niiden säilytyksen hallitsemisen *SQL Server Management dio* -hallintatyökalun avulla paikallisella tai etäpalvelimella (Microsoft 2010). SSIS-palvelu on Windows-palvelu, joka asennetaan SQL Serverin SSIS-komponenttien asennuksen yhteydessä. Palvelua ei tarvita pakettien ajamiseen, mutta oletuksena ajossa olevien pakettien ajo keskeytetään, jos palvelu pysäytetään. Varsinaisen pakettien ajamisen hoitaa SSIS-ajomoottori ja sitä täydentävät ohjelmat. (Knight et al. 2008, s. 6–7.) Kuvassa 7 *Integration Services* -palvelu osoittaa .dtsx-tiedostoon ja MSDB-tietokantaan, jolla viitataan siihen, että palvelu voi hallita tiedostojärjestelmässä tai MSDB-tietokannassa olevia paketteja (Nanda 2011, s. 19).

Kuten edellä mainittiin, SSIS-ajomoottori ja sitä täydentävät ohjelmat hoitavat SSIS-pakettien ajamisen. SSIS-ajomoottori voidaan jakaa kahteen osaan, joista ensimmäinen on vastuussa työnkulusta ja toinen tietovirrasta. Työnkulun hoitava ajomoottori on vastuussa SSIS-paketeista, säilöistä (*container*), tehtävistä ja tapahtumien käsittelystä ajon aikana (tässä mainitut komponentit selitetään tarkemmin myöhemmin tässä luvussa). Lisäksi työnkulun ajomoottori huolehtii muun muassa lokin ylläpidosta ja yhteyksistä tietolähteisiin. Tietovirran ajomoottori puolestaan tarjoaa palveluita paketin tietovirralle. Se hakee datan tietolähteistä, antaa transformaatioiden tehdä muunnoksia datalle ja vie datan erilaisiin kohteisiin. (Nanda 2011, s. 20.) Eräs huomattava ero ajomoottorien välillä on se, että tietovirrassa data säilytetään keskusmuistissa siirryttäessä komponentista toiseen, kun taas työnkulussa data kirjoitetaan aina kiintolevylle tehtävien välillä (Knight et al. 2008, s. 16).

SSIS:n oliomalli tukee sekä natiivia (*native*) että hallittua (*managed*) ohjelmointirajapintaa (*Application Programming Interface*, API). Oliomallia voi hyödyntää räätälöityjä tehtäviä tai komponentteja kirjoittamalla käyttäen mitä tahansa CLR-yhteensopivaa (*Common Language Runtime*) ohjelmointikieltä. Kuten aiemmin mainittiin, oliomalli mahdollistaa myös SSIS-pakettien ohjelmallisen käsittelyn. Sekä pakettien luominen, muokkaaminen että ajaminen on mahdollista ohjelmallisesti.

(Nanda 2011, s. 20.) SSIS:n oliomallin mahdollistamaa pakettien luomista ohjelmallisesti käsitellään tarkemmin luvussa 5.

4.2 Integration Services -paketti

Paketit ovat keskeinen osa SSIS:ää, ja ne kuvaavat latauksen työnkulun ja toimintalogiikan. Yksinkertaistettuna paketti on kokoelma toisiinsa yhdistettyjä tehtäviä (*task*), jotka suoritetaan paketissa määrätyssä järjestyksessä. Paketit ovat itse asiassa XML-rakenteisia (*Extensible Markup Language*) tiedostoja, joiden pääte tiedostojärjestelmässä on .dtsx. SSIS-paketit voidaan säilyttää joko tiedostojärjestelmässä tai *SQL Server* -tietokantapalvelimen MSDB-tietokannassa. (Knight et al. 2008, s. 7.)

Pakettien sisältö määräytyy tietolähteiden ja tarvittavan ETL-prosessin mukaan. Useimmiten ne sisältävät ainakin yhteyksiä tietolähteisiin ja -kohteisiin, SQL-ajotehtäviä ja tietovirtatehtäviä. Tarpeen tullen paketissa on mahdollista käyttää edellä mainittujen lisäksi useita muita erilaisia tehtäviä sekä lokin kirjoittamista, pakettimuuttujia ja silmukkarakenteita. Paketin sisäinen toimintalogiikka jakautuu kahteen osaan samalla tavalla kuin SSIS-ajomoottori. Työnkulkua hallitaan BIDS:n ohjausvirtavälilehdeltä (*Control Flow*) ja tietovirtaa tietovirtavälilehdeltä.

Työnkulussa määritetään tehtävien ajojärjestys ajojärjestyksen rajoittimilla (*precedence constraint*). Ajojärjestyksen rajoittimet liittävät tehtävät toisiinsa, mutta niillä voi myös ehdollistaa tehtävien suoritusta. Esimerkiksi oletusehtona on onnistuminen, jolloin seuraavaa tehtävää ei suoriteta, jos edeltävä tehtävä epäonnistuu. Lisäksi on mahdollista asettaa ajojärjestyksen rajoitin valmistumiseen, jolloin seuraava tehtävä suoritetaan riippumatta edellisen tehtävän onnistumisesta, tai epäonnistumiseen, jolloin seuraava tehtävä suoritetaan vain edellisen epäonnistuessa. Monimutkaisempiakin ehtoja on mahdollista rakentaa hyödyntäen ehtolausekkeita ajojärjestyksen rajoittimessa. Oletuksena käytetään pelkästään ajojärjestyksen rajoittimen tilaa, mutta on myös mahdollista käyttää pelkästään ehtolauseketta, rajoittimen tilaa ja ehtolauseketta, tai rajoittimen tilaa tai ehtolauseketta.

SSIS:n muuttujat ovat tärkeä osa pakettia. Ne mahdollistavat tiedon välityksen paketin tehtävien ja transformaatioiden välillä. Muuttujien näkyvyyttä paketin eri osiin voidaan hallita säilöjen avulla. Muuttujien avulla paketin toimintaa voidaan ohjata dynaamisesti ajon aikana. Ilman muuttujia pakettiin kovakoodatut yhteysasetukset

ynnä muut jouduttaisiin määrittämään uudelleen aina, kun paketti siirretään ympäristöstä toiseen. Parhaana käytäntönä onkin asettaa paketti käyttämään muuttujia, joihin haetaan tiedot ympäristöstä niille asetuksille, jotka muuttuvat ympäristöstä toiseen. (Knight et al. 2008, s. 13.)

4.3 Tehtävät

Tehtävät ovat oleellinen osa SSIS-pakettia, sillä ilman tehtäviä paketti ei tee mitään. SSIS sisältää vakiona yli kolmekymmentä erilaista tehtävää, joista yksitoista on suunnattu *SQL Server* -ylläpitäjille (Knight et al. 2008, s. 9). Tehtäviä on mahdollista luoda lisää SSIS:n oliomallia hyödyntämällä. Tärkeimmät tehtävät ovat SQL-ajotehtävä ja tietovirtatehtävä. SQL-ajotehtävällä voi sen nimen mukaisesti ajaa SQL-komentoja tietokannassa. Periaatteessa kokonaisen ETL-ketjun voisi luoda pelkästään SQL-ajotehtävillä hyödyntämällä SQL-funktiota ja -proseduureja, mutta sen luominen olisi erittäin hankalaa ja sen voisi toteuttaa myös ilman *Integration Services* -pakettia. Useimmiten SQL-ajotehtäviä käytetään kohdetaulun siivoamiseen tai muokkaamiseen ennen latausta. SQL-ajotehtävässä voidaan hyödyntää myös SSIS:n muuttujia esimerkiksi SQL-funktion parametreina. Tietovirtatehtävä sisältää nimensä mukaisesti tietovirran.

Muita huomionarvoisia tehtäviä ovat FTP-tehtävä (*File Transfer Protocol*), paketin ajotehtävä ja skriptitehtävä. FTP-tehtävällä on mahdollista ottaa yhteys FTP-palvelimeen ja hakea sieltä tiedostoja prosessin käyttöön. Paketin ajotehtävällä voidaan ajaa muita paketteja. Skriptitehtävässä voidaan käyttää *C#*- tai *Visual Basic* -ohjelmointikieliä monimuotoisten skriptien tekoon. Koska skriptitehtävä mahdollistaa kaikkien .NET-kirjastojen hyödyntämisen, sen käyttö ei rajoitu vain pienimuotoiseen skriptaamiseen, vaan sitä voi käyttää täysin haluamallaan tavalla (Knight et al. 2008, s. 55).

4.4 Säilöt

Työnkulkuun sijoitetaan tehtävien lisäksi säilöjä (*container*), joita on neljää eri tyyppiä: *Task Host Container, Sequence Container, For Loop Container ja Foreach Loop Container*. Niiden tehtävänä on auttaa SSIS:ää ryhmittämään tehtäviä loogisiin kokonaisuuksiin. *Task Host Container* on oletussäilö, johon jokainen tehtävä laitetaan automaattisesti. Näin ollen sitä ei siis voi vetää työnkulkuun BIDS:n työkalulistasta.

SSIS tarjoaa SSIS-tehtäville pääsyn muuttujiin ja tapahtumienhallintaan *Task Host Container* -säilön kautta. *Sequence container* -säilö on tarkoitettu tehtävien ryhmittämiseen loogisiin kokonaisuuksiin. Se mahdollistaa muun muassa muuttujan näkyvyyden rajoittamisen vain säilön sisälle, usean tehtävän suorituksen varmistamisen ennen seuraavan tehtävän suoritusta ja transaktion luomisen toisiinsa liittyville tehtäville, muttei koko paketille. (Knight et al. 2008, s. 117–118.) Transaktiot ovat työkokonaisuuksia, joiden sisällä olevien tehtävien tulee kaikkien suoriutua onnistuneesti. Jos jokin tehtävä epäonnistuu, peruutetaan kaikki samaan transaktioon kuuluvien tehtävien tekemät muutokset.

For Loop Container eli *For*-silmukkasäilö mahdollistaa silmukan luomisen pakettiin samankaltaisesti kuin melkein missä tahansa ohjelmointikielessä. Siinä asetetaan jokin ehto ja käydään silmukkaa niin kauan läpi, kunnes ehto täyttyy. *Foreach Loop Container* eli *Foreach*-silmukkasäilö mahdollistaa oliokokoelman läpikäynnin. Tämä kokoelma voi olla esimerkiksi kokoelma tiedostoja, SSIS-muuttujan sisällä oleva kokoelma, tai lista tauluja tai rivejä *ADO-recordset*-muuttujassa. (Knight et al. 2008, s. 119, 122.)

4.5 Tietovirta

Tietovirtoja hallitaan BIDS:n tietovirtavälilehdeltä. Jokaista tietovirtatehtävää kohden on yksi tietovirta. Yksinkertaisimmillaan tietovirta voi sisältää vain yhden lähteen ja yhden kohteen. Tällöin lähtödatan rivit kopioidaan sellaisenaan kohteeseen, tosin lähteessä ja kohteessa voi vielä määrittää halutut sarakkeet. Monimutkaisemmat tietovirrat voivat sisältää useita lähteitä ja kohteita sekä useita erilaisia transformaatioita.

