Lappeenrannan teknillinen yliopisto
Teknistaloudellinen tiedekunta
Tietotekniikan osasto

IFC-tietomallin mukaisen tiedon jäsentäminen, käsittely ja siirto

Työn ohjaaja: Juha Puustjärvi
Työn tarkastaja: Jari Porras

Joensuussa 8.4.2008

Työn tekijä: Mikko Anttonen
Hyväriläntie 16
80260 JOENSUU
puh. 013-316086

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
Teknistaloudellinen tiedekunta
Tietotekniikan osasto
Mikko Anttonen

IFC-tietomallin mukaisen tiedon jäsentäminen, käsittely ja siirto

Diplomityö

2008

75 sivua, 30 kuvaa, 7 taulukkoa ja 5 liitettä

Tarkastajat: Professori Juha Puustjärvi
Professori Jari Porras

Hakusanat: IFC, CAD, jäsennin, semanttinen web, STEP, tietokannat, XML
Keywords: IFC, CAD, database, parser, semantic web, STEP, XML

Työssä tutkittiin IFC (Industrial Foundation Classes)-tietomallin mukaisen tiedoston jäsentämistä, tiedon jatkoprosessointia ja tiedonsiirtoa sovellusten välillä. Tutkittiin, mitä vaihtoehtoja tiedon siirron toteuttamiseksi ohjelmallisesti on ja mihin suuntaan tiedon siirtäminen on menossa tulevaisuudessa. Soveltavassa osassa toteutettiin IFC-standardin mukaisen ISO10303-tiedoston (Osa 21) jäsentäminen ja tulkitseminen XML-muotoon. Sovelluksessa jäsennetään ja tulkitaan CAD-ohjelmistolla tehty IFC-tiedosto C# -ohjelmointikielellä ja tallennetaan tieto XML-tietokantaan kustannuslaskentaohjelmiston luettavaksi.

ABSTRACT

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

Parsering, processing and transfering data using IFC-data model

Thesis for the Degree of Master of Science in Technology

2008

75 pages, 30 figures, 7 tables and 5 appendices

Examiners: Professor Juha Puustjärvi
Professor Jari Porras

Keywords: IFC, CAD, database, parser, semantic web, STEP, XML

This research examined data parsering, processing and exchange between applications so, that the common data model is IFC (Industrial Foundation Classes). What possibilities at the software level there are to implement data exchange to programme and to what kind of technology data exchanging is going to develop? In the research's implementation part there was implemented IFC-standard file (ISO10303-Part 21) parsing and interpreting to XML-file. IFC-file done with the CAD-application can be parsed and interpreted with application created with C# -language and the data can be saved to XML-database to be read with the cost estimation application.

SISÄLLYSLUETTELO

1 JOHDANTO

Pyrin kuvaamaan tässä luvussa tämän työn taustaa, tutkimuskysymyksiä ja työn rajausta sekä rakennetta. Koska tiedonsiirto käsitteenä on valtavan laaja aihealue, olen rajoittanut tiedonsiirron käsittelemisen yhdelle alalle (rakennusalalle) ja sovelluksien väliseen tiedonsiirtoon. Käytännössä IFC oli minulle täysin uusi asia ja jouduin aloittamaan ”tyhjältä pöydältä”. Osittain se selittää miksi työni liitteenä on paljon selvittävää materiaalia IFC:n perusteista.

1.1 Työn tausta

Tiedonsiirto kehittyy monella teollisuuden alalla tietotekniikan ansiosta. Koska olen koulutukseltani rakennusinsinööri sekä saanut koulutuksen myös tietotekniikassa, ja koska tiedonsiirto varsinkin internetin kehityksen kautta on muuttumassa valtavasti, päätin tutkia rakennusalan sovellusten välistä tiedonsiirtoa nyt ja tulevaisuudessa. Lisäksi, koska olen tällä hetkellä ammattikorkeakoulussa tietotekniikan tuntiopettaja ja opetan nimenomaan ohjelmointia, tutkimukseni kohdistuu nimenomaan sovellusten toteutukseen koodin tasolla. Tuloksena syntyi myös sovellus, jonka avulla voin siirtää tietoa CAD-ohjelmistolla tuotetusta IFC-tiedostosta rakennusalan tarjouslaskentaohjelmaan.

1.2 Tutkimuskysymykset, työn rajaus ja rakenne

Työlläni on muutamia keskeisiä tutkimuskysymyksiä. Miten rakenteinen tieto voidaan siirtää kahden eri sovelluksen välillä, jos tieto on rakennusalan IFC-standardin tietomallin mukaisessa muodossa? Miten tietorakenteet tiedonsiirron suhteen on määritelty, jotta tieto saadaan käsiteltyä ja siirrettyä sovelluksesta toiseen? Mihin suuntaan tiedonkäsittely tässä suhteessa on kehittymässä?

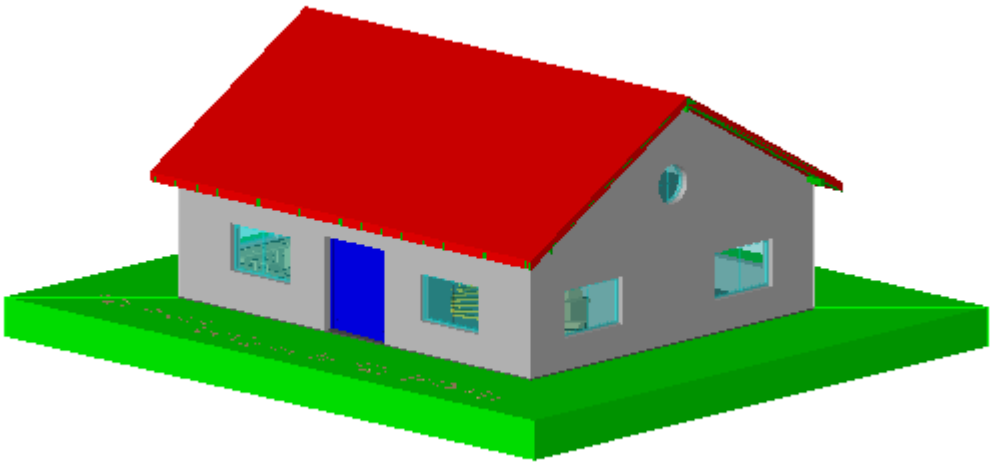
Työ on rajattu itse tiedonsiirtoon ja tiedon rakenteiden käsittelyyn eikä niinkään teknologiaan (laitteistojen kannalta) joka liittyy tiedon siirtoverkkoihin. Tiedonsiirrossa keskitytään nimenomaan ohjelmalliseen (software) osuuteen kohteena tiedon rakenteiden määrittely koodin tasolla.

Käsittelen ensin tietomallien historiaa, niiden perusperiaatteita ja sitä mihin tietomalleja tarvitaan tiedonsiirrossa. Sen jälkeen käsittelen rakennusalan standardien kehitystä sekä

varsinaisen IFC-standardin kehitystyötä ja ylipäätänsä IFC-standardin tavoitteita: mitä hyötyjä voidaan saavuttaa siirryttäessä IFC-muotoiseen tietomalliformaattiin. Eräs painopiste työlläni on tutkia mitä eri vaihtoehtoja erilaisten rakennusalan sovellusten välisen tiedonsiirron toteuttamiseen tällä hetkellä on ja mitä on kehitteillä. Käyn läpi tiedostopohjaisen tiedonsiirron ja etenen tietomalleihin. IFC-mallin mukaista tietoa voidaan tallentaa SPF- formaatissa ja XML-muodossa (uudessa ifcxml-formaatissa sekä semanttisen webin avulla XMLSchema:ssa sekä ifcOWL-kielellä.) Tieto tallennetaan monesti tietokantoihin ja sen vuoksi käsittelen luvussa 5 pelkästään tietokantoja. Työssä tehdään katsaus myös uusimpiin tiedonsiirron tekniikoihin: semanttisen webin hyväksikäyttöön tiedonsiirrossa.

Työssäni toteutan osan IFC-muotoisen datan siirrosta kustannuslaskentaohjelmiston ja CAD-ohjelmiston välille. Koska XML on tiedonsiirrossa käytetty kieli nyt ja ainakin lähitulevaisuudessa, tieto on pyrittävä tallentamaan XML – muotoon, jotta sen jatkoprosessointi olisi helpompaa sekä paikallisesti että tietoverkoissa. Työssä kuvataan lopuksi eräs tapa jäsentää, tulkita ja tallentaa tietoa IFC-tiedostosta muuntaen sen XML-muotoiseksi käyttäen C#.NET-ohjelmointikieltä.

Ajatellen työni lähtökohtaista ongelma-asettelua, koko työni perusongelma on siis siinä, miten tietoa voidaan siirtää koodin tasolla eri rakennusalan ohjelmistojen välillä IFC:n avulla. IFC:n avulla voidaan siirtää tietoa CAD-ohjelmistojen välillä tai sitten niin, että tietoa siirretään esimerkiksi CAD-ohjelmistosta kustannuslaskentaohjelmistoon tai projektinhallintaohjelmistoon. Oleellista kuitenkin on, että IFC:ssä käsitellään aina ensin 3D-mallia. Kuvassa 1 oleva esimerkki on avattu Nemetchek IfcViewer-sovelluksella.



Kuva 1. Eräs IFC:llä kuvattu kohde avattuna Nemetchek IfcViewer-sovelluksella.

Miten tietoa näiden ohjelmistojen välillä voidaan siirtää ja mikä voisi olla se formaatti joka on yleiskäyttöisin? Tiedon mallintamista on tutkittu jo monia vuosikymmeniä ja on varmaa, että tutkimustuloksia kannattaa käyttää hyväksi. Tietomallit ovat lopulta kehittyneet standardeiksi ja tärkeimpiä näistä standardeista ovat STEP sekä uusi tulokas IFC. IFC käyttää osittain STEP-standardin menetelmiä, mutta IFC:ssä on alettu käyttää myös XML-ohjelmointikieltä apuna. XML-kieli on myös niin sanotusti ”alusta riippumaton”, joka tarkoittaa sitä, että eri järjestelmissä saman syntaksin mukainen tieto voidaan tulkita samalla tavalla.

Mitä varten sitten tietoa kannattaa siirtää eri järjestelmien välillä? Ollessani eri rakennusfirmoissa töissä huomasin seikan, että uuden rakennuskohteen tarjouspyyntökierroksen aikana jokainen tarjoava firma laski suunnitellun kohteen piirustuspapereiden perusteilla rakennuskohteen massat (eli muun muassa maansiirto, betonit, teräkset, ovet, ikkunat) sekä hinnoitteli ne, lisäten erilaiset kustannukset, mitä kohteen rakentamisesta yritykselle ylipäätänsä koituu. Mitä jos suurin osa massoista voitaisiinkin saada jo suunnittelukuvien perusteella irti? Jos kuvan data kohteesta voidaan ilmaista tietotekniikan avulla rakenteisesti (tai ainakin puolirakenteisesti), voidaan rakennuskohteen massat laskea sen avulla. Tässä on jo yksi valtava hyötynäkökohta esillä.

Kun rakennuskohde kuvataan 3D-mallin mukaisesti, käyttämällä projektissa samaa kuvaa pohjana, saadaan myös muutokset näkymään kaikkialle reaaliaikaisesti. Tähän on otettu avuksi niin sanotut tuotemallipalvelimet (joita käsittelen myöhemmin), jossa eri rakentamiseen liittyvät osapuolet voivat käyttää samaa 3D-kuvaa. Jos esimerkiksi asiakas haluaa tehdä muutoksia huoneistoonsa ennen rakentamista, on periaatteessa mahdollista tuotemallipalvelimeen kytketyn kustannuslaskentaohjelmiston avulla nähdä heti mitä muutokset vaikuttavat huoneiston hintaan. Samaa periaatetta voidaan käyttää myös esimerkiksi kohteen projektiaikataulun suhteen.

Erilaiset ohjelmistot voivat lukea XML-muotoista dataa suhteellisen helposti. IFC-standardissa käytettävää ISO-10303-osa 21-formaatin tiedostomuotoa sen sijaan on hankalampi tulkita, varsinkin, jos ei ole käytettävissä suhteellisen kalliita tiedon jäsentimiä (parsereita).

Edellä luettelin syitä, miksi olen lähtenyt rakentamaan kustannuksellisesti suhteellisen halvalla tavalla omaa järjestelmääni tavoitteena laskea kustannukset rakennuskohteesta mahdollisimman nopeasti. IFC-tietomalli näyttää olevan maailmalle suhteellisen nopeasti leviävä standardi jota on ollut kehittämässä monta maata. On helppo ennustaa, että muutaman vuoden sisällä varsin moni rakentamisessa mukana oleva osapuoli on siirtymässä IFC:n käyttöön.

Mitä rakennuskohteesta voisi laskea helposti tällaisessa tapauksessa? Seinät, ikkunat ja ovet tulevat ensimmäisenä mieleen. Rakennuskohteessa on paljon muitakin laskettavia osa-alueita, mutta jostain on ensin aloitettava. Laitoin sovellukseni osalle tavoitteeksi, että ikkunat, ovet ja seinät olisi saatava laskettua ja myös jotenkin hinnoiteltua, koska ne ovat aika iso kustannuserä. Jos ne saataisiin järjestelmästä automaattisesti, myös massoitteluvirheet ja niiden hintamerkitys vähenee huomattavasti. Ovet ja ikkunat saadaan suhteellisen helposti massoiteltua ja hinnoiteltua kappaleina, kunhan vain tiedetään oven tai ikkunan koko sekä niiden tyyppi. Seinät sen sijaan ovat hieman hankalampi tapaus, koska varsinkin ulkoseinät koostuvat monesta eri kerroksesta. Tavoitteena olikin rakentaa järjestelmä, jossa voisin IFC-tiedoston perusteella hinnoitella seinän eri rakenteet ja näin saada seinän kokonaishinta yksikköä kohti järjestelmästä automaattisesti. Tämä ”välitieto” antaa kustannuslaskijalle käsityksen, mistä kustannuksista koko seinän rakenteen hinta koostuu. Kun tieto sitten siirretään varsinaiseen kustannuslaskentaohjelmistoon, voidaan seinän yksikköhinnan ja määrän

avulla laskea koko rakennettavan seinän hinta. Jos tämän ”ongelman” saa ratkaistua, voi samaa periaatetta käyttää myös esimerkiksi kattorakenteiden tai lattiarakenteiden hinnoittelussa. Myös aikataulutus voidaan suorittaa saman periaatteen mukaisesti. Koko järjestelmän rakentaminen tämän diplomityön aikana on kuitenkin liian iso urakka. Siksi tässä onkin pureuduttu mielestäni vain oleellisesti hankalimpiin kysymyksiin edellä mainitun esimerkin mukaisesti.

2. SIIRRETTÄVÄN TIEDON MUODON KEHITTYMINEN

Tässä luvussa pyrin esittelemään hieman taustaa miksi tietomallit ovat kehittyneet, mistä ne ovat lähtöisin ja miten niiden kuvaaminen on kehittynyt. Lisäksi käsittelen käsitettä ”rakenteellinen tieto” ja siihen liittyvää tiedonsiirtoa.

2.1 CAD ja graafiset mallit

Ensimmäisen CAD (Computer Aided Design) -järjestelmän kehitti vuonna 1963 Ivan Sutherland, joka tuolloin väitöskirjansa ohessa kehitteli graafisen laitteiston ja ohjelmiston nimeltään ”Sketchpad” (Eastman, 1999). Kesti noin kuusi vuotta edellisestä ajankohdasta (1969), kunnes Computervision Corporation:n toimesta valmistettiin ensimmäinen kaupallinen CAD järjestelmä (Eastman, 1999). Tietotekniikan kehittyessä myös CAD-järjestelmät ovat kehittyneet eteenpäin (Eastman, 1999). Kehityksen esteenä/veturina ovat olleet prosessorien, näyttöjen teknologian ja ohjelmistojen sekä algoritmien kehittyminen (Eastman, 1999). CAD – järjestelmissä pyritään esittämään kuva 2D (x ja y–koordinaatit samalla tasolla) tai 3D (syvyys-koordinaatisto mukana) – muodossa. Lisäksi kuvat voidaan piirtää pelkistä vektoreista, pintojen avulla tai käyttäen niin sanottua tilavuusmallinnusta.

Kolmiulotteiseen (3D) esitykseen viivamalli (”rautalankamalli” eli vektoreiden avulla piirretty malli) ei sovellu kovinkaan hyvin. Ongelmana on se, että esitys ei ole yksiselitteinen. Lisäksi 3D-rautalankamallien muodostaminen on hankalaa ja aikaa vievää. (mukaillen Laakko et al., 1998)

Pintamallilla (surface model) voidaan esittää monimutkaisia geometrisia muotoja. Pintamallit sisältävät informaatiota pintojen liittymisestä reunoihin. Reunat kuvataan tavallisesti parametrisilla käyrillä. Pintamalli muodostuu yhdistelemällä parametrisia pintalappuja (surface patches). Pintalappujen reunakäyrät ovat tavallisesti parametrisia käyriä. (Laakko et al., 1998)

Tilavuusmallit (solid models) kehitettiin alun perin graafisten mallien ongelmien takia (Laakko et al., 1998; alkup. lähde Mäntylä, 1988). Näitä ongelmia ovat muun muassa moniselitteisyys, epätäydellisyys ja rajoittunut sovellettavuus (Laakko et al., 1998; alkup. lähde Mäntylä, 1988). Tilavuusmallit pyrkivät olemaan ”täydellisiä” kappaleiden

kuvauksia, siis riittäviä vastaamaan algoritmisesti mihin tahansa geometriaa koskevaan kysymykseen (Laakko et al., 1998; alkup. lähde Mäntylä, 1988). Tilavuusmallintamisen kannalta kappale on kolmiulotteinen Euklidisen avaruuden osajoukko, joka on kuvattavissa matemaattisen mallin avulla (Laakko et al., 1998). Kappaleen matemaattisen mallin pohjalta luodaan tilavuusmalliesitys (Laakko et al., 1998).

Varsinkin koneenrakennusalalla tilavuusmallinnus on erittäin paljon käytetty ratkaisu. Rakennusalan CAD-sovelluksissa tilavuusmallinnusta käytetään harvemmin. Rakennusalan CAD-sovelluksissa pinnat ovat useimmiten suhteellisen yksinkertaisia suorakaiteen muotoisia pintoja. Edellä mainitut mallit: rautalankamalli, pintamalli ja tilaavuusmalli ovat kaikki pohjimmiltaan kuitenkin geometrisia malleja eli niissä määritellään miten CAD-kuva ilmaistaan graafisesti.

2.2 Tietomalli

Yritysten käyttämiä tietomalleja voidaan katsoa eri suunnista: liiketoiminnalliselta näkökannalta, tekniseltä kannalta tai käyttäjänäkökulmasta. Tutkimukseni kohdistuu nimenomaan tekniseen näkökulmaan ja jatkossa sanalla tietomalli tarkoitetaan nimenomaan teknistä tietomallia.

Koska tietoa pitää hallita myös muilla ohjelmistoilla kuin CAD:lla, jo varsin aikaisessa vaiheessa esimerkiksi rakennusalalla huomattiin, että eri objektien (kuten ikkunat, ovet, seinät) suhteet ja se mistä objektit koostuvat, pitää pystyä ilmaisemaan jotenkin yhteisessä ymmärrettävässä muodossa. Ensimmäiset rakennusalan tietomallia kuvaavia järjestelmiä rakennettiin Englannissa: esimerkiksi OXSYS CAD, CEDAR ja HARNESS, olivat niistä ensimmäisiä. Michiganin yliopistossa kehitettiin ARCH-MODEL, joka oli Macintosh:lle kehitetty CAD-järjestelmä. Siinä käytettiin tilavuusmallinnusta ja relaatiotietokantaa ei-graafisen datan tallentamiseen. (Eastman, 1999)

Tietomallin avulla voidaan tiedon käsittelyyn lisätä ”älykkyyttä”. Jos rakennetaan esimerkiksi tietomalli ovesta, siihen voidaan lisätä tietoa oven ominaisuuksista kuten leveys, paksuus, väri ja hinta. Samalla oven komponenttien välille voidaan määritellä yhteydet. Esimerkiksi se, miten ovi asettuu suhteessa ympäristöön, kuten seinään, voidaan määritellä tietomalliin. Tällöin tietomalliin voidaan määritellä esimerkiksi se,

että ovea ei voi piirtää tietyssä tapauksessa seinään lainkaan (esimerkiksi kaksi ovea eivät voi olla liian lähekkäin).

Tietomallin rakenne määrää, mitä sen avulla voidaan mallintaa ja mihin kaikkeen sitä voidaan käyttää. Esimerkiksi rakennuksen tietomallilla ei voi mallintaa autoa: vaikka molemmissa on ikkunoita, eivät rakennuksen tietomallin ikkunat sovellu autossa käytettäviksi. Piirustuksen tietomallilla taas voi mallintaa kuvan talosta, autosta tai vaikka maisemasta, mutta ei taloa, autoa tai maisemaa itseään. (Hietanen, 2005)

Tietomallit erottavat tiedon (datan) ja tiedon esitystavan toisistaan. Tämä tapahtuu automaattisesti, koska tietokoneen muistissa olevalla datalla ei ole mitään omaa esitystapaa. Lähin vastaava ominaisuus löytyy ihmisen ajatuksesta. Sama abstrakti ajatus voidaan ilmaista monella eri tavalla: esimerkiksi puhumalla, kirjoittamalla, piirtämällä ja elehtimällä. Itse ajatusta ei kuitenkaan voi koskaan nähdä. Kykyä erottaa tieto ja esitys toisistaan ei ole millään muulla tekniikalla: esimerkiksi paperille piirrettäessä tieto ja esitys on aina kytketty erottamattomasti toisiinsa. Käytännössä tämä tarkoittaa sitä, että tietomallissa olevalla tiedolla voi olla rajoittamaton määrä eri esityksiä eli näkymiä. (Hietanen, 2005)

Monesti rakennusalalla puhutaan tuotemallista; tuotemalli eli tuotetietomalli (product model, product data model) kuvaa tuotteen (eli rakennuksen) rakenteen ja sisältää sen tuottamiseen (suunnitteluun ja rakentamiseen) sekä käyttämiseen tarvittavan tiedon. Viime aikoina käytössä on yleistynyt englanninkielinen termi building information model (BIM), mikä kuvaa hyvin sen, että tuotemalli on nimenomaan tietojen malli. (Penttilä et al., 2006)

Tuotemallintamisessa lähdetään siitä, että CAD-järjestelmässä kuvataan tuotteen 3D-malli, joka voidaan tallentaa esimerkiksi IFC-tietomallin mukaan oliopohjaisesti tiedostoksi. Eri olioita rakennushankkeessa voivat olla esimerkiksi ovi, ikkuna, katto ja lattia. Koska olioille voidaan antaa ominaisuuksia, mahdollistaa se myös esimerkiksi sen, että rakennushankkeen objektille kuten ovelle tai ikkunalle voidaan antaa hinta tai toteutusaikataulu (esimerkiksi ominaisuudet: asennus alkaa, asennus loppuu).

Tuotemallintaminen eroaa kolmiulotteisesta (3D) mallintamisesta siten, että CAD-ohjelmilla esitetyn rakennuksen kolmiulotteisen muodon kuvauksen lisäksi tuotemalliin liittyy myös rakennuksen osien ja niihin liittyvien tietojen kuvaus. Visuaalisesti

tuotemalli ilmenee yleensä kolmiulotteisena suunnitelmana, jossa rakenteet kuvataan viivojen sijaan kolmiulotteisina kappaleina, tuoterakenteina. (Penttilä et al., 2006.)

Tuotemallintamisen perustutkimusta on tehty Suomessa jo 1980-luvun lopulta. Tuotemallintamiseen perustuvia toiminta-, menettely- ja tiedonhallintatapoja on alettu ottaa käyttöön useissa yksittäisissä rakennushankkeissa 2000-luvun alusta. Yksittäisistä pilottihankkeista ollaan tällä hetkellä siirtymässä kohti uusia toimintatapoja, joiden vaikutukset ulottuvat koko rakennusalaan. (Penttilä et al., 2006)

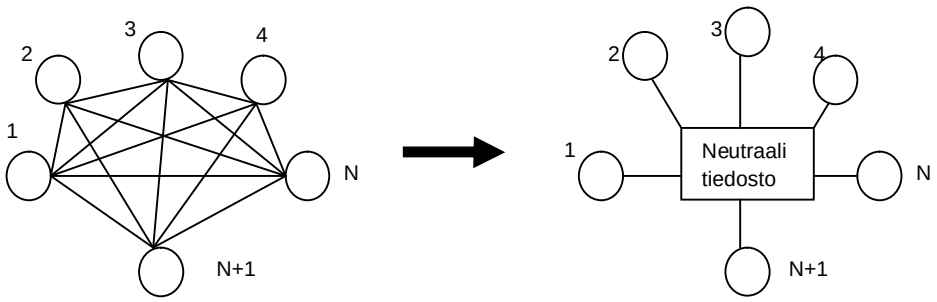
2.3 Tiedon siirron kehittyminen CAD järjestelmissä

Jotta tietoa voidaan siirtää rakenteesta (tai siten tallennusrakenteesta) toiseen, tarvitaan välille joku malli, jonka mukaisesti tieto luetaan tai tallennetaan. Avuksi tiedon lukemiseen, siirtämiseen ja tallentamiseen on kehitetty erilaisia standardisoituja tiedonsiirtotapoja.

Jo 70 – luvulla kehitettiin CAD-järjestelmien tiedonsiirtoa, koska asiakkaat vaativat, että tietoa pitäisi pystyä siirtämään ulkopuolelta projektitiedoston. Päädyttiin erilaisiin ”matalan tason” menetelmiin. Eräs menetelmä on, että kirjoitetaan osa tiedosta tekstimuotoon ja vastaanottavan sovelluksen puolelta vastaavasti jäsennetään syntynyt tekstitiedosto, suoritetaan tekstitiedostossa olleet komennot vastaanottavassa järjestelmässä ja luodaan siten objektit, kun järjestelmä on ylhäällä. Toinen menetelmä on, että kirjoitetaan oma sovelluksensa projektitiedoston lukuun, tulkitsemiseen ja tiedon kopiointiin sekä datasta muodostuneen tekstitiedoston luomiseen. Vastaanottavan puolen sovelluksella pitää olla vastaavasti tulkitseva sovellus. Kolmas menetelmä on, että rakennetaan ”alatason kutsuja”, joita voidaan suorittaa toisesta sovelluksesta ja siten purkaa tietoa (tiedostosta, jossa koko projektin tieto on) sekä vastaavasti rakentaa objektit kyseisen tiedon mukaisesti. Neljäs menetelmä on, että mahdollistetaan ”alatason kutsut” CAD-sovelluksesta sallimalla tiedon kirjoitus halutussa muodossa tai vaihtoehdoissa. Viides menetelmä on, että lisätään sovellus CAD-järjestelmän ympäristöön ja kutsutaan sitä CAD-järjestelmästä siten, että molempia järjestelmiä suoritetaan yhtä aikaa. Nykyään voidaan käyttää kaikkia edellä mainittuja tapoja. (Eastman, 1999)

2.3.1 Neutraalin tiedoston tiedonsiirto

Tärkeä vaihe sovellusten välisen tiedonsiirron kehityksessä oli CAD:n suhteen se, kun kehitettiin ”neutraalin tiedoston” tiedonsiirto. Neutraalin tiedoston tarve tuli esiin moninkertaisissa ongelmissa, kun haluttiin kehittää useimpien teknisten sovellusten välille tiedonsiirtoa. Jos oletetaan, että sovelluksien A ja B välillä halutaan siirtää tietoa molemmissa suunnissa, joudutaan kehittämään tiedonmuuntajat (translators) molempiin suuntiin. Jos oletetaan, että sovelluksia on kolme (A, B ja C) ja kaikkien sovellusten välille halutaan rakentaa tiedonsiirtomahdollisuus, tarvitaan tiedonmuuntajia kuusi kappaletta (A-B, A-C, B-A, B-C, C-A, C-B). Jos sovelluksia on neljä, tarvitaan kaksitoista tiedonmuuntajaa. Eli, jotta voidaan lisätä yksi sovellus, tarvitaan 2 x N muuntajaa, missä N on olemassa olevien sovellusten määrä (kuva 2a). Tämä tekee tiedonsiirron rakentamisen kalliiksi, jos tiedonsiirto on rakennettava monen sovelluksen välille. Jotta tietoa voitaisiin siirtää monen sovelluksen välillä ilman tätä sovellusten määrän suhteen kertautuvaa ongelmaa, alettiin kehittää yhteistä tietoformaattia eli ”neutraalia tiedostoa” (kuva 2b). Neutraalissa tiedostossa tietorakenteet on toteutettu yhteisesti sovitulla tavalla. (Eastman, 1999)



Kuva 2. ”Normaali” tapa tehdä tiedonsiirto sovelluksien välille tiedonmuuntajien (translators) avulla (a), ja neutraalin tiedoston avulla tehty tiedonsiirto (b) (Eastman, 1999)

Tärkeimpiä neutraalin tiedoston toteutuksia ovat 1980 – luvun lopulla kehitetty DXF-tiedosto (Data Interchange Format) ja 1970 – luvun lopulla kehitetty IGES (Initial Graphics Exchange Specification). DXF:n kehitti AutoCad ohjelmistonkin kehittänyt AutoDesk. Siitä tehtiin toteutukset sekä binäärisenä että tekstitiedostona. Nykyisin DXF

on varsin suosittu tiedonsiirtoformaatti varsinkin 2D -kuvissa. 3D – kuvissa sen mahdollisuudet ovat rajalliset. (Eastman, 1999)

NASA, General Electric ja Boeing kehittivät yhteisen tekstimuotoisen tiedonsiirtotiedoston IGES:n. Tiedostomuodosta tuli myöhemmin standardi NIST:n (National Institute for Standards and Technology) toimesta. Sen jälkeen kun alkuperäinen versio IGES:stä julkaistiin, julkaistiin vielä neljä muuta versiota, joissa oli laajennuksia sekä 2D että 3D – kuvia varten. Niissä oli mahdollista muun muassa tallentaa pintoja (surfaces), tilavuusmallinnusobjekteja (solids) ja ei graafisia attribuutteja. Muunto CAD järjestelmästä IGES:n neutraaliin tiedostomuotoon suoritetaan niin sanotun esiprosessorin (preprocessor) avulla ja muunto CAD – järjestelmään IGES – muodosta suoritetaan niin sanotun jälkiprosessorin (postprocessor) avulla. Oleellinen osa ohjelmiston toimittajilla on dokumentoida tiedon siirtoa varten sekä esiprosessointi- että jälkiprosessointikartat (correspondence entity maps), joissa on kuvattu se, mikä IGES – tiedostossa kuvattu tietorakenne (entiteeteistä=objekteista kuten esimerkiksi objekti piste, viiva tiedonsiirrossa) vastaa ohjelmiston omaa. On myös kehitetty joitain työkaluja, joilla voidaan jäsentää ja testata IGES-tiedoston syntaktista rakennetta. (Eastman, 1999)

