DOKUMENTTIEN KONVERTOINTI SGML-MUOTOON

Jarkko Immonen

07.12.1999

Joensuun yliopisto
Tietojenkäsittelytiede
Pro gradu -tutkielma

TIIVISTELMÄ

Dokumentaation pitkäaikaisessa arkistoinnissa on varauduttava tekniikan ja ohjelmistojen kehittymiseen. Organisaatioiden dokumentaatio on yleensä tuotettu useilla eri välineillä, josta seuraa yhteensopimattomuusongelmia. Vaikka dokumentaatio olisikin tuotettu yhdellä tekstinkäsittelyvälineellä, ei se välttämättä ole vuosien kuluttua enää käyttökelpoista. Eräs ratkaisu dokumentaation arkistointi- ja yhteensopivuusongelmiin on SGML, joka on kansainvälinen dokumenttirakenteiden määrittelyyn tarkoitettu standardi. Organisaation siirtyessä perinteisistä työvälineistä SGML:n käyttämiseen on myös vanha dokumentaatio konvertoitava SGML-muotoon.

Tämän tutkielman tarkoituksena on tutustuttaa lukija dokumentaation SGML-konvertoimiseen liittyviin käsitteisiin ja tekniikoihin. Näiden perusasioiden pohjalta vertailemme eräiden markkinoilla olevien konversiovälineiden käyttökelpoisuutta erityyppisissä konvertointitilanteissa.

Tutkielman teoreettinen osuus perustuu alan lähdekirjallisuuteen. Konversiovälineiden vertailu perustuu esimerkkidokumentaation konvertoinnista tehtyihin havaintoihin.

SISÄLLYSLUETTELO

LIITE 1: Esimerkkidokumentaatio

LIITE 2: Esimerkkidokumentaation konvertointiin tarkoitettu OmniMark-ohjelma

LIITE 3: Esimerkki OmniMark-ohjelman konvertoimasta SGML-tiedostosta

LIITE 4: Sateenkaari -dokumenttityyppimäärittely

LIITE 5: Käyttöliittymät-kurssin sivu sateenkaari-DTD:n mukaisesti merkattuna ja Tietojärjestelmien dokumentointi -kurssin sivu sateenkaari-DTD:n mukaisesti merkattuna

1 JOHDANTO

Ihmiset ovat pyrkineet säilyttämään informaatiota erilaisten tallennusvälineiden avulla savitaulujen ja papyruskääröjen ajasta nykypäivän elektronisiin tallennusvälineisiin. Vielä pari vuosikymmentä sitten suurin osa yritysten ja muiden organisaatioiden informaatiosta oli paperilla, mutta nykyään, tietotekniikan yleistyttyä, tietoa pyritään säilyttämään elektronisessa muodossa tietokoneiden kiintolevyillä ja tietokannoissa. Vanhimmat savitaulut ovat jopa tuhansia vuosia vanhoja, mutta edelleen niihin sisältyvä informaatio on lukukelpoista. Sen sijaan elektronisessa muodossa oleva dokumentti on hyödyllinen täsmälleen niin kauan, kun se voidaan löytää jostain tietovarastosta ja käsitellä jollain tietokoneohjelmalla (Karjalainen & Salminen, 1999).

Perinteinen dokumentti eroaa elektronisesta dokumentista kahdessa suhteessa (Salminen, 1995). Paperille kirjoitetuissa dokumentissa, kuten myös savitaulussa, dokumentin sisältö ja ulkoasu ovat erottamattomat. Mikäli paperidokumentti revitään, menetetään myös sen sisältö. Sen sijaan elektronisessa dokumentissa sisältö ja ulkoasu on erotettu toisistaan. Sisältö voidaan kirjoittaa ensin ja lisätä siihen myöhemmin erilaisia ulkoasumäärityksiä. Lisäksi elektronisen dokumentin sisältöä tai tulostusasua voidaan muokata aina tarpeen tullen, ehkä pitkänkin ajan jälkeen. Perinteistä dokumenttia ei sen sijaan voida muokata jälkeenpäin.

Erilaisten ja erimuotoisten elektronisten dokumenttien pitkäaikaisessa arkistoinnissa täytyy huomioida tekniikan ja ohjelmistojen kehittyminen, eikä voida olettaa että yhdellä työvälineellä, esimerkiksi jollain tekstinkäsittelyohjelmalla, kirjoitettu dokumentti olisi vuosien kuluttua vielä käyttökelpoinen. Aika ajoin informaatio on siirrettävä uudempaa teknologiaa edustavalle tallennusvälineellä tai toiseen, uudempaan tallennusmuotoon (Kuronen, 1997).

Tätä vaatimusta on lähestytty kahdella tavoin. Ensimmäisessä lähestymistavassa dokumentit tallennetaan niin, että niiden ulkoasu vastaa mahdollisimman paljon paperitulostetta. Nykyiset tekstinkäsittelyohjelmat tukevat tätä lähestymistapaa, koska ne tallentavat dokumentin ulkoasuun vaikuttavia tietoja tekstin lomaan. Täten käyttäjä voi

muotoilla dokumentin ulkoasua samalla kun hän kirjoittaa sitä ja havaita välittömästi, miltä dokumentti näyttää paperille tulostettuna. Tällaista tekstinkäsittelyä, jossa käyttäjä kertoo ohjelmalle miten dokumentti esitetään, kutsutaan proseduraaliseksi (Leinonen, 1996).

Useat tekstinkäsittelyohjelmat ovat suunniteltu käyttämään omaa tallennusmuotoa, jota muut ohjelmat tai saman ohjelman eri versiot eivät yleensä pysty lukemaan. Mikäli dokumentteja täytyy siirtää tällaisten proseduraalisten tekstinkäsittelyjärjestelmien välillä, on tekstin usein oltava puhtaassa ASCII-muodossa (American Standard Code For Information Interchange), jolloin kaikki muotoilu-informaatio kadotetaan. On siis selvää, että dokumenttien pitkäaikainen arkistoiminen perinteisten tekstinkäsittelyjärjestelmien tarjoamassa tallennusmuodossa ei ole käytännöllistä.

Toisessa lähestymistavassa ei kiinnitetä dokumentin ulkoasua, vaan keskitytään erottamaan dokumentin rakenne ja sen sisältämä varsinainen informaatio toisistaan. Kaikissa dokumenteissa on erotettavissa loogisia rakenteita tai loogisen rakenteen osia. Esimerkiksi kirja jakaantuu lukuihin ja luvut kappaleisiin. Dokumentteja, joihin liittyy tietokoneen tulkittavissa oleva rakennemäärittely, kutsutaan rakenteisiksi dokumenteiksi (structured document) (Salminen, 1992). Rakenteinen dokumentti koostuu rakenneosista ja niiden sisältämästä informaatiosta. Dokumentin rakenne noudattaa jotakin rakennekuvausta ja käyttäjän vastuulla yleensä onkin vain dokumentin laatiminen rakennetta vastaavaksi. Koska rakenteisen dokumentin tulostusasu voidaan määrittää myöhemmin, käyttäjän ei tarvitse kirjoitusvaiheessa huolehtia dokumentin ulkoasuun liittyvistä seikoista (esimerkiksi kirjainkoon vaihtamisesta otsikon ja leipätekstin välillä) vaan hän voi keskittyä dokumentin sisällön oikeellisuuteen. Rakenteisten dokumenttien käsittelyyn tarkoitettua järjestelmää kutsutaan deklaratiiviseksi (Leinonen, 1996).

SGML (Standard Generalized Markup Language) (International Organization for Standardization, 1986) on kansainvälinen dokumenttirakenteiden määrittelyyn tarkoitettu ISO-standardi ISO 8879 vuodelta 1986. Se on alunperin suunniteltu julkaisutoiminnan tarpeisiin. SGML:n avulla voidaan erottaa dokumentin looginen rakenne sen typografisesta esitystavasta. SGML ei siis ota kantaa dokumentin ulkoasuun, vaan sen avulla määritellään dokumentin informaation rakenne (Willner, 1998a). Se on kehitetty täysin laitteisto-, ohjelmisto-, järjestelmä- ja kieliriippumattomaksi, jonka vuoksi se sopii erityisen hyvin dokumenttien pitkäaikaiseen arkistointiin.