OLE DB Source eli OLE DB -lähde on useimmin SSIS-paketin tietovirrassa käytetty lähde, ja sillä voi ottaa yhteyden mihin tahansa OLE DB -yhteensopivaan tietolähteeseen. Sillä voi poimia tietokannan taulun sellaisenaan, tai ajaa sinne SQL-kyselyn suoraan tai muuttujan kautta. Kyselyssä voi käyttää myös parametreja. *Excel Source* eli *Excel*-lähde ottaa yhteyden nimensä mukaisesti *Excel*-taulukkoon. Se on kuitenkin ongelmallinen, sillä sille on vain 32-bittinen ajuri, joten paketin joutuu ajamaan 32-bittisessä tilassa. *Excel*-tiedoston lukeminen sillä voi olla muutenkin ongelmallista datatyyppien aiheuttamien ongelmien vuoksi. *Excel*-lähteen ongelmien vuoksi

onkin suositeltua käyttää *Flat File Source* eli *Flat File* -lähdettä, jos ne ovat vaihtoehtoisia. *Flat File* -lähteellä otetaan yhteys joko jollain välimerkillä sarakkeet toisistaan erottavaan tai kiinteän sarakeleveyden tiedostoon. Muut lähdetyypit ovat *Raw File Source*, *XML Source* ja *ADO.NET Source*. *Raw File* -lähde on tarkoitettu SSIS:n oman erityistyyppisen tiedoston lukuun, jota voidaan käyttää esimerkiksi tiedon tallentamiseen eri tietovirtojen välillä. XML-lähde lukee XML-tiedostoja ja ADO.NET-lähde lukee ADO.NET-yhteyden vaativia tietokantoja. (Knight et al. 2008, s. 132–139.)

SSIS tarjoaa edellä mainittuja lähteitä vastaavat kohteet ja lisäksi muutaman muun. Näistä huomionarvoisia ovat *SQL Server Destination* eli *SQL Server* -kohde ja *Recordset Destination* eli *Recordset*-kohde. *SQL Server* -kohde on optimoitu tiedon lataamiseksi *SQL Server* -palvelimeen. *SQL Server* -kohde tarjoaa lisänopeutta ja -ominaisuuksia verrattuna OLE DB -kohteeseen, mutta vaatii, että ajettava paketti on samalla palvelimella kuin *SQL Server* (Knight et al. 2008, s. 146). *Recordset*-kohde täyttää *ADO-recordset*-muuttujan, jota voi käyttää tietovirran ulkopuolella työnkulussa (Knight et al. 2008, s. 16). Sen avulla voidaan tuoda tietokannasta haettuja tietoja työnkulkuun esimerkiksi *Foreach*-silmukkasäilön läpikäytäväksi.

SSIS sisältää vakiona 29 erilaista transformaatiota, joilla tehdään tiedon tarkastukset ja muunnokset. Transformaatiot jaetaan kahteen pääkategoriaan: synkronisiin ja asynkronisiin. Synkronisissa transformaatioissa rivit viedään transformaatiossa muistipuskureihin ja samat puskurit tulevat ulos transformaation lopussa. Rivejä ei pysäytetä, ja synkroniset transformaatiot suoriutuvat nopeasti. Asynkroniset transformaatiot jakautuvat täysin blokkaaviin ja osittain blokkaaviin transformaatioihin. Osittain blokkaavat transformaatiot luovat uudet muistipuskurit transformaatiosta ulos meneviä rivejä varten. Täysin blokkaavat transformaatiot tekevät samoin, mutta pysäyttävät rivien etenemisen täysin. Ne päästävät rivit etenemään vasta, kun kaikki transformaatioon tulleet rivit on käsitelty. Täysin blokkaavat transformaatiot vievät muita transformaatioita enemmän muistia ja hidastavat tietovirtaa muita transformaatioita enemmän. (Knight et al. 2008, s. 146–147.)

Taulukossa 3 on esitelty kaikki SSIS:n mukana tulevat transformaatiot. Tämän työn käytännön osuudessa, jossa muun muassa SSIS:n transformaatioita käsitellään oh-

jelmallisesti, on toteutettu seuraavien transformaatioiden käsittely: *Aggregate*, *Conditional Split*, *Data Conversion*, *Derived Column*, *Lookup*, *Merge Join*, *Multicast*, *OLE DB Command*, *Script Component*, *Slowly Changing Dimension*, *Sort* ja *Union All*.

Taulukko 3. SSIS:n transformaatiot (Knight et al. 2008, s. 16-17, 150).

Transformaatio	**Kuvaus**
Aggregate (Aggregointi)	Koostaa rivit SQL:n GROUP BY -komennon kaltaisesti.
Audit (Auditointi)	Syöttää riveihin auditointi-informaatiota, kuten kuka ajoi paketin ja milloin.
Cache Transform (Välimuistimuunnos)	Lataa välimuistitiedoston levyltä tietovirtaan Lookup-transformaation kanssa käytettäväksi.
Character Map (Merkkikartta)	Tekee muunnoksia merkkityyppisille muuttujille. Esimerkiksi pienet kirjaimet isoiksi.
Conditional Split (Ehdollinen jakautuminen)	Jakaa rivit määritettyjen ehtojen mukaisesti.
Copy Column (Sarakkeen kopio)	Tekee kopion sarakkeesta.
Data Conversion (Datan tyyppimuunnos)	Muuntaa sarakkeen tyypin toiseksi.
Data Mining Query (Tiedonlouhintakysely)	Tekee tiedonlouhintakyselyn *SQL Server Analysis Services* -palveluun.
Derived Column (Johdettu sarake)	Luo uuden johdetun sarakkeen jonkin lausekkeen pohjalta.
Export Column (Sarakkeen vienti)	Vie sarakkeen tietovirrasta tiedostoon.
Fuzzy Grouping (Sumea ryhmittely)	Ryhmittelee tietoja sumean logiikan avulla. Esimerkiksi virheelliset tiedot on mahdollista ryhmittää oikeaan ryhmään. Vaatii toimiakseen *SQL Server Enterprise Edition* -version (tai paremman).
Fuzzy Lookup (Sumea	Hakee tietoja hakutaulusta sumean logiikan avulla ja yhdis-

haku)	tää ne tietovirtaan. Vaatii myös *SQL Server Enterprise Edition* -version (tai paremman).
Import Column (Sarakkeen tuonti)	Lukee sarakkeen tiedostosta ja tuo sen tietovirtaan.
Lookup (Haku)	Hakee tietoja hakutaulusta ja yhdistää ne tietovirtaan.
Merge (Yhdistäminen)	Yhdistää kaksi järjestettyä tietojoukkoa yhdeksi tietojoukoksi tietovirrassa.
Merge Join (Liitos)	Yhdistää kaksi järjestettyä tietojoukkoa yhdeksi tietojoukoksi käyttäen SQL:n join-operaatiota vastaavaa operaatiota.
Multicast (Monilähetys)	Kopioi tietojoukon useammalle polulle tietovirrassa.
OLE DB Command (OLE DB -komento)	Ajaa OLE DB -komennon jokaista riviä kohden OLE DB -kohteeseen.
Percentage Sampling (Prosentuaalinen otanta)	Tekee otannan tietovirrasta määritetyn prosentin perusteella.
Pivot	Kääntää jonkin rivitiedon sarakkeiksi. Esimerkiksi sarakkeessa *vuosi* olevat vuodet muutetaan sarakkeiksi 2000, 2001 ja niin edelleen.
Row Count (Rivilasku)	Tallentaa rivien määrän muuttujaan.
Row Sampling (Riviotanta)	Tekee otannan tietovirrasta määritetyn rivimäärän perusteella.
Script Component (Skriptikomponentti)	Suorittaa itse skriptatun transformaation.
Slowly Changing Dimension (Hitaasti muuttuva dimensio)	Transformaatio hitaasti muuttuville dimensioille.
Sort (Lajittelu)	Lajittelee tiedot määritettyjen sarakkeiden mukaan.
Term Extraction (Termin poiminta)	Etsii substantiiveja ja adjektiiveja tekstitiedoista. Toimii vain englanninkielisten tekstien kanssa.
Term Lookup (Termin	Vertaa tekstistä poimittuja termejä vertailutaulussa oleviin

haku)	arvoihin.
Union All (Kaikkien yhdistäminen)	Yhdistää monta tietojoukkoa yhdeksi tietojoukoksi.
Unpivot	Tekee käänteisen operaation kuin Pivot.

4.6 Yhteenveto SSIS:stä

Tässä luvussa esitetty SSIS tarjoaa työkalut latausprosessin tekoon *Microsoft SQL Server* -ympäristössä. SSIS:ssä latausprosessi suoritetaan niin sanotuissa paketeissa. SSIS:n oliomalli mahdollistaa myös räätälöityjen komponenttien tekemisen paketteihin sekä pakettien ohjelmallisen käsittelyn, joka on tämän diplomityön kannalta sen tärkein ominaisuus.

5 LATAUSPAKETIN GENEROINTI OHJELMOINTIRAJAPINNAN AVULLA

Tämän työn käytännön osuudessa tehtiin kuvassa 6 esitetyn rakenteen mukainen sovellus. Tässä luvussa esitellään sovelluksen viimeistä osuutta, joka mahdollistaa latauspaketin muodostamisen SSIS:n ohjelmointirajapinnan avulla käyttäen C#-ohjelmointikieltä.

SSIS:n oliomallin tarjoama ohjelmointirajapinta on tarkoitettu lähinnä kattamaan sellaiset skenaariot, joihin SSIS:n valmiit työkalut eivät riitä. Tällöin SSIS:n toimintaa voidaan laajentaa käyttämällä omaa ohjelmakoodia skriptaamalla tai luomalla omia komponentteja. SSIS:n tarjoamat valmiit komponentit on luotu käyttäen samaa kaikille SSIS-kehittäjille tarjolla olevaa ohjelmointirajapintaa. (Knight et al. 2008, s. 661; Nanda 2011, s. 482.) SSIS-pakettien luominen ohjelmallisesti hyödyntää samaa ohjelmointirajapintaa. Periaatteessa pakettien luominen olisi mahdollista pelkkää XML-koodia manipuloimalla, mutta se olisi tarpeettoman hankalaa ottaen huomioon, että Microsoft tarjoaa siihen valmiin ohjelmointirajapinnan.

Kuten aiemmin luvussa 4 mainittiin, SSIS:n ajomoottori jakautuu kahteen osaan, joista ensimmäinen on työnkululle ja toinen tietovirralle. Molemmat ajomoottorit on kirjoitettu natiivilla ohjelmakoodilla, koska se tarjoaa paremman suorituskyvyn. Ajomoottorien hyödyntäminen ohjelmallisesti tapahtuu kuitenkin SSIS:n hallitun oliomallin kautta, mikä helpottaa laajentamista. Jotkin SSIS:n tehtävät ja komponentit onkin kirjoitettu hallitulla ohjelmakoodilla. Työnkulun ajomoottoria ohjelmointirajapinnassa edustaa *Microsoft.SqlServer.Dts.Runtime*-nimiavaruus, joka sisältää luokat ja rajapintaluokat pakettien, tehtävien ja muiden työnkulun komponenttien luomista varten. Tietovirran ajomoottoria edustaa *Microsoft.SqlServer.Dts.Pipeline*-nimiavaruus, jonka luokilla voidaan luoda tietovirran komponentteja. (Nanda 2011, s. 483.)