Neutraalin tiedoston haittapuolina on kuitenkin muun muassa se, että rakennusalalla on erilaisia toimintatapoja järjestää piirustusdokumentit yhden projektin sisällä. Voi olla niin, että eri osapuolet käyttävät erilaisia näkymiä dataan ja siten käyttävät omia sovellettuja teknisiä piirustuksen ominaisuuksia (kuten tasoja, koordinaatistoja ja omia blokkeja). Tallennusmedia (ennen erilaiset levykkeet; nyt muun muassa CD-ROM, DVD, USB-tikku) ja sen koko on ainakin ennen aiheuttanut ongelmia siirrettäessä dataa neutraalin tiedoston kautta. Toiminnallisesti eri tavalla toimivat CAD – järjestelmät aiheuttavat myös ongelmia. Siten myös tiedonsiirron vuoksi tarvittavan tiedon koko vaihtelee järjestelmäkohtaisesti. Tiedon siirrossa saattaa tulla tietotyyppien konversion vuoksia epätarkkuutta (esimerkiksi pyöristyksiä tietotyypistä toiseen). CAD -järjestelmien toimittajat eivät ole myöskään katsoneet suopein mielin sitä, että kuva saatetaan piirtää ensin toisella järjestelmällä ja sitten siirtää toiseen. Tiedon määrä on kasvanut koko ajan ja se aiheuttaa omalta osaltaan ongelmia neutraalille tiedoston kautta tehdylle tiedon siirrolle. IGES-spesifikaatiot on kuvattu sanoin. Monet nykyiset ohjelmointikielet on kuvailtu enemmän muodollisesti, käyttäen erilaisia syntakseja

(esimerkiksi UML-kuvausmenetelmä nykyisin). Sen tapaisella yleisen syntaksin lähestymistavalla saavutetaan etuja verrattuna edellä mainittuun sanalliseen kuvaukseen. Tietorakenteet eri järjestelmien toimittajilla ovat erittäin monimutkaisia ja se aiheuttaa ongelmia siirrettäessä tietoa yhteiseen neutraalin tiedoston muotoon. CAD-järjestelmien toimittajat eivät katso suopein silmin sitä, että muutettaessa tieto neutraaliin tiedoston muotoon, karsii heidän ohjelmiston ominaisuuksia merkittävästi. Lisäksi monet järjestelmien toimittajat eivät ole halunneet, että kaikki IGES:n mahdollistavat toiminnot rakennettaisiin tiedonsiirtoa varten. Muunnokset eri tietorakenteesta toisiin tulivat erittäin monimutkaisiksi. Eri standardien kehitys oli lisäksi liian hidasta, että tiedonsiirto olisi ohjelmien kehityksen myötä saatu nopeasti tehokkaaseen käyttöön. (Eastman, 1999)

Tietorakenteiden kehittyessä ja muun muassa oliopohjaisten oliokielten synnyn vuoksi tallennettavat tietorakenteetkin ovat muuttuneet oliopohjaisiksi. Nykyiset oliopohjaiset kielet mahdollistavat muun muassa luokkien välisen perinnän ja siten tietoa voidaan abstrahoida kantaluokissa yleisemmällä tasolla. Oliopohjaisissa kielissä periytymisessä on sellaisia käsitteitä kuten erikoistaminen, joka on suhde kahden entiteettiluokan välillä. Edellä mainittu pätee myös luokkien instansseihin. Toiseen suuntaan olevaa suhdetta sanotaan yleistämiseksi. Koostamiseksi (aggregation) kutsutaan käsitettä, jossa yksittäisillä entiteeteillä voidaan kuvata jokin entiteetti. Esimerkki koosteesta voi olla: ”eläin, jolla on pitkät karvat ja joka haukkuu”, jota voidaan kutsua myös koiraksi. Attribuutteja (luokan ominaisuuksia, jotka voivat olla entiteettejä) käytetään kuvaamaan jonkin muun entiteetin ominaisuuksia (attribuutit ovat osa entiteettiä). Yhdessä olevat attribuutit määrittävät niin sanotun viittausentiteetin (entiteetti, jossa niitä käytetään) tilan. Koostumiseksi (composition) kutsutaan rakennetta, jossa attribuutit ja koosteobjekti voivat olla riippumattomia toisistaan, päinvastoin kuin erikoistamisessa (missä ne on peritty). Osaluokka saattaa myös olla koosteobjekti, josta aiheutuu näin monitasoinen kooste. (mukaillen Eastman, 1999)

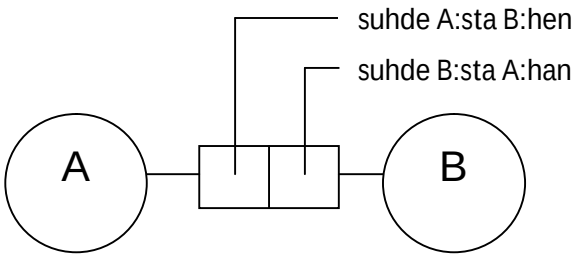
Suhteeksi (relation) kutsutaan assosiaatiota yhden objektin instanssista toiseen tai useampaan instanssiin. On olemassa muitakin suhteita kuin erikoistaminen, koostaminen ja koostuminen. Edellä mainitut suhteet ovat yleisiä erilaisissa sovelluksissa. Kuitenkin on olemassa muita tyyppejä ja suhteita, jotka ovat aihealue- tai sovelluskohtaisia ja joita tarvitaan esimerkiksi määriteltäessä rakennusalan tietomallia

(esimerkiksi suhde kahden tehtävän välillä). Tarve muiden suhteiden määrittelemiseksi instanssien välille on tuottanut joukon erilaisia suhdeasteita (käsite ”arity” tai ”cardinality”). Yksinkertaistettuna suhteet voidaan jakaa: yhden suhde yhteen (merkitään esimerkiksi 1:1), yhden suhde kahteen (1:2), yhden suhde moneen (1:m) ja monen suhde moneen (m:n) suhteisiin. (mukaillen Eastman, 1999)

Oliopohjaiset ohjelmointikielet (kuten C++) antavat hyvän lähtökohdan suunnitella ohjelmiston arkkitehtuuria edellä mainittujen erikoistamisen, koostamisen ja koostumisen kautta, mutta niissä ei ole vielä kunnolla ollut mahdollisuutta määritellä olioiden välisiä suhteita (mukaillen Eastman 1999). Vasta Java ja .NET -oliokielten (kuten C#) kautta siihen on päästy, esimerkiksi siten, että kuvataan XML-kielen avulla olioiden välisiä suhteita.

2.3.2 Tiedon käsitteellistäminen graafisen kuvauskielen avulla

Relaatiotietokannoissa on käytetty (ja käytetään edelleen) skeeman kuvaukseen niin sanottua ER-kaaviota (Entity Relationship model). Siinä entiteetit kuvataan taulukoissa, suhteet salmiakin muotoisissa kuvioissa ja attribuutit ellipseissä. ER-kaavion avulla relaatiotietokannan käsitteellinen suunnittelu on helpottunut. CAD-maailmaan ER-kaavio on kuitenkin katsottu olevan liian rajoittunut kuvausmenetelmä. Kehitettiin NIAM (Nijssen’s Information Analysis Method). Tohtori Nijssen kehitti NIAM:n 1977 tietokannan kuvausmenetelmäksi, tukien tiedonsiirtoa käyttäjän ja tietokoneen välillä käyttämällä alkeislauseita. Hänellä oli mielessään relaatiotietokannat ja hän käsitteellisti NIAM:n tarkoituksenaan muuntaa verbaalinen informaatio tietokantaskeemaksi. NIAM:n pyrkimyksenä on se, että semantiikan kuvaus voidaan tehdä tietokantariippumattomana. NIAM:ssa kuvataan entiteetit ympyröillä ja entiteettien väliset suhteet laatikoissa, kuten kuvassa 3. Alityypin ja ylätyypin välinen suhde merkitään nuolella. Lisäksi voidaan käyttää muun muassa erilaisia rooleja ja yksilöiviä pakotteita. (mukaillen Eastman, 1999)



Kuva 3. NIAM:n perussyntaksi kuvattaessa suhdetta kahden entiteetin välillä (mukaillen Eastman, 1999)

NIAM:n vahvuutena on katsottu olevan se, että se ei tee eroa suhteen (määritelty roolina) ja attribuutin välille. NIAM:lla voidaan myös määritellä monipuolisia rajoitteita ja kelpoisuussääntöjä joita tarvitaan relaatiotietokannoissa. Kuitenkin tietomallin määrittelyssä NIAM:sta on löytynyt muutamia rajoitteita. Sillä ei voida esittää useita tärkeitä rakenteita, jotka puuttuvat myös relaatiomallista. (Eastman, 1999)

2.4 Kohti rakenteellista tiedonsiirtoa

Neutraalin tiedoston tiedonsiirtomenetelmissä kuten IGES, oli tiettyjä heikkouksia, joita ei pystytty korjaamaan helposti. Tiedonsiirrolle asetettiin uusia tavoitteita: pyrittiin sisällyttämään uusien olio-ohjelmointikielien ominaisuuksia. Pyrittiin myös sisällyttämään muodollisia yhteisiä sääntöjä rakenteisiin käyttämällä uusia vastakehitettyjä tietomallin kuvaavia kieliä kuten UML (Unified Modeling Language) ja EDM (Entity Data Model). Tarkoitus oli eristää tietomalli fyysisestä tiedostomuodosta sekä tukea koko tietomallin aliryhmiä; mahdollistaen siten eri ryhmien sovellusten integroitua osaan mallia ilman, että tarvitsee toteuttaa koko tietomallia. Pyrkimyksenä oli tukea vaihtoehtoisia fyysisen tason toteutuksia mukaan lukien tiedostot, tietokannat ja tietämyspohjaiset järjestelmät. Lisäksi haluttiin sisällyttää erilaisia muita referenssimalleja yhteiseen malliin, jotka olisivat jaettuja osajoukkoja suuremmista standardoiduista malleista. (Eastman, 1999)

Deklaratiivista tietoa voidaan määritellä yksinkertaisesti sanalla *mitä*, proseduraalista tietoa voidaan määritellä sanalla *miten* ja rakenteellista tietoa (structural knowledge) voidaan kuvata sanalla *miksi* (Jonassen et al., 1993). Tietojärjestelmiä kehitettäessä ei

enää riitä, että järjestelmissä on toteutettu kysymykset: mitä ja miten, vaan nyt järjestelmien pitäisi osata jo itse vastata enemmän kysymykseen miksi. Enää ei riitä normaali ”konemainen” suoritus, vaan esimerkiksi eri järjestelmien on ymmärrettävä itsenäisesti enemmän toisistaan. Esimerkiksi sähköisen liiketoiminnan sovelluksien, varsinkin monikansallisella tasolla, pitäisi pystyä käsittelemään monenlaista tietoa ja mieluiten automaattisesti. Sovellusten ”merkityksen” rakentamiseen, semantiikalla voidaan luoda ymmärrys, esimerkiksi siitä, millainen merkitys jollain CAD-järjestelmän graafisella objektilla on toiselle sovellukselle kuten esimerkiksi kustannuslaskentaohjelmistolle. Semanttinen yhteensopivuus (semantic interoperability) on ratkaiseva elementti siinä, että voidaan tehdä esimerkiksi rakentamisen tietomallit ymmärrettäväksi ja mallitieto jaettavaksi läpi usean suunnittelualan ja heterogeenisen tietokonejärjestelmän. (Yanga & Zhang, 2006)

3 TIETOMALLIT TIEDONSIIRROSSA

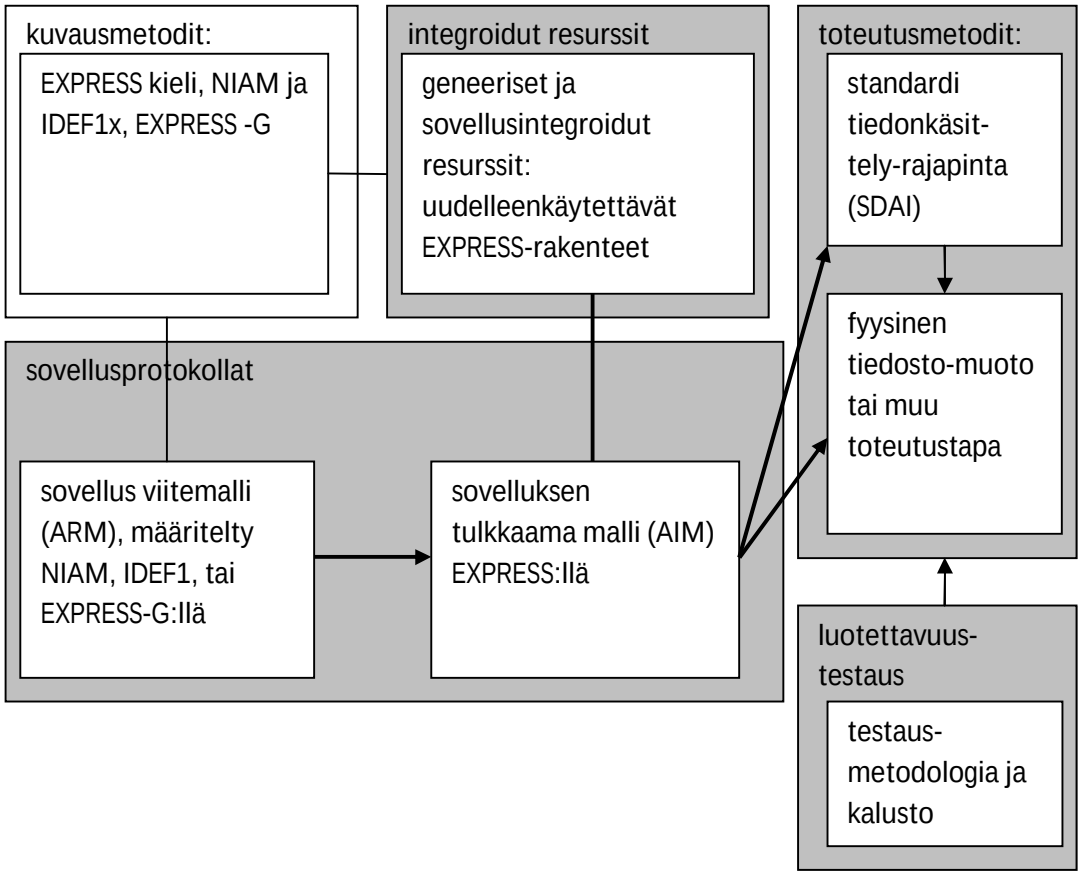
Tämä luku on ehkäpä tärkein työni tiedonsiirtoa käsittelevä luku. Tässä luvussa edetään ISO-STEP-tiedonsiirtostandardista työni tärkeimpään osuuteen eli IFC-standardin mukaiseen tiedonsiirtoon, joka on myös rakennettu STEP-standardin kuvausmenetelmin. Lisäksi siirryn luvun lopussa XML-muotoiseen tiedonsiirtoon, koska se on nykyisin käytetyin tiedonsiirron muoto.

Kuten jo aiemmin on tullut esille, jo varsin aikaisessa vaiheessa CAD:n historiassa huomattiin, että tietoa tulisi voida siirtää järjestelmästä riippumatta. 1970-luvun lopulla Yhdysvalloissa kehitettiin IGES (Initial Graphics Exchange Specification). Siitä tuli ANSI-standardi. Ongelmana IGES-standardissa oli kuitenkin se, että siinä kuvattiin graafiset elementit CAD-ohjelmistolle sopivaksi. Jos tarvittiin tiedonsiirtoa, esimerkiksi CAD – ohjelmistosta kustannuslaskentaohjelmistoon, IGES ei ollut sopiva formaatti. IGES oli kehitelty pikemminkin graafiseen kuvaukseen. Tämän jälkeen kehitettiin monia muita tiedonsiirtostandardeja kuten: SET (Standard D'Exchange et de Transfert), VDA-FS (Verband Der Automobilindustrie-Flachen-Schnittstella), DXF (Data Exchange Format). Yhdysvalloissa alettiin kehittää tiedonsiirtoformaattia PDES:ää (Product Data Exchange Standard). Tietomallin käsittämän datan siirtoon alettiin kehittää kuitenkin lopulta kansainvälistä STEP:iä.

3.1 ISO-STEP

STEP (STandard for the Exchange of Product model data) -tietomallit ovat alunperin kehittyneet ANSI-SPARC:n (American National Standards Insitute committee on database architectures and standards) työn pohjalta. Tätä työtä jatkettiin kansainvälisellä tasolla ja päädyttiin jakamaan arkkitehtuuri kolmeen eri tasoon: fyysiseen (physical), loogiseen (logical) ja sovellustasoon (application). Fyysisellä tasolla määritellään skeeman tietorakenteet tallennusmediassa. Loogisella tasolla määritellään looginen rakenne ja tiedon suhteet toteutusriippumattomalla tavalla. Lisäksi looginen taso pitää yhdistää (mapata) fyysiseen tasoon. Sovellustasolla esitetään se tieto tietomallista sovellettuna sovellukseen, joka on oleellista sovellukselle (tästä käytetään myös käsitettä näkymä=view). (Eastman, 1999)

Kuvan 4 mukainen STEP:n arkkitehtuuri koostuu eri osista, joita yhdistävät kuvassa viivat. Ohuet viivat kuvaavat kielien käyttöä ja paksut nuolet kuvaavat muuntamista, joka toteutetaan muuntimella (translator). Tiedonsiirtojärjestelmä kuvataan erilaisilla kuvausmenetelmillä, joka riippuu siitä, mitä tietomallia käytetään. Kielinä voivat olla NIAM, IDEF1x (kehitettiin aikanaan USA:n puolustuksen suunnittelua varten), EXPRESS ja EXPRESS –G kielet. Integroidut resurssit ovat yhteinen malli alijoukoista joita käytetään toistuvasti sovellusmallin määrittelyssä. Mallit, joita käytetään eri sovellusalueilla ovat geneerisiä integroituja resursseja ja ne sisältävät geometrian, materiaaliominaisuuksia ja projektiluokituksia. Teollisuuden alakohtaiset mallit ovat sovellusintegroituja resursseja. Ne voivat olla esimerkiksi elektroniikka ja rakentaminen. Sovellusprotokollat (application protocols=APs) on kehitetty erikoiseen sovelluskontekstiin käyttäen kuvausmetodeja ja integroituja resursseja. Sovellusprotokollat jaetaan kahteen näkökulmaan: sovellus viitemalliin (ARM=Application Reference Model) ja sovelluksen tulkkaamaan malliin (AIM=Application Interpreted Model). ARM kuvaa sovelluksen vaatimukset siten, että ne ovat helposti ymmärrettävissä suunnitellessa ja arvioitaessa tietomallia. ARM tulkataan AIM:ksi, jota voivat lukea sekä ihmiset että tietokoneet. AIM:ssä päätetään kaikesta geneeristen resurssien käytöstä ja yhdistetään malli sovellusintegroituihin resursseihin. Sovellusprotokolla yhdistetään toteutusmetodilla (implementation method) ja se muodostaa perustan STEP-toteutukselle. Toteutusmetodi sisältää tyypillisesti monia resursseja. STEP:n fyysinen tiedosto (SPF) ja rajapinta SPF:ään (SDAI=Standard Data Access Interface) ovat eräitä toteutusmetodeja. Lopuksi jokainen STEP sovellusprotokolla ja toteutus tarvitsevat luotettavuustestausta. Luotettavuustestaus arvioi ARM:n ja AIM:n toteutusta ja varmistaa, että STEP-kielet ja -välineet on käytetty ja tulkittu oikein. Luotettavuustestausmetodologiaa toteutetaan akkreditoiduissa organisaatioissa. Luotettavuustestausmetodologia spesifioi prosessin, johon sovellusprotokolla hyväksytään. (mukaillen Eastman, 1999)



Kuva 4. STEP tietomallin arkkitehtuuri.

STEP:ssä tietomalli kuvataan EXPRESS-kielen avulla. Tiedostomuodon tarkentimena on .exp. Lisäksi käytetään graafista EXPRESS –G -kuvausmenetelmää. Tietomallikuvausta sanotaan skeemaksi. Itse oliot eli ilmentymät, jotka noudattavat skeemaa, kuvataan STEP-standardissa P21-tiedostolla. Tietoa voidaan siirtää sovellusten välillä pelkällä P21-tiedostolla tai yksi tapa pitää tietovarastoa eri sovellusten välillä yllä, on toteuttaa se relaatiotietokannan avulla (parempi kuin pelkkä tiedosto). STEP API –kutsuja varten kehitettiin SDAI (Standard Data Access Interface).

3.1.1 ISO-STEP ja sovellusten ohjelmointikieli

Jotta tietomallin mukainen tieto voidaan siirtää ohjelmointikielen vastaaviin rakenteisiin, on mahdollista käyttää kahta erilaista ”sitomista”. Myöhäisessä sitomisessa (late binding) käytetään vain yhtä tietorakennetta kaikkiin EXPRESS-mallin määritelmiin. Entiteettien sisäiset attribuutit kerätään yleensä johonkin yhteiseen luokkaan, joka voi olla vektori (Vector) tai lista (List). Pääsy objekteihin

mahdollistetaan ajonaikaisesti. Aikaisessa sitomisessa (early binding) EXPRESS-tietomalli tehdään saataville jonkun ohjelmointikielen tietorakenteiden kautta. Tällöin jokainen EXPRESS-mallissa määritelty entiteetti kirjoitetaan käytettävässä ohjelmointikielessä luokaksi. (mukaillen Eastman, 1999)

Nour & Beucke:n artikkelin mukaisesti on esitetty vielä kolmas sidontatapa: sekoitettu sidonta (mixed binding). Siinä ajatellaan niin, että sovellukset harvoin käyttävät kaikkia rakenteita, joita skeemassa on määritelty. Välittömästi tarvittavalle osalle entiteeteistä voidaan määritellä aikainen sidonta ja lopuille entiteeteille myöhäinen sidonta. (mukaillen Nour & Beucke, 2005)

3.1.2 EXPRESS

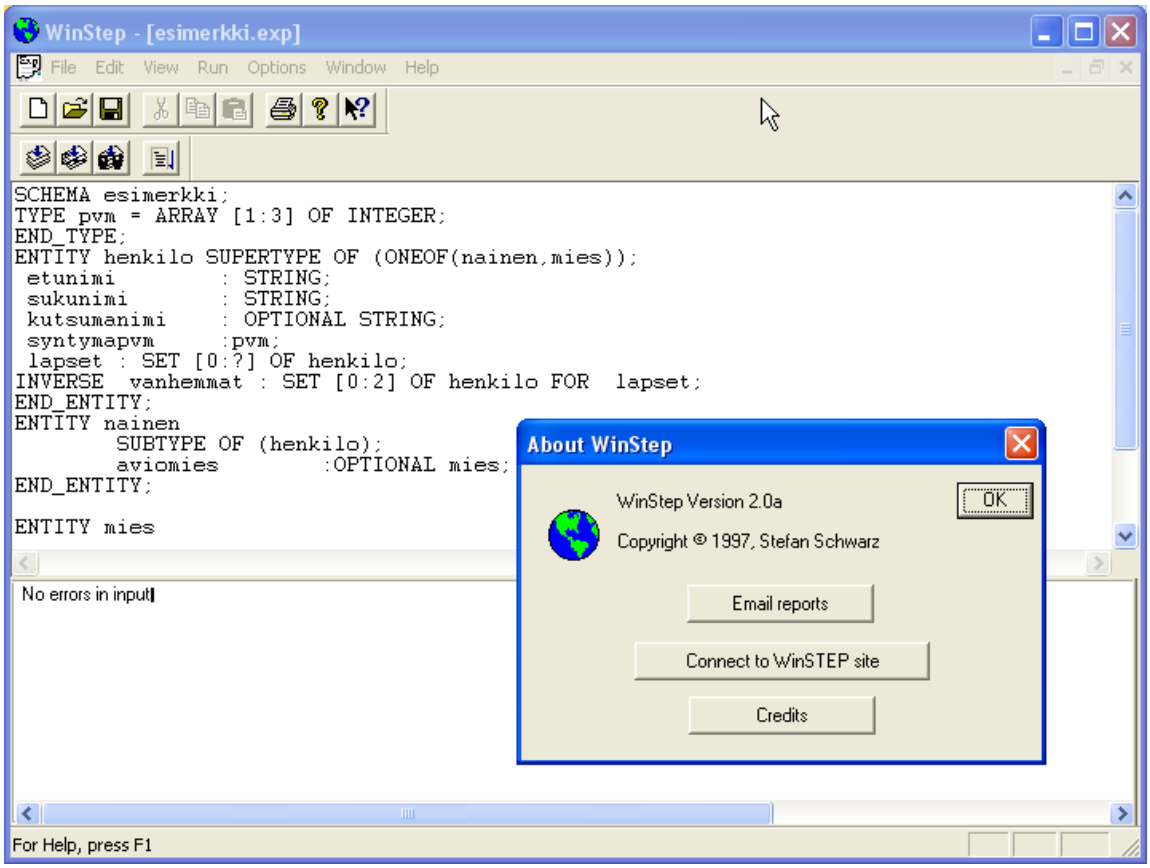
EXPRESS on samantyyppinen proseduraalinen kieli kuten Pascal, sillä erotuksella, että siinä voidaan esittää eri objektien väliset suhteet. EXPRESS on kehitetty nimenomaan tietomallin ilmaisuun ja on siinä mielessä erittäin ilmaisuvoimainen. EXPRESS:iä ei kuitenkaan pidetä varsinaisena ohjelmointikielenä vaan se on kehitetty sitä varten, että sillä voidaan kuvata tietomalli. EXPRESS:n syntaksia voi tutkia liitteestä I.

Esimerkki skeemasta, joka on luotu EXPRESS kielellä (mukaillen Eastman, 1999):

```
SCHEMA esimerkki;
TYPE pvm = ARRAY [1:3] OF INTEGER;
END_TYPE;
ENTITY henkilo SUPERTYPE OF (ONEOF(nainen,mies));
        etunimi       : STRING;
        sukunimi      : STRING;
        kutsumanimi : OPTIONAL STRING;
        syntymapvm : pvm;
        lapset        : SET [0:?] OF henkilo;
        INVERSE  vanhemmat : SET [0:2] OF henkilo FOR  lapset;
END_ENTITY;
ENTITY nainen
        SUBTYPE OF (henkilo);
        aviomies      :OPTIONAL mies;
END_ENTITY;
```

```
ENTITY mies
                SUBTYPE OF (henkilo);
                vaimo           :OPTIONAL nainen;
END_ENTITY;
END_SCHEMA;
```

Edellisen skeeman syntaksin tarkistus voidaan tehdä monilla eri kaupallisilla ohjelmilla. Seuraavassa kuvassa 5 on esimerkki WinStep:llä tehdystä EXPRESS:n oikeellisuuden tarkistuksesta.



Kuva 5. EXPRESS–muotoisen skeemaan oikeellisuuden tarkistus

EXPRESS on kieli, joka kehitettiin kuvaamaan STEP standardin mukaisesti niin sanottuun AIM:ään (Application Interpreted Model). EXPRESS:n kehitti Douglas Schenk ja sitä on kehitelty eteenpäin vuodesta 1986 lähtien. (Eastman, 1999)

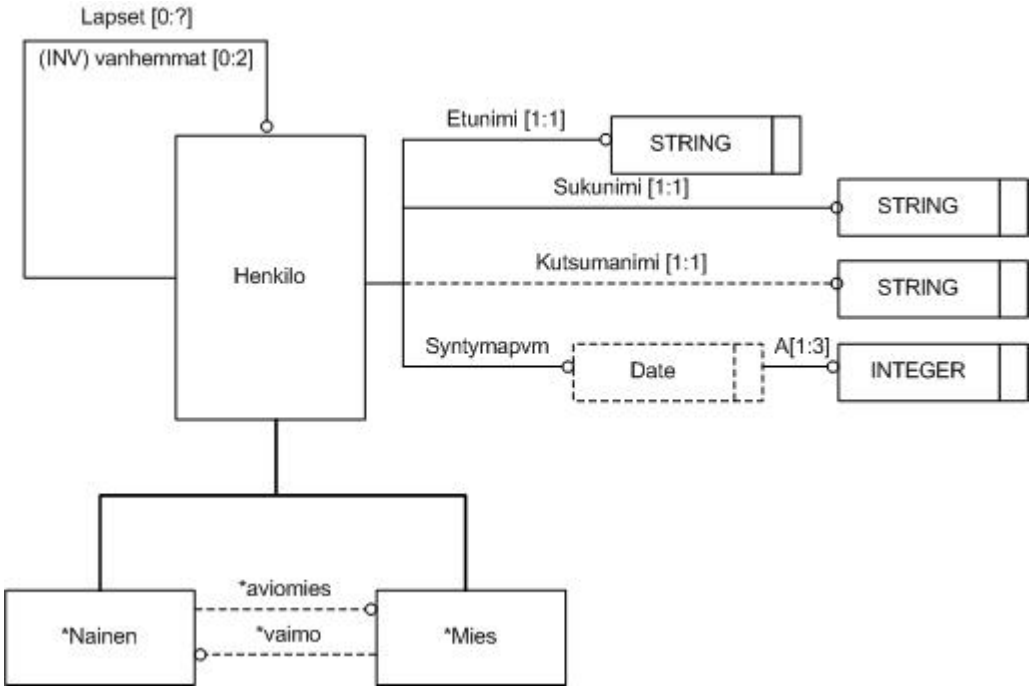
EXPRESS-kieli on saavuttanut jalansijan tiedonsiirrossa kahden sovelluksen välillä tiedostotyyppisen tiedonsiirron tapauksessa. Jos tieto tallennetaan tietokantaan, EXPRESS-kielessä on joitain puutteita. Siitä puuttuu mahdollisuus käyttäjän valita jokin osajoukko tietueista, jota halutaan käsitellä sovelluksessa. EXPRESS-kielestä puuttuu myös kyky tarkistaa tietomallin yhtäpitävyys ajoittain, kun tehdään muutoksia. Kyky samanaikaisuuden hallintaan puuttuu (kun mallia päivitetään monelta taholta samaan aikaan). Kyky toteuttaa suurempia malleja (joissa on osallisena useampia sovelluksia) puuttuu. Edellä mainittu puute on siinä mielessä huono rakennusalalla, koska usein siellä tarvetta sellaiseen on. Kaikki edellä mainitut puutteet ovat tulleet esille, kun on alettu kehittämään varsinaisesta siirrettävästä tiedosta erillistä tietomallia. EXPRESS-kielessä ei voida myöskään suorittaa niitä funktioita, sääntöjä ja suhteita, jotka mallissa on määritelty. Arkkitehtuurin mukaisesti mallin toteuttavan muuntajan on otettava ne huomioon sovelluksessa käytetyssä varsinaisessa ohjelmointikielessä, mutta EXPRESS-kieltä ei siinä mielessä voida suorittaa. Jotkin SDAI (Standard Data Access Interface)-kirjastot mahdollistavat funktioiden, sääntöjen ja WHERE sekä DERIVE -lauseiden suorituksen. (Eastman, 1999)

3.1.3 EXPRESS –G

EXPRESS –G on kehitetty, jotta tietomalli voidaan ilmaista graafisesti. Se, mikä on kirjoitettu EXPRESS:llä, voidaan kuvata EXPRESS –G:llä, mutta sitä kaikkea mitä on kuvattu EXPRESS –G:llä ei voida kirjoittaa EXPRESS:llä. Usein tietomallia ei pystytä esittämään sellaisella yleisellä tietokantojen kuvausmenetelmällä, kuten ER–kaaviolla. Tällöin EXPRESS –G jääkin usein ainoaksi mahdollisuudeksi. EXPRESS –G:tä käytetään useasti nimenomaan ARM:n (Application Reference Model:n) kuvaamiseen.