SGML:n käytännön sovellukset perustuvat toimialakohtaisiin rakennemäärittelyihin (Virta, 1995). Näitä ovat esimerkiksi ATA-100 (siviili-ilmailu), AECMA 1000D (sotilasilmailu), SAE J2008 (kuljetusvälineteollisuus), TCIF (tietoliikenneteollisuus) ja AAP (kustannusteollisuus). Myös CERN:n alkujaan kehittämä HTML (Hypertext Markup Language) on SGML-pohjainen merkintäkieli, joka ei kuitenkaan tue loogista rakennetta.

SGML:n ottaminen tuotantokäyttöön perinteisten proseduraalisten tekstinkäsittelyjärjestelmien tilalle on ainakin isoissa organisaatiossa vaikea ja monitahoinen tehtävä. Organisaation on määriteltävä omaan tietotarpeeseensa sopiva dokumentin rakenne tai mahdollisesti useita, mikäli julkisesti saatavilla olevat rakennemäärittelyt eivät sovi sen tarpeisiin. Lisäksi henkilöstölle on järjestettävä koulutus uusien työvälineiden käyttöön. Täten SGML:n tuomien hyötyjen täytyy olla selvästi todennettavissa. Vaikka siirtyminen SGML:n käyttöön on suuri strateginen päätös, sen organisaatiolle tuomia todellisia etuja ovat tekstin monikäyttöisyys, dokumenttien käsittelyvaiheen automatisointi, sovellus- ja laiteriippumattomuus, informaation pitkäaikainen arkistointi, dokumenttien siirrettävyys ja tiedon hallinta (Salminen, 1995; Severson, 1995). Esimerkiksi Boeing 747 -lentokoneen tekninen dokumentaatio käsittää noin 1800 ylläpidettävää ohjekirjaa, joiden 30 miljoonaa sivua revisioidaan kolmen kuukauden välein (Virta, 1995). Tämä ylläpitotyö olisi todennäköisesti mahdotonta, mikäli dokumentaatio olisi proseduraalisissa tekstinkäsittelyjärjestelmissä.

Siirtyminen SGML:n käyttöön johtaa myös kysymykseen vanhojen dokumenttien arkistointimuodosta. Jotta SGML:stä saatava hyöty olisi maksimaalinen, täytyy vanhat dokumentit muuntaa valitun rakenteen mukaiseksi eli konvertoida SGML-muotoon. Yleensä dokumentin muuntamisella esitysmuodosta toiseen käsitetään esimerkiksi Word-dokumentin muuntamista RTF-muotoon (Rich Text Format), joka on täysin automatisoitu prosessi. Dokumentin konvertointi SGML-muotoon poikkeaa edellisestä, koska sen tarkoituksena on tuottaa dokumentin sisältämälle informaatiolle lisäarvoa. Täten perinteisen dokumentin konvertointi SGML-muotoon ei ole aivan yksinkertainen prosessi, vaan se vaatii tarkkaa dokumentin informaation analysointia ja sen loogisten rakenteiden selvittämistä (Severson, 1995).

Tämän tutkielman tarkoituksena on tutustua SGML-konvertointiin liittyviin käsitteisiin ja strategioihin sekä vertailla saatavilla olevia konvertointityövälineitä. Seuraavassa luvussa tutustumme SGML:n peruskäsitteisiin ja konvertoinnin kannalta tärkeisiin työvälineluokkiin. Luvussa 3 tarkastelemme yksinkertaista konversioprosessia ja perehdymme konversion tekniseen toteuttamiseen liittyviin kysymyksiin. Luvussa 4 tutustumme konvertointiin projektitasolla. Luvussa 5 analysoimme esimerkkidokumentaation ja vertailemme sen avulla eri konversiovälineitä. Lopuksi teemme yhteenvedon tutkielmassa käsitellyistä asioista.

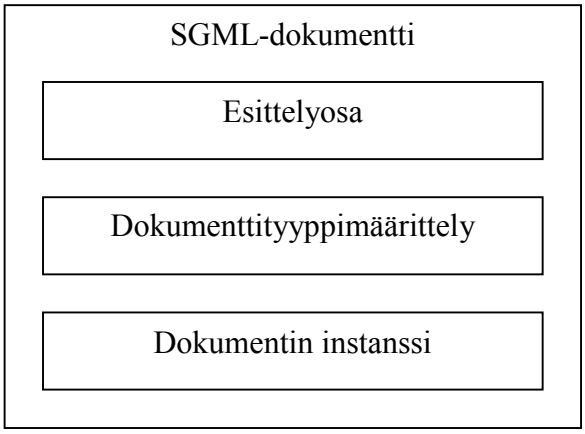
2 SGML:N KÄSITTEITÄ

SGML-standardi sisältää metakielen, jonka avulla voidaan muodostaa dokumenttien loogisen rakenteen kuvaava kielioppi. Se ei ota kantaa dokumenttien ulkoisen esitystavan tai sen rakenteen määritykseen eikä rakenteen osien käsittelyyn. SGML ei siis ole merkkauskieli, vaan tapa kuvata merkkauskieli.

Tässä luvussa perehdymme SGML:n peruskäsitteisiin ja tutkimme yksinkertaisen esimerkin avulla miten dokumentin rakenne määritellään. Lisäksi tutustumme konvertoinnissa tarvittaviin SGML-työvälineisiin. Goldfarb (1990) antaa yksityiskohtaisen kuvauksen SGML-standardista.

2.1 SGML-dokumentti

SGML erottaa dokumenttien rakenteen, sisällön ja ulkoasun toisistaan. SGML-dokumentti sisältää tavallisesti kolme osaa: SGML-esittelyosan, dokumenttityyppimäärittelyn ja merkatun sisällön eli dokumentin instanssin. Kuvassa 2.1 on esitetty SGML-dokumentin osien suhteet.



Kuva 2.1. SGML-dokumentin osat.

Esittelyosa (declaration) kertoo dokumenttityyppimäärittelyn konkreettisen syntaksin (concrete syntax) eli dokumentin merkkauksessa käytettävän syntaksin (Tyrväinen, 1998).

Esittelyosassa määritellään esimerkiksi mitä merkkivalikoimaa dokumentissa käytetään ja mitkä merkit ovat merkkauksessa erotinmerkkejä. Lisäksi eri tietokoneissa käytettyjen erilaisten merkistöjen eroista aiheutuvat ongelmat ratkaistaan esittelyosassa (Kuronen, 1997). Useimmat SGML-jäsentäjät (katso luku 2.2) eivät vaadi esittelyosaa, jolloin käytetään oletussyntaksia (reference concrete syntax) (DeRose, 1997). Myöhemmin esitettävissä määrittelyissä ja esimerkeissä käytämme SGML:n oletussyntaksia (lukuunottamatta nimeämiseen liittyviä pituusrajoituksia).

Dokumenttityyppimäärittely (document type definition, DTD) kuvaa dokumenttiluokan dokumenttien rakenteen SGML-kielellä. Kuvaus on deskriptiivinen ohjelma eli sääntöjoukko, joka määrää dokumentin elementtien loogisen rakenteen ja niiden väliset suhteet (Tyrväinen, 1998). Yksinkertaistettuna dokumenttityyppimäärittelyssä esitellään siis dokumentin merkkaukseen käytettävät tunnisteet ja niiden väliset suhteet. Dokumenttityyppimäärittely kuuluu yleensä *prologiin* (prolog), jossa esitellään dokumentin jäsennyksessä käytettävät säännöt (Goldfarb, 1990).