5.1 SSIS-komponenttien ja -yhteyksien luominen

Tämän työn kannalta oleellisimmat SSIS-komponentit ovat ohjausvirran osalta *sequence container*, *execute SQL task* ja *data flow task*. Tietovirran osalta oleellisimmat komponentit ovat lähteistä *flat file source* ja *ole db source*, transformaatioista

aggregate, *conditional split*, *data conversion*, *derived column*, *lookup*, *merge join*, *multicast*, *ole db command*, *slowly changing dimension*, *sort* ja *union all*, ja kohteista *ole db destination*. Komponenttien lisäksi tarvitaan *ole db connection*- ja *flat file connection* -yhteydet muun muassa lähteitä, kohteita ja *lookup*-komponenttia varten. Edellä mainituilla komponenteilla pystytään täyttämään useimpien ETL-prosessien tarpeet. Edellä olevasta listasta puuttuu *script component*, joka on tärkeä komponentti siinä mielessä, että sen avulla voidaan toteuttaa sellaisiakin transformaatioita, joita muilla komponenteilla ei voi. Sitä ei kuitenkaan pysty SSIS:n 2008-version paketissa luomaan ohjelmallisesti, koska skriptin ja komponentin metatiedon välinen tiivis yhteys hajoaa helposti, jos komponentin tekee ohjelmallisesti (Guea 2008). Jos *script component* puuttuisi SSIS:n käytettävistä komponenteista, joutuisivat ETL-kehittäjät joko tekemään mittavia kiertoratkaisuja tai ohjelmoimaan toistuvasti uusia komponentteja, jotka eivät välttämättä eroaisi toisistaan merkittävästi. *Script component* -komponentin puuttuminen ei kuitenkaan ole ongelma tämän työn kannalta, sillä:

- *script component* on luonteeltaan sellainen, että sitä tarvitsee yleensä vain erikoistapauksissa (Knight et al. 2008, s. 17)
- sen automaattinen generointi olisi lähes mahdotonta ottaen huomioon, että sen sisältö riippuu niin monesta asiasta
- se vaatii kehittäjän kirjoittamaa ohjelmakoodia
- sen voi korvata luomalla oman räätälöidyn komponentin.

Lisäksi *script component* -komponentin tai korvaavan rakenteen automaattinen generointi jää tämän diplomityön rajauksen ulkopuolelle, sillä tässä työssä ei pyritäkään latauspaketin kehittämisen täydelliseen automatisointiin.

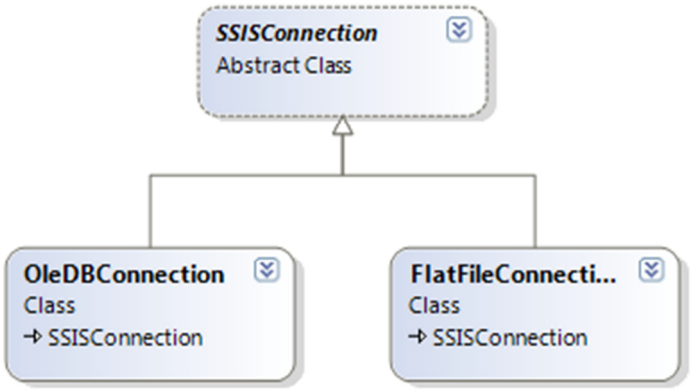
5.1.1 Yhteyksien konfigurointi ja rakenne

SSIS:n ohjelmointirajapinnassa yhteydet toimivat *ConnectionManager*-luokan kautta. *ConnectionManager*-luokka edustaa fyysisiä yhteyksiä ulkoisiin tietolähteisiin. Luokka erottaa yhteyksien toteutuksen yksityiskohdat ajomoottorista, jolloin ajomoottori voi olla vuorovaikutuksessa jokaisen yhteyden kanssa johdonmukaisella ja ennustettavalla tavalla. Toisin kuin muissa ohjelmointitavoissa, joissa yhteysluokat yhdistetään erilaisilla avausmetodeilla tietolähteisiin, SSIS:n ajomoottori hallinnoi paketin kaikkia yhteyksiä sen ajon aikana. Yhteydet luodaan käyttäen *Connections*-

luokan *Add*-metodia. *Connections*-luokka sisältää kaikki pakettiin lisätyt yhteydet. (Microsoft 2008b.)

ConnectionManager-luokan konfiguroimiseen tarvitaan yleensä vain nimi ja yhteyden tiedot (Microsoft 2008b). Yhteyden tietoina toimii *string*-tyyppinen muuttuja, joka sisältää *flat file connectionin* tapauksessa tiedostopolun ja -nimen, ja *ole db connectionin* tapauksessa palvelimen osoitteen, siellä olevan tietokannan, *ole db* -yhteystyypin ja tarvittaessa yhteyden tunnuksen ja salasanan. *Flat file connection* eroaa *ole db connectionista* sillä tavalla, että siinä pitää konfiguroida myös tiedoston rakenne. Tiedostosta tulee määrittää muun muassa sen jokaisen sarakkeen sisältämä tietotyyppi, tietotyypin pituus niiltä tietotyypeiltä, joilla se on oleellista, ja millä tavalla tiedon sarakkeet erotetaan toisistaan.

Kuva 8 esittää yhteyksien luokkakaavion. *SSISConnection*-abstraktiluokka sisältää muun muassa *ConnectionManager*-luokan olion, jonka varsinaiset yhteysluokat perivät. *SSISConnection*-abstraktiluokka mahdollistaa siten kaikkien yhteysolioiden tallentamisen *SSISConnection*-tyyppiseen listaan, jolloin yhteyksien käsittely on helpompaa. Tällä hetkellä yhteyksistä ei ole toteutettu muita kuin OLE DB -yhteys ja tiedostoyhteys, mutta tulevaisuudessa myös muut yhteystyypit voidaan toteuttaa.



Kuva 8. Yhteyksien luokkakaavio.

5.1.2 Ohjausvirran komponenttien konfigurointi ja rakenne

Ohjausvirran komponentit, joihin kuuluvat tehtävät ja säilöt, voidaan lisätä viiteen erilaiseen ajomoottorin olioon. Näitä ovat *Package*, *Sequence*, *ForLoop*, *ForEachLoop* ja *DtsEventHandler*. Vastaavat säilöt SSIS-paketissa ovat *sequence container*, *for loop container* ja *foreach loop container*. Myös *Package* eli paketti ja

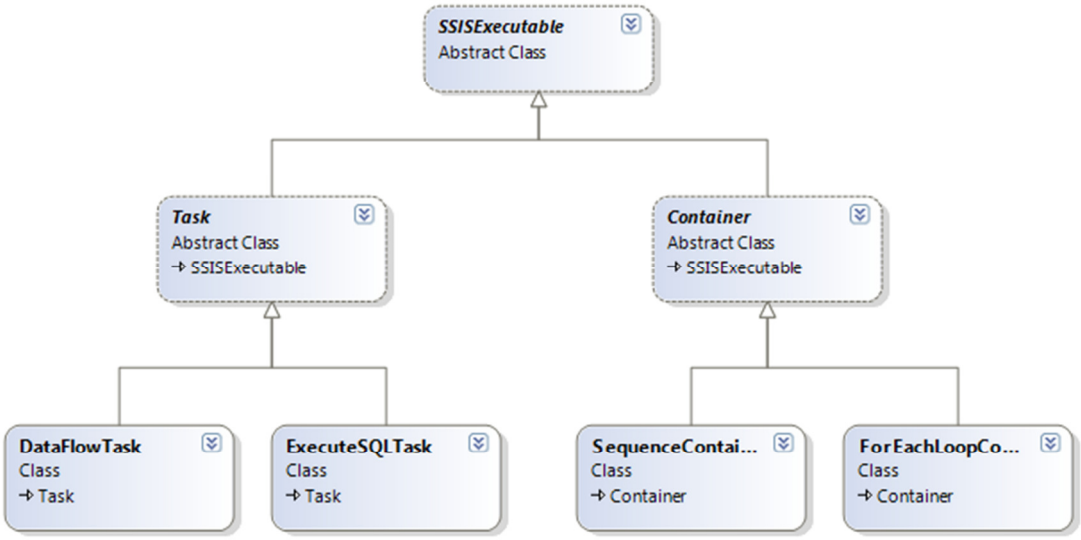
DtsEventHandler eli tapahtumien käsittelijä lasketaan säilöiksi, ja ne kaikki perivät *Executables*-propertyn. Propertyt ovat C#:ssa luokan jäseniä, jotka tarjoavat joustavan mekanismin *private*-tyyppisten kenttien käsittelyyn (Microsoft 2010). *Executables*-property sisältää säilön ajettavat oliot, jotka periytyvät *Executable*-luokasta. Ajomoottori käyttää *Executable*-luokan *Execute*- ja *Validate*-metodeja luokan prosessointiin. Komponentin lisäys säilöön tapahtuu *Executables*-propertyn *Add*-metodilla, jolle annetaan parametrina komponentin tyypin kertova *string*-tyyppinen muuttuja. (Microsoft 2008b.)

Sequence container -säilön konfiguroimiseksi ei sen luomisen jälkeen yleensä tarvitse tehdä muuta, kuin antaa sille nimi. Konfiguroitavissa olevia valintoja on nimen lisäksi useita, mutta tässä työssä niitä ei käytetä, koska monia niistä ei tarvita kovinkaan usein eikä niiden automaattinen generointi yksinkertaisen mallin pohjalta olisi mahdollista. Sama pätee *data flow* -tehtävään. *Execute SQL* -tehtävä sen sijaan vaatii konfiguroimista: se vaatii vähintäänkin tiedon tietokantayhteydestä, SQL-komennon lähteen määrittämisen ja itse SQL-komennon. Tietokantayhteytenä voi toimia mikä tahansa OLE DB- tai ADO.NET -tyyppinen (*ActiveX Data Objects for .NET*) yhteys. SQL-komennon lähteen määritys vaikuttaa siihen, miten SQL-komento määritetään. Vaihtoehtoina ovat suora SQL-komento, jolloin komento annetaan tekstinä, SSIS-muuttuja, jolloin komento on tallennettuna muuttujaan, tai tiedosto, jolloin komento on tallennettu tiedostoon.