EXPRESS –G:ssä kaikki muut tyypit, paitsi perustyypit (Real, Integer, Boolean, String, Logical), on kuvattu katkoviivalla piirretyillä laatikoilla. Perustyypit kuvataan jatkuvalla viivalla piirretyllä laatikolla. Laatikko, jossa on ympyrä laidalla, kuvaa referenssiä tyyppiin samassa skeemassa tai kuvassa. Suhteet kuvataan viivoilla, jotka yhdistävät laatikoita. Yli- ja alityyppien suhteet kuvataan paksuilla viivoilla, kun taas muut suhteet kuvataan ohuilla viivoilla. EXPRESS –G:n MS-Vision mukaisen syntaksin merkinnät ovat koostettuna liitteessä II.

Edellä kuvatussa luvussa EXPRESS:llä kuvattu Henkilo–entiteetti voitaisiin kuvata kuvan 6 mukaan seuraavasti:



Kuva 6. Henkilo–entiteetti kuvattuna EXPRESS –G:llä (mukaillen Eastman, 1999)

3.2 IFC

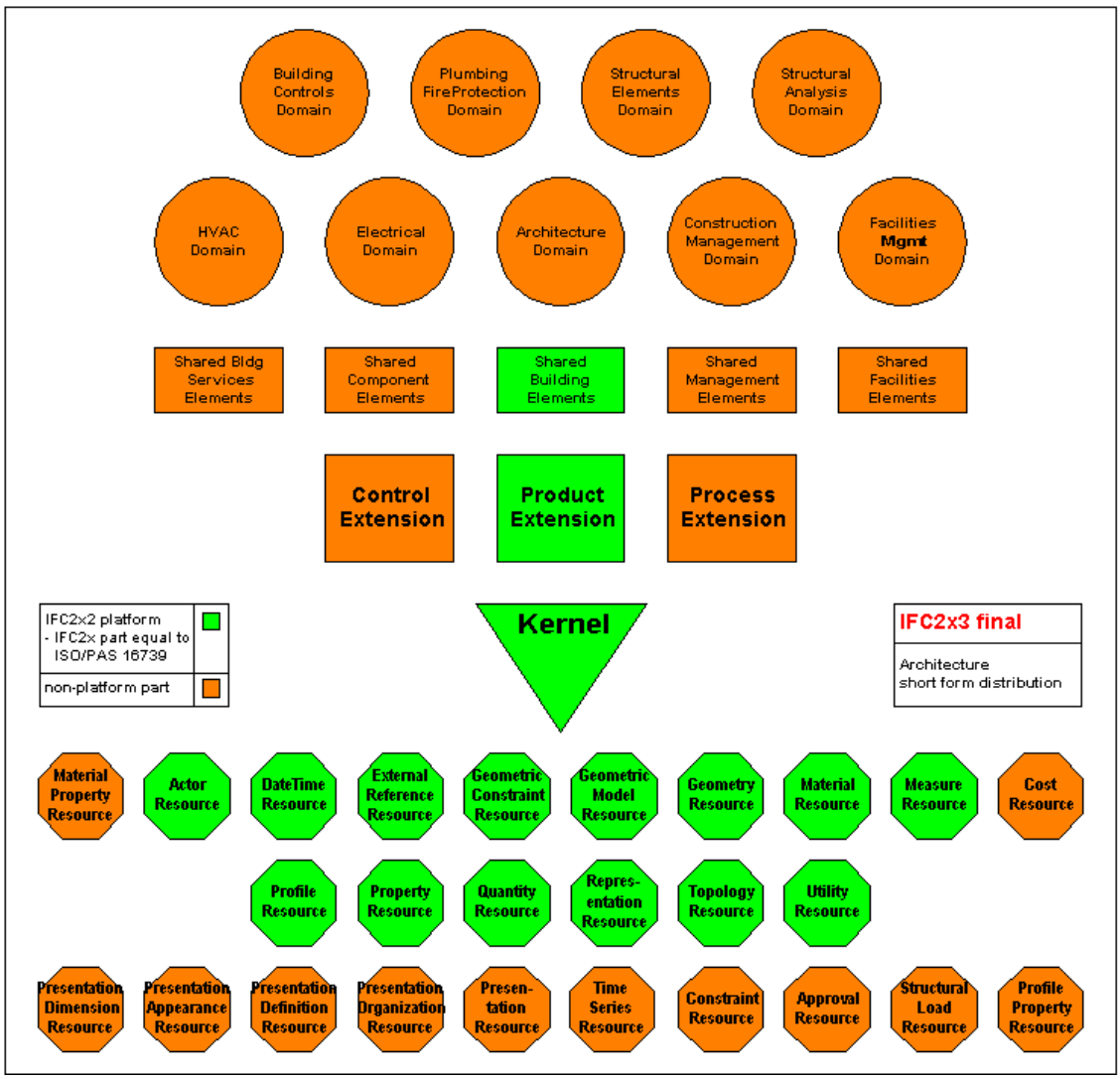
IFC-standardi (Industry Foundation Classes) on kansainvälinen tiedonsiirtostandardi rakentamisen ja kiinteistöpidon eri tietojärjestelmien välillä (Penttilä et al., 2006). IFC pyrkii kattamaan rakennushankkeiden ja rakennuksen koko elinkaaren ja eri osapuolien näkökulmat (Laine, 2008). IFC 2 x 3:n kattamia (joskaan ei täydellisesti) rakentamisen ja kiinteistöpidon alueita ovat: rakennesuunnittelu (arkkitehtisuunnittelu), rakennesuunnittelu, talotekninen suunnittelu, rakentamisen valmistelu (tuotannonsuunnittelu ja rakentaminen) ja kiinteistönpito (Laine, 2008).

IFC määrittelee tietokonesovelluksista riippumattoman tavan siirtää kolmiulotteista tuotetietoa sovellusten kesken (Penttilä et al., 2006). Keskeinen IFC:n kehittämisen tavoite liittyy ”interoperability” käsitteeseen, eli tietoa on kyettävä tallentamaan ja siirtämään ohjelmien välillä ohjelmariippumattomasti (Penttilä et al., 2006). Oleellista kuitenkin on, että tallennettaessa tietoa IFC-tietomallin mukaisesti, kaikki tieto tallennetaan oliopohjaisesti.

Pyrkimyksenä on tallentaa myös muunlaista tietoa eri entiteetteihin, kuin mitä graafisen tietomallin mukainen tieto on. Esimerkiksi talonrakennuksessa, voitaisiin tallentaa eri piirto-objektit kuten ovi, ikkuna ja seinä tarkemmin, kuin mitä esimerkiksi rakennusvalvonta Suomessa tällä hetkellä vaatii. Piirto-objekteihin voidaan tallentaa esimerkiksi rakennusmateriaalit, joita käytetään sekä esimerkiksi kaikkea muuta tietoa, mitä tarvitaan rakennuksen koko elinkaaren aikana.

3.2.1 IFC:n eri osa-alueet

IFC jaetaan osa-alueisiin kuvan 7 mukaisesti. Mallin eri osat on kuvattu standardissa sekä EXPRESS-kielellä, että EXPRESS –G-kuvauksilla. (mukaillen IAI, 2000)



Kuva 7. IFC:n rakenne osa-alueittain (IAI, 2006)

Resource-taso on matalin taso IFC-malliarkkitehtuurissa ja sitä (viittauksia resursseihin) voidaan käyttää muiden tasojen luokissa. Resurssit ovat riippumattomia domain:sta (alueesta tai toimialasta). Kaikki resurssit esittävät yksittäisiä liiketoimintakäsitteitä. Esimerkiksi IfcCostResource käsittää kaikki kustannuksiin liittyvät asiat. Samalla tavalla kaikki geometriaan liittyvät asiat on IfcGeometryResource:ssa. Laajennetut grafiikkaan liittyvät asiat ovat IfcProfileResource skeemassa. (IAI, 2000)

Core-tasolla määritellyt luokat ovat sellaisia, joihin voidaan viitata Domain ja Interoperability-tasoilta (Interoperability-taso on taso, jossa kuvan 7 mukaisesti ovat määritelty eri Shared Elements-laatikot). Core-taso on perusrakenne IFC-objektimallissa ja siinä määritellään yleisimmät käsitteet, joita voidaan erikoistaa korkeimmilla tasoilla IFC-objektimallissa. Core–taso koostuu kahdesta erikoistumisen tasosta: Kernel sekä Core –laajennukset (Extension). Core–tason suunnittelun yksi tavoite on määritellä kaikille malleille yhteiset sisällöt. Sisällöt on määritelty siten, että ne voidaan myöhemmin määritellä uudelleen tai käyttää erilaisissa interoperability ja domain-malleissa. Niiden avulla voidaan myös esiharmonisoida domain-malleja luomalla yhteisiä käsitteitä, ja ne mahdollistavat myös päivitettävyyden uusille IFC-julkaisuille. (IAI, 2000)

Kernel -tasolla on määritelty kaikki peruskäsitteet, jotka ovat ominaisia sen hetkiselle IFC-julkaisulle. Siinä myös määritellään mallin rakenne ja jakautuminen eri luokkiin. Käsitteet, jotka on määritelty Kernel-tasolla, on erikoistettu korkeammalla tasolla. Siinä on myös peruskäsitteet objektien varaamisesta, suhteista, tyyppimäärittelyistä, attribuuteista ja rooleista. Kernel voidaan nähdä mallina (template), jonka mukaan muut skeemat mallissa on kehitetty (mukaan lukien kaikki laajennusmallit). Sen rakenteet ovat erittäin geneerisiä ja ne eivät ole AEC/FM-spesifisiä, vaikkakin niitä käytetään vain AEC/FM tarkoituksiin johtuen Core-laajennusten erikoistamisesta. Kernel rakenteet ovat pakollisia kaikille IFC-implementaatioille. Kernel on Core-mallin perusta. Kernel-luokista saatetaan viitata Resource-tason luokkiin, mutta niistä ei viitata Core-tasolle tai korkeammalle tasolle. IfcKernel määrittelee IFC-arkkitehtuurin abstrakteimman tason. Se kapseloi yleisemmät käsitteet (esim. objektit, suhteet ja ominaisuudet), jotka ovat oleellisia IFC:n ymmärtämiselle. Mallin avainrakenteet IFC 2x Kernel:ssä on esitetty liitteessä III. (IAI 2006).

3.2.2 P21-muoto (ISO 10303-21)

IFC:ssä voidaan käyttää yhtenä tiedostomuotona STEP-standardin osa 21:n (ISO 10303-21) tiedostomuotoa. Edellä mainittua tiedostomuotoa käytetään ilmaistaessa IFC-luokilla tehtyjen objektien instansseja (varsinainen CAD-kuvan data). P21 (osa 21) on tällä hetkellä käytetyin tiedostomuoto IFC-tiedonsiirrossa. Tiedoston tarkennin on ifc tai stp.

Tiedosto jakaantuu seuraaviin oleellisimpiin osiin (Nour & Beucke, 2005):

```
ISO-10303-21;

HEADER;
…           /*Tässä HEADER -osa*/
ENDSEC;

DATA;
…           /*Tässä DATA -osa*/
ENDSEC;
END-ISO-10303-21;
```

HEADER –osa

HEADER –osassa voi olla seuraavia määritelmiä:

```
FILE_DESCRIPTION /*tiedoston kuvaus*/
FILE_NAME /*tiedoston nimi*/
FILE_SCHEMA  /*käytetyn skeeman nimi*/
(FILE_POPULATION)
(SECTON_LANGUAGE)
(SECTION_CONTEXT)
```

Suluissa olevat osuudet toimivat vain versiosta 3 eteenpäin.

IFC2x mallin mukaisesti HEADER osa voi olla IFC-tiedostossa seuraavanlainen (toim. Liebich, 2004):

```
ISO-10303-21;

HEADER;
/*kuvaus*/                    (‘IFC2X_PLATFORM’),
/*toteutuksen taso*/          ’2:1);
FILE_NAME(
/*nimi*/                      ’testi_projekti’,
/*aikaleima*/                 ’2007-10-17T21:25:27+06:00’,
/*tekijä*/                    (’tekijän nimi’),
/*organisaatio*/              (’tekijän organisaatio’),
/*työkalun versio*/           ’IFC työkalun nimi’,
/*alkuperäinen järjestelmä*/  ’lähettävän sovelluksen nimi’,
/*tekijänoikeus*/             ’tekijänoikeudellinen nimi’);
ENDSEC;
FILE_SCHEMA (
/*skeeman nimi*/              (’IFC2X_FINAL’));
```

DATA-osa

Data-osa on varsinainen ilmentymien kuvaava osuus, ja siinä eri ilmentymät eri riveillä ovat muotoa:

#positiivinen kokonaisluku = OLIO(parametri1,parametri2, …);

Parametrit voivat olla paitsi olion kuvaavia attribuutteja, myös viittauksia muihin olioihin (#-merkki edessä) tai kokoelmiin (viittaus suluissa muodossa (#positiivinen kokonaisluku)). Kokoelmia voivat siinä tapauksessa olla EXPRESS- kielen SET, BAG, LIST ja ARRAY. Jos parametria ei ole asetettu, laitetaan olion parametrin arvoksi $. INVERSE, DERIVED ja uudelleenmääriteltyjä attribuutteja ei kirjoiteta uudelleen, koska ne voidaan johtaa muista. Alityypeissä uudelleen määritellyt tai perityt attribuutit kuvataan *:llä. Luetellut, boolean tai loogiset arvot ilmoitetaan muodossa: .arvo. Esimerkiksi: .TRUE. String-tyyppiset arvot ilmoitetaan muodossa ’merkkijono’.

Esimerkki 1:

```
#268436967= IFCORGANIZATION($,'Yritys Oy',$,$,$);
```

#268436971= IFCPERSONANDORGANIZATION(#268436965,#268436967,$);

Edellä IFCPERSONANDORGANIZATION:n toinen parametri(#268436967) viittaa edellisen rivin IFCORGANIZATION:n ilmentymään.

Esimerkki 2:
#268436973= IFCSIUNIT(*,.LENGTHUNIT.,.MILLI.,.METRE.);

* viittaa kantaluokan attribuutteihin ja esimerkiksi .MILLI lueteltuun arvoon.

Esimerkki 3:

#268437060= IFCCARTESIANPOINT((0.,0.));
#268437062= IFCCARTESIANPOINT((3200.,0.));
#268437064= IFCPOLYLINE((#268437060,#268437062));

Suluissa oleva (#268437060,#268437062) viittaa kokoelmaan IFCCARTESIANPOINT -olioista. Eli IFCPOLYLINE muodostaa viivan alkupisteen ja loppupisteen välille.

3.3 Erilaisten tietomallien välinen tiedonsiirto

On olemassa erilaisia näkemyksiä siitä, pitäisikö rakennusalalla noudattaa vain yhtä tietomallia, vai voitaisiinko käyttää yhtä aikaa montaa tietomallia. Käytännössä tietomallin rakentaminen on kuitenkin erittäin kohdealuekohtaista. Esimerkiksi rakennesuunnittelija tarvitsee erilaisen tietomallin kuin arkkitehti. Tietyiltä osin tietomalli voi olla samanlainen. Erilaisten tietomallien väliseen tiedonsiirtoon on kehitetty EXPRESS –X-kieli, jolla voidaan muuntaa vastaavuudet eri tietomallien skeemojen välille. Sovelluksen ja tietomallin välistä muuntamista sanotaan ulkopuoliseksi muuntamiseksi (*external mapping*). Kahden eri tietomallin välistä muuntamista taas sanotaan sisäiseksi muuntamiseksi (*internal mapping*). (Eastman, 1999)

Kahden eri STEP-skeeman väliseen muuntamiseen on kehitetty EXPRESS –M-kieli. Esimerkki skeemojen väliseen muuntamiseen on taulukossa 1 oleva koodi. (mukaillen Eastman 1999)

Taulukko 1. Lähde- ja kohdeskeema, joiden väliin laaditaan muunninkoodi.

LÄHDE:	**KOHDE:**
ENTITY viiva; alku:point; loppu; point; END_ENTITY;	ENTITY viiva_vektori; alkaa : cartesian_point; päättyy:cartesian:point; END_ENTITY;

EXPRESS –M-kielellä muunto edellä kuvatusta lähteestä kohteeseen tehtäisiin seuraavasti (mukaillen Eastman 1999):

```
MAP viiva_vektori <- viiva
  alkaa := {cartesian_point} alku;
  loppuu := {cartesian_point} loppu;
END_MAP
```

EXPRESS –M-kieltä ei luotu tietokannan rakenteen muunnokseen, vaan se toimi enemmänkin tiedostopohjaisten skeemojen välisenä muunnoskielenä. Kehitettiin erilaisille tietonäkymille EXPRESS –V-laajennus ja edelleen EXPRESS –X, joka on kehitetty nimenomaan eri tietomallien väliseen muuntamiseen. (mukaillen Eastman, 1999)

3.4 XML-pohjaisen tiedon tallennus ja siirto

XML-kieltä (eXtensible Markup Language) voidaan käyttää kahteen tarkoitukseen: tiedon sisällön (ja rakenteen) määrittämiseen ja tiedonsiirtoon. XML:n eräs etu on se, että sillä voidaan luoda sovellusriippumattomia dokumentteja ja dataa. XML on niin sanottu ”merkintäkieli” (markup language) ja se koostuu sanoista tai merkeistä, joiden ympärillä käytetään tageja. Eräs etu XML:llä verrattuna HTML-kieleen on myös se, että siinä voidaan eriyttää varsinainen data ja käyttöliittymän kuvaus (voidaan tehdä esimerkiksi XSLT ja/tai XHTML -kielellä) toisistaan.

3.4.1 XML-kieli tiedon tallentamisen formaattina

XML-kieli taipuu monenlaisen tiedon esittämiseen. Tietoa voidaan esittää relaatiotietokannan vaatimassa muodossa tai esimerkiksi puumaisesti. Myös oliopohjainen tieto voidaan esittää XML-muodossa. XML-kieli on lisäksi rakenteellisesti suhteellisen selkeä ja yksinkertainen. Niin sanottu well-formed (oikein

muotoiltu) XML-dokumentti on sellainen, jossa on yksi kantasolmu (=tagi), jokainen aloitettu tagi loppuu lopetustagilla, jokaiseen attribuutin arvo on lainausmerkeissä, tyhjillä elementeillä on myös lopetusmerkki (/), kaikki entiteetit on määritelty ja lapsielementit ovat sisäkkäisiä (Singh & Huhns, 2005). Loppujen lopuksi edellä mainittuja ”perussääntöjä” on suhteellisen vähän ja silloin myös virheellisten XML-dokumenttien luontimahdollisuudet ovat mitättömät. XML-spesifikaatiossa on kuitenkin määritelty lisäksi lukuisia muita sääntöjä, joita tulisi noudattaa. Käyttämällä kehitysympäristöjä, XML-dokumenteista tulee kuitenkin yleensä automaattisesti hyvin muotoiltuja.

Perussyntaksi XML-dokumentissa on seuraava:

<kantasolmu>

<lapsielementti attribuutti=arvo>

lapsielementin arvo

</lapsielementti>

</kantasolmu>

3.4.2 XML-kieli tiedonsiirrossa

Vaikkakin XML-muotoinen tiedonsiirto voidaan toteuttaa kuten edellä mainittu ”neutraalin tiedoston” avulla suoritettu tiedonsiirto, antaa XML paljon muitakin etuja. Osaksi XML:n käyttö tiedonsiirtoformaattina pohjautuu siihen, että web-palvelimet pystyvät käsittelemään tekstimuotoista dataa joustavasti ja esimerkiksi ilman suurempia konfigurointeja palomuureissa. Esimerkiksi vanha COM (Component Object Model) – komponenttien avulla tehty tiedonsiirto pohjautui binäärisen tiedon siirtoon, jolloin tiedonsiirron toteuttaminen oli hankalaa varsinkin tietoturvallisesti. Tiedonsiirtoon sanomapohjaisesti XML-kielellä on kehitetty niin sanottu SOAP (Simple Object Access Protocol) -protokolla. Sitä käytetään nykyaikaisten (hajautettujen sovellusten) web-palvelu (web-service) -tyyppisten sovellusten viestintäprotokollana.

3.4.3 XML-kielten skeemat

XML:n ollessa rakenteinen kieli, sillä voidaan määritellä myös metadataa eli tietoa varsinaisesta tiedon rakenteesta. XML-dokumenttien rakenteen oikeellisuus voidaan tarkistaa niin sanotuilla skeemoilla. Jos XML-dokumentin rakenne on skeemassa määritellyn kaltainen, sanotaan XML-dokumentin olevan validi. Skeematekniikoita on

olemassa monenlaisia. Ensimmäinen skeematekniikka oli DTD, joka kehitettiin SGML-kielen muotoisena (josta myös XML-kieli aikoinaan kehittyi) (Evjen et al, 2007). Toinen skeematekniikka on XMLSchema, jota käytetään nykyisin yhä enemmän (Evjen et al, 2007). Etuna siinä verrattuna DTD:hen on muun muassa, että XMLSchemassa on enemmän tietotyyppejä ja itse XMLSchema on myös XML-muotoinen. Lisäksi on olemassa esimerkiksi RELAX NG-skeema (Evjen et al, 2007). Skeemoja käytetään tiedonsiirron suhteen nimenomaan siihen, että ne voivat toimia yhteisenä rajapintana lähettäjän ja vastaanottajan välillä. Jos molemmissa päissä, eli sekä lähettävässä että vastaanottavassa sovelluksessa on sama määrä kenttiä, samaa tietotyyppiä ja sama tietorakenne (esimerkiksi lapsielementit, attribuutit määritelty rakenteellisesti samalla tavalla), ei tiedonsiirrossa voi tulla niin helposti virhettä, ainakaan lähetyksessä ja vastaanotossa.

3.4.4 IfcXML

IfcXML on vaihtoehto STEP-standardimuotoiselle (SPF = Step Physical File, joka ISO 10303-21 standardin mukainen) tiedostolle. IfcXML-tiedostoille voidaan antaa tiedoston tarkennin .xml, .ifcxml tai .ifx. IfcXML:stä oletetaan tulevan keino jolla IFC-objektimallin mukainen tieto voidaan kuvata esimerkiksi aikatauluissa, määräluetteloissa ja tuotetietojen esittämisessä. Lisäksi ifcXML:ää voidaan käyttää XML- muotoisten viestien välityksessä, esimerkiksi sähköisen liiketoiminnan sovelluksissa. Jos johonkin muuhun toiminnalliseen kokonaisuuteen (joka on rakennettu XML-pohjaisesti; kuten esimerkiksi paikkatietojärjestelmä), halutaan rakentaa yhteyksiä, voidaan helpommin kommunikoida XML-pohjaisesti ifcXML:n avulla. (IAI, 2007)

IfcXML:ssä esitetään standardin 10303 Part 28 Edition 2 (”osa 28”) mukaista dataa. Standardissa on esitetty XML-skeema, joka on tehty EXPRESS (ISO 10303 part 1):stä automaattisella konversiolla. Muutos EXPRESS-skeemasta XML-skeemaan esitetään erillisessä konfiguraatiotiedostossa, jossa on esitetty muuttamisprosessin spesifikaatiot. Edellä mainittu konfiguraatiotiedosto on standardoitu ja se julkaistaan jokaisen uuden IFC-skeeman julkaisemisen yhteydessä. (IAI, 2007)

IFC-muotoisen datan käsittelyssä on sovelluksia toteutettu seuraavilla tavoilla: Express STEP SPF-tiedostomuotoisen datan suoran käsittelyn toteutukset (rakennettu erilaisia

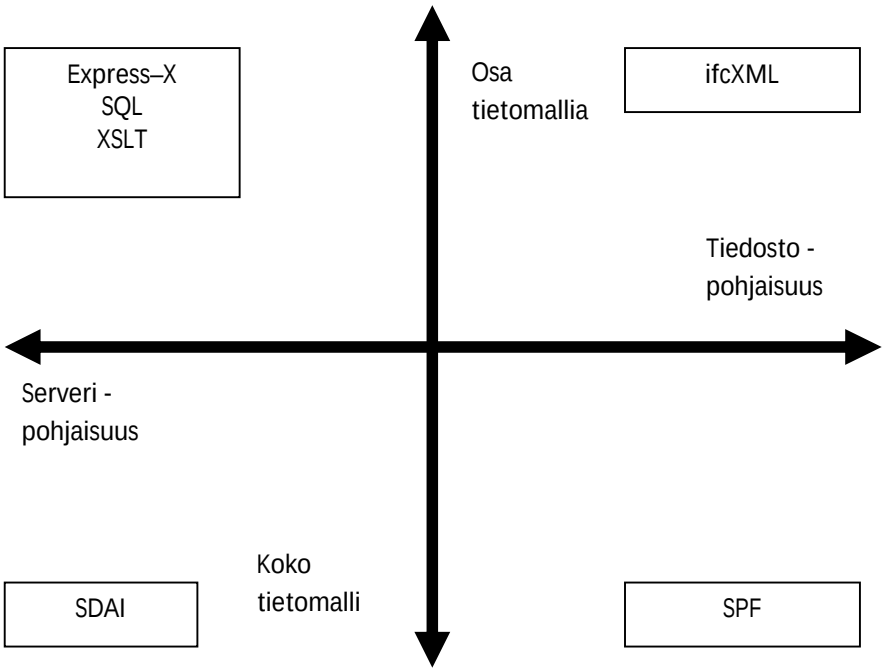
ToolKit:ejä), Express SDAI (rajapinta STEP:iin)-toteutukset tietomallipalvelimilla, Express-X:llä ja muilla tietokannan kysely-/muokkauskielillä tehdyt toteutukset sekä ifcXML-pohjaiset sovellukset. Verrattuna muihin sovellusalueisiin ifcXML-muotoiset sovellukset ovat ainakin toistaiseksi harvinaisempia. (IAI, 2007)

Tiedostokokona ifcXML-muotoinen tiedosto on 2-10 kertaa suurempi, kuin perinteinen ISO 10303-21 standardin mukainen tiedosto. Taulukossa 2 on vertailtu EXPRESS:llä kuvattua skeemaa ifcXML:llä kuvattuun skeemaan. Taulukossa 3 taas on vertailtu P21-formaatissa olevia instansseja vastaavaan ifcXML-dokumenttiin.

ifcXML:n roolia ei nähdä siinä, että koko tietomallin oliot pitäisi voida kuvata sen avulla, mutta sillä voidaan kuvata ja esittää tarvittavaa osaa tietomallin objekteista (esimerkiksi raporteissa tai osittaisessa tiedonsiirrossa). Ainakin vielä laajan XML-muotoisen datan esittäminen saattaa olla hidasta joidenkin sovellusten (esimerkiksi parsereiden) avulla. Jos suorituskyky on pullonkaula, IFC-mallin mukainen tieto kannattaa toteuttaa tietomallipalvelimelle tai erilliseen tietokantaan. Jotkin tietokannat tukevat suoria EXPRESS-kyselyitä tai XSLT-muunnoksia. (IAI, 2007)

XML-parsereita on kuitenkin kahdenlaisia: sellaisia, jotka lukevat koko XML-datan kerralla muistiin tai sellaisia, joissa XML-tiedostoa voidaan lukea ”pala” kerrallaan. Viimeksi mainituista esimerkiksi SAX-parseri on sellainen. Jos tietoa voitaisiin lukea järkevissä kokonaisuuksissa suhteessa skeemaan, voi tiedoston tulkinnan suorituskyky parantua huomattavasti. SAX-parseria ei kannata kuitenkaan käyttää kaikenlaisissa sovelluksissa: kannattaako se, riippuu sovelluksessa kerrallaan tarvittavan tiedon määrästä.

IfcXML-skeema on kevyempi versio EXPRESS-skeemasta. Verrattuna EXPRESS-skeemaan, joitakin rajoitteita kuten säännöt (rules), päinvaistaiset suhteet (inverse) ja perityt (derived) attribuutit puuttuvat ifcXML:stä. IfcXML:n tieto validoidaan ifcXML-skeemaa (.xsd -tiedosto) vasten ja siitä saattaa puuttua osin yhteensopivuus IFC:n EXPRESS-skeemaan (.exp tiedosto). Seuraavassa kuvassa 8 on vertailtu eri IFC-toteutuksia eri sovellusalueiden suhteen. XML-kielellä voidaan ilmaista myös niin sanottua semantiikkaa ontologioiden avulla, esimerkiksi käyttäen niin sanottua ifcOWL-kieltä. Luvussa neljä on käsitelty hieman tätä. Kuvasta 8 ifcOWL kuitenkin puuttuu vielä. (mukaillen IAI, 2007)



Kuva 8. IFC-toteutuksien vertailu eri sovellusalueiden suhteen (IAI, 2007)

Taulukko 2. Esimerkki muunnoksesta EXPRESS-tietomalliskeemasta XML-skeemaan (IAI, 2007)

Alkuperäinen IFC EXPRESS- muoto	**ifcXML-skeema (generoitu Part28-standardin mukaisesti)**
ENTITY IfcProperty ABSTRACT SUPERTYPE OF (ONEOF (IfcComplexProperty ,IfcSimpleProperty)); Name : IfcIdentifier; Description : OPTIONAL IfcText; END_ENTITY;	<xs:element name="IfcProperty" type="ifc:IfcProperty" abstract="true" substitutionGroup="ex:Entity" nillable="true"/> <xs:complexType name="IfcProperty" abstract="true"> <xs:complexContent> <xs:extension base="ex:Entity"> <xs:sequence> <xs:element name="Name" type="ifc:IfcIdentifier"/> <xs:element name="Description" type="ifc:IfcText" nillable="true" minOccurs="0"/> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType>

Taulukko 3. Esimerkki tietomallin mukaisen datan ilmaisemisesta SPF-muodossa (P21-formaatti) ja ifcXML- muodossa (IAI, 2007)

P21 -data	**ifcXML**
#84=IFCPROPERTYSINGLEVALUE ('**Red**',$,IFCINTEGER(**255**),$); #85=IFCPROPERTYSINGLEVALUE ('**Green**',$,IFCINTEGER(**0**),$); #86=IFCPROPERTYSINGLEVALUE ('**Blue**',$,IFCINTEGER(**0**),$); #87=IFCCOMPLEXPROPERTY ('**Color**',$,'**Color**',(#84,#85,#86));	<IfcComplexProperty id="i87"> <Name>**Color**</Name> <UsageName>**Color**</UsageName> <HasProperties> <IfcPropertySingleValue> <Name>**Red**</Name> <NominalValue> <IfcInteger>**255**</IfcInteger> </NominalValue> </IfcPropertySingleValue> <IfcPropertySingleValue> <Name>**Green**</Name> <NominalValue> <IfcInteger>**0**</IfcInteger> </NominalValue> </IfcPropertySingleValue> <IfcPropertySingleValue> <Name>**Blue**</Name> <NominalValue> <IfcInteger>**0**</IfcInteger> </NominalValue> </IfcPropertySingleValue> </HasProperties> </IfcComplexProperty>

4 IFC-TIEDON KÄSITTELY JA SIIRTO TULEVAISUUDEN TIETOVERKOISSA

Semanttinen web on tällä hetkellä ”kuuma” puheenaihe puhuttaessa Internetistä. Seuraavissa luvuissa pyrin kuvaamaan mikä semanttisen webin ja rakennusalan suhde on nyt, mitä siitä on hyötyä tiedonsiirron suhteen ja miten se vaikuttaa rakennusalan sovelluksien tiedonsiirron kehittymiseen tulevaisuudessa.