Dokumentin instanssi (document instance) tarkoittaa dokumentin SGML-merkattua sisältöä, joka noudattaa dokumenttityyppimäärittelyssä kuvattuja sääntöjä. SGML-dokumentin sisältö koostuu siis DTD:ssä määritellyistä rakenneosista ja varsinaisesta dokumentin sisältämästä informaatiosta.

Jokainen SGML-dokumentti noudattaa jotakin dokumenttityyppimäärittelyä, jossa kaikki kyseiseen dokumenttiluokkaan kuuluvat rakenneosat määritellään. Seuraavaksi tutustumme dokumenttityyppimäärittelyssä esiteltäviin rakenneosiin ja dokumentin merkkaamiseen DTD:n mukaiseksi.

E*lementti* (element) on dokumentin rakenneosa, joka sisältää jonkin loogisesti erotettavan osan tekstiä. Elementtien nimiä käytetään dokumentin merkkauksen yhteydessä tunnisteina. Alkutunniste muodostuu oletussyntaksin mukaisesti "<"- ja ">"-merkeistä ja niiden väliin sijoitetusta elementin nimestä. Lopputunniste eroaa alkutunnisteesta elementin nimen eteen sijoitetun "/"-merkin avulla. Elementit voivat sisältää joko alielementtejä tai informaatiosisältöä. Alla olevassa esimerkissä `lista`-elementti sisältää kolme `alkio`-elementtiä, jotka vuorostaan sisältävät informaatiota. Listan alku merkataan `<lista>`-alkutunnisteella ja listan loppu `</lista>`-lopputunnisteella. Listaan kuuluvat

alkiot merkataan kukin <alkio>- ja </alkio>-tunnisteilla, joiden välissä on listan alkion sisältämä informaatio.

```
<lista>
     <alkio>Ensimmäinen kohta</alkio>
     <alkio>Toinen kohta</alkio>
     <alkio>Kolmas kohta</alkio>
</lista>
```

Attribuutit (attribute) ovat elementeille annettavia parametreja, jotka kuvaavat elementtien ominaisuuksia (Chesnutt & Lowry, 1998). Käytettävät attribuutit tulee määritellä elementtikohtaisesti dokumenttityyppimäärittelyssä. Mikäli attribuutille ei anneta arvoa dokumentin merkkaamisen yhteydessä, saa se dokumenttityyppimäärittelyssä määritellyn oletusarvon. Edellä esitetylle `lista`-elementille voidaan määritellä attribuutti, joka kuvaa listatyypin (esitetäänkö listan alkiot numeroituna listana vai luettelomerkein) (DeRose, 1997). Attribuutin arvo ilmaistaan elementin alkutunnisteen sisällä sijoittamalla attribuutin nimen perään ko. attribuutin arvo lainausmerkeissä.

```
<lista tyyppi="luettelomerkki">
     <alkio>Ensimmäinen kohta</alkio>
     <alkio>Toinen kohta</alkio>
</lista>
```

Entiteetti (entity) on informaatiokokonaisuus, johon viitataan yhtenä yksikkönä. SGML-jäsennin korvaa dokumentin entiteettiesiintymät dokumenttityyppimäärittelyssä määritellyllä sisällöllä. Entiteettimäärittelyjen avulla on mahdollista liittää dokumenttiin ulkopuolista informaatiota. Entiteettimäärittelyjä käytetään esimerkiksi usein toistuvien merkkijonojen esittämiseen, merkkivalikoiman laajentamiseen ja alidokumenttien lisäämiseen päädokumenttiin (Korhonen & Kuisma, 1999).

Seuraavassa on esimerkki SGML-merkatusta dokumentista:

```
<!doctype artikkeli system "/home/tko/jimmonen/sgml/artikkeli.dtd">
<artikkeli tyyppi="essee">
  <yleistiedot>
    <organisaatio>Joensuun yliopisto</organisaatio>
```

```
  <kirjoittaja>Jarkko Immonen</kirjoittaja>
  <paivays>21.10.1999</paivays>
  <nimi>Esimerkkidokumentti</nimi>
</yleistiedot>
<abstrakti>T&apisteet;ss&apisteet; esimerkkidokumentissa
m&apisteet;&apisteet;ritell&apisteet;&apisteet;n muutamia
SGML-standardiin liittyvi&apisteet; k&apisteet;sitteit&apisteet;
(dokumenttityyppim&apisteet;&apisteet;rittely, elementti, attribuutti
ja entiteetti)
</abstrakti>
<runko>
  <otsikko>Dokumenttityyppim&apisteet;&apisteet;rittely eli DTD
  </otsikko>
  <kappale>DTD kuvaa dokumenttiluokan dokumenttien rakenteen
  SGML-kielell&apisteet;. Kuvaus on deskriptiivinen ohjelma eli
  s&apisteet;&apisteet;nt&opisteet;joukko, joka
  m&apisteet;&apisteet;r&apisteet;&apisteet; dokumentin elementtien
  loogisen rakenteen ja niiden v&apisteet;liset suhteet.
  </kappale>
  <otsikko>Elementti</otsikko>
  <kappale>Elementti on dokumentin rakenneosa, joka
  sis&apisteet;lt&apisteet;&apisteet; jonkin loogisesti muista
  erotettavan osan teksti&apisteet;. Elementtien nimi&apisteet;
  k&apisteet;ytet&apisteet;&apisteet;n dokumentin merkkauksen
  yhteydess&apisteet; tunnisteina.
  </kappale>
  <otsikko>Attribuutti</otsikko>
  <kappale>Attribuutit ovat elementeille annettavia parametraja,
  jotka kuvaavat elementtien ominaisuuksia.
  K&apisteet;ytett&apisteet;v&apisteet;t attribuutit tulee
  m&apisteet;&apisteet;ritell&apisteet; elementtikohtaisesti
  dokumenttityyppim&apisteet;&apisteet;rittelyss&apisteet;.
  </kappale>
  <otsikko>Entiteetti</otsikko>
  <kappale>Entiteetti on informaatiokokonaisuus, johon viitataan
  yhten&apisteet; yksikk&opisteet;n&apisteet;. SGML-j&apisteet;sennin
  korvaa dokumentin entiteettiesiintym&apisteet;t
  dokumenttityyppim&apisteet;&apisteet;rittelyss&apisteet;
  m&apisteet;&apisteet;ritelyll&apisteet;
  sis&apisteet;ll&opisteet;ll&apisteet;.
  Entiteettim&apisteet;&apisteet;rittelyjen avulla on mahdollista
```

```
    liitt&apisteet;&apisteet; dokumenttiin ulkopuolista informaatiota.
    </kappale>
  </runko>
</artikkeli>
```

Kuten esimerkin ensimmäiseltä riviltä (<!doctype artikkeli ... >) näemme, dokumentin rakenne perustuu `artikkeli`-dokumenttityyppimäärittelyyn. Dokumentissa esiintyy kaikkia aiemmin kuvattuja dokumentin rakenneosia: elementtejä, attribuutteja ja entiteettejä. Dokumentin sisältämä informaatio on kokonaisuudessaan `artikkeli`-elementin alku- ja lopputunnisteiden välissä. `<Artikkeli>`-alkutunnisteessa elementin `tyyppi`-attribuutille annetaan arvo "essee". Skandinaaviset merkit on dokumentissa korvattu entiteettiviittauksilla, joissa entiteetin nimen edessä on `&`-merkki ja perässä puolipiste, esimerkiksi pieni ä saadaan `&apisteet;` -viittauksella.

Esimerkkidokumentti koostuu yleistiedoista, abstraktista ja rungosta. Yleistiedot sisältävät kirjoittajan nimen ja organisaation sekä dokumentin nimen ja päiväyksen. Abstraktissa tiivistetään dokumentin sisältö ja rungossa esitetään varsinainen artikkelin informaatio.