Tehtävät muutetaan vielä luomisen jälkeen tyyppimuunnoksella *Executable*-tyypin oliosta *TaskHost*-tyypin olioon. *TaskHost*-luokka tarjoaa tehtävälle uusia propertyja ja metodeja, jotka erottavat ne selkeämmin muista *Executable*-tyypin olioista. *TaskHost*-luokka mahdollistaa myös itse tehtävään käsiksi pääsemisen *InnerObject*-propertyn kautta. Tämä on tarpeellista joidenkin tehtävien, kuten tietovirtatehtävän, kanssa ja saattaa tarjota suorituskykyetua, mutta monet tehtävät voidaan pitää *TaskHost*-oliona, jolloin voidaan kirjoittaa geneerisempää koodia. (Microsoft 2008b.)

Kuva 9 esittää ohjausvirran komponenttien luokkakaaviota. Ohjausvirran komponentit periytyvät *SSISExecutable*-abstraktiluokasta *Task*- tai *Container*-abstraktiluokkien kautta. Rakenne poikkeaa hieman SSIS:n ohjelmointirajapinnan periytymisjärjestyksestä, jossa ohjausvirran komponentit periytyvät suoraan *Executable*-luokasta. Tällä

on haettu selkeyttä, sillä tehtävät eroavat säilöistä siten, että niihin ei voi sijoittaa toisia *Executable*-luokan olioita. Samankaltaisesti kuin yhteysluokkien tapauksessa, tämä luokkarakenne yksinkertaistaa luokista muodostettujen olioiden tallentamista eri ohjausvirran komponentteja sisältävään listaan. Tehtävistä on tällä hetkellä toteutettu tietovirtatehtävä ja SQL-ajotehtävä. Säilöistä puolestaan on toteutettu *Sequence Container* -säilö ja *ForEachLoop*-säilö.



Kuva 9. Ohjausvirran komponenttien luokkakaavio.

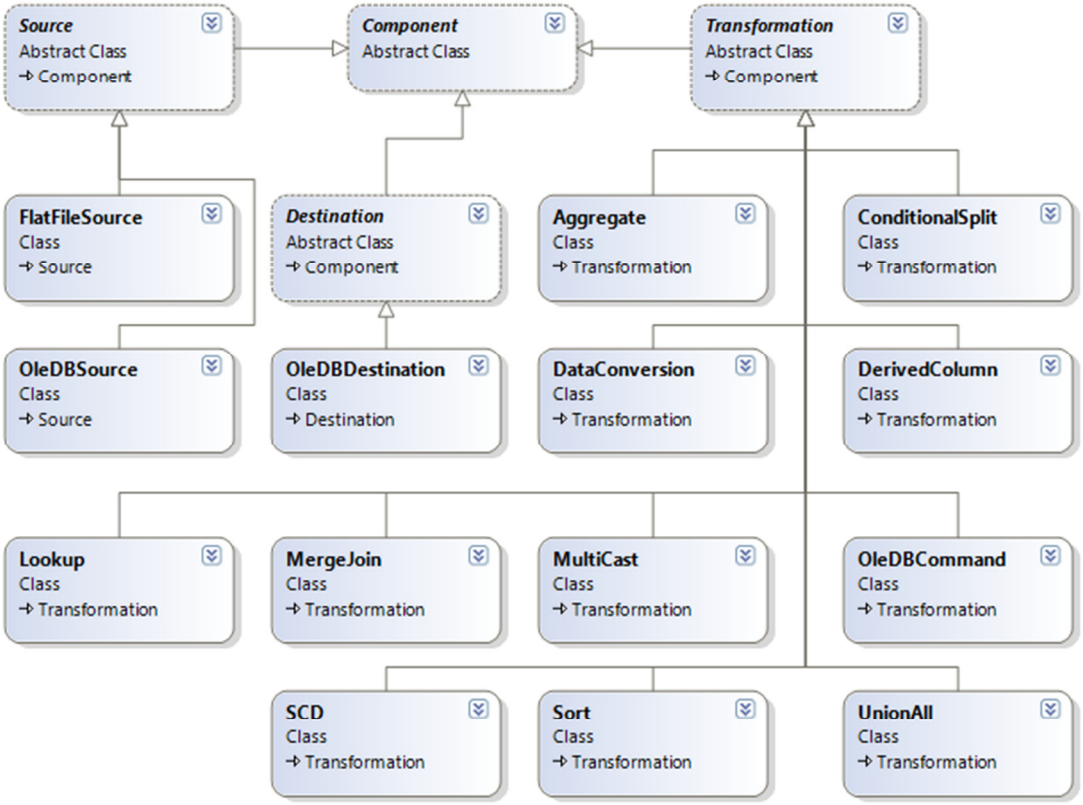
5.1.3 Tietovirran komponenttien konfigurointi ja rakenne

Tietovirran komponentit lisätään tietovirtaan käyttäen tietovirtatehtävän *ComponentMetaDataCollection*-propertyn *New*-metodia, joka palauttaa komponentin *IDTSComponentMetaData100*-rajapintaluokan. Tässä vaiheessa *IDTSComponentMetaData100*-rajapintaluokka ei kuitenkaan määritä komponentin tyyppiä, vaan se määritetään antamalla komponentin luokkatunnus *ComponentClassID*-propertylle. Tätä propertya käytetään komponentin luomiseen ajoaikana. (Microsoft 2008b.)

Suunnitteluajan ilmentymän komponentista saa kutsumalla *IDTSComponentMetaData100*-rajapintaluokan *Instantiate*-metodia, joka luo ilmentymän *ComponentClassID*-propertyn perusteella. Osa komponentin metatiedosta tuleekin muokata suoraan, mutta suunnitteluajan ilmentymän metodeja tulisi käyttää komponentin metatiedon suoran muokkaamisen sijaan, sillä muokkaamalla metatietoa suoraan ohitetaan komponentin mahdollisuus tarkistaa muutosten oikeellisuus. Suunnitteluajan ilmentymän luomisen jälkeen tulee kutsua ilmentymän *ProvideComponentProperties*-metodia,

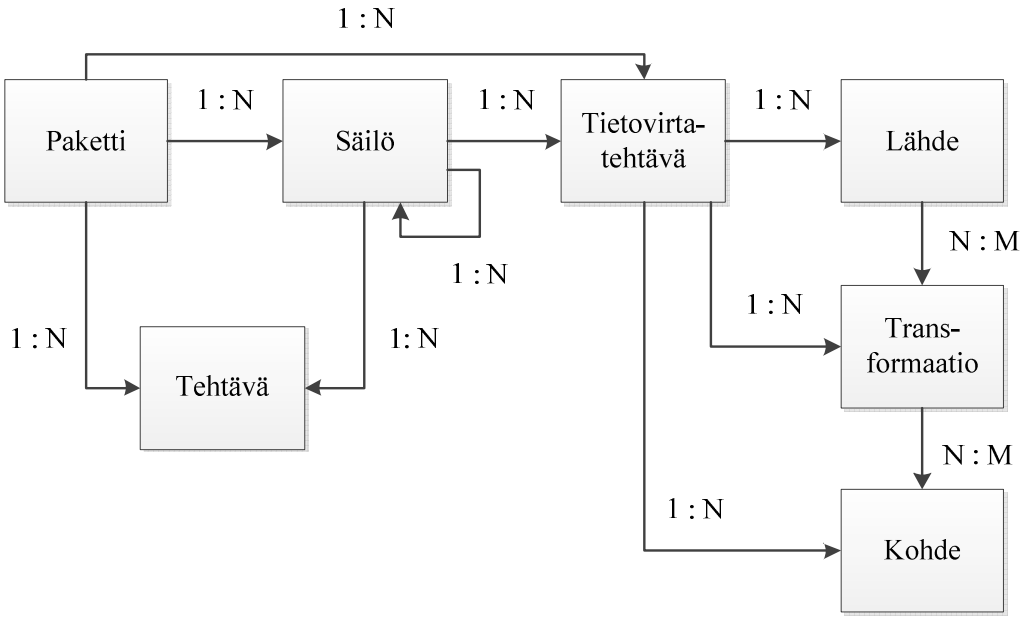
joka toimii kuin konstruktori, koska se alustaa komponentin. *CustomPropertyCollection*-property sisältää komponentin ominaisuudet, ja niitä voi asettaa kutsumalla *SetComponentProperty*-metodia. Seuraavaksi joillekin komponenteille, kuten lähteille, tulee määrittää yhteys ulkoiseen tietolähteeseen. Se onnistuu määrittämällä komponentin *ConnectionManager*-propertyyn oikea yhteys. Komponentin alustaminen päätetään hakemalla ulkoisista lähteistä komponentin sarakkeille metatieto. Tämä ei ole tarpeellista useille transformaatiolle ja kohteille, sillä ne saavat sarakkeiden metatiedon edeltävältä komponentilta. (Microsoft 2008b.)

Kuva 10 esittää tietovirran komponenttien luokkakaavion. Tietovirran komponentit periytyvät *Component*-abstraktiluokasta *Source*-, *Transformation*- ja *Destination*-abstraktiluokkien kautta. Tämäkin rakenne eroaa hieman SSIS:n ohjelmointirajapinnan rakenteesta. SSIS:n ohjelmointirajapinnassa ei ole *Source*-, *Transformation*- ja *Destination*-abstraktiluokkia vastaavia luokkia, mutta tässäkin tapauksessa välissä olevat luokat selkeyttävät rakennetta. Myös tietovirran komponenttien osalta abstrakteista luokista periytyminen helpottaa olioiden käsittelyä. Kuvasta 10 näkee tällä hetkellä toteutettuna olevat komponentit.



Kuva 10. Tietovirran komponenttien luokkakaavio.

Eri komponenttien suhteet toisiinsa on esitetty kuvassa 11. Ne vastaavat SSIS-paketin rakennetta. Kuvasta on jätetty selkeyden vuoksi pois yhteydet, koska ne käytännössä liittyvät jokaiseen paketin komponenttiin. Yhteen pakettiin voi laittaa monta säilöä, tehtävää ja tietovirtatehtävää. Säilöihin voi laittaa toisia säilöjä ja tehtäviä, mukaan lukien tietovirtatehtävän. Tietovirtatehtävä on erotettu kaaviossa muista tehtävistä siksi, että vain tietovirtatehtävään voi laittaa lähteitä, transformaatioita ja kohteita. Periaatteessa tietovirtatehtävässä voi olla esimerkiksi pelkästään transformaatioita, mutta käytännössä se yleensä sisältää yhden tai useamman lähteen, joitakin transformaatioita ja yhden kohteen.



Kuva 11. Komponenttien väliset suhteet.

5.2 Yhteenveto latausprosessin generoimisesta

Tässä luvussa esiteltiin latauspakettien generoimisen kannalta oleelliset SSIS-ohjelmointirajapinnan komponentit. Kaikkia SSIS:n komponentteja varten ei ohjelmoitu toiminnallisuutta, koska joitakin komponentteja käytetään vain harvoin, joten niiden toiminnallisuuden sisällyttäminen tähän työhön olisi lisännyt laajuutta huomattavasti verrattuna saavutettuun hyötyyn.