4.1 Erilaiset tiedonsiirtoteknologiat nyt ja tulevaisuudessa

Pelkistetysti voidaan sanoa, että tietoa voidaan siirtää IFC-sovellusten välillä: pelkkien IFC-tiedostojen avulla, ifcXML-tiedostojen avulla, käyttämällä STEP–rajapintaa (työasema- tai serveripohjaisesti), tietokantaa apuna käyttämällä tai viimeisimpänä uutena tulevaisuuden visiona suorittamalla tiedonsiirto agenttipohjaisesti semanttisen webin tekniikoilla. Viimeisimpään vaihtoehtoon on alettu kehittää ifcOWL-ontologiaa.

Seuraavan taulukon 4 mukaisesti yhteensopivuus ja sovelluksien yhteiskäyttö (interoperation) etenee teknologioiden osalta (esimerkiksi niin sanottujen virtuaaliorganisaatioiden tiedonsiirron osalta) neljänteen vaiheeseen.

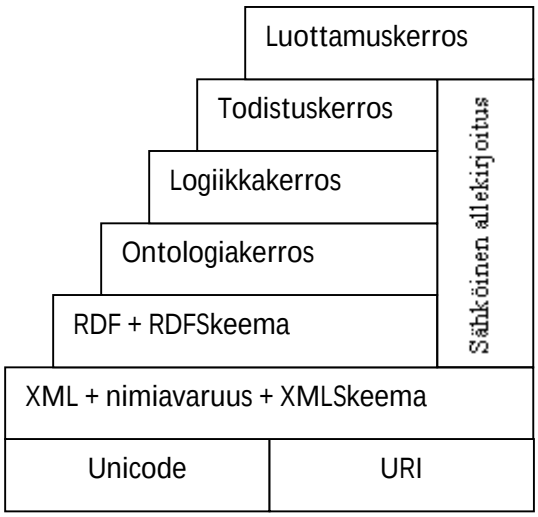
Taulukko 4. Teknologioiden yhteiskäytön kehitysvaiheiden tarkastelua osa-alueittain (Singh & Huhns, 2005)

Sukupolvi/ Osa-alue	**Ensimmäinen**	**Toinen**	**Kolmas**	**Neljäs**
Kommunikaatio	TCP/IP	CORBA	HTTP	Sanomanvälitys
Informaatio	SQL	XML	RDF	OWL
Sovellus	RPC	EDI	SOAP	Protokollat
Konfiguraatio	Kovakoodattu	Hakemistopalvelu	UDDI	Valinta

4.2 Semanttinen web rakentamisen eri osapuolten apuna

Tulevaisuuden webin visiona on rakentaa niin sanottu semanttinen web, jota voidaan käyttää esimerkiksi virtuaaliorganisaation eri osapuolten välisessä tiedonsiirrossa. Niin

sanotun agenttipohjaisen tiedonsiirron avulla voidaan ketjuttaa yksittäiset atomiset palvelut ja agentit ja orkestroida suureksi monimutkaiseksi toisistaan riippuvaiseksi palveluksi, jolloin saadaan esimerkiksi: ”linkitettyä yksittäiselle rakentajalle kunnalliset säädökset suhteessa rakentajan omaan projektiin” (mukaillen Gehre et al., 2006). Jotta edellä mainitut tavoitteet voitaisiin saavuttaa, pyritään yhteensopivuuteen, tiedon: syntaksin suhteen (XML-kielen avulla), rakenteen suhteen (RDF-kielen avulla) ja semantiikan suhteen (OWL -ontologiakielen avulla). Puhutaankin niin sanotusta semanttisen webin pinosta (kuva 9).



Kuva 9. Semanttisen webin pino (Berners-Lee, 2000)

Unicode ja URI muodostavat perustan, jonka päälle on rakennettu syntaksikerros, joka on rakennettu XML:n, XMLSkeeman ja nimiavaruuksien avulla. Tietomallikerros on rakennettu RDF:n ja RDFSkeeman avulla. Tällä hetkellä rakennetaan ontologiakerrosta. Todistuskerros ja luottamuskerros ovat kerroksia, joita tullaan rakentamaan tulevaisuudessa.

XML-kieli ja sen syntaksi on perusta koko semanttiselle webille. Sekä RDF ja OWL pyritään kuvaamaan myös XML-pohjaisesti. RDF-kielellä voidaan yhdistää jaettua semantiikkaa ja OWL-kieli toimii semantiikan määrittäjänä. Myös RDFS (RDF-skeema) – kielellä voidaan määritellä semantiikkaa, mutta rajoitetusti (OWL-kieli on monipuolisempi). Rajapinnat eri web palvelujen välillä kuvataan XML-skeemalla. Myös IFC:lle on alettu kehittämään OWL-ontologiaa: puhutaan niin sanotusta ifcOWL-

kielestä. Ongelmana on kuitenkin se, että ifcOWL-kielelle ei ole lähdetty rakentamaan yhteistä standardia monen osapuolen kesken kuten IFC-kielen muille formaateille. On tosin myös käsitys, että on hyvä, että standardia ei ole alettu kehittämään, koska se hidastaisi ifcOWL:n kehittymistä. Esimerkkejä tiedonsiirrosta on tehty muullakin tavalla kuin ifcOWL-syntaksin mukaisesti. (mukaillen Yanga & Zhang, 2006)

Web palvelut (web services) ovat osa semanttisen webin pinon teknologiaa: ne ovat modulaarisia sovelluksia, joita voidaan julkaista, sijoittaa ja käyttää läpi webin (mukaillen Gehre et al., 2006). Web-palvelujen avulla voidaan rakentaa sovelluksia, jotka vastaavat yksittäisiin funktiokutsuihin tai monimutkaisiin liiketoiminnan prosessikutsuihin webin kautta, käyttämällä niin sanottua SOAP-protokollaa, joka sekin on XML-muotoinen, TCP/IP:n päälle rakennettu oma protokollansa. Sovellukset voidaan löytää niin sanotun UDDI-rekisterin avulla. Niissä on esitetty web-palvelun sisältö ja kuinka niitä voidaan käyttää hyväksi.

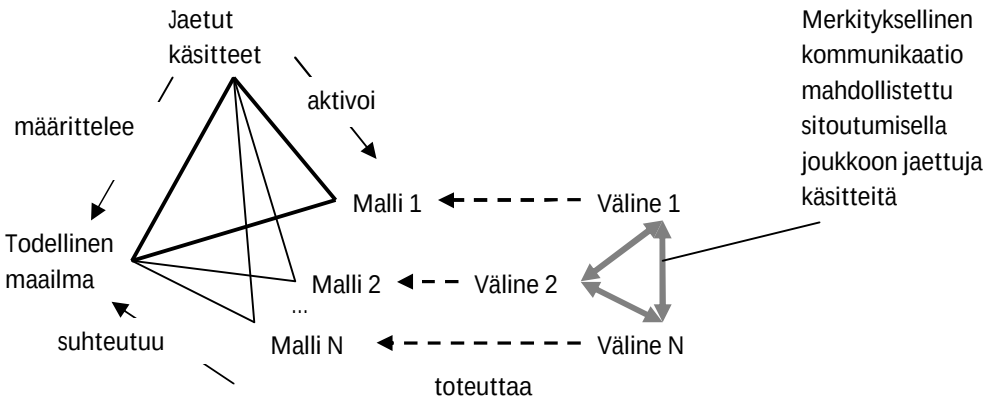
Liiketoimintaprosessien kuvaamiseen on kehitetty erilaisia rajapintoja ja kieliä. Rajapintoja ovat esimerkiksi ebXML, RosettaNet. Prosessien kuvaamiseen kehitettyjä kieliä taas ovat BPEL4WS, OWL-S, PSL ja WSCI.

Taustalla semanttisen webin kehitykseen on muun muassa virtuaaliorganisaatioiden välisen tiedonsiirron ratkaiseminen. Virtuaaliorganisaatioilla tiedonsiirrossa on niin sanottu yhteensopivuuden (interoperability) ongelma. Virtuaaliorganisaatiolla on kuitenkin etuja: se on väline markkinavetoiseen yhteistyöhön ja komplementaarisuuteen (täydentyvyyteen). Dynaaminen virtuaaliorganisaation partnereiden osallistuminen edellyttää, että organisaatiot voivat liittyä virtuaaliorganisaation verkostoon ja irtaantua siitä dynaamisella tavalla. Sen vuoksi juuri oikea-aikainen informaatio saatavilla olevista palveluista ja osapuolista on tarpeellista. Verkoston informaatiota pitää päivittää säännöllisesti. Prosessien ja resurssien jako on pääasiallinen haaste virtuaaliorganisaatioiden yhteensopivuudelle. Ei ole yksittäistä organisaatiollista rakennetta kaikille virtuaaliyrityksille. Edellä mainituista seikoista johtuen, metodit, palvelut ja rakenteelliset määritelmät (luotaessa yhteensopivuuden rakennetta virtuaaliorganisaatioita varten) täytyy olla joustavia, vankkoja ja vikasietoisia; kiinnittäen erikoista huomiota ylläpidettävyyteen. Tämän hetkiset huippuunsa kehitetyt ratkaisut käyvät käsiksi ongelmaan vain tekniseltä kannalta. Tukimetodeita ja työvälineitä on saatavissa, joilla voidaan yhdistää prosesseja ja päästä kiinni

rajapintoihin. Näin mahdollistetaan helppo palvelujen löytyminen ja se, että voidaan kuvata rajapintojen parametreja automatisoidulla koneen tulkitsemalla tavalla. Kuitenkin, sellainen tekninen yhteensopivuus takaa pääsyn vain yhteensopivuuteen ja parhaimmillaankin mahdollistaa vain syntaktisen yhteensopivuuden. Jotta päästään joustavampaan informaation hallintaan ja ”tarkoitukseen”, riittävä semanttinen yhteensopivuus täytyy huomioida niin, että se mahdollistaa kommunikaation taidokkaasti (tai yksityiskohtaisesti) laaditulle semanttiselle käsitteistön sanastolle ja siten ontologiselle tasolle. (mukaillen Gehre et al., 2006)

Kuva 10 kuvastaa, kuinka yhteinen ontologia voi mahdollistaa integraation semanttisesti heterogeenisille työvälineille. Korkealla tasolla suhde termien, mallien ja todellisuuden välillä voidaan esittää niin sanotulla Ogden-kolmiolla (Ogden & Richards, 1994).

Tietotekniikassa on termejä, joita käytetään käsitteiden määrittelyssä. Ne ovat symboleita, jotka määrittelevät käsitteitä; esim. termi rakennus tarkoittaa käsitettä ”rakennus”, joka on tosielämän artefakti. Eri välineet käyttävät tyypillisesti eri malleja suhteessa tosielämän artefaktiin, riippuen niiden erikoisista tarkoituksista ja aihealueesta (aiheita kuten: arkkitehtuuri, rakenteellinen analyysi, energia-analyysi, kustannusten arviointi). Siten jokaiselle välineelle on oma Ogdenin-kolmio, mutta sitoutumalla jaettuun käsitteiden joukkoon, niiden on mahdollista ymmärtää toisiansa ja kommunikoida molemminpuolisen aihealueen avulla. (mukaillen Gehre et al., 2006)



Kuva 10. Ogdenin-kolmio: yhteensopivuus eri välineiden avulla saavutetaan yhteisellä ontologisella tasolla.

4.3 ifcOWL verrattuna EXPRESS-kuvaukseen

”ifcOWL-skeema ontologia” perustuu puhtaasti niihin abstrakteihin kuvauksiin, joita IFC:stä on jo EXPRESS-muotoisilla kuvauksilla tehty. Sen ydintoiminnallisuus on kuvata IFC-mallin hierarkia objekteista, ominaisuuksista ja suhteista käyttäen OWL-kielen (Ontology Web Language) termejä ja määritelmiä. Se on täysin verrattavissa EXPRESS:llä tehtyyn skeemaan. (Gehre et al., 2006)

Alla pari esimerkkiä kuvattu ensin EXPRESS-skeemalla ja sitten ifcOWL-skeemalla (Gehre et al., 2006):

```
ENTITY IfcRoot
END_ENTITY
```

Jos sama kuvataan ifcOWL-kielellä, saadaan seuraava koodi:

```
<owl:Class ID=”IfcRoot”/>
```

Abstraktien ja tavallisten entiteettien välillä ei tehdä eroa.

```
ENTITY IfcObject
SUBTYPE OF (IfcRoot);
END_ENTITY;
```

muuttuu ifcOWL:ään

```
<owl:Class rdf:ID="IfcObject">
<rdfs:subClassOf rdf:resource="#IfcRoot"/>
</owl:Class>
```

Instanssien käsittelyn esimerkki P21-standardin EXPRESS-kuvaukseen verrattuna RDF-syntaksin vastaavaan: (mukaillen Gehre et al., 2006):

```
#672=IFCCARTESIANPOINT((0.,0.));
...
#679=IFCPOLYLINE((#672,#673,#674,#675,#676,#677,#678));
```

vastaava RDF-syntaksi:

```
<ifc:PolygonalBoundary>
 <ifc:IfcPolyline rdf:ID="E679">
  <ifc:Points>
   <ifc:IfcCartesianPoint rdf:ID="E672">
    . . .
```

4.4 Nykytila ontologiapohjaisten sovellusten kehitystyössä

Ontologiasovellusten kehittäminen on osoittautunut kovemmaksi työksi kuin luultiin. Saatavissa olevia ontologioita, jotka mallintavat esimerkiksi eri teollisuuden aihealueita, on saatavissa vähän tai ei ollenkaan. Ontologioiden kehittäminen on yhä edelleen tutkimusaluetta lähes kolmenkymmenen vuoden tutkimuksen jälkeen. Toimivia sovelluksia on vaikea löytää. Yksi syy saattaa olla se, että julkisesti on tarjolla vain rajoitetusti ontologioiden mallinnus- ja prosessointivälineitä. Kuitenkin uusi ontologiakieli OWL näyttää tuovan uusia etuja, ja työvälineetkin ovat kehittyneet. Erityisesti Protégé-kehitysympäristö ja Jena-ontologiakehysrakenne ovat lupaavia ontologiaperusteisia tietoteknisiä järjestelmiä. Kuitenkin jopa Protégé de-facto -standardina ontologian mallintamiseen, tuo mukanaan monia ongelmia jos kehitetään monimutkaista OWL-pohjaista ontologiaa, jossa käytetään viittauksia muihin ontologioihin. (mukaillen Gehre et al., 2006)

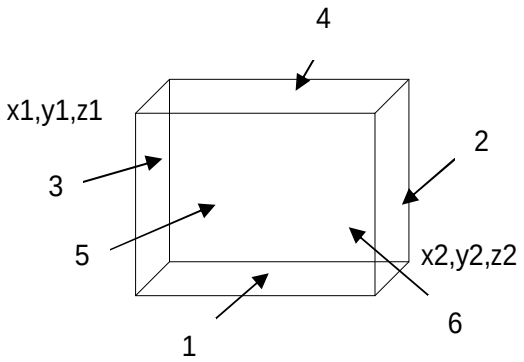
5 TIETOKANNAT IFC- TIEDON TALLENNUSPAIKKANA

Kuten kaikissa tiedonhallintaan liittyvissä liiketoiminnallisissa sovelluksissa, myös CAD-maailmassa tiedolle on saatava varma säilytyspaikka. Käsittelen tässä luvussa sitä, millaiseen tietokantaan CAD-sovelluksista tuleva tieto on tallennettava ja mitä varten. Lopuksi käsittelen tietomallipalvelimia ja sitä, mitä etuja niillä voidaan saavuttaa suhteessa tiedonsiirtoon.

5.1 Relaatiotietokannat

Relaatiotietokannat ovat tunnettuja siitä ominaisuudestaan, että varsinainen tieto ja tiedon säilytyspaikka ovat eristetyt täysin toisistaan. Relaatiotietokannassa kaikki tieto tallennetaan tauluihin. Suhteessa STEP-tietomallin mukaiseen tietoon, yksi entiteetti olisi luonnollista sijoittaa aina yhteen tauluun. Relaatiotietokannassa taulujen väliset suhteet voivat olla yhden suhde yhteen (merkitään: 1:1), yhden suhde moneen (1:m) tai monen suhde moneen (m:n); jossa viimeinen suhde kuitenkin täytyy purkaa kahteen 1:m yhteyteen.

Seuraavasta esimerkistä (kuva 11 ja taulukko 5) käy ilmi relaatiotietokantojen kömpelyys ja tehottomuus suhteessa 3 -ulotteiseen dataan (Rao, 1994):



Kuva 11. Kuution pinnat (Rao, 1994)

Taulukko 5. Objektien pinnat (Rao, 1994)

Objekti	**Pinta**
suorakaide 1	1
suorakaide 2	2
suorakaide 3	3
suorakaide 4	4
suorakaide 5	5
suorakaide 6	6

Jos edellä olevan kuution vastakkaisten kulmien pisteiden koordinaatit ovat (x1,y1,z1) ja (x2,y2,z2), tieto eri pintojen koordinaateista jouduttaisiin tallentamaan taulukon 6 mukaiseen tauluun.

Taulukko 6. Kuution pintojen koordinaatit 2-ulotteisessa taulukossa (Rao, 1994)

Pinta	**x-alkupiste**	**y-alkupiste**	**z-alkupiste**	**x-loppupiste**	**y-loppupiste**	**z-loppupiste**
1	x1	y1	z1	x2	y1	z2
2	x2	y1	z1	x2	y2	z2
3	x1	y1	z1	x1	y2	z2
4	x1	y1	z1	x2	y2	z1
5	x1	y2	z1	x2	y2	z2
6	x1	y1	z2	x2	y2	z2

Ongelmaksi edellisessä taulukossa muodostuu se, että kolmiulotteinen tieto yritetään tallentaa kaksiulotteiseen tauluun. Tämä on ongelmallista esimerkiksi tiedon käsittelyn tehokkuuden kannalta.

Toinen ongelma relaatiotiekannoissa on se, että oliopohjainen data joudutaan siirtämään (mapping) erilaisiin tietorakenteisiin, eli olioista tauluihin (impedance mismatch problem). Tämä laskee tietokannan suorituskykyä suhteessa oliopohjaisiin tietokantoihin. (Rao, 1994)

Määrittelin tekemäni sovelluksen (kuvattu luvussa 6) kuitenkin siten, että tieto on tarvittaessa helposti tallennettavissa myös relaatiotietokantaan. Koska sovelluksen ei tarvitse lukea kaikkea tietoa P21-tiedostosta, tietokannan suorituskyky ei muodostu ongelmaksi.

5.2 Oliotietokannat

Oliotietokannat on havaittu monessakin suhteessa paremmaksi STEP-tietomallin mukaisen tiedon säilytyspaikaksi. Kun käsitellään monimutkaista tai jatkossa kehittyvää dataa, oliotietokannat ovat viisaampi ratkaisu. Oliotietokantoja käytetään etenkin:

tiedonkäsittelyn järjestelmissä, CAD/CAM-suunnittelujärjestelmissä, ohjelmistojen tukiympäristöissä ja CASE-työvälineissä. (Rao, 1994)

Useimmiten EXPRESS-kielen elementit oliotietokannassa on muutettu olioiksi. Olioihin voidaan tehdä luonti-, muokkaus- tai hakutoimintoja. Tietokantoihin on myös mahdollista määritellä sessiokohtaisia (istuntokohtaisia) operaatioita kuten: tapahtuman nauhoitus ja nauhoituksen lopetus, session aloitus ja lopetus. Transaktioiden käsittely voidaan lisätä kirjoitus- ja lukutapahtumiin. Voidaan aloittaa transaktio vain lukutilassa ja jatkaa siitä luku-/kirjoitustilaan. Transaktioissa voidaan suorittaa monia tapahtumia (kuten monissa relaatiotietokannoissakin); ne voidaan suorittaa loppuun tai keskeyttää. Mallia (skeemaa) voidaan operoida; kuten toiminnoilla: avaa tiedon säilytyspaikka, sulje tiedon säilytyspaikka, uudelleennimeä malli. (mukaillen Eastman, 1999)

Objektien identiteetti voidaan järjestää oliotietokannoissa siten, että se ei riipu objektien attribuuteista. Myös tässä oliotietokannat eroavat relaatiotietokannoista. Objektien identiteetin avulla mahdollistetaan objektien hajautus. On olemassa kaksi tapaa määritellä objektien identiteetti: voidaan tehdä sijainnista riippuva yksilöinti tai sijainnista riippumaton yksilöinti. Viimeisessä tapauksessa objektia voidaan käyttää riippumatta systeemistä, jos jokaisella objektilla on oma yksilöivä tunniste (OID = object identifier). Siinä tapauksessa yksilöivän tunnisteen pituus saattaa olla iso: esimerkiksi 40 tai 64 bittiä. (mukaillen Rao, 1994)

IFD (International Framework for Dictionaries; nimeä käytetään organisaatiosta tai IFD kirjastosta) on kehittänyt kirjaston, jolla voidaan yksilöidä rakennuksissa käytettävät objektit yksiselitteisesti. Yksilöivänä tunnisteena käytetään GUID:ia (=Global Unique Identifier) tai UUID:tä (Universal Unique Identifier). Objektin tunniste sijoitetaan nimeen (name-tag) tai otsikkoon (label). GUID on 128 bittinen numero pakattuna 22 merkkiin. Algoritmina käytetään OMG:n (Object Management Group):n kehittämää algoritmia, jossa tunnisteen luovan metodin siemenenä (parametrina) on koneen IP-osoite. Näin voidaan olla varmoja, että ei käytetä kahta samanlaista tunnistetta. IFD:lle on kuitenkin määritelty paljon laajempia tehtäviä. IFD:n tavoitteena on olla ”sovelluskehyksenä jossa eri ontologiat voivat elää yhtä aikaa”. IFD-standardi (ISO 12006-3) on määritelty myös EXPRESS:llä ja sitä pidetään ”täydennyksenä IFC:lle”. (Bjørkhaug & Håvard, 2007)

5.3 Tietomallipalvelimet

Tietomallipalvelimet ovat palvelimia, joissa viestintä eri palvelimien välillä tapahtuu tietomallipohjaisesti. Etuna tietomallipalvelimella (verrattuna pelkästään tietomallin käyttöön) on se, että voidaan hallita muun muassa: käyttäjien oikeuksia, rooleja, versiointia ja omistajuutta. Voidaan jakaa tiedoitusluonteista tietoa tarvittaessa eri osapuolille. Voidaan myös tehdä linkkejä sisäisesti tai ulkopuolelle. Käyttäjä voi määritellä itse tarvitsemansa tiedon kokonaisuuden. Tietomallipalvelinten avulla voidaan hallita monimutkaisuutta. Niiden avulla voidaan myös tukea rakennuksen koko elinkaarta. Käyttäjä on vuorovaikutuksessa palvelimeen 3D-mallin kautta. 3D-malli on käyttöliittymä myös muuhun tietoon (kuten esimerkiksi kiinteistönpito ja määrälaskenta). (mukaillen Kiviniemi et al., 2005)

Tietomallipalvelimien ohjelmoimista testattiin jo 2000-luvun alussa VTT:n projektissa (IFC Model Server vuonna 2001). Siinä tietokantana oli SQL Server 2000. Tietomallipalvelimen sovellusarkkitehtuuri koostui, paitsi edellä mainitusta tietokannasta, tietokantatason komponentista (DALC= Data Acces Layer Component) ja Web-palvelutason komponentista (WSLC=Web Service Layer Component). Loppukäyttäjät ovat yhteydessä web-service tyyppisten sovellusten käyttäjinä webin kautta sovelluksilla, jotka käyttävät SOAP-protokollaa. Koodia kyseisestä projektista löytyy edelleen webistä. (Adachi, 2005)

Muitakin tietomallipalvelinratkaisuja sovelluspuolella on kehitetty; yksi niistä on WebSTEP (Karstila & Hemiö, 2002). Toinen tietomallipalvelinratkaisu on EPM 2003 (EPM, 2003). Kaikki edellä mainitut tietomallipalvelimet käyttävät tiedonsiirtoon tekniikoita kuten XML, SOAP ja STEP (mukaillen Adachi, 2002).

Tietomallipalvelimiin on kehitetty myös prosessipohjainen kehys SABLE (Simple Access to the Building Lifecycle Exchange), jonka avulla voidaan rakentaa yhteisen rajapinnan kautta Web Service-tyyppisiä sovelluksia, jotka viestivät SOAP:n (Simple Object Access Protocol) avulla (mukaillen Houbaux, 2003). SABLE:ssa on rakennettu (ja rakennetaan) ”standardoitu sovellusrajapinta” verkoston tasolla eri osapuolille aihealueen mukaan. Näitä aihealueita voivat olla muun muassa: kustannuslaskenta, lämpö-, vesi- ja ilmastointisuunnittelu (mukaillen Houbaux, 2003). Asiakaspuolella

(client) kyseiset rajapinnat on ohjelmoitu siten, että clienttien ”näkymät” ovat niin sanottuja BLIS- näkymiä (BLIS, 2001).

Tietomallipalvelimien avulla voidaan yhdistää monta tietomallia samaan verkostoon. Varsinaisten loppukäyttäjien sovellusrajapinnoiksi on tehty kehitystyötä muun muassa VBE II-projektissa, jonka tavoitteena oli kehittää muutamia rajapintoja loppukäyttäjien sovelluksille ja testata niitä monien tietomallipalvelinten ympäristössä käyttäen todellista projektitietoa, SABLE API:a (Application Programming Interface = sovellusrajapinta) ja olemassa olevia mallipalvelimia. (mukaillen Kiviniemi et al., 2005)

6 ESIMERKKI IFC-TIEDON SIIRROSTA JA PROSESSOINNISTA

Tämä luku on varsinainen työni soveltava osuus, jossa esittelen luomani IFC-tiedoston jäsentimen sekä miten tallensin tietoa, minne ja minkä vuoksi. Tiedonsiirron osapuolena on myös vastaanottava sovellus, mutta en pitänyt sitä varsinainen työni painopisteenä, koska kyseessä on ”vanha” COM-pohjainen sovellus ja pyrkimyksenä oli pikemminkin opiskella uutta tiedonsiirron kieltä, XML:ää.

Työssäni kehitin sovelluksen rakennusalan tarjouslaskennan hinnoittelua sekä määrätietojen hakua varten niiltä osin kuin se on järkevää/mahdollista käyttäen IFC -muotoista dataa. Sovelluksen avulla voidaan hakea ifc- muotoisesta tiedostosta määrät tietyistä IFC-kirjaston olioista oliotietokantaan, jonka tallennusformaatti on XML. Sovellus rakennettiin käyttäen Microsoftin Visual Studio 2005:sta ja .NET Framework 2:sta. Tiedot on nyt helppo siirtää eteenpäin XML-muotoisena varsinaiseen tarjouslaskentasovellukseen. Lisäksi tarvittaessa tieto on helppo siirtää esimerkiksi .NET:n DataSet-olioiden kautta relaatiotietokantaan. CAD-ohjelmistona käytin Archicad:n versio 10:tä sekä 11:sta, että AutoDesk:n Revit-ohjelmistoa. Tavoitteena ei ollut millään tavalla tallentaa tietoa IFC-tiedostoon vaan pelkästään lukea CAD-ohjelmistolla tehtyjen entiteettien määrä sekä tallentaa kyseiset määrät siihen muotoon, että ne ovat helposti käsiteltävissä varsinaisessa tarjouslaskentasovelluksessa.

6.1 Hinnoittelusovelluksen perusrakenne

Oliotietokannat, jotka on kehitetty STEP-tietomalleja varten, ovat suhteellisen hinnakkaita. Se on yksi syy, jonka vuoksi halusin rakentaa sovelluksen alusta asti itse. Käytin M.Nour:n & K.Beucke:n kehittämää tapaa ladata P21-formaattinen tieto kolmeen erilliseen taulukkoon, josta tieto vietiin edelleen oliotietokantaan (Nour & Beucke, 2005). Koska EXPRESS-mallin mukaisesti ilmaistu IFC-tietomalli ei sisällä moniperintää, onnistuu IFC-skeeman mukaisten luokkien väliset suhteet ilmaista myös sellaisilla ohjelmointikielillä, kuten esimerkiksi Java (tai C#) (mukaillen Eastman, 1999). Kyseisissä kielissä ei ole myöskään mahdollista määritellä moniperintää (toisin kuten esimerkiksi C++:ssa), vaikkakin niissä voidaan tarvittaessa implementoida monta rajapintaa (mukaillen Eastman, 1999)

Nour & Beucken-algoritmissa taulukon alkiot on viety Java-kielen vastaaviin tietorakenteisiin. Työssäni käytin kuitenkin C#-kieltä. C#-kielessä tietorakenteet ovat aika lähellä Javan vastaavia, joten olioiden muuttaminen C#-kielelle taulukon 7 mukaisesti oli suhteellisen helppoa. Sovelluksessani olen käyttänyt ainakin toistaiseksi pelkästään aikaista sidontaa. Siten rakennan vaihe kerrallaan sovellusta kirjoittaen skeemassa määritellyn entiteetin vastaavaksi C#-kielen luokaksi. Jotta näin voidaan tehdä, ”haittapuolena” on se, että ohjelmoijan täytyy koodia kirjoittaessaan tietää suhteellisen tarkasti mitä EXPRESS-skeemassa on esitetty.

6.2 Algoritmi

STEP-tiedosto (Osa 21) on muotoa:

#olion tunniste = IFCxxxx(arvo1, arvo2, arvo3, … arvo N);

Olion tunniste on maksimissaan yhdeksän numeroa pitkä yksilöllinen tunniste oliolle. IFCxxx kuvaa entiteetin nimeä EXPRESS skeemassa. Arvot arvo1, arvo2, arvo3, … arvo N kuvaavat edellä mainitun entiteetin attribuutteja, sisältäen myös viittauksia muihin elementteihin. Esimerkkinä on osa ArchiCad:n tekemästä IFC-muotoisesta tiedostosta. Materiaalin taso koostuu materiaalista ja paksuudesta. Materiaalin tasojoukko materiaalien tasoista, ja sen käyttö on kuvattu IFCMATERIALLAYERSETUSAGE-oliossa (sisältää muun muassa tasojoukon linjan kohdan=100.):

#268437031= IFCMATERIAL('Default Wall');
#268437033= IFCMATERIALLAYER(**#268437031**,200.,$);
#268437035= IFCMATERIALLAYERSET((**#268437033**),'Basic Wall:Wall 1');
#268437037=
IFCMATERIALLAYERSETUSAGE(**#268437035**,.AXIS2.,.NEGATIVE.,100.);
#268437043=
IFCWALLSTANDARDCASE('1yfo$GLYvBcPyGlEsOcDZs',#268436972,'Basic Wall:Wall 1:1107',$,'Basic Wall:Wall 1:896',#268437041,#268437099,'1107');
#268437102=
IFCRELASSOCIATESMATERIAL('30Ml_tjpvF_RzWxLKV_1YX',#268436972,$,$,(**#268437043**),**#268437037**);

Tason käyttö ja seinäolio (IFCWALLSTANDARDCASE) ovat sidotut toisiinsa IFCRELASSOCIATESMATERIAL –oliossa.