Seuraavaksi tutkimme edellisen esimerkin merkkaukseen käytettyä dokumenttityyppimäärittelyä. Dokumenttityyppimäärittely `artikkeli` sisältää kymmenen elementin (`artikkeli`, `yleistiedot`, `abstrakti`, `runko`, `kirjoittaja`, `organisaatio`, `nimi`, `paivays`, `otsikko` ja `kappale`) ja neljän entiteetin (`Apisteet`, `apisteet`, `Opisteet` ja `opisteet`) määrittelyt. Lisäksi se sisältää yhden `artikkeli`-elementtiin liittyvän attribuutti-määrittelyn (`tyyppi`).

```
<!DOCTYPE artikkeli [

<!ENTITY    Apisteet         SDATA "[Auml  ]"                          >
<!ENTITY    apisteet         SDATA "[auml  ]"                          >
<!ENTITY    Opisteet         SDATA "[Ouml  ]"                          >
<!ENTITY    opisteet         SDATA "[ouml  ]"                          >

<!ELEMENT   artikkeli        - -       (yleistiedot,abstrakti?,runko)  >
<!ELEMENT   yleistiedot      - O       (kirjoittaja+ & organisaatio? &
                                       nimi & paivays?)                >
```

```
<!ELEMENT   abstrakti        - O      (#PCDATA)                          >
<!ELEMENT   runko            - -      ((otsikko*,kappale*)+)             >
<!ELEMENT   kirjoittaja      - -      (#PCDATA)                          >
<!ELEMENT   organisaatio     - -      (#PCDATA)                          >
<!ELEMENT   nimi             - -      (#PCDATA)                          >
<!ELEMENT   paivays          - -      (#PCDATA)                          >
<!ELEMENT   otsikko          - O      (#PCDATA)                          >
<!ELEMENT   kappale          - O      (#PCDATA)                          >

<!ATTLIST   artikkeli tyyppi          (essee|muistio|poytak) "essee"    >
]>
```

Entiteetit `Apisteet` ja `Opisteet` korvataan dokumentin instanssia käsiteltäessä isoilla Ä- ja Ö-kirjaimilla ja vastaavasti entieteetit `apisteet` ja `opisteet` pienillä ä- ja ö-kirjaimilla. Entiteettien määrittelyssä esiintyvää merkkijonoa SDATA (specific character data) käytetään erikoismerkkien (esimerkiksi skandinaaviset kirjaimet) esittelyyn.

Elementtien määrittelyssä esiintyvät miinus-merkit ja O-kirjaimet kuvaavat elementin alku- ja lopputunnisteiden pakollisuutta. Näitä kutsutaan elementin minimointisäännöiksi. Esimerkiksi `otsikko`-elementin alkutunniste on pakollinen (-), mutta lopputunniste voidaan jättää merkkaamatta (O, omitted). Seuraavassa esimerkissä otsikot on merkattu kahdella eri tavalla, jotka kummatkin ovat syntaktisesti oikein. Ensimmäisessä otsikossa sekä alku- että lopputunniste on merkattu, mutta jälkimmäisessä otsikossa lopputunniste on jätetty pois.

```
<otsikko>Esimerkki</otsikko>
<kappale>T&apisteet;ss&apisteet; on teksti&apisteet;</kappale>

<otsikko>Toinen esimerkki
<kappale>Lis&apisteet;&apisteet; teksti&apisteet;</kappale>
```

Elementtiin sisältyvien alielementtien ei välttämättä tarvitse esiintyä elementin sisällä tietyssä järjestyksessä tai yhtä monta kertaa. Elementtimäärittelyssä voidaan käyttää tiettyjä operaattoreita määräämään alielementtien järjestys tai esiintymiskertojen lukumäärä.

Alielementtien esiintymisjärjestykseen voidaan vaikuttaa seuraavilla operaattoreilla (El Andaloussi & Maler, 1996):

- *Peräkkäis-operaattori* (sequence connector) "," määrää, että alielementtien on esiinnyttävä peräkkäin luetellussa järjestyksessä (a, b ➔ ensin a ja sitten b).
- *Ja-operaattori* (and connector) "&" ei sido alielementtien esiintymisjärjestystä, mutta kaikkien alielementtien on esiinnyttävä (a & b ➔ ensin a ja sitten b tai päinvastoin).
- *Tai-operaattori* (or connector) "|" kertoo, että jonkin, mutta vain yhden, luetellun alielementin täytyy esiintyä (a | b ➔ joko a tai b).

Alielementtien lukumäärään voidaan vaikuttaa seuraavilla operaattoreilla (El Andaloussi & Maler, 1996):

- "?"-operaattori ilmaisee valinnaisuutta eli alielementti ei esiinny tai se esiintyy vain kerran (0-1 kertaa).
- "*"-operaattorilla ilmaistaan valinnaisuutta ja toistettavuutta eli alielementti ei esiinny ollenkaan tai se voi esiintyä useaan kertaan (0-n kertaa).
- "+"-operaattori määrää, että alielementin täytyy esiintyä vähintään kerran, mutta se voi esiintyä useamminkin (1-n kertaa).

Ylläesitetyn DTD:n mukaisesti `artikkeli`-elementti koostuu `yleistiedot`-elementistä, valinnaisesta `abstrakti`-elementistä ja `runko`-elementistä juuri tässä järjestyksessä (eli esimerkiksi runko ei voi esiintyä dokumentissa ennen abstraktia). Dokumentin ei tosin tarvitse sisältää abstraktia, mutta yleistiedot ja runko ovat pakollisia.

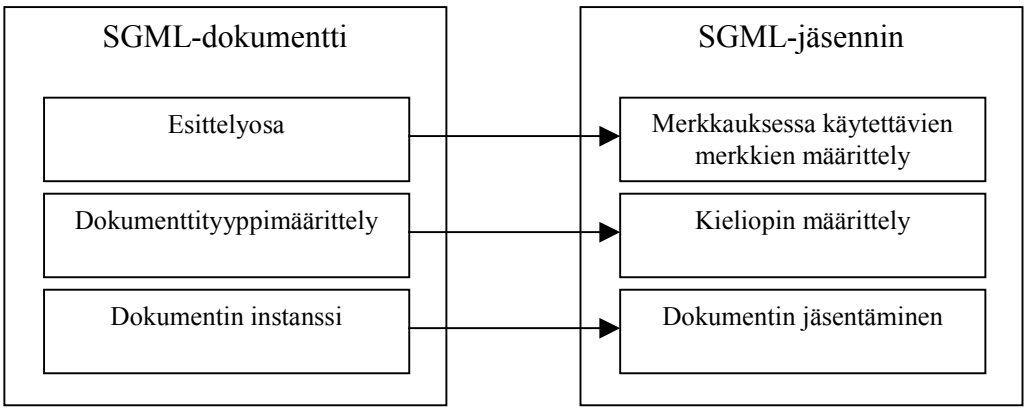
`Yleistiedot`-elementti koostuu yhdestä tai useammasta kirjoittajasta, organisaatiosta, artikkelin nimestä ja päiväyksestä missä järjestyksessä tahansa. Näistä elementeistä organisaatio ja päiväys ovat valinnaisia. `Abstrakti`-elementti ei enää jakaannu alielementteihin eli se sisältää informaatiota, tässä tapauksessa artikkelin tiivistelmän. Elementin määrittelyssä esiintyvä `#PCDATA` (parsed character data) tarkoittaa merkkijonoa, josta jäsennin erottaa entiteettiviittaukset. Runko koostuu valinnaisesta otsikosta ja valinnaisesta kappaleesta tässä järjestyksessä, mutta näiden alielementtien yhdistelmän täytyy esiintyä vähintään kerran.

Dokumenttityyppimäärittelyn viimeisellä rivillä määritellään `artikkeli`-elementille artikkelin tyyppiä kuvaava parametri "`tyyppi`". Kyseinen attribuutti voi saada arvot "essee", "muistio" tai "poytak". Mikäli arvoa ei määritellä, saa attribuutti arvokseen oletusarvon "essee".