6 MÄÄRITYSDOKUMENTTIPOHJAN SUUNNITTELU

Tämän työn käytännön osuudessa tehtiin kuvassa 6 esitetyn rakenteen mukainen sovellus. Tässä luvussa käsitellään kuvan ensimmäistä osaa, käsittelysääntöjä. Hovi et al. (2009, s. 55) mainitsevat kuusi tietoa, joita käsittelysäännöissä tulee ainakin kuvata: lähtöjärjestelmä, lähtötaulu tai tiedosto, lähtösarake, käsittelysääntö, tietovarastotaulu ja tietovarastosarake. Käsittelysäännöt tallennetaan tässä luvussa esitettyyn määritysdokumenttiin.

Määritysdokumentissa yksi tärkeä vaatimus on se, että ei-teknisen henkilön on helppo ja nopea täyttää se. Tämä vaatimus juontaa siitä, että tietovarastointiprojektin alussa vaaditaan nopeaa dokumentaatiota tietojärjestelmistä (Vassiliadis et al. 2002, s. 16). Tämän takia esimerkiksi Akkaouin et al. (2009) ja Muñozin et al. (2008) mallit eivät sovellu määritysdokumenteiksi, koska ne ottavat kantaa toteutuskysymyksiin. Vassiliadisin et al. (2002) malli on parempi siinä mielessä, että se ei ota kantaa toteutukseen samalla tarkkuudella, mutta sen käyttö vaatisi opettelua eikä se siten sovellu erityisen hyvin ei-tekniselle henkilölle. Näiden lisäksi kaikki edellä mainitut lähestymistavat vaatisivat erillisten suunnittelutyökalujen tekemisen tai ostamisen. Näistä syistä johtuen määrittelydokumenttien pohjaksi valittiin *Excel*-dokumenttipohja. *Excel*-taulukoiden tärkeimpinä etuina ovat selkeys tietyin varauksin ja tunnettuus.

6.1 Vaatimukset

Määritysdokumentin vaatimukset ovat yksinkertaisuus, selkeys, helppokäyttöisyys, laajennettavuus ja mahtuminen pieniresoluutioiselle näytölle. Yksinkertaisuus, selkeys ja helppokäyttöisyys liittyvät toisiinsa: helppokäyttöinen käyttöliittymä on yksinkertainen ja selkeä. Laajennettavuuden vaatimus täytetään muun muassa sillä, ettei dokumentissa olevia käsittelysääntöjä ole kiinteästi asetettu dokumenttiin, jolloin jos uusia ominaisuuksia tarvitsee määrittää, voidaan olemassa olevia osia siirtää. *Excel*-dokumentti täyttää nämä vaatimukset, kunhan se suunnitellaan hyvin.

Pieniresoluutioiselle näytölle mahtumisen vaatimus tulee siitä, että dokumenttipohjaa on voitava täyttää kannettavalla tietokoneella esimerkiksi asiakaspalaverin aikana. Tarkoituksena on, että koko dokumentti mahtuisi leveydeltään näytölle niin, ettei sitä

tarvitsisi vierittää leveyssuunnassa. Käytettävyysanalyyseissä on havaittu, että käyttäjät eivät pidä web-sivujen vierittämisestä leveyssuunnassa, mutta korkeussuunnassa vierittäminen ei ärsytä samalla tavalla (Nielsen 2002). Tämän voi olettaa pätevän myös tällaisen dokumentin käytettävyyteen.

6.2 Sisältö

Koska nopea täyttäminen on iso vaatimus dokumentissa, tehtiin tässä työssä kaksi eri dokumenttipohjaa. Ensimmäinen on pikaista käsittelysääntöjen dokumentointia varten ja toinen tarkennuksia varten. Molempien dokumenttipohjien sommittelu on sama ja myös sisältö näyttää ulkoisin puolin samalta. Niiden ero johtuu *Excelin* tarjoamasta tietojen kelpoisuuden tarkistamisesta ja näkyy tietotyyppien määrittämisessä. *Excel* mahdollistaa soluun syötettävien tietojen kelpoisuuden tarkistamisen toisissa soluissa määritettyjen arvojen pohjalta. Solun yhteyteen tulee alasvetovalikko, josta sopiva arvo voidaan valita. Koska *SQL Server* tukee useita erilaisia tietotyyppejä ja monet niistä ovat samankaltaisia, rajataan ensimmäisen dokumenttipohjan tietotyypit neljään päätyyppiin: merkkijonoon, kokonaislukuun, desimaalilukuun ja päivämäärään. Näin solun alasvetovalikko ei täyty kaikista mahdollisista tietotyypeistä ja sopivan tietotyypin valikoiminen on nopeampaa. Koska sovellus hoitaa kohdetaulujen automaattisen generoinnin määrityksen pohjalta, ja koska tietotyyppien tulee olla SSIS-paketin metatiedoissa, laajennettiin toisen dokumenttipohjan tietotyyppien määrä käsittämään kaikki *SQL Server* -tietokannan tukemat tietotyypit. Koska dokumenttipohjat ovat ulkoisesti samannäköisiä, ensimmäisestä dokumenttipohjasta voi helposti kopioida kaikki tiedot toiseen dokumenttipohjaan, jossa tietotyyppejä voi tarkentaa.

Kuten aiemmin mainittiin, tulee käsittelysäännöistä dokumentoida ainakin lähtöjärjestelmä, lähtötaulu tai tiedosto, lähtösarake, käsittelysääntö, tietovarastotaulu ja tietovarastosarake (Hovi et al. 2009, s. 55). Koska tarkoituksena on generoida latauspaketin runko käsittelysääntöjen pohjalta, tarvitaan myös lisää tietoa. Koska kaikki tiedot eivät ole välttämättä saatavilla alun dokumentoinnissa tai niiden täyttäminen hidastaisi turhaan dokumentin täyttämistä, ne voidaan täyttää myös myöhemmin, mutta kuitenkin ennen kuin latauspaketin runko generoidaan. Kun lähtöjärjestelmä on tietokanta, tarvitaan tietokantapalvelimen yhteystiedot ja tietokannan nimi. Jos

lataus tehdään tiedostosta, tiedoston määrittämiseksi tarvitaan tiedoston sijainti, tiedoston tyyppi, joka kertoo millä tavalla tiedostossa olevat sarakkeet erotetaan toisistaan, sarakkeiden erotinmerkki ja rivien erotinmerkki. Kiinteäsarakkeisten tiedostojen tapauksessa sarakkeiden erotinmerkillä ei ole merkitystä.

Kuva 12 havainnollistaa määritysdokumentin rakenteen. *Source type* kertoo lähtötietojen tyypin. Tällä hetkellä tuettuna ovat tiedostot tai tietokannan taulut. *Source server*- ja *Source database*-kenttiä (lähdepalvelin, lähdetietokanta) käytetään lähtötietokannan tietojen määrittämiseen. Lähdetiedoston määritys hoidetaan kentillä *Source file* (lähdetiedosto), *File type* (tiedoston tyyppi), *Column Delimiter* (sarakkeiden erotinmerkki) ja *Row Delimiter* (rivien erotinmerkki). Vaikka tällä hetkellä lähtötaulun ja -tiedoston määritykset näkyvät samaan aikaan taulukossa, niistä vain toiset ovat käytössä kerrallaan riippuen *Source type* -kentästä. Kohdetietokannan ja -taulun määritys tehdään kentillä *Destination server* (kohdepalvelin), *Destination database* (kohdetietokanta), *Destination table name* (kohdetaulun nimi) ja *Type* (tyyppi). Tuetut taulujen tyypit ovat *staging*-, dimensio- ja faktataulutyypit. *Column name* (sarakkeen nimi), *Column type* (sarakkeen tyyppi), *Length* (sarakkeen pituus), *Special options* (erityisvalinnat) ja *Comments* (kommentit) on tarkoitettu lähtö- ja kohdetaulun tai -tiedoston konfigurointiin.

Source type	Database table					
Source server	Palvelin					
Source database	Tietokanta	**Source**				
		Column name	**Column type**	**Length**	**Special options**	**Comments**
Source file		He_Vuosi	int		1I1	Dimension.Henkilo
File type		He_Henkilotunnus	varchar	11	1I2	Dimension.Henkilo
Column Delimiter		He_HenkiloId	int			Dimension.Henkilo
Row Delimiter		Pa_Palkka	float			Staging.Palkkatiedot
		Pa_Vuosi	int		1R1	Staging.Palkkatiedot
Destination server	Palvelin	Pa_Henkilotunnus	varchar	11	1R2	Staging.Palkkatiedot
Destination database	Tietokanta					
Destination table name	Facts.Palkka	**Destination**				
Type	Fact	**Column name**	**Column type**	**Length**	**Special options**	**Comments**
		Id	int		Identity	
		HenkiloId	int		not null	He_HenkiloId
		Palkka	float			Pa_Palkka

Kuva 12. Esimerkki latauksen käsittelysääntöjen määrityksestä määritysdokumentilla.

Sarakkeen pituus määrittää kiinteäsarakkeisen tiedoston sarakkeen leveyden ja esimerkiksi tietotyypin merkkien määrän, kun kyseessä on merkkijono. Tässä vaiheessa erityisvalintoja käytetään lähtösarakkeissa määrittämään, miten eri tauluista tulevat

tiedot yhdistetään toisiinsa. Erityisvalinnoissa toisiinsa liitettävät sarakkeet määritetään seuraavanlaisella syntaksilla: <liitoksen numero><liitoksen tyyppi><sarakkeen vastaavuus>. Liitoksen numero kertoo, mitkä sarakkeet kuuluvat samaan liitokseen. Liitoksen tyyppi on joko I, L, F tai R. I tulee *Inner Join* -liitoksesta, L *Left join* -liitoksesta, F *Full join* -liitoksesta ja R määrittää sarakkeen tulevan liitoksen oikealta puolelta. Sarakkeen vastaavuus määrää, mitkä sarakkeet vastaavat toisiaan. Tätä on havainnollistettu taulukossa 4: siinä määritetään, että *Dimension.Henkilo*-taulusta tulevat *He_Vuosi*- ja *He_Henkilotunnus*-sarakkeet yhdistetään *inner join* -liitoksella *Staging.Palkkatiedot*-taulun *Pa_Vuosi*- ja *Pa_Henkilotunnus*-sarakkeisiin. Koska lähtösarakkeet voivat tulla useammasta taulusta tietokannan ollessa kyseessä, käytetään kommenttikenttää kertomaan lähtötaulun nimi. Jos yhtä saraketta on tarkoitus käyttää useammassa liitoksessa, eri liitokset erotetaan toisistaan pilkulla. Jos useammasta taulusta liitetään vastaavat sarakkeet toisiinsa, ne voidaan numeroida yhdellä liitosnumerolla ja ensimmäistä vasemmalta puolelta tulevaa liitosta lukuun ottamatta ne merkitään oikealta puolelta tuleviksi. Esimerkissä kyse on surrogaattiavaimen hakemisesta faktatauluun.

Taulukko 4. Esimerkki lähtösarakkeiden määrityksestä.