Algoritmissa olion tunnisteista tehdään ensimmäinen taulukko, jossa sijaitsevat kaikki IFC-elementit. Jokainen ensimmäisen taulukon IFC-elementti osoittaa taulukkoon, jossa sijaitsevat argumentit(arvo1, arvo2, arvo3, …arvoN), joita tarvitaan kyseisen olion muodostamiseen. Kolmas taulukko muodostetaan sellaisista argumenteista, joissa parametrina on jokin toinen IFC-elementti. Kaikki kolme taulukkoa on kuvattu kuvassa 12. (Nour & Beucke, 2005)

Kuva 12. Kolme taulukkoa, jotka viittaavat toisiinsa yllämainitussa järjestyksessä (Nour & Beucke, 2005)

Kun IFC-tiedosto käydään läpi, muodostetaan edellä mainitut taulukot. Sen jälkeen on helppo etsiä ja käydä läpi rakenteet eri olioiden välillä. Sovelluksessa eri EXPRESS-elementit voidaan muuttaa taulukon 7 mukaisesti.

Taulukko 7. EXPRESS-elementtien muunto C#-kielen rakenteiksi (mukaillen Eastman, 1999)

EXPRESS-elementti	**olio STEP P21**	**C#-kielessä**
ARRAY	lista	List
BAG	lista	List
BINARY	boolean	bool
ENTITY	entiteetin olio	class
ENTITY attribuuttina	entiteetin instanssin nimi	referenssi objektiin
ENUMERATION	lueteltu vakio	class
INTEGER	integer	int
LIST	lista	List
LOGICAL	lueteltu vakio	class
NUMBER	reaaliluku	double
REAL	reaaliluku	double
SELECT	entiteettinä	class (aikainen sidonta)
SET	lista	List
STRING	String	String
TYPE	entiteettinä	class (aikainen sidonta)

6.3 Tietokanta

Tietokantana toimii kevyt oliopohjainen tietokanta, johon oliot tallennetaan XML-muodossa. Tietokannan koodaamisesta on otettu mallia TechRepublic:n artikkelista: ”Developing an obect oriented database in less than 140 lines of C#” (Smith, 2007).

Tietokannasta puuttuu monen käyttäjän yhtäaikainen mahdollisuus käyttää tietokantaa, mutta siihen ei nyt tässä tapauksessa sovelluksessa ollut tarvettakaan. Tietokanta on kuitenkin suhteellisen helppo muuttaa monen käyttäjän tietokannaksi lisäämällä muun muassa olioiden luokkien lukitukset olioiden muokkauksiin. Monen käyttäjän tietokannaksi muuttamisen jatkokehittelyssä kannattaisi kuitenkin tehdä verkkotapahtumat arkkitehtuurisesti omaksi komponentiksi.

Tietokannalle on asetettu muutamia vaatimuksia. Tietokannan avulla täytyy voida tallentaa objektin tila ilman, että ylimääräistä tietoa asetetaan objektille tai kerrotaan tietokannalle tarkasti, kuinka objekti tallennetaan. Lisäksi täytyy olla mahdollista hakea objekti säilytyspaikasta ja välittää objekti käyttöön ilman, että käyttäjä määrittelee kentät tietokannasta objektille. Täytyy olla myös mahdollista käyttää suoraan luokkaesitystä (eli toisin kuten relaatiotietokannan SQL-kyselyissä, jotka koodataan upotettuna ohjelmointikieleen lainausmerkkien sisälle). Lisäksi luokkien käyttö pitää

olla tyyppiturvallista. Tietokannassa täytyy olla mahdollista käyttää kyselyjä lainausmerkeissä kuten edellä (mutta tyyppiturvallisesti). Pitää olla mahdollista poistaa objekteja tietokannasta optiolla ”deep delete”, joka tarkoittaa syvää poistamista, jolloin objektin mukana poistetaan muut siihen liittyvät objektit. Täytyy myös olla mahdollista päivittää objekteja tietokannassa. (Smith, 2007)

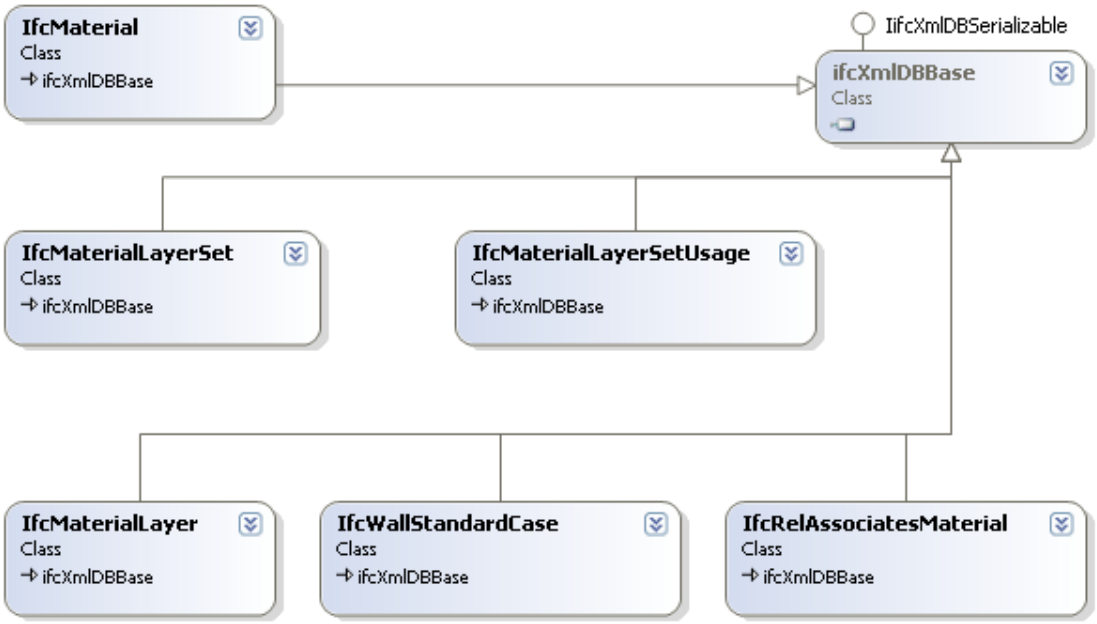
Koska työssäni pyritään saamaan objektien määrät talteen ja koska objektien ominaisuuksia pitää pystyä myös tallentamaan, sopivat edellä luetellut vaatimukset sovellukseeni. Varsinaisen IFC-tiedoston muokkaamiseen sovelluksessa ei ole tarvetta ja useimmiten kyseiset toiminnot onkin rakennettu toimivaksi CAD-sovelluksen sisältä. Rakennettu oliotietokanta on suhteellisen nopea koska koodin määrä on melko pieni ja tietokannan tietueiden haku, lisäys, muokkaus, poisto toiminnossa on käytetty .NET:n XPath-predikaattihakuja, toteutettuna delegaateilla (delegaatti toimii periaatteessa samalla tavalla kuin funktion osoitin C++:ssa, mutta delegaatti on tehty tietotyypiksi .NET:ssä). Tietoa voidaan käsitellä siten XML-dokumentista XPath-polkujen avulla.

Tieto tallennetaan XML-dokumenttiin, joka on muotoa:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Database>

</Database>
```

Sovelluksessa jokainen IFC-entiteetti muodostaa vastaavan luokan C#-kielellä. Edellinen esimerkki materiaalien liittämisestä seinään on toteutettu kuvan 13 mukaisesti.



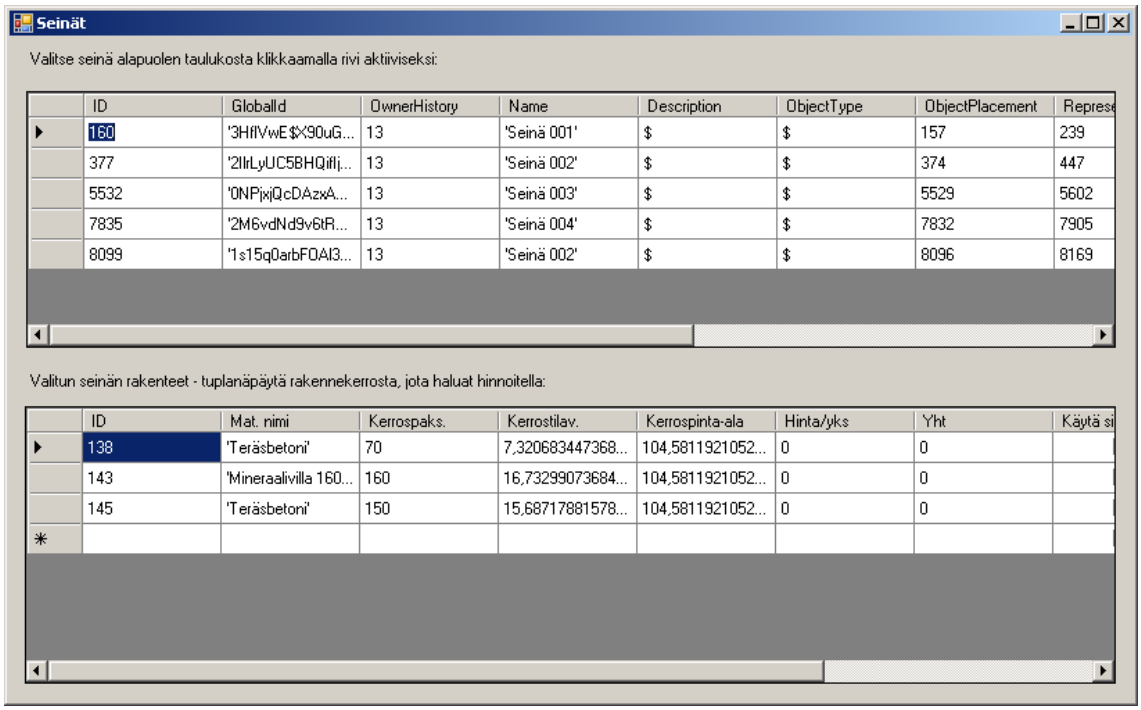
Kuva 13. UML-kaavio luokista materiaalien liittämisestä seinään

Jotta kaikki objektit olisivat tietokannan mukaisia (tai siten sellaisia, että kyseessä olevat objektit ovat joiltain osin samanlaisia), objektien luokat toteuttavat yhteisen rajapinnan IifcXmlDBSerializable:

```
public interface IifcXmlDBSerializable
{
      string ID {  get; set; }
      System.Xml.XmlNode Node { get; set; }
      void Save(bool persistData);
}
```

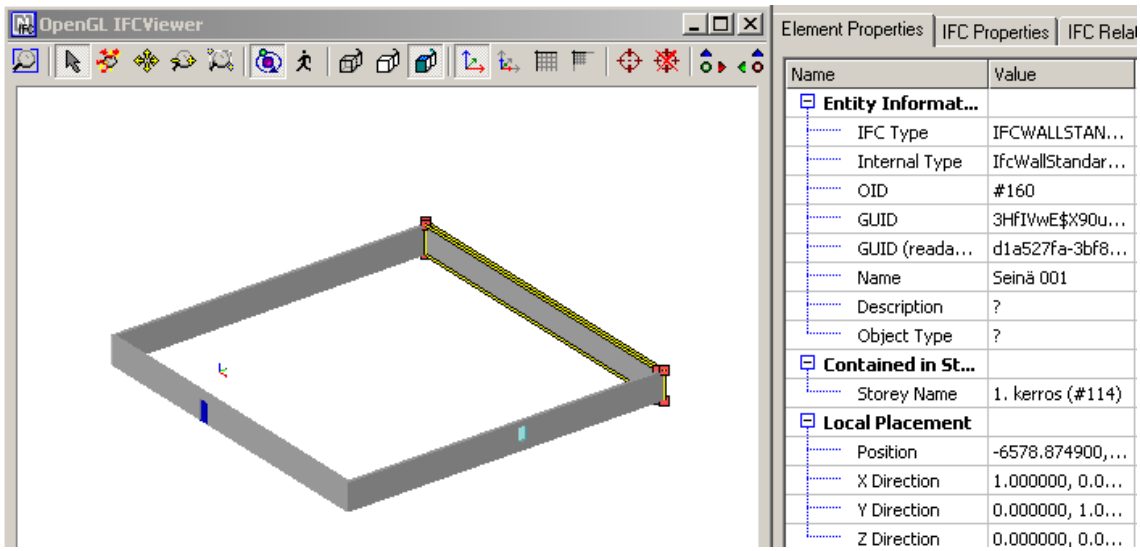
ID yksilöi eri objektit. Node on XML-dokumentin solmu (elementti). Save-metodilla voidaan tallentaa tieto XML-dokumenttiin. Luokassa ifcXmlDBBase on toteutettu tietokannan objekteille kohdistetut eri toiminnat: haku, lisäys, poisto, päivitys.

Seuraavassa on esitetty osa seinän rakenteiden määrien hakutoiminnoista. Kuva, jota käsitellään, on piirretty käyttäen Archicad versio 10:tä. Sovellukseni käyttöliittymässä haku on toteutettu kuvan 14 mukaisesti.



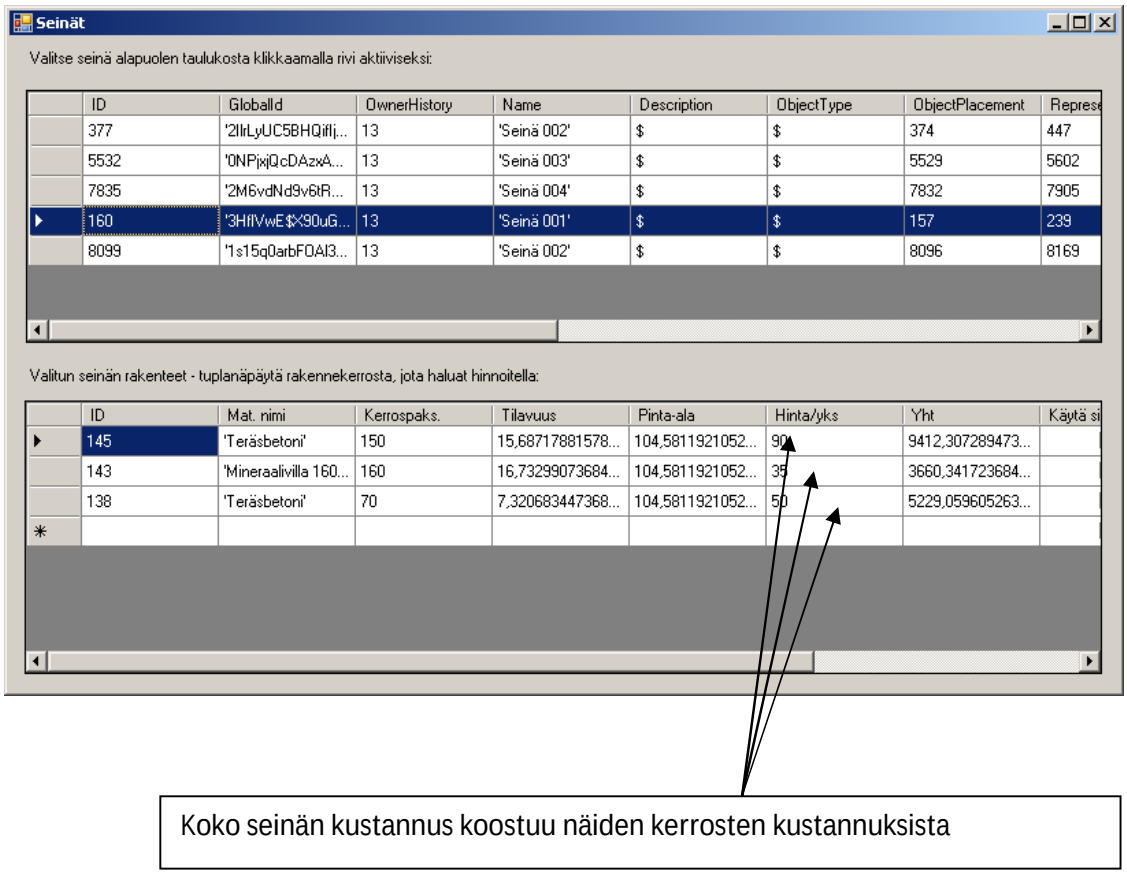
Kuva 14. Seinän eri rakenteiden tutkiminen käyttöliittymässä.

Kuvassa 15 seinää tarkastellaan Nemetschek IfcViewer:llä.



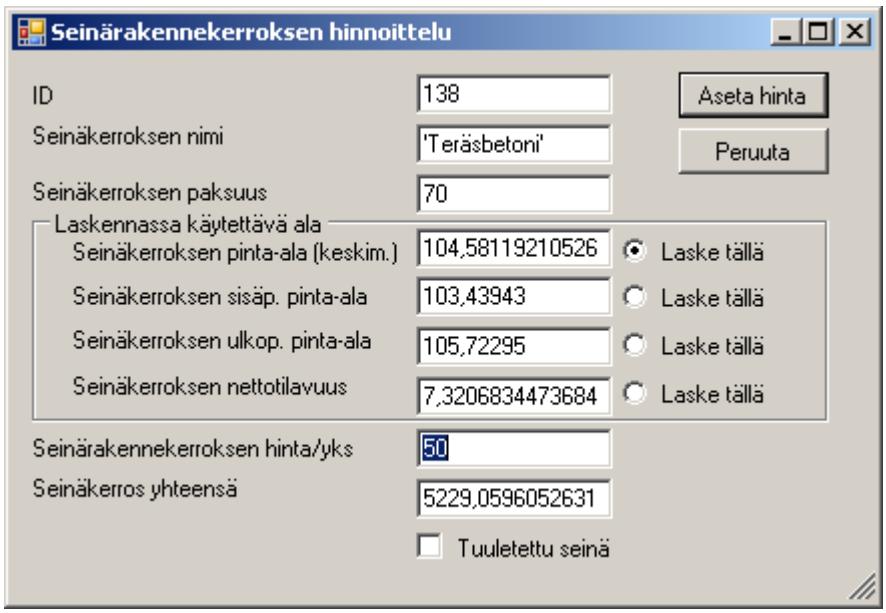
Kuva 15. Seinä tarkasteltuna Nemetschek IfcViewer:llä.

Kuvassa 16 käyn läpi seinien rakenteiden hinnoittelun rakentamassani sovelluksessa.



Kuva 16. Seinä voidaan hinnoitella rakennekerroksittain.

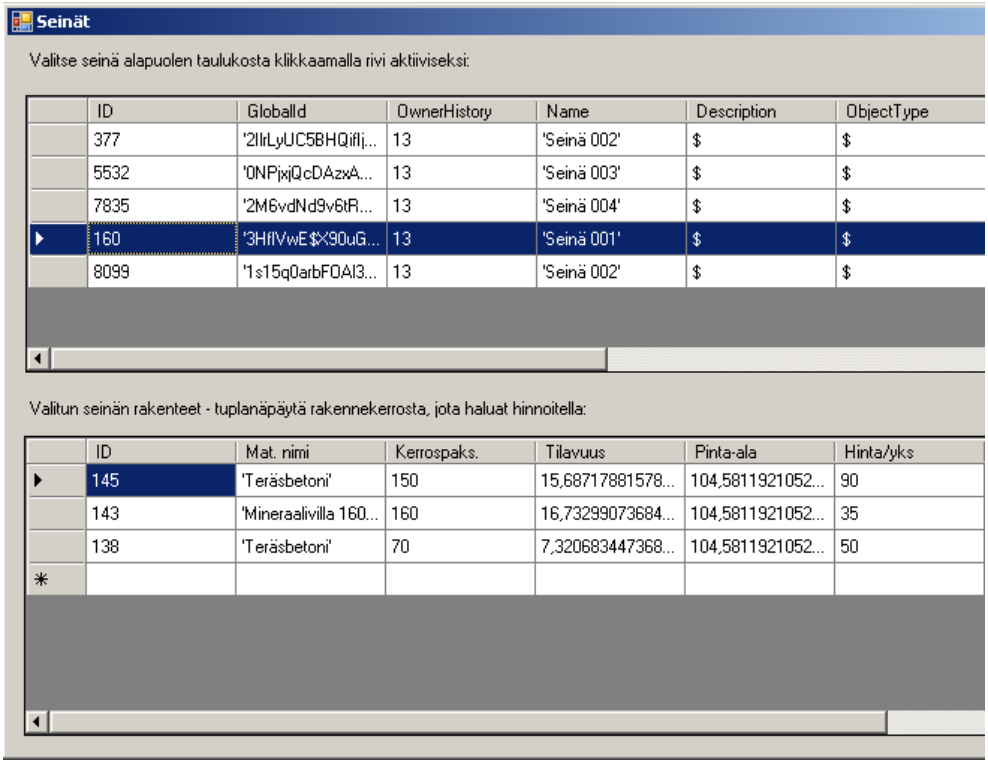
Tuplanäpäytetään seinärakennetta 138 seinästä, jonka ID on 160. Tuplanäpäytetään siis alemmasta taulukosta riviä (ID 138, materiaali Teräsbetoni). Avautuu kuvassa 17 näkyvä seinän rakenteen hinnoitteluikkuna.



Kuva 17. Seinärakenteen kerroksen hinnoittelu.

Paksussa seinässä oleellista on myös se, lasketaanko kerroksen pinta-ala rakennuksen ulkoseinän ulkopinnan, sisäpinnan tai keskilinjan mukaisesti. Tämä asia on käsitelty kohdassa: ”Laskennassa käytettävä ala”, jossa kustannuslaskija voi valita edellä mainitut vaihtoehdot.

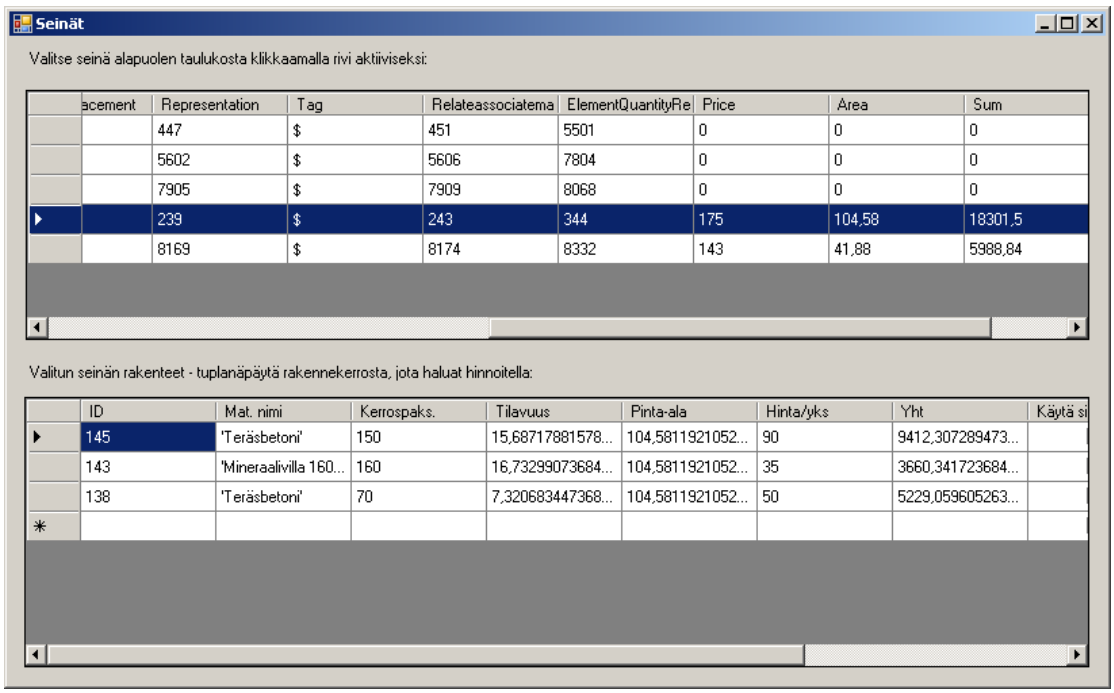
Kuvassa 18 näkyy tilanne, jossa seinä on hinnoiteltu.



Kuva 18. Seinän eri rakennekerrokset on hinnoiteltu.

Nyt nähdään, että seinä koostuu rakenteista joiden hinta on 50 €/m², 90 €/m² ja 35 €/m² (hinnat kuvitteellisia) eli yhteensä 175 €/m². Tämä kaikki saadaan aikaan sen vuoksi, että noudatin sovelluksessani rakennusalan IFC-mallia.

Kuvassa 19 nähdään lopullinen tilanne: Price-kentässä seinän kohdalla on laskettu kaikkien rakenteiden yksikköhinnat yhteensä (175), laskettu pinta-ala Area-kentässä (104,58) ja seinän kokonaishinta (18301,5).



Kuva 19. Koko seinän hinta koostuu rakennekerroksien hinnoista.

Tietyn seinän kerrosrakenteet ovat sovelluksessani haettavissa pelkistetysti seuraavasti:

```
//muuttujien esittely lomakkeen yksityisinä jäsenmuuttujina
private List<IfcRelAssociatesMaterial> ralasms = null;
private List<IfcMaterialLayerSetUsage> matlsetsu = null;
private List<IfcMaterialLayerSet> matlsets = null;
private List<IfcMaterialLayer> matls = null;
private List<IfcMaterial> mats = null;

/*Välillä muuttujien luonti ja haettu johonkin tiettyyn seinään
liittyvät materiaalisuhteet kokoelmaan ralasms*/

foreach(IfcRelAssociatesMaterial relas in ralasms)
   foreach (IfcMaterialLayerSetUsage malasetu in matlsetsu)
       if (relas.RelatingMaterial == malasetu.ID)
           foreach (IfcMaterialLayerSet malas in matlsets)
               if (malasetu.ForLayerSetRef == malas.ID)
                   foreach (IfcMaterialLayer ml in matls)
                       if (malas.ID == ml.MaterialLayerSetRef)
                       {
                           foreach (IfcMaterial ma in mats)
                               if (ml.MaterialRef == ma.ID)
                               {
                                 /*tähän koodi mitä tehdään löydetylle
                                 datalle*/
                               }
                       }
```

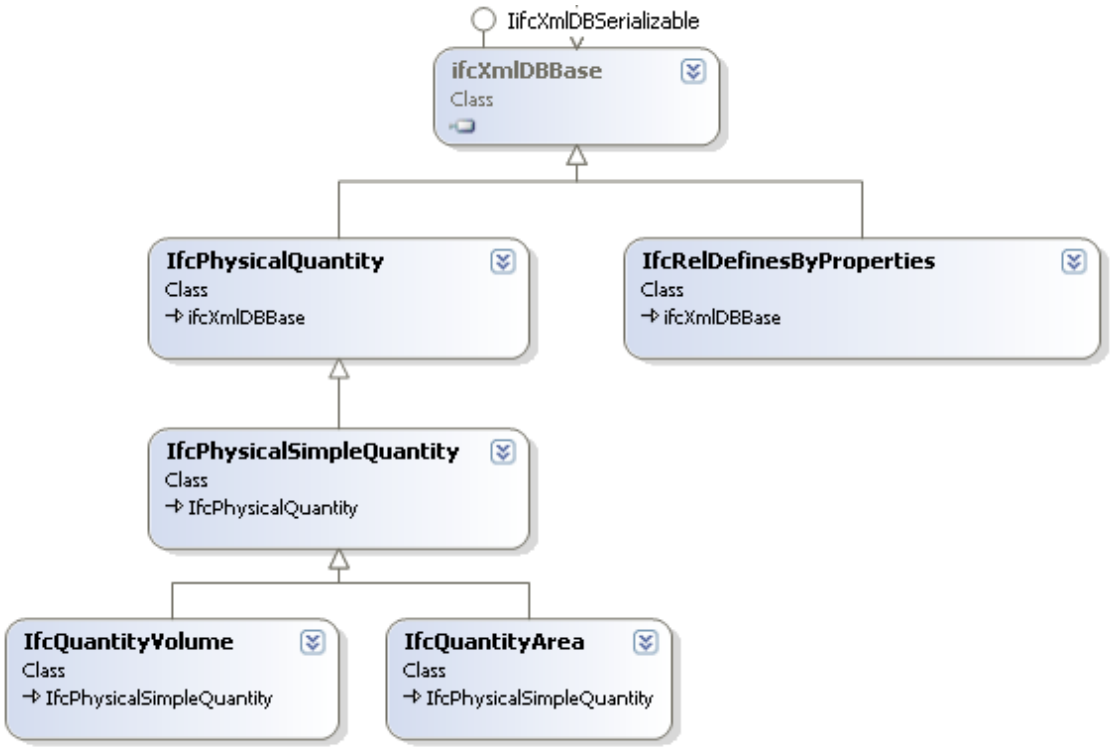
Tietokannassa vastaava XML-koodi on muodossa:

```
<XmlDB.IfcRelAssociatesMaterial>
    <IfcRelAssociatesMaterial
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <ID>243</ID>
      <GlobalId>'0jNSaUMKPCZelhUm2JV7gq'</GlobalId>
      <OwnerHistory>13</OwnerHistory>
      <Name>$</Name>
      <Description>$</Description>
      <RelatingMaterial>149</RelatingMaterial>
    </IfcRelAssociatesMaterial>
jne,…
</XmlDB.IfcRelAssociatesMaterial>
<XmlDB.IfcMaterialLayerSetUsage>
<IfcMaterialLayerSetUsage xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <ID>149</ID>
      <ForLayerSetRef>147</ForLayerSetRef>
      <LayerSetDirection>.AXIS2.</LayerSetDirection>
      <DirectionSense>.POSITIVE.</DirectionSense>
      <OffsetFromReferenceLine>0</OffsetFromReferenceLine>
    </IfcMaterialLayerSetUsage>
jne,…
</XmlDB.IfcMaterialLayerSetUsage>
<IfcMaterialLayerSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <ID>147</ID>
      <LayerSetName>'US201 betoniseinä + mineraalivi'</LayerSetName>
</IfcMaterialLayerSet>
jne,…
<XmlDB.IfcMaterialLayer>
<IfcMaterialLayer xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <ID>138</ID>
      <MaterialRef>124</MaterialRef>
      <MaterialLayerSetRef>147</MaterialLayerSetRef>
      <LayerThickness>70</LayerThickness>
      <IsVentilated>false</IsVentilated>
      <Price>0</Price>
    </IfcMaterialLayer>
</XmlDB.IfcMaterialLayer>
jne,…
<XmlDB.IfcMaterial>
<IfcMaterial xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <ID>124</ID>
      <Name>'Teräsbetoni'</Name>
    </IfcMaterial>
    <IfcMaterial xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
</XmlDB.IfcMaterial>
```

Koska tieto on XML-dokumentissa sellaisessa muodossa, että jokaisesta entiteetistä löytyy yksilöivä kenttä sekä tarvittaessa ”yhteyskenttä” johonkin muuhun entiteettiin, on tieto helppo tallentaa myös relaatiotietokantaan siten, että jokainen entiteetti

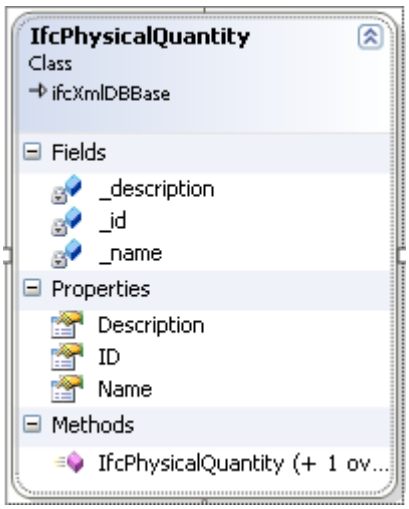
muodostaa yhden taulun rakenteen kenttinä entiteetin attribuutit. Edellä mainittuna ”yhteyskenttänä” muihin entiteetteihin voi toimia relaatiotietokannan viiteavainkenttä.

Luokissa pitää ottaa myös perintä käyttöön, sikäli kun sovellus sellaisia luokkia käyttää. Tällöin voidaan hyödyntää IFC-luokkarakenteen perintää tehokkaasti. Esimerkiksi tapauksessa että sain laskettua määrät seinärakenteiden paksuuksista, jouduin rakentamaan IFC-luokat C#-kielellä kuvan 20 mukaisesti.



Kuva 20. Seinärakenteen määrien kapseloivien luokkien perintä.

IfcPhysicalQuantity on yleinen määräluokka. IfcPhysicalSimpleQuantity on objekteille määritelty yksittäisten määrien kapseloiva luokka. Sen sisältö on kuvan 21 mukainen.



Yksityiset jäsenmuuttujat:
_id objektin yksilöivä tunniste
_name objektin nimi
_description objektin kuvaus

Julkiset ominaisuudet:
ID objektin yksilöivä tunniste (aksessorit set ja get välittävät arvon _id jäsenmuuttujalle/-muuttujalta)
Name objektin nimi
Description objektin kuvaus

Lisäksi julkinen parametrillinen muodostin on muotoa:

```
public IfcPhysicalQuantity(string id, string
_Name, string _Description)
```

Kuva 21. Luokan IfcPhysicalQuantity jäsenet.

IfcPhysicalSimpleQuantity:n muodostin on rakennettu siten, että edellä mainitut kantaluokan IfcPhysicalQuantity muodostimelle välitetään molemmilla luokilla yhteisesti käytettävät jäsenet (base on kantaluokan muodostimen kutsu):

```
public IfcPhysicalSimpleQuantity(string id, string _Name, string
_Description, string _Unit, string _ElementRef):
base(id,_Name,_Description)
```

Jos objektille on määritelty pinta-ala, käytetään IfcQuantityArea-luokkaa, jonka muodostin on muotoa:

```
public IfcQuantityArea(string id, string _Name, string _Description,
string _Unit, double _AreaValue, string _ElementRef): base(id, _Name,
_Description, _Unit, _ElementRef)
```

Jos objektille on määritelty tilavuus, käytetään IfcQuantityVolume-luokkaa, jonka muodostin on muotoa:

```
public IfcQuantityVolume(string id, string _Name, string _Description,
string _Unit, double _VolumeValue, string _ElementRef): base(id,
_Name, _Description, _Unit, _ElementRef)
```

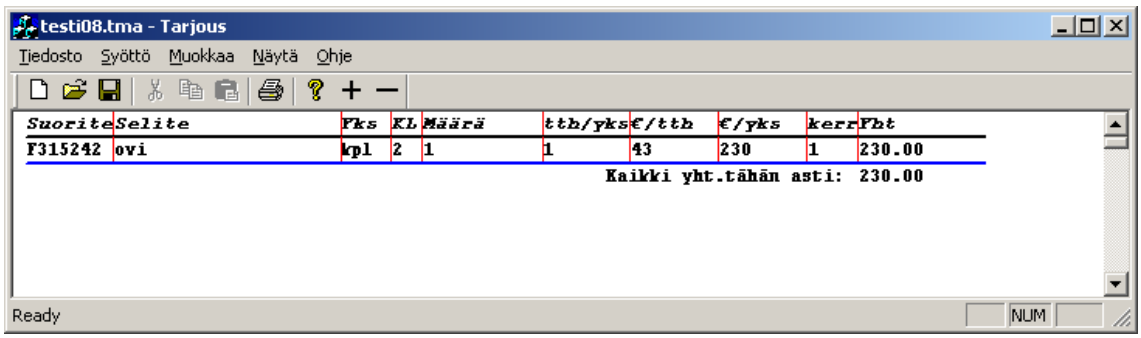
Periaatteessa olisi mahdollista mennä niinkin tarkkaan kuvaukseen, että lasketaan muun muassa betoniteräkset, betoni, raudoituskiinnikkeet, villojen kiinnikkeet IFC-mallin mukaisesti, mutta käytännössä rakentamisessa ei voida mennä tälle tasolle (se ei ole

kustannustehokasta). Tässä suhteessa rakennusala poikkeaa muusta teollisuudesta kuten esimerkiksi tapauksessa, jossa rakennetaan kännykkä. Kännykän osia on huomattavasti vähemmän verrattuna rakennettavaan taloon.

6.4 Tiedon vastaanottava sovellus

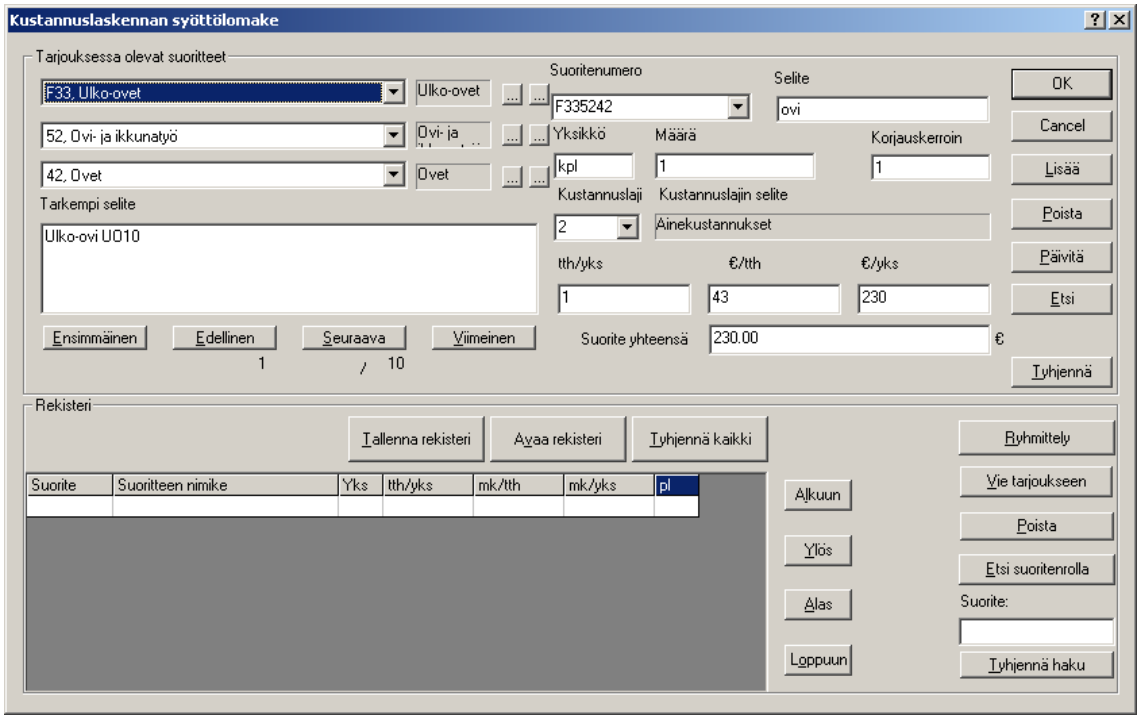
Sovellus, joka käyttää XML-muotoon luotua tietoa on kustannuslaskentaohjelmisto. Se on tehty Visual C++ 6:lla ja se käyttää Microsoftin MSXML-jäsennintä, jotta XML-tiedosto saadaan luettua. Koska MSXML:n teknologia on COM-pohjaista ja siitä ollaan nykyisin tietokantoja käsittelevissä sovelluksissa pikemminkin pääsemässä eroon, en käsittele jäsentimen koodaamista tässä erityisemmin. XML-dokumentin jäsentäminen on suhteellisen helppo toimenpide, mutta varsinkin uusissa Java ja .NET-kielissä XML-dokumentin lukeminen ja siihen tallentaminen on helppoa.

Alla kuvassa 22 on kuvattu oven tuonti XML-tiedostosta (valikosta Tiedosto, Tuo XML:stä):



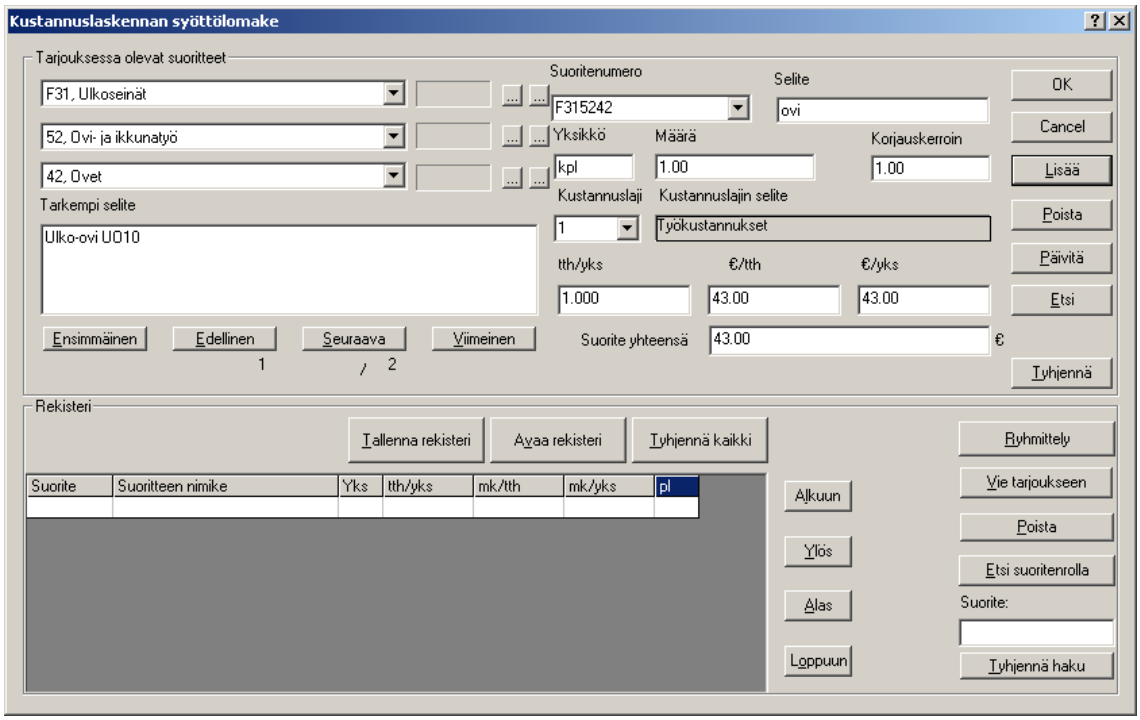
Kuva 22. Tieto tuotu kustannuslaskentaohjelmistoon.

Ovea tuplanäpäytettäessä avataan varsinainen kuvan 23 mukainen kustannuslaskentaohjelmiston hinnoitteluikkuna.



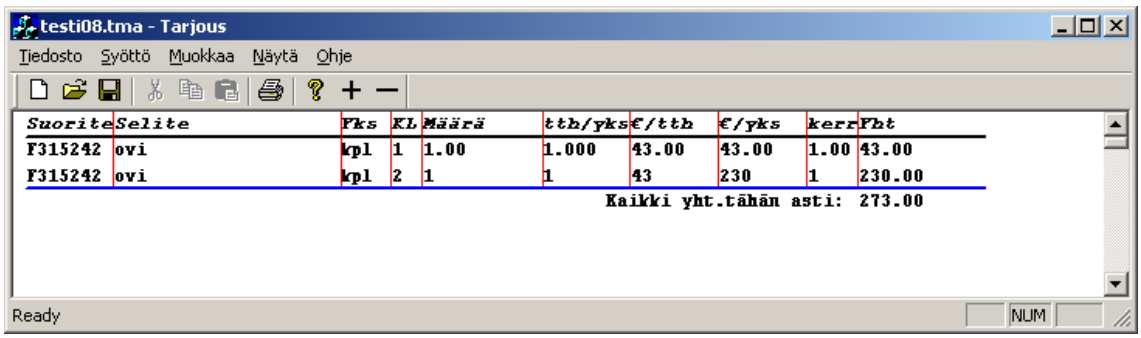
Kuva 23. Oven materiaalin hinnoittelu.

Varsinainen kustannuslaskentaohjelmisto on rakennettu niin, että kustannuksia voidaan kasata samalle suoritenumerolle (esimerkissä F335242). Lisätään samalle suoritenumerolla seuraavaksi kuvan 24 mukaisesti työkustannukset.



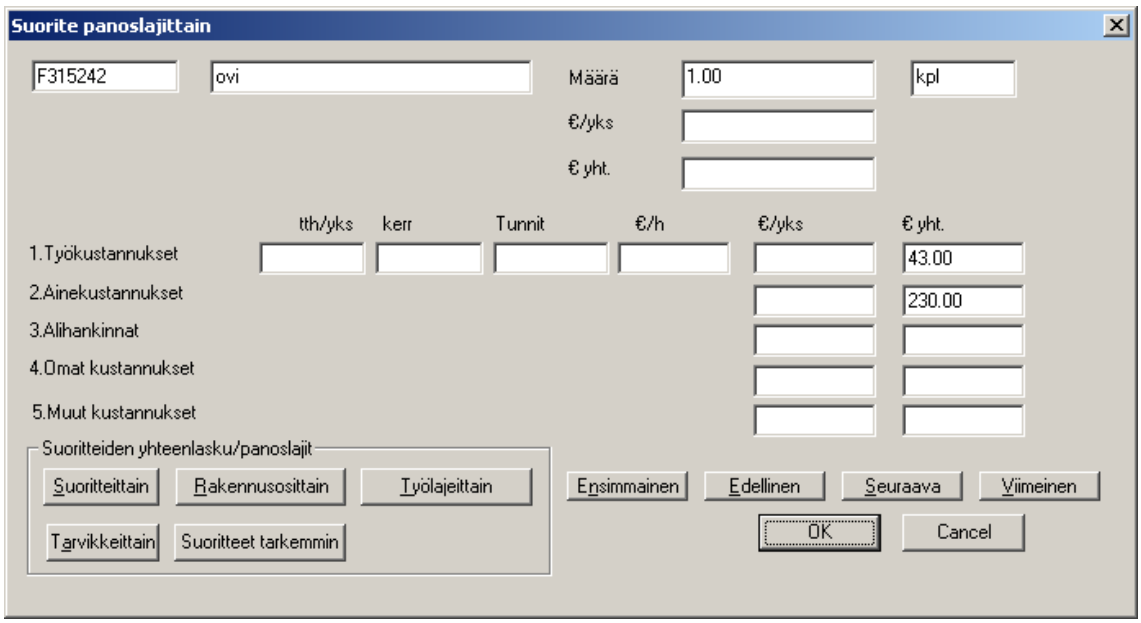
Kuva 24. Oven työn hinnoittelu.

Tarjouksessa olevat suoritteet muodostuvat Talo 90-järjestelmän mukaisesti rakennusosasta (F33=Ulko-ovet), työlajista (52=Ovi-ja ikkunatyö) sekä tarvikkeesta (42=Ovet). Sekä työ-, että materiaalikustannukset edellä käsitellyn oven mukaisesti on nyt laskettu yhteen kuvan 25 mukaisesti.



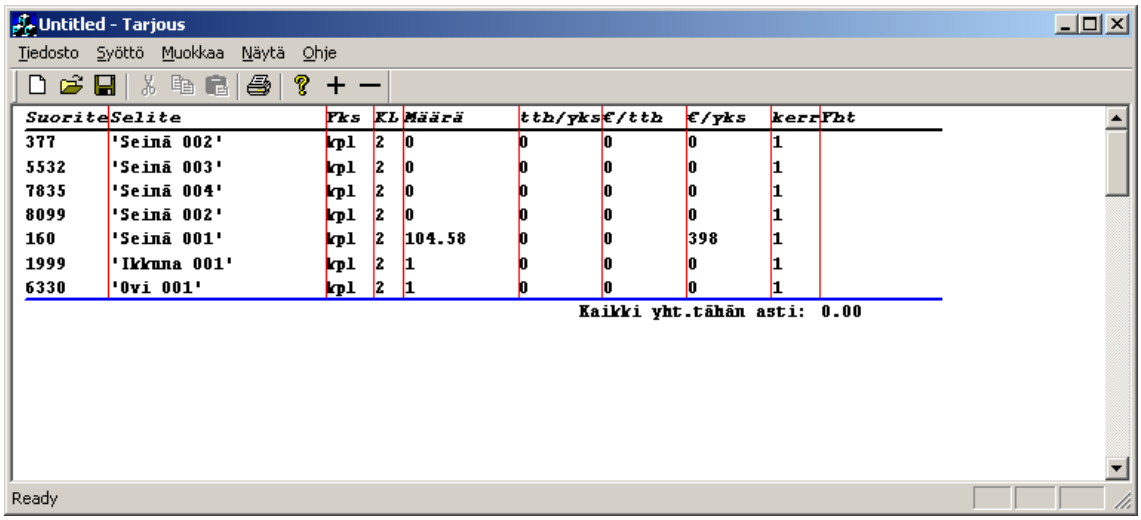
Kuva 25. Tarjous koostuu nyt tähän mennessä oven materiaalin sekä työn hinnasta.

Kun kaikki eri kustannuslajit on käyty läpi, voidaan suoritetta tutkia kustannuslajeittain kuvan 26 mukaisesti.



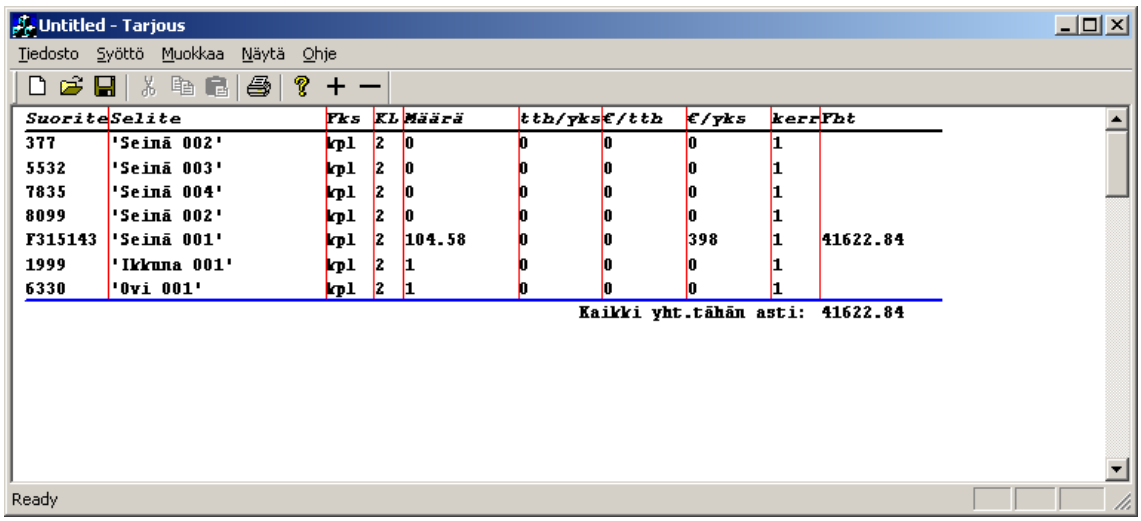
Kuva 26. Suoritetta voidaan tutkia eri panoslajeittain (=kustannuslajeittain)

Samalla tavalla voin tuoda sovellukseeni tällä hetkellä ovet, ikkunat ja seinät. Kuvassa tuotu erään pienen rakennuskohteen kaikki ovet, ikkunat ja seinät. Kuvan 27 mukaisesti vain yksi seinä (Seinä 001) on hinnoiteltu. Seinää ei ole kuitenkaan vielä tarkasteltu syöttölomakkeella (ei laskettu vielä mitään Yht-sarakkeeseen).



Kuva 27. Yksi seinä hinnoiteltu ja tuotu varsinaiseen kustannuslaskentaohjelmistoon.

Seuraavaksi päivitetään seinän laskennalliset tiedot kuvan 28 mukaisesti. Tätä ennen suoritenumero on objektin id:n arvo.



Kuva 28. Seinä käyty päivittämässä materiaalin osalta

Rakennusalalla ei ole olemassa PDM (tuotetiedonhallinta)-järjestelmää aivan samassa tarkoituksessa kuin sellaisilla teollisuuden aloilla, joilla valmistetaan teollisesti eri osista koostuvaa kokonaisuutta. Onkin nähty pikemmin niin, että juuri alalle kehitetty tietomalli (BIM=building industrial model) toisi ratkaisun muun muassa tiedonsiirron suhteen (eri ohjelmistojen välille). Kun esimerkiksi muovi –tai metallialalla PDM-järjestelmän avulla tietoa voidaan siirtää helposti järjestelmään rakennetun toiminnallisuuden kautta, on rakennusalalla nähty ratkaisuksi tietomallipalvelimet

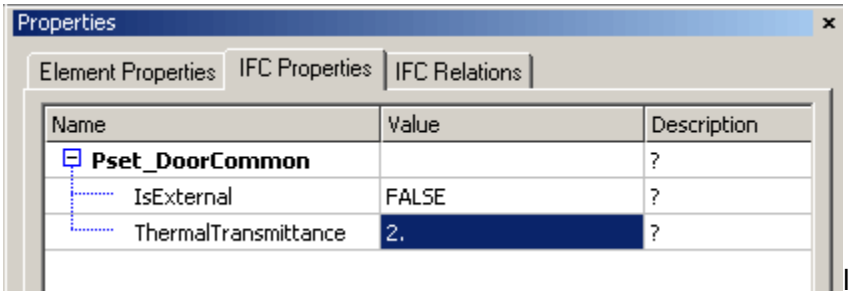
(kuvattu aikaisemmin). Niissä käyttäjä on aina tekemisissä rakennuskohteesta kuvatun 3D-mallin kanssa.

Joitakin samoja piirteitä kuitenkin rakennuskohteiden määrittelemisessä on, kuin PDM-järjestelmässä. Esimerkiksi tuoterakennetta voidaan jakaa hierarkiassa erilaisilla nimikkeistöillä. Kustannuslaskennassa käytetään erilaisia nimikkeistöjä. Esimerkiksi Suomessa on tällä hetkellä käytössä sekä Talo 90-, että Talo 2000-järjestelmä. Talo 2000-järjestelmän nimikkeistö on pyritty rakentamaan siten, että se tukisi paremmin rakentamisen tuote- ja prosessimallinnusta tietotekniikan keinoin (mukaillen Talo-nimikkeistöryhmä & Haahtela-kehitys Oy, 2008). PDM-järjestelmät on monesti rakennettu erilaisten tietokantojen päälle, ja myös kustannuslaskentaohjelmistoissa on monesti taustalla esimerkiksi relaatiotietokanta. Rakennusalalla ei ole kuitenkaan vielä laajalti, ainakaan Suomessa, ollut käytössä vielä sellaista menetelmää, että tuoterakenteen nimikkeet olisivat samat rakennuksesta CAD:llä tehdyissä kuvissa että kustannuslaskentaohjelmistoissa. Yleensä rakennuskohteesta tehdyissä kuvissa ei ole käytetty tunnuksia eri osille. Tulevaisuudessa tilanne saattaa tältäkin osin muuttua (esimerkiksi ArchiCad-ohjelmistossa littera voidaan merkitä objektille). On mahdollista, että kehitetään kansainvälinen standardi juuri tähän tarkoitukseen. Talo 2000-järjestelmä on jo nyt osiltaan ISO 12006-2 yhteensopiva ja soveltuu kansainvälisiin hankkeisiin (Talo-nimikkeistöryhmä & Haahtela-kehitys Oy, 2008). Sovellukseni nimikkeistö on rakennettu toistaiseksi Talo 90-järjestelmän mukaisesti.

6.5 Tiedon tallentaminen tuotemallin mukaiseen tiedostoon

Miten sitten laskettu hinta – mihin sen arvo kannattaa sijoittaa, että sitä voidaan käyttää eteenpäin muissa IFC:tä tukevissa ohjelmistoissa? Sovelluksessani ei sitä ominaisuutta tarvittu, mutta käyn asian läpi, koska se on oleellista, jos tieto halutaan esittää samassa IFC-tiedostossa, ja näin jakaa useille eri ohjelmistoille ja osapuolille. Useissa CAD-ohjelmistoissa voidaan tehdä itse niin sanottuja ominaisuusjoukkoja (Property Sets), joihin voidaan sijoittaa itse luotuja ominaisuuksia esimerkiksi IfcPropertySingleValue-luokan avulla. Voidaan luoda esimerkiksi kenttä, johon sijoitetaan lasketun seinän hinta.

Kuvassa 29 on tarkasteltuna kaikille oville yhteinen ominaisuusjoukko Pset_DoorCommon, jossa on määritelty 2 ominaisuutta: IsExternal (onko kyseessä ulko-ovi) ja ThermalTransmittance (lämpöjohtavuuskerroin).

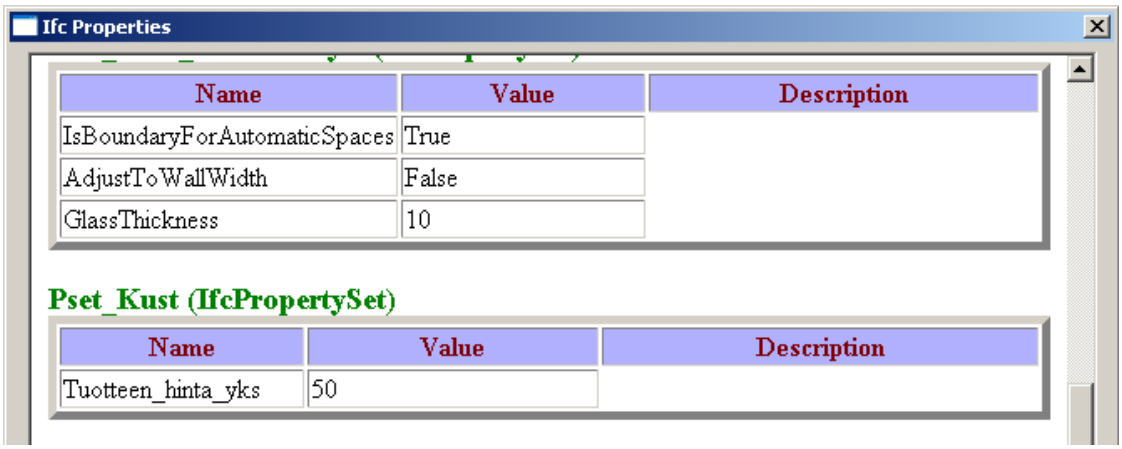


Kuva 29. Ovien yhteinen ominaisuusjoukko tarkasteltuna Nemetschek IfcViewer:llä.

Jos ominaisuusjoukkoja halutaan tehdä itse ja näin viedä tietoa eteenpäin eri sovellusten välillä P21-formaatin tiedostossa, voidaan se tehdä koodaamalla itse ominaisuusjoukko ja ominaisuudet (Yanga, 2002). Seuraavassa esimerkissä on luotu itse tehty ominaisuusjoukko Pset_kust, johon on asetettu yksi ominaisuus: Tuotteen_hinta_yks (=tuotteen hinta yksikköä kohti reaalilukuna). Rivillä 745 on määritelty suhde ikkunaobjektin (IFCWINDOW) ja edellä mainitun ominaisuusjoukon välille. Ominaisuusjoukko ja suhde pitää yksilöidä yksilöllisellä tunnuksella (1.parametri). Ohjelmointikielissä löytyy erilaisia metodeita, joilla kyseisen tyyppinen parametri voidaan luoda dynaamisesti. Toinen vaihtoehto on, että käytetään jo aiemmin mainittua IFD-kirjastoa apuna.

#24=IFCOWNERHISTORY(#23,#21,$,.NOCHANGE.,$,$,$,1205859038);
…
#244=IFCWINDOW('BmnNEQmCm6G000001_qtS0',#24,'WINDOW Standard(1)',",",#247,#384,$,1510.,1010.);
…
#742=IFCRELCONNECTSPATHELEMENTS('0BB9C4s2HANPujhtB5Wevz',#24,$,$,$,#394,#184,(1),(1),.ATEND.,.ATSTART.);
#743=IFCPROPERTYSINGLEVALUE('Tuotteen_hinta_yks',$,IFCREAL(50.),$);
#744=IFCPROPERTYSET('0BB9C4s2HANPujhtB5Wevy',#24,'Pset_Kust',$,(#743));
#745=IFCRELDEFINESBYPROPERTIES('1Yl5Uiyun7f8NkhOFqdvdj',#24,$,$,(#244),#744);
ENDSEC;
END-ISO-10303-21;

Kuvassa 30 edellä luotu ominaisuus ja ominaisuusjoukko tarkasteltuna DDS-CAD Viewer:llä.



Kuva 30. Itse tehty ominaisuusjoukko tarkasteltuna DDS-CAD Viewer versio 6.4:llä.

Samalla tavalla eri objekteihin voidaan liittää mitä tahansa muuta tietoa. Esimerkiksi ikkunaan voitaisiin lisätä aikataulu (esimerkiksi ominaisuudet asennus_alkaa, asennus_loppuu). Monissa CAD-ohjelmistoissa ominaisuusjoukkoja voidaan luoda ohjatusti. Oleellista IFC-projektissa on kuitenkin se, että eri ohjelmistojen välisen tiedonsiirron ominaisuusjoukoista ja ominaisuuksista sovitaan etukäteen ennen projektin aloittamista. Standardia kannattaa noudattaa mahdollisimman paljon. Tietoa voidaan käyttää siten mahdollisesti uudelleen helpommin (esimerkiksi tiedon analysoinnissa). Esimerkiksi IFCPRODUCT:ssa on määritelty kenttä IFCCOST, jolloin tuotteen hinnan suhteen kannattaa käyttää kenttää cost. Esimerkiksi Revit-ohjelmistossa on tehty jo oletuksena näin.

7 JOHTOPÄÄTÖKSET

Pyrkimys muuttaa tietomallien määrittely XML-skeeman muotoiseksi, on karsinut tietomallin monimuotoisuutta. Onkin huomattu, että EXPRESS on parempi kieli jossain suhteessa määritellä tietomalli STEP-standardin mukaisesti. Jää nähtäväksi, miten tarkkaan EXPRESS-malli voidaan toteuttaa tulevaisuudessa semanttisen webin tekniikoilla. Sovellustani voisi parantaa siten, että EXPRESS-tietomalli olisi ladattavissa osaksi sovellusta ja siten jatkossa mahdollistettaisiin uusien tietomallien yhteensopivuus. Eli kun tulee uusi IFC:n versio, se voitaisiin ladata sovellukseen, jolloin eri entiteettien attribuutit luettaisiin EXPRESS-tiedostosta ja funktiokutsut olisivat sovelluksessa sen mukaisia. Toisin sanoen: tarvitaan EXPRESS-jäsennin (parseri), jonka avulla *.exp tiedoston tietomalli luetaan ensin ja muodostetaan sitten sen mukaisesti entiteetit (sovelluksessa luokat). Siten sovellusta voisi kehittää muuttamalla aikaisen sidonnan sekoitetuksi sidonnaksi (mixed binding). Tätä työtä aloittelinkin jo, mutta tähän versioon sovellusta en ehtinyt vielä sitä liittää. Vielä parempi olisi, jos tietomalli voitaisiin lukea XML-muodossa ifcOWL:nä. Näin sovellus toteuttaisi semanttista yhteistoiminnallisuutta (semantic interoperability). Koska XML-dokumentteihin löytyy paljon hyviä ja helposti käytettäviä jäsentimiä, XML-muodossa esitetty tietomalli on helpommin luettavissa ja tulkittavissa.

Jos verrataan skeemoja IFC2xFINAL ja IFC2x3 näyttää siltä, että sovelluksessa tarvittavan IFC-tietomallin olioiden muodostamisessa on tapahtunut erittäin pieniä muutoksia. Jotta sain sovelluksen toimimaan jälkimmäisessä skeemassa, ei tarvinnut tehdä kuin muutamia vähäisiä muutoksia. IFC-tietomalli alkaakin olla jo sen verran vakiintunut, että perusluokkien osalta tapahtuu enää pieniä muutoksia. Pikemminkin on niin, että IFC:tä laajennetaan muille alueille (kuten esimerkiksi rakennusten sähköteknisiin kuvauksiin). Jotta pystyin menemään sovelluksessani tarkempiin yksityiskohtiin, jouduin tallentamaan IFC-tiedoston tietoa kuitenkin tietokantaan, jotta pääsin kustannuslaskentaohjelmiston vaatimaan tarkkuuteen. XML-tietokanta on joustava hyvinkin erilaisten tietojen tallennuspaikkana ja sopii siinä mielessä hyvin CAD-järjestelmän tietojen analyysiin.

Pyrkimys esittää IFC:n mukaisesti koko CAD-kuva XML-muotoisesti (esimerkiksi ifcxml-tiedostomuodossa), kasvattaa tiedoston kokoa huomattavasti. Koska CAD-kuvissa käytettävän tiedon määrä on erittäin suuri (esimerkiksi talojen piirustuksissa), kannattaa tietoa käsitellä STEP P21-formaatin mukaisesti (ifc -päätteinen tiedosto). Tiedon lukemiseksi STEP P21-tiedostosta tämä työ osoittaa, että Nour & Beucken algoritmi ja Charles Eastman:n ohjeet EXPRESS-kielen kuvauksen mukaisista muutoksista ohjelmointikielen vastaaviksi elementeiksi toimivat myös C#-kielellä tehdyssä sovelluksessa. Tieto voidaan muuttaa jatkokäsittelyssä haluttuun muotoonsa. Sovelluksessani se muutettiin XML-muotoon, vieläpä siinä muodossa, että se on helppo tallentaa myös relaatiotietokantaan. Relaatiotietokannat ovat nykypäivänä käytetyin tietokantamuoto ja niille on ominaista, että tieto tallennetaan aina tauluihin, jolloin tieto on helposti sieltä luettavissa ja ladattavissa. Niin sanotuissa 3–tasosovelluksissa (eräs ohjelmistoarkkitehtuurin sovelluksen rakennetta kuvaava tekniikka) on kolme tasoa: tietokannan toiminnallisuutta määrittelevät luokat, käyttöliittymän toiminnallisuutta määrittelevät luokat sekä niin sanottu liiketoiminnallinen taso, johon määritellään varsinainen sovelluksen toimintalogiikka. Mielestäni viimeisen tason mukaisesti IFC:n mukaista tietoa kannattaisikin käsitellä juuri sillä tasolla objekteina ja kokoelmina, kuten olen edellä esittänyt sovelluksestani. Jos tieto halutaan tallentaa tietokantaan, se voidaan tallentaa oliotietokantaan tai relaatiotietokantaan. Koko tietomallin mukaista tietoa ei kannata käsitellä sovelluksissa, jos sitä ei tarvita. Sovelluksessani tarvittiin vain kuvasta saatua määrätietoa, jolloin muita luokkia ei tarvitse ainakaan suoranaisesti käsitellä.

Sovellukseni rakenne osoittautui kaiken kaikkiaan yllättävän joustavaksi. Luokkia voi rakentaa vaihe kerrallaan eteenpäin ja haetun tiedon voi tarkistaa helposti XML-tiedostosta. Tieto voidaan lisäksi tarvittaessa tallentaa erittäin helposti nyt .NET-kirjaston DataSet-olioihin ja tarvittaessa eteenpäin niin sanotun Data Adapter-olion kautta tietokantaan. .NET:ssä pystytään DataSet-olion sisältö palauttamaan myös hajautetussa sovelluksessa SOAP-muodossa erittäin helposti. Siinä mielessä näen, että tallennus XML-tiedostoihin oli onnistunut valinta. Koska tietokantasovellukseni on arkkitehtuurisesti jaettu komponenttiin, joka käsittelee tietokantaa, että erilliseen käyttöliittymäsovellukseen, pystyn siirtämään tietokantakomponentin web-sovellukseen helposti. Silloin myös XML:n vahvuudet tiedonsiirtoformaattina pääsevät oikeuksiinsa.

Tulevaisuuden kehitystyössä voidaan käyttää varmasti hyväksi sitä, että jo nyt moniin tietokantoihin voidaan tehdä ”XML-kyselyjä” suoraan (niin sanotulla XQuery-kielellä sekä XPath-syntaksin mukaisesti). Lisäksi, kun XMLSchemaa käyttämällä voidaan tiedonsiirto suorittaa sekä lähetyksessä että vastaanotossa samanmuotoisesti, tiedonsiirrossa ei voi tulla enää niin paljon virheitä. Jossain vaiheessa, kun sovelluksessa on mielestäni tarpeellinen määrä siirrettäviä objekteja koodattu, luon XML-dokumenttia varten skeeman XMLSchema -muotoisesti. Nykyisissä kehitysympäristöissä se voidaan tehdä myös automaattisesti XML-dokumentin perusteella. Kokeilin myös ifcxml-tiedoston lukemista ja tulkintaa, ja tulin siihen tulokseen, että jos kaikki tieto laitetaan xml-tiedostoon ja vielä monisisäkkäisinä elementteinä, muuttuu sovelluksen suorituskyky ratkaisevasti. Tiedostokoko vähänkin suuremmassa CAD-kuvassa saattaa olla jo erittäin iso. Tietyissä sovelluksissa käyttämällä SAX-jäsennintä ifcxml-tiedoston lukeminen ja tulkinta voi olla kuitenkin riittävän nopeaa. Uskon, että lähitulevaisuudessa P21-formaatti IFC-muotoisessa tiedonsiirrossa tulee pysymään vielä pitkään.

Jos tietoa halutaan tallentaa IFC-tietomallin mukaisesti siten, että tieto jaetaan useiden sovellusten kesken, on IFC:ssä siihen rakennettu sopiva menetelmä: ominaisuusjoukot (Property Sets). Hienoa kyseisessä menetelmässä on se, että ominaisuusjoukkoja ja niihin ominaisuuksia voidaan tehdä itse ja rakentaa näin sopivia kokonaisuuksia projektispesifisesti. Moniin CAD-ohjelmistoihin kuitenkin ei ole ainakaan vielä saatu rakennettua kyseisiä toimintoja helpoksi käyttää. Toivon mukaan tässä ei tapahdu samanlaista oman järjestelmän suojelua, kuin neutraalin tiedoston tiedonsiirrossa, jolloin esimerkiksi IGES-tiedostomuodossa eri järjestelmien valmistajat suojasivat omaa tiedostoformaattiaan ja järjestelmäänsä. Jo nyt on nähtävissä se, että jotkin suurempien valmistajien tuotteet eivät esimerkiksi näytä itse tehtyjä ominaisuusjoukkoja lainkaan, puhumattakaan että ominaisuusjoukoissa olevia ominaisuuksien arvoja voisi päivittää tai ominaisuuksia voisi lisätä (ainakaan helposti).

Tietomalliserverin toimintaa en tässä työssä tutkinut tekniseltä kannalta tarkemmin ja lisäksi tietoa kyseisestä asiasta löytyy vielä niukalti. Sovellukseni XML-tietokanta olisi periaatteessa tarvittaessa muutettavissa siten, että se tukisi hajautettujen ja samanaikaisten sovellusten käyttöä. Tietomalliserverit tuovat kuitenkin niin paljon etuja

(kuten esimerkiksi voidaan hallita käyttäjien oikeuksia, rooleja, versiointia ja omistajuutta sekä hallita monimutkaisia projekteja keskitetysti) mukanaan, että viime kädessä vähänkin suuremmissa projekteissa niiden käyttö on järkevää.

XML on tuonut ”puuttuvan” osan ohjelmointikieliin monessakin suhteessa. Kuten luvussa kaksi tuli esille, olio-ohjelmointikielet muuttivat sovellusten rakennetta arkkitehtuurisesti. Ongelmana aikaisemmin oli se, mitä tehdään luokista tehdyille olioille, jotta rakenteellinen tieto saadaan luettua, tallennettua ja muokattua helposti. Sovelluksen luokkarakenne voi olla tehty vaikka miten hyvin ja kapseloidusti, mutta ongelmana on ollut nimenomaan instanssien käsittely. XMLSkeema toimii rakenteena esimerkiksi tietokannan ja ohjelmointikielen välillä: näin voidaan sopia yhteinen rajapinta ja rakenne, jonka mukaisesti instanssit voidaan johtaa eteenpäin. Tämä mahdollistaa luotettavamman ja helpomman tiedonsiirron sovellusten välille. Semanttinen web tulee olemaan tulevaisuudessa sopiva ratkaisu tyydyttämään tiedonsiirtotarpeet joita webissä tarvitaan. Tällä hetkellä kuitenkin sovelluksia on vielä tehty erittäin vähän ja niitä ei vielä voi siinä mielessä arvioida luotettavasti.

LÄHTEET

Adachi, Yoshinobu 2001-2002, IMSvr: IFC Model Server project by Secom (Japan) and VTT (Finland). [verkkosivusto]. [viitattu 20.2.2008]. osoite: http://cic.vtt.fi/projects/ifcsvr/

Adachi, Yoshinobu, 2005, Introduction of IFC Model Server. [verkkodokumentti]. [viitattu 20.2.2008] Saatavissa: http://cic.vtt.fi/vera/Seminaarit/2002.04.24_IAI_Summit/Adachi.pdf

Berners-Lee Tim, 2000, Semantic Web Wedding Cake (or “layer cake”) in a conference talk, XML 2000 conference [XML 2000 konferenssi]. Washington DC., [viitattu 20.2.2008].

Bindu R. Rao, 1994, Object-oriented databases: technology, applications, and products. McGraw-Hill Companies. 253 s. ISBN 0-07-051279-5

Bjørkhaug Lars ja Bell Håvard, 2007, IFD in a Nutshell. [verkkosivusto]. [viitattu 20.2.2008]. Saatavissa: http://dev.ifd-library.org/index.php/Ifd:IFD_in_a_Nutshell

BLIS: View definitions. [verkkosivusto]. [viitattu 20.2.2008]. Saatavissa: http://www.blis-project.org/views/index.htm

Eastman, Charles M., 1999. Building Product Models: Computer Environments Supporting Design and Construction. CRC Press LLC. 411 s. ISBN 0849302595

EPM: Express Data Manager by EPM Technology, 2003, [Tuotteen verkkosivusto]. [viitattu 20.2.2008]. Tietoa URL: http://www.epmtech.jotne.com/products/

Evjen Bill, Sharkey Kent, Thangarathinam Thiru, Kay Michael, Vernet Alessandro, Ferguson Sam, 2007, Professional XML. Wiley Publishing. 856 s. ISBN 978-0-471-77777-9

Gehre Alexander, Katranuschkov Peter, Wix Jeffrey, Beetz Jakob, 2006, IST-004664, InteliGrid, Interoperability of Virtual Organizations on a Complex Semantic Grid. [verkkodokumentti]. [viitattu 20.2.2008]. Saatavissa: http://www.itcon.org/data/works/att/2007_30.content.05471.pdf

Hietanen, J., 2005. Tietomallit ja rakennusten suunnittelu. Filosofinen selvitys tieto- ja viestintätekniikan mahdollisuuksista. Helsinki: Rakennustieto Oy. 95 s. ISBN 9516827837

Houbaux, Patrick, 2003, SABLE: Simple Access to Building Lifecycle Exchange project by Eurostep. [Projektin verkkosivusto]. [viitattu 20.2.2008]. Tietoa URL: http://www.blis-project.org/~sable/

IAI (International Alliance of Interoperability), 2000, Industry Foundation Classes - Release 2x IFC Technical Guide, Enabling Interoperability in the AEC/FM. [verkkodokumentti]. [viitattu 20.2.2008]. Saatavissa: http://www.iai-international.org/Model/documentation/IFC_2x_Technical_Guide.pdf

International Alliance for Interoperability, 2006, Industry Foundation Classes, IFC2x Edition 3. [verkkodokumentti]. [viitattu 20.2.2008]. Saatavissa: http://www.iai-international.org/Model/R2x3_final/index.htm

International Alliance of Interoperability Modelling Support Group, toim. Liebich Thomas, 2004, IFC 2x Edition 2 Model Implementation Guide. [verkkodokumentti]. [viitattu 20.2.2008]. Saatavissa: http://www.iai-international.org/Model/files/20040318_Ifc2x_ModelImplGuide_V1-7.pdf

International Alliance of Interoperability, Modeling Support Group, Nisbet, Liebich Thomas, 2007, ifcXML Implementation Guide Version 2.0. [verkkodokumentti]. [viitattu 20.2.2008]. Saatavissa: http://www.iai-tech.org/products/ifc_specification/ifcxml-releases/

Jonassen, D.H., toim. Beissner, K. & Yacci, M., 1993, Structural Knowledge: Techniques for Representing, Conveying, and Acquiring, Lawrence Erlbaum Associates. 265 s. ISBN 0805813608

Karstila Kari & Hemiö Tero, 2002, WebSTEP: WebSTEP IFC Model Server project by Eurostep. Saatavissa: http://www.eurostep.com/prodserv/ems/ems.html

Kiviniemi A., Fischer M., Bazjanac V., 2005, Integration of Multiple Product Models: IFC Model Servers as a Potential Solution. [verkkodokumentti]. [viitattu 20.2.2008]. Saatavissa: http://itc.scix.net/data/works/att/w78-2005-KN-w2-1-Kiviniemi.pdf

Laakko Timo, Sukuvaara Antti, Borgman Jukka, Simolin Teemu, Björkstrand Roy, Konkola Marcus, Tuomi Jukka, Kaikonen Hannu, 1998, Tuotteen 3D-CAD-suunnittelu, Werner Söderström Oy. 311 s. ISBN 951-0-23217-3

Laine Tuomas, 2008, Tuotemallintaminen talotekniikkasuunnittelussa, Rakennustieto Oy Rati. 48 s. ISBN 978-951-682-859-9

Mäntylä M., 1988, An Introduction to Solid Modelling. Computer Science Press. 401 s.

Nour M. & Beucke K., 2005, Manipulating IFC model data in conjunction with CAD. [verkkodokumentti]. [viitattu 20.2.2008]. Saatavissa: http://www.irbdirekt.de/daten/iconda/06079005492.pdf

Penttilä Hannu, Nissinen Sampsa, Mittaviiva Oy, Niemioja Seppo, System Studio Oy, 2006. Tuotemallintaminen rakennushankkeessa. Yleiset periaatteet. Rakennusteollisuus RT ry, Rakennustietosäätiö RTS, Rakennustieto Oy, Helsinki. 64 s. ISBN:951-682-796-9

Singh M. P. & Huhns M. N., 2005, Service-Oriented Computing: Semantics, Processes, Agents, John Wiley and Sons. 549 s. ISBN 0470091487

Smith Zack, 2007, Developing an object oriented database in less than 140 lines of C#. TechRepublic. [verkkodokumentti]. [viitattu 20.2.2008]. Saatavissa: http://downloads.techrepublic.com.com/abstract.aspx?docid=277997

Talo-nimikkeistöryhmä, Haahtela-kehitys Oy, 2008, Talo 2000 -nimikkeistö – Yleisseloste, Rakennustieto Oy Rati. 127 s. ISBN 978-951-682-850-6

Yanga Q.Z., toim. Bjork Bo-Christer, 2002, IFC-compliant design information modeling and sharing. [verkkodokumentti]. [viitattu 21.3.2008]. Saatavissa esimerkiksi: http://www.itcon.org/2003/1

Yanga Q.Z., Zhang Y., 2006, Semantic interoperability in building design: Methods and tools. [verkkodokumentti]. [viitattu 20.2.2008]. Saatavissa esimerkiksi: http://www.sciencedirect.com/

EXPRESS –kielen perusteita (mukaillen Eastman, 1999)

Perustietotyypit

Perustietotyypit kielessä ovat: NUMBER, REAL, INTEGER, STRING, LOGICAL, BOOLEAN ja BINARY.

INTEGER on kokonaisluku. REAL on rationaalinen, irrationaalinen tai tieteellinen luku. Tieteelliset luvut voidaan kuvata tietyllä tarkkuudella. NUMBER on INTEGER:n ja REAL:n kantatyyppi. STRING on lista yksittäisiä merkkejä. LOGICAL tyyppinen muuttuja voi saada arvoja TRUE, FALSE tai UNKNOWN. BOOLEAN tyyppinen muuttuja voi saada arvot TRUE tai FALSE. BINARY on binääriarvojen vektori käyttäjän määrittelemällä koodauksella. Sekä STRING ja BINARY ovat muuttujan pituisia vektoreja. Niillä voi olla määritelty pituus. Jos pituus on määritelty lisäksi sanalla FIXED, muuttujien arvot pitää olla juuri sen mittaisia kun muuttujan määrittelyssä on sanottu. Mutta jos muuttujan pituutta ei ole määritelty, sen pituus voi vaihdella.

Esimerkkejä muuttuja määrittelystä:

```
s1: STRING; (*vaihtelevan pituinen merkkijonomuuttuja*)
s2: STRING(10); (*vaihtelevan pituinen merkkijonomuuttuja maksimissaan 10 merkkiä pitkä*)
b1:BINARY(10) FIXED; (*täsmälleen 10 bitin pituinen muuttuja b1*)
```

EXPRESS:ssä on myös lueteltuja vakioita; alla esimerkki:

```
TYPE kompassin_suunta =
        ENUMERATION OF (etela, pohjoinen, ita, lansi);
END_TYPE;
```

Muuttujat ja muut struktuurit voidaan koostaa isoimmiksi ryhmiksi käyttäen konstruktoreita. Konstruktorit ryhmittelevät saman tyyppisiä asioita. EXPRESS:ssä näitä ovat: ARRAY, BAG, LIST ja SET.

ARRAY:tä käytetään kun elementit laitetaan määrätyn pituiseen listaan. ARRAY:t voivat olla yhteenliitettyjä; esimerkiksi:

matrix : ARRAY[1:4] OF ARRAY[1:4] OF REAL;

ARRAY:ssä sekä alempi, että ylempi raja pitää määritellä (alempi < ylempi). Taulukossa on (yläraja-alaraja + 1) alkiota. Alkioihin voi viitata syntaksilla matrix[3][1] (vertaa edellinen matrix:n määritelmä).

BAG on järjestämätön elementtien kokoelma. Kaksoisarvoja sallitaan. Ala- ja ylärajaa ei tarvitse määritellä. Jos alaraja on määritelty, pitää BAG:ssä myös olla alarajan verran elementtejä. Jos yläraja on määritelty, voi BAG:ssä olla maksimissaan ylärajan verran elementtejä. Jos ylä- ja alarajaa ei ole määritelty, oletetaan että rajat ovat [0:?]. ? tarkoittaa rajoittamatonta määrää. Esimerkiksi:

pisteiden_bag : BAG[1:?] OF piste; (*tarkoittaa, että pisteitä BAG:ssa oltava minimissään 1*)

LIST on järjestetty kokoelma elementtejä, samalla tavalla kuin ARRAY, mutta LIST:iä voi olla muuttuvan pituinen. Esimerkki:

pisteiden_lista: LIST[0:?] OF piste;

SET on ei-järjestetty kokoelma elementtejä. Tupla-arvoja ei sallita. SET:t voivat olla vakion tai muuttuvien kokoisia. Esimerkiksi:

nimien_set: SET OF [1:100] OF nimi;

Kaikkien konstruktoreiden yleistys on AGGREGATE. Sitä voidaan käyttää määrittelyssä, missä mitä tahansa konstuktoria voidaan käyttää.

Perustyyppejä voidaan käyttää määriteltäessä korkeamman tason omia tyyppejä, esimerkiksi:

```
TYPE alue = REAL;
END_TYPE;

TYPE nimi = STRING;
END_TYPE;
```

Tyypit voivat myös olla erityyppisiä. Silloin voidaan käyttää sanaa SELECT. Esimerkiksi:

```
TYPE numero = SELECT(REAL,INTEGER);
END_TYPE;
```

Entiteetit

Yleinen objektin tyyppi, mistä oliot muodostetaan on Entity. Sen avulla voidaan määritellä hyvin erilaisia ja monimutkaisia elementtejä. Esimerkki:

```
ENTITY piste;
        x,y,z : REAL;
END_ENTITY;
```

Entiteetit voidaan periyttää (SUPERTYPE OF) tai ne voivat periä (SUBTYPE OF) muita entiteettejä. Esimerkiksi:

```
ENTITY homogeeninen_piste
        SUBTYPE OF (piste);
        w: REAL;
END_ENTITY;
```

Entiteetti ei voi olla saman entiteetin ylätyyppi ja alityyppi. Alityyppi (SUBTYPE) perii kaikki ylätyypin (SUPERTYPE) attribuutit, mukaan lukien DERIVED ja INVERSE attribuutit. Rajoitukset jotka on määritelty WHERE lauseissa periytyvät myös. Sääntöjen ylikirjoittaminen ei ole sallittu EXPRESS:ssä.

Attribuutit

EXPRESS:ssä voidaan määritellä kolmenlaisia attribuutteja:
Explicit:arvoja voidaan käsitellä suoraan.
Derived:arvot voidaan laskea muista attribuuteista. Huom! tämä ei vastaa niin sanottua suojattua jäsenmuuttujaa ohjelmointikielessä.
Inverse: kuvataan mol. puolinen yhteys attribuuttien välillä.
Jos sallitaan attribuutille tyhjä arvo, se määritellään OPTIONAL sanalla.

Eksplisiittiset (explicit) attribuutit

Eksplisiittiset attribuutit voidaan jakaa

- yksinkertaisiin tyyppeihin (simple types): NUMBER, REAL, INTEGER, STRING, LOGICAL, BOOLEAN ja BINARY.
- Koostetyyppeihin (aggregate types): Nämä ovat yhden tyypin koosteita käyttäen jotain konstruktoria.
- Entiteettityyppeihin (entity types): Nämä objektit kuvataan entiteetteinä. Entiteettiä käytettäessä attribuuttina voidaan käyttää luomaan suhdetta kahden entiteetin välillä.

Johdetut (derived) attribuutit

Johdetut attribuutit ovat johdettuja toisien entiteettien arvoista. Johdetut attribuutit määritellään sanalla DERIVE, esim:

```
ENTITY 3dympyra;
        keskipiste : piste;
        sade: REAL;
        keskiviiva: vektori;
DERIVE
        ala: REAL := pii * sade ** 2;
END_ENTITY;
```

UNIQUE:lla voidaan määritellä attribuutti sellaiseksi, että jokaisella ENTITY:n instanssilla attribuutti saa eri arvonsa (eli arvo ei voi toistua).

Päinvastaiset (inverse) attribuutit

Monesti attribuutti kuvaa yhteyttä kahden elementin välillä, esim.

```
ENTITY suora;
        piste_ref : ARRAY[1:2] OF piste1;
END_ENTITY;
```

Edellinen esimerkki kuvaa suora instanssin ja kahden pisteen välistä suhdetta. Joskus on vaatimus, että edellisen kaltainen suhde pitää esittää kaksisuuntaisena. Tämä tarkoittaa suhteessa edelliseen esimerkkiin sitä, että suhde on myös pisteestä suoraan. Tämä voitaisiin esittää esim. seuraavasti:

```
ENTITY piste1;
        x,y,z : REAL;
        suora_ref : SET OF suora;
END_ENTITY;
```

Edellisen esimerkin mukaan vain pisteet, jotka ovat osa suoraa voivat olla piste1 –tyyppisiä.

INVERSE –attribuutti määrää symmetrisen säännön olemassa olevan suhteen ja attribuutin välille. Se luo automaattisesti suhteen ja ylläpitää suhdetta attribuuttien välillä. Edellinen esimerkki voitaisiin esittää seuraavasti:

```
ENTITY piste1;
        x,y,z : REAL;
INVERSE
        suora_ref : SET OF line FOR piste_ref;
END_ENTITY;
```

Edellisen esimerkin mukaisesti jos piste_ref:iä päivitetään, suora_ref piste1:ssä päivittyy automaattisesti.

INVERSE-attribuutin tapainen kuvaus/toiminto puuttuu ns. ER-kuvausmenetelmästä ja mitään yksinkertaista ratkaisua toteuttaa myös relaatiotietokannoissa em. päivitys molempiin suuntiin on hankala.

Säännöt (RULES)

EXPRESS tukee toimintaa, jossa voidaan määritellä useampia sääntöjä jotka toteuttavat monentyyppisiä semanttisia ehtoja jotka ovat tärkeitä tietomalleissa. Johdetut (derived) attribuutit kuvaavat yhden tämäntapaisen säännön. WHERE lauseella voidaan ilmaista eri arvojen kombinaatioita entiteetin attribuuteissa. Esim.