2.2 SGML-välineitä

Tekstitiedostojen konvertointi SGML-muotoon on yleensä paljon ihmistyötä vaativa prosessi, joten on järkevää pyrkiä automatisoimaan sitä soveltuvin osin erilaisilla työvälineillä. Seuraavaksi tutustumme kahteen konvertoinnissa käytettävään apuvälineluokkaan: SGML-jäsentimeen ja konversio-ohjelmaan.

SGML-jäsentimen (SGML parser) tehtävänä on tarkastaa dokumentin rakenteen ja dokumenttityyppimäärittelyn vastaavuus (kuva 2.2). Dokumentissa esiintyvien loogisten rakenteiden merkkaaminen dokumenttityyppimäärittelyn mukaiseksi ja käytettävien rakenteiden yhtenäisyys ovat edellytys rakenteisten dokumenttien hyödyntämiselle (Kunnari, Lyytikäinen & Pietinen, 1995). Mikäli dokumentin rakenne ei vastaa DTD:n mukaista rakennetta, jäsennin yleensä tulostaa ilmoitukset löydetyistä virheistä. Virheilmoitusten antaman informaation perusteella käyttäjällä on mahdollisuus korjata dokumentti vaadittavan rakenteen mukaiseksi.



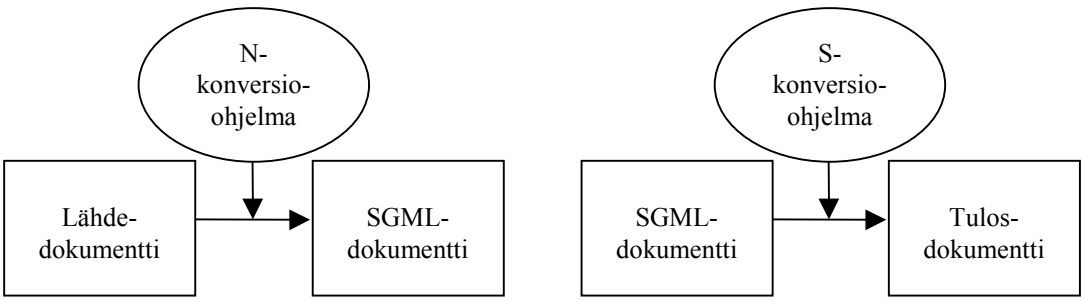
Kuva 2.2. SGML-dokumentin ja SGML-jäsentimen suhde (McGrath, 1998).

SGML-standardi jakaa SGML-jäsentimet kahteen luokkaan: SGML-jäsentimiin ja tarkastaviin SGML-jäsentimiin (validating SGML parser). SGML-jäsentimiksi luokitellaan ohjelmat, jotka tunnistavat dokumentin instanssista SGML:n mukaiset merkkaukset ja tarkastaviksi SGML-jäsentimiksi ohjelmat, jotka pystyvät löytämään ja ilmoittamaan käyttäjälle myös instanssissa esiintyvät virheet (Ensign, Goldfarb & Pepper, 1998).

Konversio-ohjelman (converter) tarkoitus on muuntaa yksi tai useampi dokumentti esitysmuodosta toiseen. Konversio-ohjelmat voidaan erotella pääsääntöisesti kolmeen eri ryhmään:

- Ohjelmat, jotka muuntavat ei-SGML-dokumentteja SGML-muotoon (up-conversion).
- Ohjelmat, jotka muuntavat SGML-dokumentin yhdestä SGML-muodosta toiseen SGML-muotoon (transform).
- Ohjelmat, jotka tuottavat SGML-dokumentista tulostuskelpoisen version (down-conversion).

Travis ja Waldt (1995) kutsuvat ensimmäisen ryhmän ohjelmia konversio-ohjelmiksi ja toisen ja kolmannen ryhmän ohjelmia muunnosohjelmiksi. Ensign, Goldfarb ja Pepper (1998) kutsuvat lähde- ja tulosdokumentin esitysmuotojen perusteella (kuva 2.3) ensimmäisen ryhmän ohjelmia N-konversio-ohjelmiksi (N-converter, Non-SGML source document converter) ja muita konversio-ohjelmia S-konversio-ohjelmiksi (S-converter, SGML source document converter).



Kuva 2.3. Eri konversio-ohjelmien lähde- ja tulosdokumenttien erot (Ensign, Goldfarb & Pepper, 1998).

N-konversio-ohjelmat jaetaan vielä yleisiin N-konversio-ohjelmiin (general N-converter) ja erityisiin N-konversio-ohjelmiin (specific N-converter) (Ensign, Goldfarb & Pepper, 1998). Yleinen N-konversio-ohjelma pystyy muuntamaan lähdedokumentit valitun dokumenttityyppimäärittelyn mukaiseksi, mutta erityiset N-konversio-ohjelmat muuntavat lähdeaineiston aina yhteen muotoon.

Koska käsittelemme tässä tutkielmassa tekstitiedostojen muuntamista SGML-muotoon, rajoitumme vertailemaan N-konversio-ohjelmia ja tarkoitamme jatkossa termillä konversio-ohjelma em. ohjelmia.

3 KONVERSION TEKNINEN TOTEUTTAMINEN

Jokainen konvertointitilanne on pääsääntöisesti erilainen, joten ei ole olemassa yhtä ainoaa oikeaa tapaa suorittaa konvertointi. Erilaisissa tilanteissa on tärkeää valita tilanteeseen sopiva konvertointistrategia ja käyttää oikeita työvälineitä. Konvertoinnin onnistuminen ja laadukkuus riippuukin pitkälti konvertoijan asiantuntemuksesta, mutta myös huolellisesta lähdeaineiston analysoinnista.

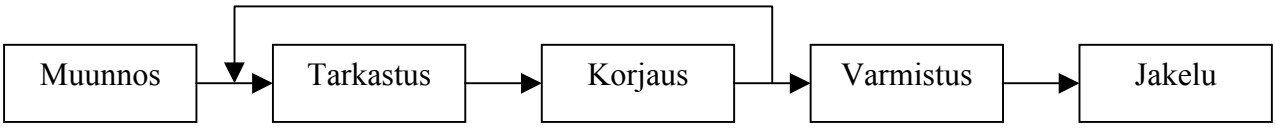
Tässä luvussa tarkastelemme aluksi yksinkertaisen konversioprosessin jakamista eri vaiheisiin ja tutkimme miten tietomassan konvertointi etenee kussakin vaiheessa. Tämän jälkeen perehdymme konvertoitavan aineiston analysoinnissa ja konversio-ohjelman tekemisessä huomioitaviin seikkoihin. Lopuksi tarkastelemme erilaisia konvertointiin liittyviä tekniikoita ja strategioita.

3.1 Perusprosessi

Yksinkertaisimmillaan konvertointi voidaan tehdä esimerkiksi käyttäen tekstinkäsittelyohjelmaa tai awk-skriptiä (Travis & Waldt, 1995). Tällöin aineistossa esiintyvät tunnisteet korvataan vastaavilla valitun esitysmuodon mukaisilla tunnisteilla. Helpoissa tapauksissa aineistoa ei tarvitse läpikäydä kuin yhden tai kaksi kertaa.

Konvertointi on kuitenkin harvoin näin helppoa. Yksityiskohtaisemman aineiston, kuten teknisen dokumentaation, konvertointi vaatii suuremman joukon työvälineitä ja useampia läpikäyntejä sekä enemmän ihmistyötä, jolloin konvertoinnin kustannukset ovat selvästi korkeammat.

Travis ja Waldt (1995) jakavat yksinkertaisen konversioprosessin viiteen eri vaiheeseen, jotka ovat muunnosvaihe (convert), tarkastusvaihe (parse), korjausvaihe (correct), varmistusvaihe (verify) ja jakeluvaihe (deliver). Suurempien aineistojen kohdalla tässä esitettyjä vaiheita joudutaan mahdollisesti jakamaan pienempiin osiin ja toistamaan eri vaiheita useasti. Kuvassa 3.1 on esitetty perusprosessin eteneminen.