Nimi	**Tyyppi**	**Pituus**	**Erityisvalinnat**	**Kommentit**
He_Vuosi	int		1I1	Dimension.Henkilo
He_Henkilotunnus	varchar	11	1I2	Dimension.Henkilo
He_HenkiloId	int			Dimension.Henkilo
Pa_Palkka	float			Staging.Palkkatiedot
Pa_Vuosi	int		1R1	Staging.Palkkatiedot
Pa_Henkilotunnus	varchar	11	1R2	Staging.Palkkatiedot

Tietovarastotaulu määritetään tietovaraston yhteystietojen yhteydessä. Tietovarastosta tarvitsee tietää sen palvelimen yhteystiedot ja siellä olevan tietokannan nimi. Lisäksi tulee määrittää tietovarastotaulun nimi sekä se, onko se *staging*-, dimensio- vai faktataulu. Tarkoituksena on, että kaikki *staging*-lataukset määritetään yhteen *Excel*-tiedostoon eri välilehdille, kuten myös dimensio- sekä faktalataukset omiin tiedostoihinsa. Tämä sen takia, että eri lataustyyppien tulee seurata loogisesti toisiaan jär-

jestyksessä *staging*, dimensio ja fakta. Sovellus ei kuitenkaan estä määrittämästä kaikkia samassa *Excel*-tiedostossa, mutta SSIS:n parhaat käytännöt suosittelevat isäntäpaketin käyttöä, joka ajaa lapsipaketit (*staging*-, dimensio- ja faktapaketit). (Knight et al. 2008, s. 408).

Tietovarastotaulun sarakkeet määritetään samankaltaisesti kuin lähtötaulut (taulukko 5). Poikkeuksena ovat kuitenkin nimi-, erityisvalinta- sekä kommenttikentät. Siinä missä lähtötauluja hakiessa nimikenttä määritti taulusta haettavan sarakkeen nimen, tietovarastotaulua luotaessa se määrittää luotavien sarakkeiden nimet. Tietovarastosarakkeissa erityisvalinnat määrittävät taulun luontiin vaikuttavat valinnat, kuten identiteettisarakkeen määrittämisen. Kommenttikenttä määrittää, mistä tieto sarakkeeseen tulee. Lisäksi siinä voidaan määrittää *derived column* -transformaation tukema lauseke, jolla sarakkeista tulevia tietoja voidaan muokata halutulla tavalla. *Derived column* -transformaatio mahdollistaa melko monimutkaisiakin operaatioita, mutta ei silläkään voi tehdä kaikkea. Monimutkaisten lausekkeiden tapauksessa niiden editointi lienee parempi jättää BIDS:iin, koska BIDS tarjoaa lausekkeita varten syntaksin tarkistuksen.

Taulukko 5. Esimerkki tietovarastotaulun sarakkeiden määrityksestä.

Nimi	**Tyyppi**	**Pituus**	**Erityisvalinnat**	**Kommentit**
Id	int		Identity	
HenkiloId	int		not null	He_HenkiloId
Palkka	float			Pa_Palkka

Koska pienen resoluution tuki on yksi vaatimus dokumentille, taulujen määritykset on sijoitettu allekkain. Näin käyttäjät joutuvat vierittämään taulukkoa mahdollisimman harvoin sivusuunnassa.

6.3 Sisäänluku

Sisäänluvussa ei tässä vaiheessa kiinnitetä tarkemmin huomiota tietojen oikeellisuuteen, vaan oletetaan, että sen täyttäjä noudattaa ohjeistusta ja siten tietää, mitä dokumenttiin tulee laittaa. Sisäänluvussa on kuitenkin huomioitu SQL-injektiohyökkäyksen (engl. *SQL Injection*) mahdollisuus, vaikka ainakin sovelluksen

alkuaikoina voidaan olettaa, että käyttäjä on luotettava. Näin ei kuitenkaan ole myöhemmässä vaiheessa, mikäli sovellus tulee yleisempään käyttöön, joten SQL-injektion esto on kirjoitettu dokumentin sisäänluvun hoitavaan ohjelman osuuteen. SQL-injektio on mahdollinen, sillä sovellus mahdollistaa kohdetaulujen luomisen tietokantaan automaattisesti.

SQL-injektio on ongelma varsinkin web-sovelluksissa. Hyökkäys tapahtuu syöttämällä SQL-komentoja web-sovelluksen syötekenttiin. Jos sovelluksessa ei ole huomioitu SQL-injektion mahdollisuutta, ja syötekenttään laitettu komento ajetaan sellaisenaan sovelluksen tietokantaan, hyökkääjä voi mahdollisesti päästä käsiksi sellaiseen tietoon, johon peruskäyttäjällä ei ole oikeuksia. Lisäksi hyökkääjä voi selvittää tietokannan rakenteen ja lisätä tai poistaa dataa tietokannasta. SQL-injektion voi estää muun muassa seuraamalla ja muokkaamalla käyttäjän syötteitä. (Ciampa et al. 2010, s. 43.) Syötteestä voi tehdä turvallisemman poistamalla kiellettyjä merkkejä tai avainsanoja. Lisäksi C# tarjoaa mahdollisuuden parametrisoituihin SQL-komentoihin, joissa käyttäjän syötettä ei lisätä sellaisenaan SQL-komentoon, vaan se annetaan SQL-komennolle parametriksi. Tässä diplomityössä tehdyssä sovelluksessa käytetään molempia edellä mainituista tavoista.

Muuten sisäänluvussa hyödynnetään Microsoft *Excelin* tarjoamaa ohjelmointirajapintaa. Rajapinta mahdollistaa muun muassa taulukkojen lukemisen, muokkaamisen ja tallentamisen ohjelmallisesti *Microsoft.Office.Interop.Excel*-kirjaston kautta. Tässä työssä rajapintaa käytetään avaamaan *Excel*-taulukko ja lukemaan sen välilehdiltä tietojen käsittelysäännöt. Välilehdiltä etsitään avainsanoja, joiden perusteella tietyt solut luetaan sisään. Luettujen solujen perusteella luodaan oliot sisään luettavia tauluja ja kohdetaulua varten.

6.4 Yhteenveto määritysdokumenttipohjasta

Tässä luvussa esiteltiin, miten työssä suunniteltiin yksinkertainen, selkeä ja laajennettavissa sekä helposti omaksuttavissa oleva määritysdokumenttipohja latausprosessin käsittelysääntöjen tallentamiseksi. Yksinkertaisuudestaan huolimatta, tai ehkä siitä johtuen, tämänkin dokumentin luettavuus kärsii, jos tietovarastotauluissa on paljon sarakkeita ja tiedot tulevat useista tauluista useilla eri liitoksilla. Toisaalta

luettavuuden heikkeneminen ei haittaa kokonaisuutta merkittävästi, koska dokumentti on tarkoitettu pääasiassa generoimaan latausprosessin runko.

7 LATAUSPAKETIN MUODOSTAMINEN MÄÄRITYSDOKUMENTTIEN POHJALTA

Edellisissä luvuissa esiteltiin, miten ETL-prosessi on mahdollista generoida käyttäen SSIS:n ohjelmointirajapintaa, sekä määritysdokumenttipohja tietovaraston latausprosessin käsittelysääntöjä varten. Kummastakin on vain vähän hyötyä ilman tapaa muodostaa määritysdokumentin pohjalta malli, jonka pohjalta itse latauspaketti generoidaan. Tässä luvussa esitetään kuvan 6 latausrungon muodostaminen. Tässä työssä muodostetaan kolme erilaista latausrunkoa: *staging*-, dimensio- ja faktalatausrungot.

7.1 Staging-latausrungon muodostaminen

Staging-latauksen runko on tässä työssä tehtävistä latausrungoista yksinkertaisin, mikä johtuu suoraan siitä, että *staging*-lataus on moniulotteisen tietovaraston latauksista yksinkertaisin. *Staging*-latauksen perusrakenne jakautuu *staging*-taulun tyhjennykseen, tietojen poimimiseen lähteestä, mahdollisiin yksinkertaisiin muokkauksiin ja tietojen lataamiseen *staging*-kannan tauluun. SSIS:ssä työläin osuus *staging*-latauksessa on yleensä tietojen poiminnan määrittäminen lähteestä. Jos lataus tehdään useasta lähdejärjestelmän tietokannan taulusta, voi poiminnan määrittäminen olla monimutkainen, mutta ei välttämättä kovin työläs. Jos lataus tehdään tiedostosta, jonka sarakkeet ovat kiinteälevyiset, poiminnan määrittäminen sieltä voi olla erittäin työlästä. Tämä johtuu siitä, että jokainen sarake ja sarakkeen tietotyyppi joudutaan määrittämään yksitellen. Vaikka yhden tiedoston poiminnan määrittämiseen ei välttämättä kulu kymmentä minuuttia enempää, tiedostojen määrä moninkertaistaa määrittämiseen kuluvan ajan. Lisäksi ongelmana on se, että joissain tapauksissa pieni muutos tiedoston määrityksessä voi aiheuttaa sen, että tiedoston poiminta joudutaan määrittämään alusta alkaen uudelleen.

Tässä työssä kaikkien lataustyyppien rungot muodostavat ensimmäiseksi yhteydet tietolähteisiin ja tietovarastoon. Sen jälkeen *staging*-latauksen runko tyhjentää ensin kohdetaulun määritystiedostosta saatavien tietojen perusteella. Kohdetaulun tyhjentämiseen käytetään yksinkertaista *truncate table* -komentoa, jonka yhteyteen haetaan tyhjennettävän taulun nimi määritysdokumentissa. Taulun tyhjennyksen jälkeen avataan tiedosto tai yhteys lähdekantaan. Sen jälkeen luodaan *derived*

lumn -transformaatio, joka tekee määritysdokumentin kommenttikentän tietojen pohjalta muunnoksia tietoihin. Tämän jälkeen tiedot kirjoitetaan *staging*-tauluun.

7.2 Dimensiolatausrungon muodostaminen

Dimensiolatauksen perusrakenne on melko yksinkertainen, mutta rakenne vaihtelee enemmän kuin *staging*-latauksessa. Rakenne koostuu yleensä seuraavista vaiheista: tietojen poiminta lähtö- tai *staging*-tauluista, mahdolliset muunnokset ja SCD-lataus (*Slowly Changing Dimension*) dimensiotauluun. Dimensiolatauksen rungon monimutkaisuus vaihtelee sen mukaan, millainen dimensio on kyseessä, ja mistä arvot dimensioon tulevat. Jos dimensio on sellainen, että sitä ylläpidetään käsin esimerkiksi siirtotiedostoissa, sen sisäänlataus on yksinkertaista. Tällöin voidaan vain hakea tiedot siirtotiedostosta ja kirjoittaa muutokset dimensiotauluun. Jos dimension luonnollinen avain muodostuu useasta eri taulusta tulevista sarakkeista ja dimensio vaatii useita muunnoksia dataan, dimensiolataus on monimutkainen. Tällöin myös vastaavan faktataulun lataaminen on monimutkaista. Dimensiolatauksiin ei olekaan yhtä suoraviivaista ratkaisua kuin *staging*-latauksiin. Sekä tämän että automaattisen generoinnin rajoituksien vuoksi tässä työssä tehty dimensiolatauksen runko ei pyri kattamaan kaikkia vaihtoehtoja. Se voi tehdä yksinkertaisen yhdestä taulusta tulevan dimensiolatauksen tai yhdistää useasta taulusta tulevaa tietoa.