```
ENTITY vektori;
        a,b,c :         REAL;
WHERE
        pituus1 : a ** 2 + b ** 2 + c ** 2 = 1.0;
END_ENTITY;
```

Edellisessä esimerkissä pituus1:lle on asetettu LOGICAL –tyyppinen eheyssääntö (kaikki WHERE lauseet ovat LOGICAL –tyyppisiä). Kun tarkistus suoritetaan WHERE-lauseessa, palautetaan joku arvoista: TRUE, FALSE tai UNKNOWN.

UNKNOW:ia käytetään kun joku attribuuteista puuttuu. Jos pituus1 ei ole TRUE, silloin vektori:n instanssi ei mukaudu ko. spesifikaatioon.

EXPRESS:ssä on käytössä esim. sääntöihin monia funktioita:

ABS, ACOS, ASIN, ATAN, BLENGTH, COS, EXISTS, EXP, FORMAT, HIBOUND, HIINDEX, LENGHT, LOBOUND, LIKE, LOG, LOG10, LOG2, LOINDEX, NVL, ODD, ROLESOF, SIN, SIZEOF, SQLRT, TAN, TYPEOF, USEDIN, VALUES

Jos halutaan viitata entiteetissä oleviin entiteetin omiin attribuutteihin, voidaan käyttää sanaa SELF. Silloin käytetään allaolevan esimerkin mukaista syntaksia:

SELF\piste.x;

Edellä viitataan SELF:llä sen hetkiseen entiteettiin. Merkintä \piste viittaa entiteettiin joka on peritty sen hetkisestä entiteetistä. Eli edellä olevassa lauseessa viitataan perityn entiteetin attribuuttiin x.

SUHTEET (RELATIONS)

EXPRESS:ssä ei ole rakennetta, jolla voidaan kuvata suhteita (tyyppien tai entiteettien välillä). Kuten useimmissa ohjelmointikielissä, suhteet on kuvattava attribuuteilla.

Rajoitteet entiteettien ylä- ja alatyypille

EXPRESS mahdollistaa seuraavia rajoitteita (CONSTRAINTS) SUBTYPE lauseelle:

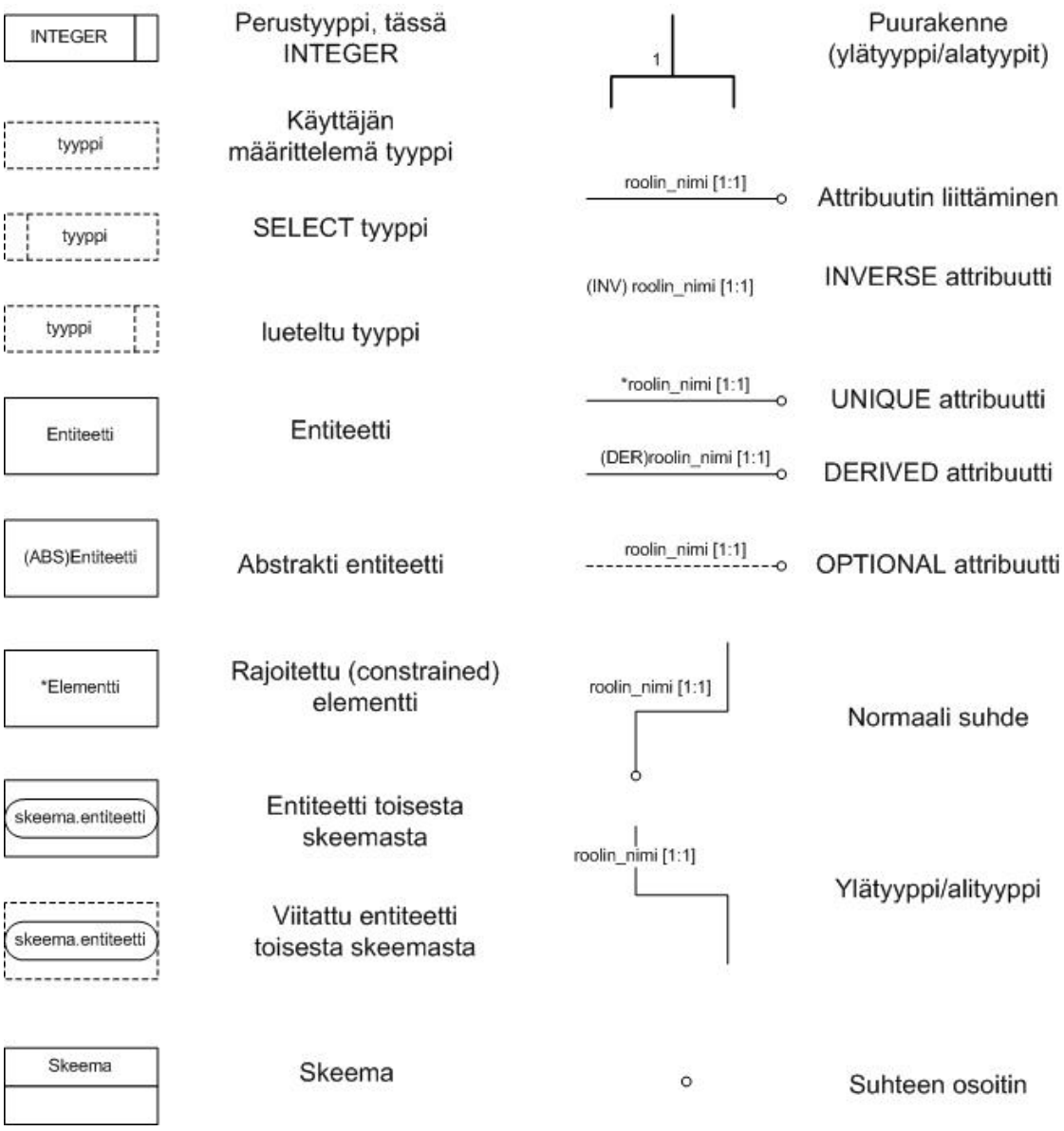
- ONEOF: määrittää joukon alityyppejä poissulkeviksi. Instanssi voi olla vain yhtä alityyppiä.
- AND: käytetään ilmaisemaan loogista lausetta eli instanssi voi olla molempia alityyppejä.
- ANDOR: määrittää että sääntöä ei ole ja että instanssi voi kuulua mihin tahansa alijoukkoon (mihin tahansa SUBTYPE:ihin). Jos rajoitetta ei ole määritetty, ANDOR otaksutaan oletukseksi.

Edellä mainitut alityyppien rajoitteet saattavat olla hyödyllisiä tietomalleissa, koska ne sallivat eri kombinaatioiden määrittelyjä joissa entiteetit on sijoitettu.

Abstraktit ylityypit

Abstraktit ylätyypit määritellään sanalla ABSTRACT.

EXPRESS -G



EXPRESS –G syntaksi (MS-Visio:n syntaksi) (IAI, 2000)

Mallin avainrakenteet IFC 2x Kernel:ssä

Kernel on IFC mallin skeema joka mahdollistaa perustiedon kaikille aliluokille. Näissä perustiedoissa määritellään objektien perustieto, suhteiden (kuten esim. koostumusten ja assosiaatioiden) määrittely eri asioiden välillä, tyyppitiedon määrittely objekteihin, ominaisuuksien määrittely (, joilla määritellään objekti) sekä tietyille objekteille muotoesitykset ja sijainti projektin sisällä. (IAI, 2000)

IfcRoot

Abstrakti kantaluokka kaikille entiteeteille on IfcRoot. Siinä on määritelty peruskäsitteet identifikaatio, omistajuus ja muutostieto sekä vapaaehtoinen entiteetin nimeämistieto. (IAI, 2000)

Objektit (objects), suhteet (relations) ja ominaisuudet (properties) on peritty IfcRoot:sta. Objektit ovat mikä tahansa semanttinen kokonaisuus tai asia IFC-mallin sisällä. Suhteet on yleistetty ja objektisoitu eri asioiden välille. Ominaisuudet ovat ominaisuuksia tai ominaisuusjoukkoja (property sets) joita voidaan liittää objekteihin. Kaikki edellä mainitut asiat on määritelty IFC:n abstrakteissa luokissa IfcObject, IfcRelationship IfcPropertyDefinition. (IAI, 2000)

IfcObject

Objektien yläluokkana on abstrakti ylätyyppi IfcObject. Siihen sisältyy konkreettisia asioita kuten seinä, palkki jotka ovat fyysisesti olemassa olevia asioita tai käsitteellisiä asioita kuten piirtoverkko (grid) tai piirustusalueen rajat. Sitä voi käyttää myös prosessikäsitteisiin kuten työtehtäviin, kontrolleihin (controls=käsitteet kuten esim. ohjeet, spesifikaatiot, säädökset, rajoitteet tai muut vaatimukset jotka liittyvät objektiin), resursseihin (resources, ovat käsite, joka kuvaa objektin käyttöä pääasiassa prosessin sisällä, kuten esim. työvoiman), tekijöihin (actors) kuten esimerkiksi henkilöihin jotka ovat mukana suunnittelu- tai rakennusprosessissa. IfcObject:n 2. tason (aliluokkia) luokkia ovat em. asioita kapseloivat luokat: IfcProduct, IfcProcess, IfcControl, IfcResource, IfcActor, IfcGroup, IfcProject. (IAI, 2000)

IfcProxy

IfcProxy on erikoinen alityyppi tuotteita (muut kapseloidaan IfcProduct:ssa), joita ei kuvata semanttisesti IFC:n sisällä. (IAI, 2000)

IfcRelationship

Tässä luokassa kapseloidaan objektien välinen suhde yhteen luokkaan. Tämä mahdollistaa myös sen, että suhteen semantiikka irrotetaan objektien attribuuteista. (IAI, 2000)

IfcPropertyDefinition

Objektien ominaisuudet kapseloidaan luokkaan IfcPropertyDefinition. (IAI, 2000)

IfcRelationship

Tämä luokka kapseloi objektien väliset suhteet. Luokalla on viisi aliluokkaa: IfcRelAssigns, IfcRelAssociates, IfcRelDecomposes, IfcRelDefines, IfcRelConnects. (IAI, 2000)

IfcRelAssigns

Tämä luokka kapseloi “linkin” objektien ja sen aliobjektien (ts. aliluokan ilmentymien) välillä. Linkissä määritellään jako missä yksi objekti (client = asiakas) käyttää toisten objektien (supplier = toimittaja) palveluja tai yksi objekti navigoi muihin objekteihin.

Jakaminen voi tapahtua tuotteille, prosesseille, kontrolleille, resursseille, aktoreille ja ryhmille. Edellisten jaot on kapseloitu luokkiin: IfcRelAssignsToProduct, IfcRelAssignsToProcess, IfcRelAssignsToControl, IfcRelAssignsToResource, IfcRelAssignsToActor, IfcRelAssignsToGroup. Luokat ovat aliluokkia abstraktille luokalle IfcRelAssigns. Suhteet muodostetaan IfcObject luokan aliluokille. Esimerkiksi IfcProduct muodostaa suhteen kapseloimalla sen luokkaan IfcRelAssignsToProduct. (IAI, 2000)

IfcRelAssociates

Assosiaatiosuhde mahdollistaa linkin objektien välille joissa on viittaus mihin tahansa objektiin tai ominaisuuteen. Viittaus tämän luokan avulla saadaan luokitteluun, kirjastoon tai dokumenttiin. Ne on kapseloitu luokkiin IfcRelAssociatesClassification, IfcRelAssociatesLibrary ja IfcRelAssociatesDocument. Objektit, jotka mahdollistavat viittauksen toiminnallisuuden ovat IFC-mallin resurssitasolla. (IAI, 2000)

IfcRelDecomposes

Jakosuhde mahdollistaa linkit objektien välillä jotka osallistuvat yhden suhde moneen suhteeseen. Tässä suhteessa on yksi kantaobjekti joka määrittää kokonaisuuden ja yksi tai useampi objekti aliobjekteina. Jakosuhdetta on olemassa kahdenlaista: koostumus (IfcRelAggregates kapseloi, jossa kantaobjekti ja lapsiobjekti voivat olla instansseja eri luokista IfcObject luokan alityypeille) ja upotus (IfcRelNests kapseloi, jossa kantaobjekti ja lapsiobjekti ovat kaikkia saman luokan instansseja IfcObject luokan alityypeille). (IAI, 2000)

IfcRelDefines

Määrittelysuhde mahdollistaa linkit objektien ja ominaisuusjoukkojen (property set) välillä. On olemassa kahdenlaisia määrittelysuhteita: ominaisuuksien määrittely (IfcRelDefinesByProperties) ominaisuusjoukolle ja tyyppimäärittely (IfcRelDefinesByType) yksittäiselle oliolle. (IAI, 2000)

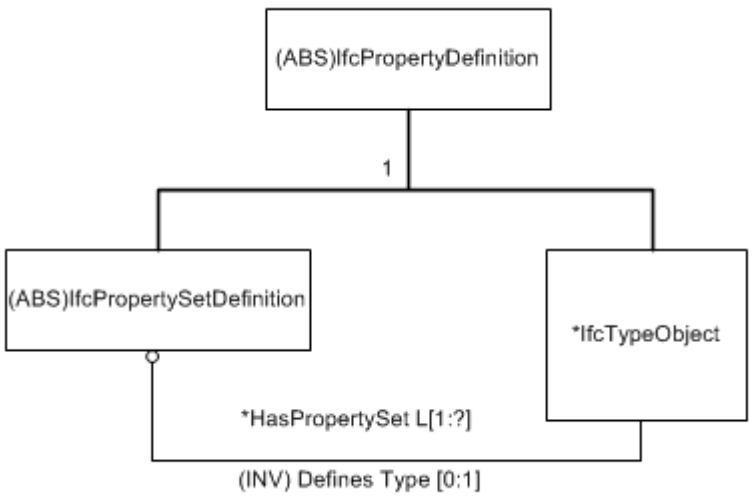
IfcRelConnects

Yhteyssuhde mahdollistaa linkit objektien välille jotka ovat yhteydessä jollain tavalla. Yhteys voi olla fyysinen tai looginen. Jaksosuhde (IfcRelSequence kapseloi)

mahdollistaa linkit kahden prosessin välillä. Jaksosuhde mahdollistetaan joka kerta kun tarvitaan suhde kahden prosessin välille (jaksosuhde on yhden suhde yhteen suhde kahden prosessin välillä, mutta prosessi ei voi olla suhteessa itseensä). (IAI, 2000)

IfcPropertyDefinition

IFC:ssä on olemassa kahdenlaisia ominaisuusmäärittelyjä. Ne peritään luokasta IfcPropertyDefinition. Niitä ovat tyyppiobjekti (IfcTypeObject kapseloi, joka määrittelee erityistä tietoa tyyppistä), ominaisuusjoukko (IfcPropertySet kapseloi, joka määrittää jaettavat ja laajennettavat ominaisuusjoukot liitettäväksi objekteihin). (IAI, 2000)



IfcPropertyDefinition-luokan perimä (IAI, 2000)

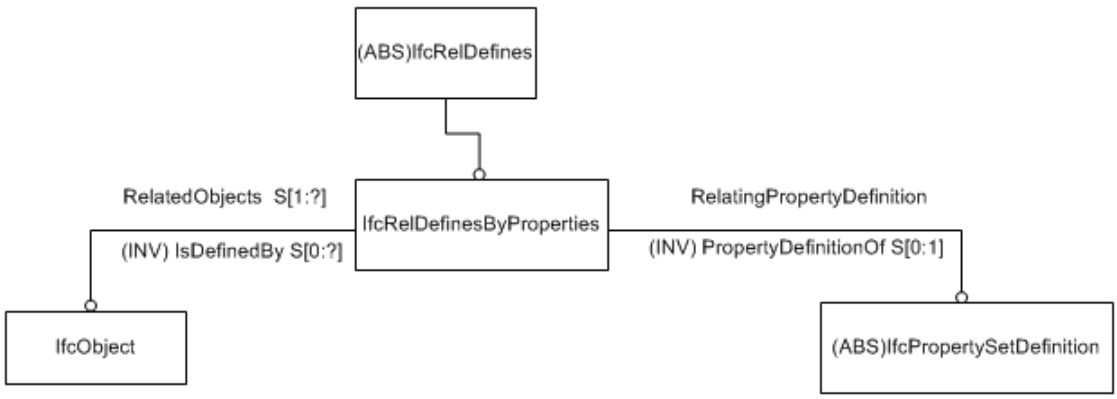
Tiedonsiirron kannalta ominaisuusmäärittelyt ovat oleellisia koska siten voidaan määritellä ja siirtää omaa tietoa objekteista sekä osana IFC:tä että ulkopuolelta IFC:n.

IfcPropertySetDefinition

Voidaan muodostaa ominaisuusjoukko joka toimii yhdessä siten että sitä voidaan kuvata yhtenä objektina. Ominaisuusjoukko voidaan muodostaa IFC mallin sisällä tai myös ulkopuolelta mallin laajentaen siten IFC mallia. (IAI, 2000)

Jos ominaisuusjoukko määritellään IFC mallin sisällä, tyyppimäärittelymekanismi on luetteloitu vakio (enumeration list) mahdollisista ominaisuusjoukoista joita voidaan liittää luokkaan, missä jokaisella arvolla luetelluista vakioista on nimi joka on määritelty ominaisuusjoukossa Nimi attribuutissa. (IAI, 2000)

IfcPropertySetDefinition objektit voidaan liittää muihin olioihin käyttäen luokkaa IfcRelDefinesByProperties. Tämä mahdollistaa sen, että objekti ja ominaisuusmäärittely voivat olla toisistaan riippumattomia ja silloin IfcPropertySetDefinition voidaan liittää objektiin kun halutaan. Etu suhdeluokkaa käyttäen on se, että objektin ei tarvitse sisältää mitään viittauksia ominaisuusjoukkoihin silloin kun niitä ei tarvita. Tässä tapauksessa monta objektia voi käyttää samaa ominaisuusjoukkoa. (IAI, 2000)

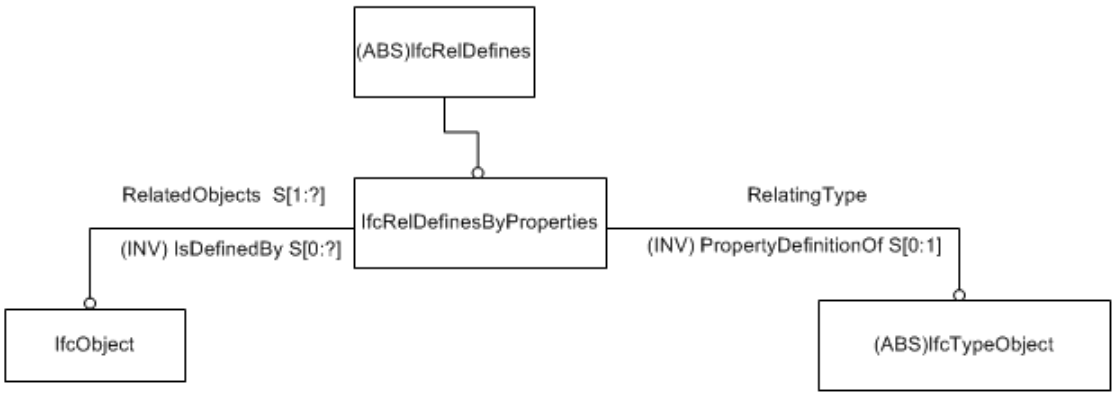


IfcRelDefinesByProperties luokan suhde objektiin (IfcObject) ja abstrakteihin luokkiin IfcRelDefines sekä IfcPropertySetDefinition (IAI, 2000)

IfcTypeObject

IFC:ssä voidaan erikoisesti kuvata ja täsmentää aiemmin geneerisesti määriteltyä tyyppiä. Tällä tavalla voidaan tehdä ominaisuusjoukkoja jotka yhdessä muodostavat

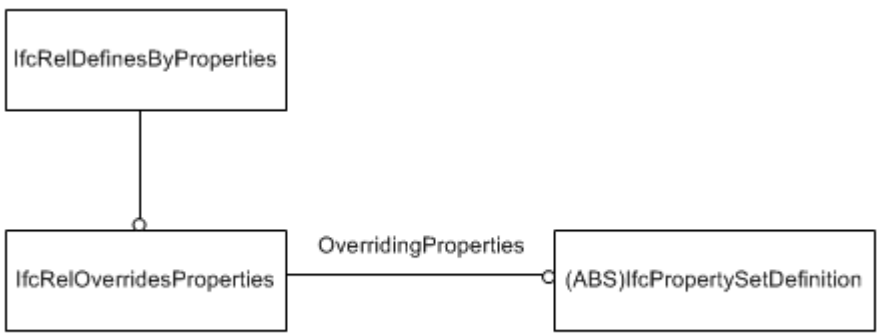
kokonaisuuden. Jos monta samanlaista objektia ovat samanlaisia ominaisuuksiltaan, kannattaa käyttää yksittäistä IfcTypeObject luokan oliota johon voidaan viitata eri objekteista. (IAI, 2000)



IfcRelDefinesByProperties luokan suhde IfcTypeObject luokkaan. (IAI, 2000)

IfcRelOverridesProperties

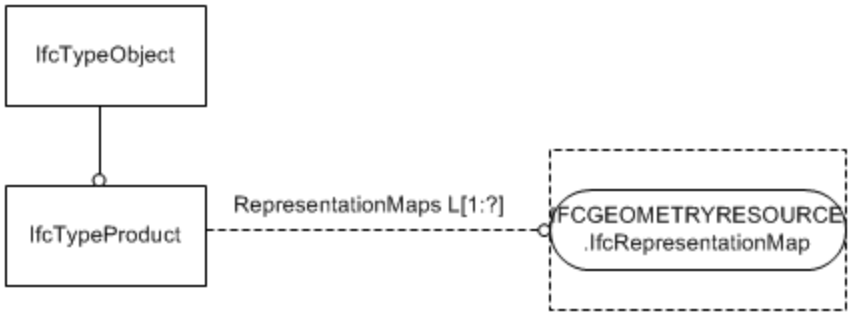
Ominaisuusjoukko voidaan liittää moneen objektiin. Kuitenkin, saattaa olla tapaus jossa jotkut ominaisuudet ominaisuusjoukosta vaihtelevat kun muut ominaisuudet taas pysyvät vakioina. Jos ei haluta, että määritellään uusia ominaisuusjoukkoja ja liitetä niitä objekteihin, on mahdollista ylikirjoittaa ne ominaisuudet joiden arvot vaihtelevat. Tämä tehdään ylikirjoittamalla ominaisuusjoukko IfcRelOverridesProperties luokan avulla. (IAI, 2000)



Ominaisuusjoukon ylikirjoittaminen IfcRelOverridesProperties luokan avulla. (IAI, 2000)

IfcTypeProduct

Tällä luokalla mahdollistetaan se, että IfcPropertyDefinition:lla on useampia muotoesityksiä RepresentationMaps attribuutin avulla. Edellä mainittu attribuutti on tyyppiä IfcRepresentationMap joka on luokka IfcGeometryResource-skeemassa joka määrittelee ryhmän geometrisia objekteja jotka muodostavat toiminnallisen kokonaisuuden. CAD järjestelmissä tällaisia objekteja ovat symbolit (tai blokit). (IAI, 2000)



IfcRepresentationMap IfcGeometryResource-skeemassa. (IAI, 2000)

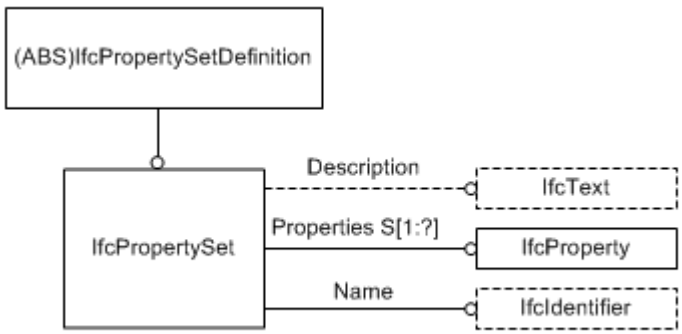
IfcPropertySet

IfcPropertySet luokka kapseloi kokoelman (tai joukon) ominaisuuksia. Ominaisuusjoukko voidaan määrittää ulkoisesti tai staattisesti IFC mallin sisällä. IFC mallissa on määritelty muutamia ominaisuusjoukkoja. Täydellisen IFC mallin katsotaan käsittävän sekä EXPRESS kielellä kuvatut ominaisuusjoukot että hajautetut ominaisuusjoukot. (IAI, 2000)

Ominaisuusjoukon määrittely kuvaa tavan missä informaatio voi olla IFC:n tiedonsiirron tiedostossa tai IFC:n mukaisessa tietokannassa. Ominaisuusjoukon määrittelyä voidaan myös käyttää tiedon siirron määrittelyyn, jos siirrossa siirretään ominaisuusjoukon tietoa. (IAI, 2000)

IfcPropertySet sisältää yhdistelmän ominaisuuksia. IfcPropertySet:ssä pitää olla minimissään yksi ominaisuus, jotta se voidaan määritellä. (IAI, 2000)

IfcPropertySet:llä on nimi. Sitä käytetään property set:n tunnistamiseen. Lisäksi voidaan lisätä kerronnallinen tarkentava kuvaus property set:stä. (IAI, 2000)



IfcPropertySet:n kuvaus. (IAI, 2000)

IfcProperty

IfcProperty kapseloi ominaisuuden joka voi olla: yksittäinen arvo, lueteltu vakio, tietyissä rajoissa oleva arvo, joukko arvoja, olion viittaus, moniarvoinen (complex) ominaisuus. IfcProperty on abstrakti kantaluokka, joten siitä ei voi tehdä oliota itsestään vaan siitä on perittävä aliluokkia joista voidaan tehdä olioita. Jokaisella oliolla on oltava nimi (Name) jolla identifioida ominaisuus. (IAI, 2000)

IfcProperty:n aliluokkia ovat: IfcPropertySingleValue, IfcPropertyEnumeratedValue, IfcProperty-BoundedValue, IfcPropertyTableValue, IfcPropertyReferenceValue, IfcComplexProperty. (IAI, 2000)

IfcPropertySingleValue

Tämä luokka kapseloi yksittäisen ominaisuuden, jolla voi olla joko: nimi, nimi ja arvo tai nimi, arvo ja yksikkö arvot. Yksikkö ei ole pakollinen. (IAI, 2000)

IfcPropertyEnumeratedValue

Luokka sallii valita arvon ennalta määritellystä listasta, jossa on lueteltu vakioita. Arvo tallennetaan EnumerationValue attribuuttiin ja arvo valitaan listasta joka määritellään IfcPropertyEnumeration objektissa. IfcPropertyEnumeration:ssa voidaan määritellä

myös yksikkö, mutta ei ole mahdollista että luettelossa on arvoja joilla on eri yksikkö. (IAI, 2000)

IfcPropertyBoundedValue

Luokassa voidaan määritellä ominaisuus joka saa arvoja ylä- ja alarajan väliltä. (IAI, 2000)

IfcPropertyTableValue

Sallii joukon arvoja, jossa jokainen arvo on riippumaton toisesta. IfcPropertyTableValue:een voidaan tehdä kaksiulotteinen taulukko, johon tallennetaan arvoja tai siihen voidaan sijoittaa esimerkiksi tietyn funktion toteuttavia arvoja tuloksineen(, jolloin 1. rivin alkiot olisivat x ja 2.rivin alkiot f(x)). (IAI, 2000)

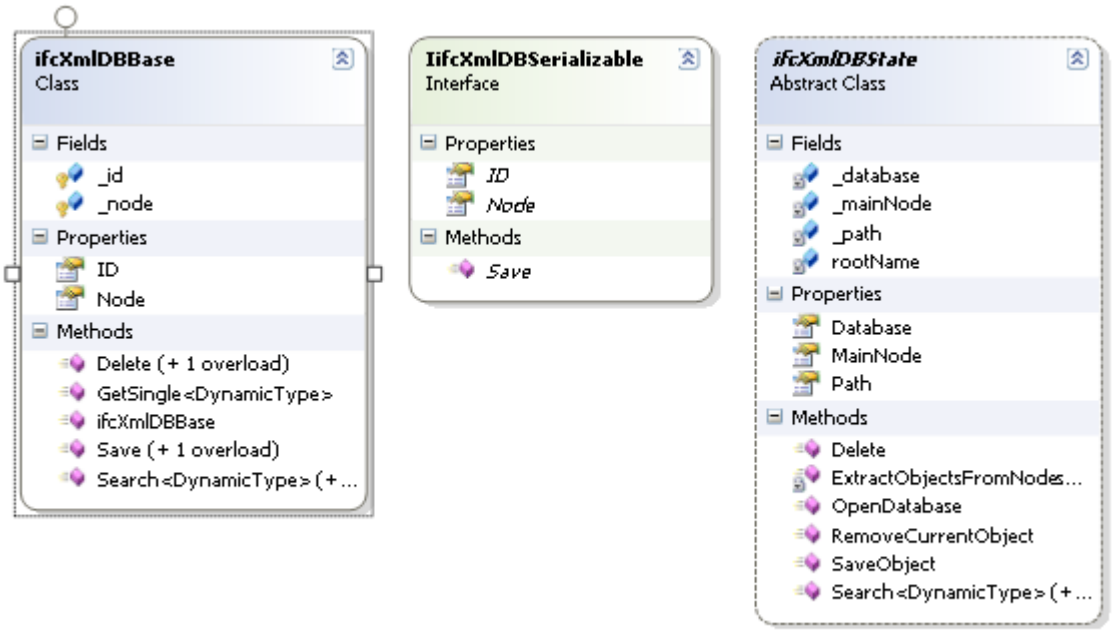
IfcPropertyReferenceValue

Luokka mahdollistaa viittauksen objekteihin joiden rakenne määritellään pysyvässä osassa IFC mallia. Tämä mahdollistetaan määrittelemällä suhde objektiin johon viitataan. Kyseisen luokan IfcPropertyReferenceSelect attribuutti mahdollistaa objektin tyypin (luokan) valinnan. IfcPropertyReferenceSelect attribuutin tarkoituksena on että sen kautta päästään käsiksi viitattavan luokan ominaisuuksiin kuten muihinkin ominaisuuksiin. IfcPropertyReferenceSelect:llä voidaan päästä moniin eri resurssitason luokkiin käsiksi. Rajoitteen tekee se IFC arkkitehtuurin yleinenkin rajoite että objekti voi viitata vain samalla tai alemmalla tasolla olevaan objektiin. Koska IfcPropertyReferenceSelect on resurssitason luokka ja resurssitaso on alin luokka, ei IfcPropertyReferenceSelect:n avulla voida viitata kuin resurssitason objekteihin. (IAI, 2000)

IfcComplexProperty

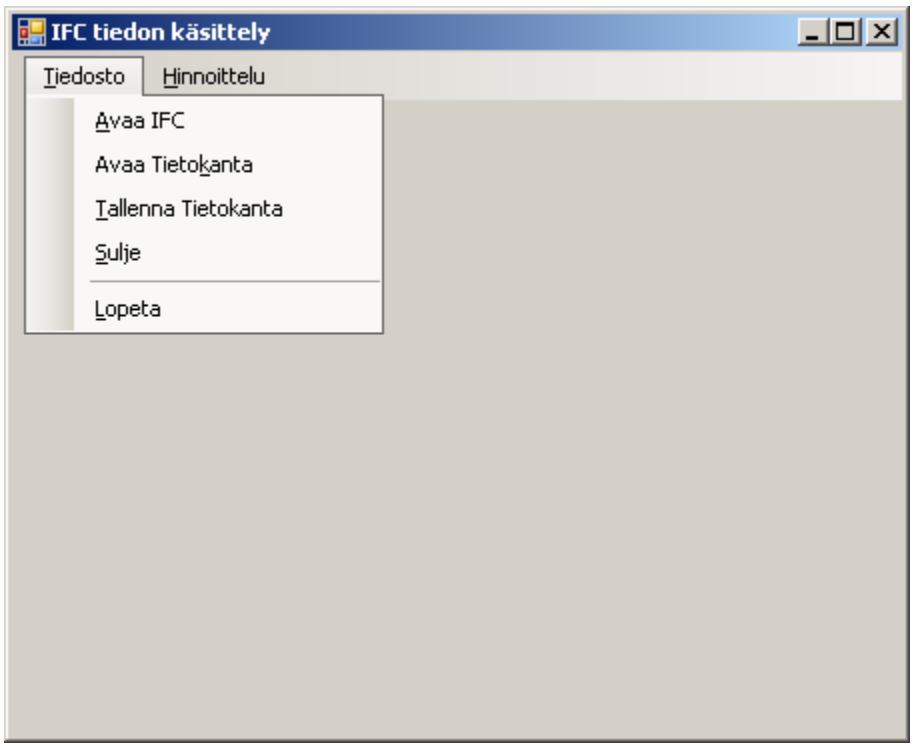
Luokan avulla voidaan laajentaa objektiin liitettävien ominaisuuksien määrää määrittelemällä mekanismi jolla voidaan tuoda muita ominaisuuksia nimettyihin ryhmiin.

IfcComplexProperty voi olla puurakenteessa yksi osa ominaisuusjoukkoa tai sen aliluokkina voi olla yksittäisiä ominaisuuksia (IfcProperty luokan aliluokkia).

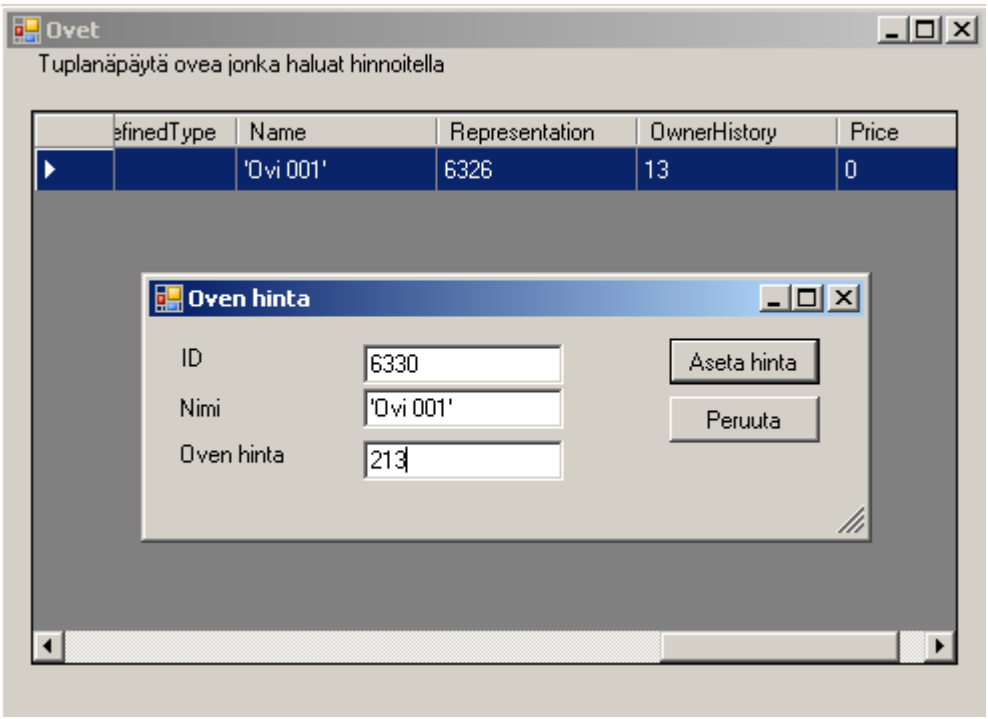


Tietokantasovelluksen rakenne UML-kaaviona.

Luokka ifcXmlDBBase toteuttaa rajapinnan IifcXmlDBSerializable. Luokan ifcXmlDBBase-oliot käyttävät abstraktin luokan ifcXmlDBState staattisia metodeja joilla tieto voidaan hakea, lisätä, muokata ja poistaa. Koko tietokanta on tehty Visual Studio .NET:ssä erilliseksi projektiksi, joka on käännetty dll:ksi. Tietokantaohjelmistoa käyttävissä sovelluksissa joudutaan siten tekemään viittaus tietokantaohjelmistoon. Varsinainen sovelluslogiikka voidaan kuitenkin pitää erillään omassa sovelluksessa.



Sovelluksen käyttöliittymä Tiedosto-valikon osalta. Avaa IFC-toiminto avaa ifc-tiedoston, käsittelee sen ja vie tiedon XML-tietokantaan. Tallenna Tietokanta-toiminnolla voidaan tallentaa tietokanta. Avaa Tietokanta-toiminnolla voidaan avata jo tehty tietokanta. Hinnoittelu-valikon kautta voidaan tutkia eri entiteettien määrät ja hinnoitella entiteetit.



Eri entiteeteille voidaan asettaa myös hinta.