Kuva 3.1. Konvertoinnin perusprosessi (Travis & Waldt, 1995).

Muunnosvaiheessa dokumentin sisältö muunnetaan SGML-muotoon, jotta sitä voidaan käsitellä SGML-välineillä. Mikäli dokumentti on pitkä tai vaikea konvertoida, kannattaa se jakaa erillisiin osiin, jolloin jokainen osa voidaan muuntaa erikseen (Severson, 1995). Esimerkiksi kirja voidaan jakaa lukuihin ja konvertoida jokainen luku toisista riippumatta. Toisaalta myös vaikeasti konvertoitavat osat voidaan erottaa muusta aineistosta, jolloin eri osien konvertointiin voidaan käyttää kyseiseen tilanteeseen soveltuvaa konvertointitekniikkaa. Käsittelemme muunnosvaiheessa käytettäviä konvertointi-tekniikoita tarkemmin luvussa 3.4.

Koska SGML-konvertoinnin tarkoituksena on saattaa informaatio tietyn rakenteen mukaiseksi ja täten antaa sille lisäarvoa, täytyy muunnosvaiheen tuloksena saadun SGML-merkatun dokumentin noudattaa dokumenttityyppimäärittelyn sääntöjä. Muunnosvaiheen tuloksena saadun dokumentin laadukkuus riippuu pitkälti konvertoijan asiantuntemuksesta, käytetyn muunnosohjelman ominaisuuksista ja muunnoksen suorittamisen huolellisuudesta. Muunnettu dokumentti ei aina vastaa DTD:n mukaisia määrittelyjä: siitä voi puuttua dokumenttityyppimäärittelyssä vaadittuja merkkauksia tai dokumentti voi sisältää virheellisiä merkkauksia.

Tarkastusvaiheessa merkkauksen oikeellisuus tarkastetaan SGML-jäsentimen avulla. Jäsennin läpikäy SGML-merkatun dokumentin ja tarkastaa, onko dokumentin sisältö kyseisen dokumenttityyppimäärittelyn mukainen. Mikäli jäsennin löytää syntaktisia virheitä dokumentin rakenteesta, se tulostaa virheilmoitukset, joiden avulla konvertoija löytää virheelliset kohdat. Tyypillinen virhe on esimerkiksi dokumenttityyppimäärittelyssä pakolliseksi merkityn lopputunnisteen puuttuminen. Virheiden korjaamisen jälkeen dokumentti jäsennetään uudelleen ja prosessia toistetaan kunnes dokumentti vastaa syntaktisesti DTD:ssä kuvattua rakennetta.

SGML-jäsentimen avulla pystytään selvittämään merkatun aineiston rakenteellinen oikeellisuus, mutta ei tiedon oikeellisuutta tai riittävyyttä. Vaikka SGML-jäsennin ei löydäkään merkatusta aineistosta virheitä, täytyy aineisto vielä läpikäydä sisällön oikeellisuuden varmistamiseksi. Tietokone ei tähän pysty, vaan informaation oikeellisuuden varmistamiseen tarvitaan aina ihmisen työpanosta.

Varmistusvaihe onkin konvertoijan suorittamaa käsityötä, jonka tarkoituksena on varmistaa tiedon laadukkuus ja semanttinen oikeellisuus. Konvertoija vertaa lähdeaineistoa SGML-merkattuun aineistoon ja tarkastaa, että rakenne on merkattu oikein, ja ettei informaatiosta puutu mitään oleellista, eikä se sisällä ylimääräistä, epärelevanttia tietoa. Varmistusvaihe vastaa pitkälti korjauslukemista, mutta sen suorittaminen on nopeampaa, koska jokaista sanaa tai merkkiä ei välttämättä tarvitse läpikäydä.

Jakeluvaiheen tavoitteena on saattaa aineisto siihen käyttöön, mihin se alunperin on tarkoitettu. Tässä vaiheessa suoritetaan tarvittavat lopputoimenpiteet, jotta dokumentti on loppukäyttäjän saatavilla ja käytettävissä. Jakeluvaiheeseen sisältyy esimerkiksi pitkästä dokumentista lohkottujen osien yhdistämistä tai dokumentin siirtämistä eri hakemistoon tai palvelimelle.

3.2 Dokumenttianalyysi

Dokumentaation konvertointi alkuperäisestä muodostaan SGML-merkattuun muotoon vaatii huolellista dokumenttien analysointia informaation rakenteiden määrittelemiseksi. Mitä huolellisemmin ja tarkemmin analyysi tehdään, sitä paremmin dokumenttityyppimäärittelyn rakenne vastaa informaation rakennetta. Resurssien puutteessa massiivisia aineistoja ei kuitenkaan aina ole mahdollista analysoida kokonaan, jolloin kaikkia lähdeaineistossa esiintyviä tilanteita ei välttämättä ole huomioitu dokumenttityyppimäärittelyssä, eikä myöskään konversio-ohjelmassa.

Dokumenttianalyysillä (document analysis) tarkoitetaan dokumentin (tai dokumenttien) informaatiorakenteen ja tyypin määrittämistä (Korhonen & Kuisma, 1999). Dokumenttianalyysin pohjalta voidaan määritellä organisaation tietotarpeeseen soveltuva

dokumenttityyppimäärittely. Dokumenttianalyysissä selviää myös konvertoinnissa tärkeää tietoa lähdeaineistosta, kuten sen muoto ja laatu sekä lähdedokumenttien rakenteisuuden aste (Severson, 1995).

Dokumenttianalyysi on iteratiivinen prosessi, joka vaatii yleensä useita läpikäyntejä ennen kuin sen tuloksena muodostuva dokumenttityyppimäärittely on tarpeeksi kattava. Travis ja Waldt (1995) jakavat analyysiprosessin neljään vaiheeseen:

- *Esianalyysissä* (preliminary analysis) analysoidaan laaja otos aineistosta ja määritellään millaisia rakenteita dokumenteissa esiintyy ja mitkä ovat niiden väliset suhteet. Esianalyysin tuloksena syntyy dokumenttianalyysiraportti (document analysis report).
- *Ryhmäanalyysin* (expanded group analysis) tarkoituksena on laajentaa esianalyysia ja varmistaa, että esianalyysin mukainen rakenne tukee kaikkien osapuolien informaatiotarpeita.
- *DTD:n kehittäminen* (DTD development) tapahtuu edeltäneiden analyysien perustella.
- *Testauksen ja toteutuksen* (testing and implementation) tarkoituksena on testata, että kehitetty dokumenttityyppimäärittely soveltuu organisaation informaatiotarpeeseen.

Yllä esitetyn prosessin vaiheita toistetaan kunnes syntynyt dokumenttityyppimäärittely soveltuu organisaation tarpeeseen ja kattaa lähdeaineistossa esiintyvät tärkeimmät rakenteet.

Dokumentin analysointiin voidaan käyttää kolmea eri lähestymistapaa. Ulkoasun perusteella analysoitaessa (format tagging) erotetaan dokumentissa esiintyvät elementit esimerkiksi lihavoinnin, alleviivauksen, kirjasinkoon tai tasauksen avulla. Rakenteen mukaisessa analysoinnissa (structure tagging) keskitytään dokumentin elementtien hierarkiaan. Tällöin luodaan omat tunnisteet esimerkiksi eri tason otsikoille. Sisällön mukaisessa analysoinnissa (content tagging) keskitytään dokumentin sisällön ja sen tarkoituksenmukaisuuden tunnistamiseen. Dokumenttianalyysissä on hyvä soveltaa kaikkia edellä esitettyjä lähestymistapoja, jolloin dokumenttien eri piirteiden informaatioarvo saadaan huomioitua kattavasti.

Dokumentin rakennetta voidaan havainnollistaa esimerkiksi erilaisilla kaavioilla (tree diagram, box diagram ja railroad model diagram) tai esimerkatuilla sivuilla (sample marked-up pages) (Travis & Waldt, 1995). Rakennemallien tarkoituksena on visualisoida elementtien suhteita, jolloin ihmisen on helpompi omaksua kokonaisrakenne.