Eri tauluista tulevien tietojen yhdistäminen toisiinsa erityyppisillä *join*-operaatioilla tekee tietovirrasta käytännössä binääripuun, jossa tietolähteet ovat lehtisolmuja. Algoritmi, jolla puu muodostetaan, on seuraavanlainen: ensin muodostetaan lähtötaulujen oliot ja lisätään ne listaan, jossa niiden yhteyteen liitetään myös tieto siitä, minkä taulujen kanssa ne liitetään keskenään. Sitten liitosnumerot käydään numerojärjestyksessä läpi ja yhdistetään kaikki samaan liitokseen kuuluvat lähteet keskenään. Kun yksi liitosnumero on valmis, sitä varten tehdään oma olio, joka sisältää tiedot sen seuraavista liitoksista. Tämä olio lisätään läpikäytävien olioiden listaan, ja sieltä poistetaan jo käsitellyt oliot. Tätä jatketaan, kunnes kaikki liitokset on tehty ja saadaan viimeinen liitos. Viimeinen liitos yhdistetään *derived lumn* -transformaatioon, joka tekee määritysdokumentissa määritetyt muutokset sarakkeille. Sen jälkeen tiedot viedään dimensiotauluun *slowly changing dimension* -transformaation avulla.

7.3 Faktalatausrungon muodostaminen

Faktalatauksen perusrakenne on yleensä melko suoraviivainen. Lähtö- tai *staging*-tauluista haetaan tiedot, luonnolliset avaimet korvataan dimensiotauluista saaduilla surrogaattiavaimilla ja rivit kirjoitetaan faktatauluun. Se, mitä latauksessa tehdään faktataulussa oleville riveille, vaihtelee. Joissain tapauksissa faktataulu tyhjennetään kokonaan, joskus taulusta poistetaan joitain rivejä ja joskus olemassa oleviin tietoihin ei kosketa ja uudet tiedot ladataan osittain. Faktalatauksen runko muodostuu lähdetaulujen lataamisesta, luonnollisten avainten korvaamisesta surrogaattiavaimilla, mahdollisista muunnoksista faktatietoon ja lopulta tietojen kirjoittamisesta kohdetauluun.

Tässä työssä tehdyssä sovelluksessa faktalatauksen rakenne on samankaltainen dimensiolatauksen kanssa ja se käyttää samaa algoritmia liitoksissa. Erona on, että tietovirran tiedot ladataan kohdetauluun suoraan ilman *slowly changing on* -transformaatiota.

7.4 Sovelluksen arkkitehtuuri

Sovellus jakautuu kolmeen moduuliin: *ExcelReader*, *PackageBuilder* ja *PackageGenerator* (kuva 13). *ExcelReader* on vastuussa *Excel*-tiedoston lukemisesta, *PackageBuilder* paketin logiikan muodostamisesta ja *PackageGenerator* itse paketin muodostamisesta. Kuvaan 6 verrattuna *ExcelReader*- ja *PackageBuilder*-osien toiminta asettuu ensimmäiseen generointiin ja latauspaketin runkoon. *PackageGenerator* puolestaan vastaa viimeistä generointia ja SSIS-pakettia.

Kuva 13. Sovelluksen arkkitehtuuri.

Arkkitehtuurissa *PackageBuilder*-moduuli on keskeisin ja se käyttää sekä *ExcelReader*- että *PackageGenerator*-moduulien luokkia. Se on luonnollinen valinta keskeiseksi osaksi arkkitehtuurissa, sillä sen vastuuna on paketin logiikan muodostaminen. Muut moduulit lähinnä palvelevat *PackageBuilder*-moduulin tarpeita.

7.5 Yhteenveto latauspaketin muodostamisesta

Tässä luvussa esiteltiin, miten erityyppisten latauspakettien rungot muodostetaan määritysdokumenttiin täytettyjen käsittelysääntöjen pohjalta. Näistä *staging*-lataus pysyy aina samankaltaisena, kun taas dimensio- ja faktalatauksessa on enemmän vaihtelua. Yksinkertaisimmillaan erityyppiset lataukset ovat melko samankaltaisia, mutta monimutkaisimmillaan erot ovat huomattavia.

8 POHDINTA

Tässä työssä tutkittiin ETL-prosessin kehittämisen nopeuttamista. Työn teoriaosuudessa luotiin katsaus tietovarastoihin, ETL-prosessiin sekä Microsoftin *SQL Server Integration Services* -ETL-työkaluun. Näiden lisäksi olemassa olevaa tutkimustietoa käytiin läpi ehdotettujen nopeutusratkaisujen löytämiseksi. Tutkimukset kuitenkin esittävät eriäviä mielipiteitä siitä, miten ETL-prosessi tulisi mallintaa ja mihin mallien tulisi ottaa kantaa. Kahdessa tutkimuksessa on esitetty toimiva tapa muodostaa ETL-prosessi käsitemallin pohjalta. Muodostus on kuitenkin puoliautomaattista, sillä ETL on täynnä yksityiskohtia, joita pitää määrittää toimivan prosessin aikaansaamiseksi. Näitä yksityiskohtia on mahdollista piilottaa ETL:n käsitemallista, mutta niiden tulee silti olla tarjolla jossain. Tästä muodostuukin ongelma esitetyissä malleissa, sillä valittuja kuvauskieliä ei ole suunniteltu ETL-prosessien mallintamiseen. Siksi tässä työssä esitettiin tutkimustiedon pohjalta uusi arkkitehtuuri, joka hyödyntäisi alusta alkaen ETL:n kuvaamiseen tarkoitettua TDL-kieltä (*Transformation Description Language*).

Työn käytännön osuudessa toteutettiin sovellus, joka lukee *Excel*-taulukossa määritetyt ETL-prosessin käsittelysäännöt ja luo niiden pohjalta ETL-prosessin rungon SSIS-pakettiin. Sovellus pohjautuu tutkimustiedon pohjalta esitettyyn arkkitehtuuriin, mutta diplomityön rajausten vuoksi koko arkkitehtuuria ei voitu tässä työssä toteuttaa.

Yksi puute sovelluksessa on kunnollisen käyttöliittymän puute. Tällä hetkellä sovellus ottaa vastaan määritystiedoston sekä kohdetiedoston komentoriviparametreina. Lisäksi esimerkiksi generoidussa faktataulun latauksessa kaikki dimensioiden surrogaattiavaimet haetaan toistaiseksi käyttäen erillisiä lähteitä ja liitoksia faktataulun tietovirtaan. Tämä ei tosin ole huono tapa varsinkaan suurten dimensioiden tapauksessa, mutta monet lähteet ja *merge join* -liitokset voisi korvata *lookup*-transformaatioilla ja samalla yksinkertaistaa latauspakettia. *Lookup*-transformaatio on tehokkaampi pienemmillä dimensioilla, kun vertailutaulu mahtuu kokonaisuudessaan keskusmuistiin, mutta keskusmuistin loppuessa *lookup*-transformaatio hidastuu huomattavasti, toisin kuin *merge join* -transformaatio (Veerman et al. 2009, s. 235).

Tätä varten voisi tehdä valinnan, jolla käyttäjä voi valita, kummalla tavalla surrogaattiavaimet haetaan kunkin dimensiotaulun kohdalla.

8.1 Jatkokehitys

Jatkokehitysmahdollisuudet työn pohjalta jakautuvat kolmeen eri vaihtoehtoon. Ensimmäinen on kuvassa 5 esitetyn arkkitehtuurin toteuttaminen kokonaisuudessaan. Toinen on määritysdokumenttipohjan ja ETL-prosessin rungon generoimisen kehittäminen edelleen ja kolmas TDL-kielen (*Transformation Description Language*) kehittäminen. Ensimmäisessä jatkokehitysvaihtoehdossa voidaan harkita käsittelysääntöjen ja ETL-prosessin rungon generoimisen tärkeyttä. Periaatteessa jo pelkästään TDL-kielen kehittäminen ja sen pohjalta eri ETL-välineiden ohjelmakoodin generoiminen on hyödyllistä, kuten tutkimuksissa Akkaoui et al. (2011) ja Muñoz et al. (2009) on havaittu. ETL-prosessin rungon generoimisella käsittelysääntöjen pohjalta on kuitenkin potentiaalia nopeuttaa ETL-prosessin kehitystyötä eniten. Lisäksi tässä diplomityössä on jo toteutettu alku rungon generoimiselle, joten siitä olisi hyvä jatkaa. ETL-prosessin rungon generoimisen voisi integroida sovelluksen käyttöliittymään, jolloin käyttöliittymä voisi tarjota käyttäjälle erilaisia latausrunkoja. On selvää, että ensimmäinen vaihtoehto on myös kaikkein työläin ja se vaatii asiantuntemusta useista eri aihepiireistä ja välineistä. Asiantuntemusta vaaditaan muun muassa ohjelmointikielen luomisesta ja kääntämisestä sekä kaikista ETL-välineistä ja niiden käyttämistä kielistä, joihin TDL halutaan kääntää.

Määritysdokumenttipohjan ja ETL-prosessin rungon generoimisen edelleen kehittäminen keskittyy käytännössä ETL-prosessin kehittämisen nopeuttamiseen, joka onkin ollut tämän työn käytännön osuuden tarkoitus. Käsittelysäännöistä voisi tutkia, kuinka monipuolisia niistä voisi tehdä ja kuinka pitkälle hiottu runko niiden pohjalta olisi mahdollista saada aikaan. Koko pakettia koskevia säännöksiä varten voisi lisätä kokonaan oman välilehden määritysdokumenttipohjaan. Yksi tällainen sääntö voisi olla esimerkiksi faktatauluista tietojen poistamiseen liittyvä sääntö, joka kertoo, mitä poistetaan ja miten. Lisäksi esimerkiksi olemassa olevien SSIS-pakettien muokkaaminen puuttuu tällä hetkellä sovelluksesta ja sen toteuttaminen olisi hyödyllistä määrityksiin tulevia muutoksia ajatellen. Myös sovelluksessa tällä hetkellä olevat puutteet tulisi korjata.

Pelkkä TDL-kielen kehittäminenkin olisi hyödyllistä, mutta sen tulisi saada tarpeeksi tunnettuutta, jotta työ ei menisi hukkaan. Kun kielelle saisi laajan tuen, syntyisi todennäköisesti useita sovelluksia, jotka hoitaisivat kielen kääntämisen eri ETL-välineiden ohjelmakoodiksi. Tämä tilanne helpottaisi uuteen toteutusvälineeseen siirtymistä, vaikkei poistaisikaan tarvetta eri toteutusvälineiden asiantuntijoille. Todennäköisesti joitakin harvemmin käytettyjä tai selkeästi yhden toteutusvälineen erikoisominaisuuksia jouduttaisiin silti hallitsemaan ETL-kielen omalla toteutusvälineellä.

9 JOHTOPÄÄTÖKSET

ETL-työkalut kaipaavat samanlaista ratkaisua kuin BPMN tarjosi BPEL:n ongelmiin. Siinä missä BPEL-työkalujen eri esitystavat aiheuttivat hankaluuksia, eri valmistajien vain omissa ympäristöissään toimivat ETL-työkalut vaikeuttavat siirtymistä työkalusta toiseen. On tarve kuvata ETL-prosessit välineistä riippumattomasti, mutta tähän mennessä ehdotetut ratkaisut eivät riitä, sillä muihin käyttötarkoituksiin tarkoitetut kuvauskielet eivät sovellu riittävän hyvin ETL-prosessien mallintamiseen. Tarvitaan uusi kieli, joka on tarkoitettu alusta lähtien ETL-prosessien kuvaamiseen. Tässä työssä on esitetty ratkaisuksi TDL-työnimellä (*Transformation Description Language*) kulkevan kielen mahdollisuutta.

Tutkimuskysymykseen ”*miten lyhentää tietovaraston latausprosessin kehittämiseen kuluvaa aikaa ohjelmallisesti*” vastaus jakautuu kahteen tapaan. Ensimmäinen tapa liittyy ETL-prosessin mallintamiseen kuvauskielellä, jonka pohjalta on mahdollista generoida valmis ETL-ohjelmakoodi. Tässä tavassa latausprosessin kehittäminen nopeutuu varsinkin ETL-välineestä toiseen siirryttäessä, koska toteutusmenetelmät on mahdollista kopioida suoraan vastaavista projekteista. Toinen tapa on ETL-prosessin rungon generoiminen automaattisesti käsittelysääntöjen perusteella. Näin on tehty tämän työn käytännön osuudessa. Näiden tapojen yhdistäminen on mahdollista tässä työssä esitetyllä tavalla.

LÄHTEET

Akkaoui, Zineb El & al. 2011. A Model-Driven Framework for ETL Process Development. Teoksessa: Song, I., Cuzzocrea, A. & Davis, K. C. (toim.) Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP. New York, NY, USA: ACM. 2011. S. 67–74. ISBN 978-1-4503-0963-9.

Akkaoui, Zineb El & Zimányi, Esteban. 2009. Defining ETL Workflows using BPMN and BPEL. Teoksessa: Song & Zimányi 2009, s. 41–48.

Aureolis. 2010. Aureolis – Business Intelligence for Better Information [verkkodokumentti]. Julkaistu 2010 [viitattu 13.5.2012]. Saatavissa: http://www.aureolis.com/pages/fi/etusivu.php.

Ciampa, A., Corrado, A. V. & Di Penta, M. 2010. A heuristic-based approach for detecting SQL-injection vulnerabilities in Web applications. Teoksessa: Lee, S-W., Monga, M. & Jürjens, J. (toim.) Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems. New York, NY, USA: ACM. 2010. S. 43–49. ISBN 978-1-60558-965-7.

Codd, E. F. 1971. Normalized data base structure: a brief tutorial. Teoksessa: Codd, Edgar F. & Dean, Albert L. (toim.) Proceedings of the 1971 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control. New York, NY, USA: ACM. 1971. S. 1–17. ISBN tuntematon.

Guea, Silviu. 2008. Generating SSIS script component code programmatically. Julkaisussa: SQL Server Integration Services [online-keskustelupalsta]. Microsoft, 9.12.2008 [viitattu 22.4.2012]. Saatavissa: http://social.msdn.microsoft.com/Forums/br/sqlintegrationservices/thread/21c052e0-7673-40e8-b1ee-bc9c717db644.

Hovi, Ari. 1997. Data Warehousing – Tietovarastointitekniikka. Espoo: Suomen Atk-kustannus Oy. 124 s. ISBN 951-762-509-X.

Hovi, A., Hervonen, H. & Koistinen, H. 2009. Tietovarastot ja Business Intelligence. Jyväskylä: WSOYpro/Docendo-tuotteet. 196 s. ISBN 978-951-0-34792-8.

Inmon, W.H. 2005. Building the Data Warehouse, 4th Edition. Indianapolis, Indiana: Wiley Publishing, Inc. 576 s. ISBN 978-0-7645-9944-6.

ISO/IEC 19501-1:2012. 2012. Information technology - Object Management Group Unified Modeling Language (OMG UML), Infrastructure. Geneve: International Organization for Standardization. 220 s.

ISO/IEC 19501-2:2012. 2012. Information technology - Object Management Group Unified Modeling Language (OMG UML), Superstructure. Geneve: International Organization for Standardization. 740 s.

Kimball, Ralph & Caserta, Joe. 2004. The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data. Indianapolis, Indiana: Wiley Publishing, Inc. 528 s. ISBN 978-0-7645-6757-5.

Knight, Brian & al. 2008. Professional Microsoft SQL Server 2008 Integration Services. Indianapolis, Indiana: Wiley Publishing, Inc. 1008 s. ISBN 978-0-470-24795-2.

Linstedt, Dan E. 2002. Data Vault Series 1 – Data Vault Overview [verkkodokumentti]. Julkaistu 2002 [viitattu 20.11.2011]. Saatavissa: http://www.tdan.com/view-articles/5054/.

Luján-Mora, S., Trujillo, J. & Song, I. 2002a. Extending the UML for Multidimensional Modeling. Teoksessa: Jézéquel, J., Hußmann, H. & Cook, S. (toim.) UML '02 Proceedings of the 5th International Conference on The Unified Modeling Language. London, UK: Springer-Verlag. 2002. S. 290–304. ISBN 3-540-44254-5.

Luján-Mora, S., Trujillo, J. & Song, I. 2002b. Multidimensional Modeling with UML Package Diagrams. Teoksessa: Spaccapietra, S., March, S. T. & Kambayashi, Y. (toim) ER '02 Proceedings of the 21st International Conference on Conceptual Modeling. London, UK: Springer-Verlag. 2002. S. 199–213. ISBN 3-540-44277-4.

Malinowski, Elzbieta & Zimányi, Esteban. 2008. Advanced Data Warehouse Design. Berlin, Heidelberg: Springer-Verlag. 444 s. ISBN 978-3-540-74404-7.

Microsoft. 2008a. Architecture of Integration Services [verkkodokumentti]. Julkaistu 2008 [viitattu 8.5.2012]. Saatavissa: http://msdn.microsoft.com/en-us/library/bb522498.aspx.

Microsoft. 2008b. Building Packages Programmatically [verkkodokumentti]. Julkaistu 2008 [viitattu 8.5.2012]. Saatavissa: http://msdn.microsoft.com/en-us/library/ms345167.aspx (etusivu).

Microsoft. 2010. Properties (C# Programming Guide) [verkkodokumentti]. Julkaistu 2010 [viitattu 13.5.2012]. Saatavissa http://msdn.microsoft.com/en-us/library/x9fsa0sw.aspx.

Microsoft. 2012. Microsoft Releases SQL Server 2012 to Help Customers Manage "Any Data, Any Size, Anywhere" [verkkodokumentti]. Julkaistu 2012 [viitattu 8.5.2012]. Saatavissa: http://www.microsoft.com/en-us/news/press/2012/mar12/03-06SQLServer12PR.aspx.

Muñoz, Lilia & al. 2008. Modelling ETL Processes of Data Warehouses with UML Activity Diagrams. Teoksessa: Meersman, R., Tari Z. & Herrero P. (toim.) Proceedings of the OTM Confederated International Workshops and Posters on On the Move to Meaningful Internet Systems: 2008 Workshops: ADI, AWeSoMe, COMBEK, EI2N, IWSSA, MONET, OnToContent + QSI, ORM, PerSys, RDDS, SEMELS, and SWWS. Berlin, Heidelberg: Springer-Verlag, 2008. S. 44–53. ISBN 978-3-540-88874-1.

Muñoz, Lilia & al. 2009. Automatic generation of ETL processes from conceptual models. Teoksessa: Song & Zimányi 2009, s. 33–40.

Nanda, Ashwani. 2010. Hands-On Microsoft SQL Server 2008 Integration Services, Second Edition. USA: McGraw-Hill Osborne Media. 720 s. ISBN 978-0-07-173640-4.

Nissen, Gary. 2003. Is Hand-Coded ETL the Way to Go? [verkkodokumentti]. Julkaistu 2003, päivitetty 11.1.2011 [viitattu 13.5.2012]. Saatavissa: http://www.kimballgroup.com/html/articles_search/articles2003/0305IE.html.

O'Brien, James A. & Marakas, George M. 2009. Management information systems. Boston, MA: McGraw-Hill/Irwin. 659 s. ISBN 978-0-07-128043-3.

Rainardi, Vincent. 2007. Building a Data Warehouse with Examples in SQL Server. Berkeley, CA: Apress, Inc. 523 s. ISBN 978-1-59059-931-0.

Simitsis, Alkis & Vassiliadis, Panos. A Methodology for the Conceptual Modeling of ETL Processes. Teoksessa: Eder, J., Mittermeir, R. & Pernici, B. (toim.) CAiSE '03 Workshop-Proceedings. University of Maribor Press. S. 305–316. ISBN 86-435-0552-8.

Song, Il-Yeol & Zimányi, Esteban (toim.) 2009. Proceedings of the ACM twelfth international workshop on Data warehousing and OLAP. New York, NY, USA: ACM. 106 s. ISBN 978-1-60558-801-8.

Trujillo, Juan & al. 2001. Designing Data Warehouses with OO Conceptual Models. Computer, 34: 12. S. 66–75. ISSN 0018-9162.

Trujillo, Juan & Luján-Mora, Sergio. 2003. A UML Based Approach for Modeling ETL Processes in Data Warehouses. Teoksessa: Song, Il-Yeol & al. (toim) Conceptual Modeling – ER 2003 22nd International Conference on Conceptual Modeling, Chicago, IL, USA, October 13-16, 2003, Proceedings. Berlin, Heidelberg: Springer-Verlag. 2003. S. 307–320. ISBN 978-3-540-20299-8.

Vassiliadis, Panos et al. 2000. Arktos II: A Design tool for ETL scenarios [verkkodokumentti]. Julkaistu 2000, päivitetty 4.9.2008 [Viitattu 5.5.2012]. Saatavissa: http://www.cs.uoi.gr/~pvassil/projects/arktos_II/index.html.

Vassiliadis, P., Simitsis, A. & Skiadopoulos, S. 2002. Conceptual modeling for ETL processes. Teoksessa: Song, Il-Yeol & Theodoratos, Dimitri (toim) CIKM '02 Eleventh ACM International Conference on Information and Knowledge Management, McLean, VA, USA — November 04 - 09, 2002. New York, NY, USA: ACM. S. 14–21. ISBN 1-58113-590-4.

Veerman, Erik et al. 2009. Microsoft SQL Server 2008 Integration Services: Problem, Design, Solution. Indianapolis, Indiana: Wiley Publishing, Inc. 480 s. ISBN: 978-0-470-52576-0.