3.3 Konversiomäärittely

Edellä havaitsimme, että konversioprosessia ei koskaan voida toteuttaa täysin automaattisesti, vaan se vaatii viimeistään varmistusvaiheessa ihmisen työpanosta. Konversio-ohjelman tarkoituksena on automatisoida ja nopeuttaa muunnosvaihetta. Kuitenkin konversio-ohjelman luomiseksi ohjelmoija tarvitsee tarkkaa tietoa siitä, miten alkuperäisestä aineistosta löydetään loogiset rakenteet ja miten nämä rakenteet tulee esittää SGML-muodossa.

Konversiomäärittelyllä (conversion specification document, conversion spec) tarkoitetaan dokumenttia, jossa kuvataan yksityiskohtaisesti kuinka lähdeaineiston automaattinen konvertointi SGML-muotoon suoritetaan (Travis & Waldt, 1995). Konversiomäärittely voi pohjautua pitkälti dokumenttityyppimäärittelyyn, jota tarkennetaan lähdeaineiston käsittelyyn liittyvillä säännöillä. Konversiomäärittelyssä kuvataan lähdeaineistosta löytyvät tunnisteet ja niiden käsittelytavat sekä se, miten informaatio esitetään dokumenttityyppimäärittelyn vaatimassa muodossa.

Koska konversiomäärittely yleensä muuttuu konvertointiprojektin edetessä, siinä esitetyt säännöt ja kuvaukset tulisi esittää loogisessa järjestyksessä, jotta dokumentin ylläpito ja sääntöjen muuttaminen olisi helppoa. Selkeyden vuoksi dokumentin rakenne on hyvä laatia dokumenttityyppimäärittelyn rakenneosien, ei lähdeaineiston, pohjalle. Yksinkertaisimmillaan konversiomäärittely on taulukko, jossa riveillä on esitetty DTD:ssä määritellyt tunnisteet ja sarakkeilla sellaisten tapausten kuvaukset, jolloin kyseinen tunniste merkataan dokumenttiin (Willner, 1998b). Muunnossäännöt voidaan kuvata sanallisesti tai käyttäen jotain formaalimpaa esitystä. Huolimatta siitä mitä tapaa käytetään, on sääntöjen oltava täsmällisiä, jotta konversio-ohjelman luominen on mahdollista.

Alla on yksinkertainen esimerkki säännöistä, jollaisia konversiomäärittelyssä voi esiintyä. Näiden sääntöjen mukaisesti konvertoitavat dokumentit sisältävät vain sisennettyjä otsikoita ja kappaleita. Otsikot merkataan `<otsikko>`-tunnisteilla ja kappaleet `<kappale>`-tunnisteilla (vrt. Travis, 1997).

`<otsikko>` Tunniste merkataan, kun dokumentissa esiintyy rivi, joka alkaa vähintään kolmella välilyönnillä joita seuraa yksi tai useampia kirjaimia tai numeroita ja rivinvaihtomerkki. Alku- ja lopputunnisteen väliin sijoitetaan ensimmäisestä ei-tyhjeestä alkava ja rivinvaihtoon päättyvä merkkijono.

`<kappale>` Tunniste merkataan, kun dokumentissa esiintyy kappale, jossa on vähintään kaksi riviä tekstiä. Kappaletta edeltää ja seuraa yksi tyhjä rivi (rivi sisältää vain ja ainoastaan rivinvaihtomerkin). Alkutunniste merkataan tekstiä edeltävän rivin ja lopputunniste tekstiä seuraavan rivin rivinvaihtomerkin tilalle.

Lähdedokumenttien loogisten rakenteiden esittämisessä on usein huomattavia eroja, jonka vuoksi konversiomäärittelyn tekeminen ei välttämättä ole aivan yksinkertainen tehtävä. Esimerkiksi kappaleiden sisennykset voivat olla yhdessä dokumentissa toteutettu välilyönnein ja toisessa tabulaattorien avulla. Konversio-ohjelman sääntöjä muodostettaessa tällaiset tapaukset täytyy huomioida, jotta muunnoksen tarkastus- ja korjausvaiheet eivät vaadi kohtuuttomasti panostusta.

Pienissä konvertointiprojekteissa tai yksinkertaisten aineistojen konvertoinnissa ei konversiomäärittelyä välttämättä tarvita, mutta suurien tietomassojen ja mahdollisesti useita osastoja käsittävien organisaatioiden konvertointiprojektit vaativat tarkan konversiomäärittelyn onnistuakseen. Konversiomäärittelyn avulla konvertointiprojektiin osallistuvat henkilöt ymmärtävät toisiaan paremmin ja heidän on helpompi kommunikoida keskenään. Hyvin tehtyä konversiomäärittelyä voidaan tietyiltä osiltaan hyödyntää myös tulevissa konvertointiprojekteissa. Esimerkiksi edellisen konvertoinnin ongelmakohdat pystytään havaitsemaan paremmin sen avulla. Lisäksi konversiomäärittely on erityisen hyödyllinen konversio-ohjelman ohjelmoijalle, joka saa tarkat ohjeet sääntöjen koodaamiseksi muunnosohjelmaan.

3.4 Konvertointitekniikat

Dokumenttien konvertointi SGML-muotoon voidaan tehdä joko manuaalisesti tai käyttäen erillistä konversio-ohjelmaa, jonka tarkoitus on vähentää muunnosprosessiin tarvittavaa ihmistyötä. Ihminen on hitaampi ja kalliimpi työntekijä kuin tietokone, joten on järkevää pyrkiä minimoimaan ihmistyön määrää automatisoimalla muunnosvaihetta, varsinkin jos aineisto on pitkä. Jos muunnettavaa tekstiä ei ole paljon, on muunnos todennäköisesti helpompi ja nopeampi suorittaa käsin.

Toisaalta myös lähdeaineiston laatu vaikuttaa valittavaan konvertointitapaan. Mikäli aineisto on moniselitteistä ja epäjohdonmukaista, on tarvittavan konversio-ohjelman tekeminen hankalaa. Kaikkien aineistossa esiintyvien tilanteiden etsiminen ja koodaaminen muunnosohjelmaan on aikaa vievää ja tekee muunnosohjelmasta helposti suuren ja raskaan.

3.4.1 Manuaalinen konvertointi

Manuaalinen konvertointi (manual conversion) on konvertoijan suorittamaa käsityötä, jossa lähdeaineisto käydään alusta loppuun esimerkiksi tekstieditorin avulla ja muokataan se valitun dokumenttityyppimäärittelyn mukaiseksi (Travis & Waldt, 1995). Yksinkertaisimmillaan tämä tarkoittaa lähdeaineistossa esiintyvien tunnisteiden korvaamista dokumenttityyppimäärittelyn mukaisilla tunnisteilla.

Manuaalista konvertointia voidaan tehostaa SGML-editorin avulla, joka pystyy jäsentämään dokumentin ja paljastamaan havaitsemansa virheet jo muunnosvaiheessa. Lisäksi editoreilla voidaan yleensä lisätä dokumenttityyppimäärittelyn mukaisia merkkauksia helposti dokumenttiin. Tällainen *interaktiivinen konvertointi* (interactive assisted conversion) on eräs manuaalisen konvertoinnin muoto.

Manuaalista konvertointia käytettäessä konvertoitavan aineiston muodolle ei aseteta vaatimuksia, koska dokumentin merkkaus, ja merkkauksen oikeellisuus, on konvertoijan vastuulla. Itse asiassa mikäli konvertoitavan dokumentin rakennetta ei ole lainkaan merkitty, vaan se koostuu ainoastaan informaatiosisällöstä, on manuaalinen konvertointi usein ainoa mahdollinen tapa suorittaa konvertointi (Travis & Waldt, 1995).

Manuaalinen konvertointi soveltuu hyvin tilanteisiin, jossa tarvittavan konversio-ohjelman tekeminen on vaikeaa tai mahdotonta. Tällaisia tilanteita aiheuttavat esimerkiksi rakenteellisesti huonokuntoiset aineistot, monimutkaiset taulukot ja vain harvoin aineistossa esiintyvät erikoistapaukset. Lisäksi manuaalinen konvertointi on käyttökelpoinen konvertoitaessa lyhyitä aineistoja, sillä konvertointityö on tällöin todennäköisesti nopeampi suorittaa käsin kuin tehdä muunnokseen soveltuva konversio-ohjelma.

Aineiston konvertointi manuaalisesti on kuitenkin hidasta ja kallista. Tietokone on ihmistä nopeampi käsittelemään tapauksia, joissa muunnokselle on olemassa selvät säännöt. Usein onkin järkevä automatisoida muunnosta ja soveltaa manuaalista konvertointia vain tilanteisiin, joita konversio-ohjelma ei osaa käsitellä (Travis & Waldt, 1995).

3.4.2 Ohjelmallinen konvertointi

Ohjelmallinen konvertointi (programmatic conversion) tarkoittaa muunnosta, jossa lähdeaineiston muunnoksen SGML-muotoon suorittaa erityinen konversio-ohjelma. Konversio-ohjelma voi olla esimerkiksi skripti tai jollain perinteisellä ohjelmointikielellä tehty ohjelma, joka läpikäy lähdedokumentin ja muuntaa sen tiettyjen, ohjelmaan koodattujen sääntöjen mukaan halutun muotoiseksi, SGML-merkatuksi tulosdokumentiksi.

Markkinoilla on myös erityisiä ohjelmistoja, joiden tarkoitus on helpottaa konversio-ohjelman tekemistä ja itse konversiota. Ohjelmat ovat tapahtumapohjaisia (event-driven), jossa käyttäjän on määriteltävä mikä toimenpide suoritetaan kun prosessoitavassa dokumentissa havaitaan etsitty merkkijono (Kuikka & Nikunen, 1997). Eräissä välineissä ei dokumenttien konvertoimiseksi tarvitse kirjoittaa varsinaista ohjelmakoodia, mutta

yleisesti voimme ajatella ohjelmallisen konversion vaativan ohjelmointitaitoa. Tarkoitamme jatkossa termillä konversio-ohjelma, sellaista ohjelmaa tai ohjelmakoodia, joka muuntaa lähdedokumentin SGML-merkatuksi tulosdokumentiksi ja termillä konversioväline em. konversio-ohjelmien kehittämiseen tarkoitettua kehitysympäristöä.

Mikäli konversio-ohjelman tekemiseen käytetään perinteisiä ohjelmointikieliä, on järkevää valita ohjelmointikieli, joka sisältää tehokkaat tekstin muokkausrutiinit. Tällaisia ohjelmointikieliä ovat esimerkiksi Lisp, Snobol ja Perl (Travis & Waldt, 1995). Perinteisten ohjelmointikielten heikkoutena konversio-ohjelmaa tehdessä on SGML-tuen puute. Konversiovälineet sisältävät yleensä SGML-jäsentimen, jonka avulla ohjelman kehitysvaiheessa suoritetun konversion syntaktinen oikeellisuus on nopeasti selvitettävissä.

Ohjelmallinen konvertointi onnistuu parhaiten rakenteellisesti hyväkuntoiselle lähdeaineistolle. Tällöin konversio-ohjelmaan voidaan luoda yksiselitteiset säännöt, joiden mukaan tunnisteet merkataan tulosdokumenttiin. Mikäli aineisto on rakenteellisesti huonokuntoinen, on konversio-ohjelman tekeminen vaikeaa ja siitä tulee helposti laaja. Varsinkin laajoissa aineistoissa on usein poikkeuksia ja erikoistilanteita, jotka esiintyvät dokumentaatiossa vain muutamia kertoja. Tällaiset tapaukset on järkevää konvertoida manuaalisesti.

Joskus lähdeaineiston rakenneinformaatio on selvästi havaittavissa, jolloin sääntöjen koodaaminen konversio-ohjelmaan on helppoa. Dokumentin rakenneosat voidaan usein tunnistaa automaattisesti ulkoasun perusteella. Esimerkiksi otsikot ovat yleensä kirjoitettu suuremmalla kirjasinkoolla kuin leipäteksti, joten konversio-ohjelmaan voidaan muodostaa sääntö, joka merkkaa tulosdokumenttiin vaikka kirjasinkoolla 14 kirjoitetut kappaleet `<otsikko>`-tunnisteella ja kirjasinkoolla 12 kirjoitetut kappaleet `<kappale>`-tunnisteella.

Konversio-ohjelmaan koodattavat säännöt voivat perustua myöskin tekstissä esiintyviin avainsanoihin. Mikäli merkattavat elementit tunnistetaan informaation, ei tyylin tai esitysmuodon perustella, puhutaan *ohjelmallisesta sisällöntunnistuskonvertoinnista* (programmatic content interrogation conversion) (Travis & Waldt, 1995). Tällöin esimerkiksi kappale, joka alkaa "Huom!"-merkkijonolla voidaan merkata `<huom>`-tunnisteella `<kappale>`-tunnisteen sijaan.

3.5 Konvertointistrategiat

Konversioprosessi ei aina ole niin suoraviivainen kuin edellä kuvattiin, koska konvertointitilanteet vaihtelevat suuresti eri organisaatioissa. Kuhunkin konversioprosessiin saattaa sisältyä erilaisia välivaiheita, jotka vaativat enemmän tai vähemmän ihmistyötä onnistuakseen. Myöskin lähdeaineistoissa on eroja: dokumentteja voi olla tuotettu usealla eri työvälineellä tai ne voivat olla rakenteellisesti epämääräistä.

Konvertointi voidaan toteuttaa käyttäen tilanteeseen soveltuvaa konvertointistrategiaa. Millainen strategia valitaan riippuu pitkälti konvertoitavan aineiston muodosta, laajuudesta ja käyttötarkoituksesta. Pienten ja yhtenäisten aineistojen konvertointi on yleensä parasta toteuttaa suoraviivaisesti muunnosvaiheesta aina jakeluvaiheeseen. Suuremmissa aineistoissa, joissa dokumenttien rakenne ja muoto ei välttämättä ole yhtenäinen, konvertoinnin helpottamiseksi ja koko prosessin tehostamiseksi voidaan käyttää tilanteeseen sopivaa konvertointistrategiaa.

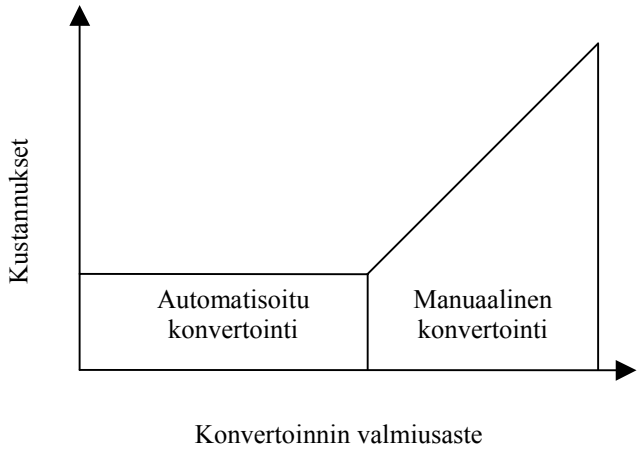
Seuraavaksi tutustumme kolmeen eri strategiaan, mutta konvertointitilanteiden heterogeenisuudesta johtuen erilaisia konvertointistrategioita on olemassa huomattavasti enemmän.

3.5.1 Jälkivarmistus

Yksinkertaisessa konversioprosessissa varsinaisen muunnoksen jälkeen läpikäydään vielä useita vaiheita ennen kuin konvertoitu aineisto on lopullisessa muodossa. Viimeiset vaiheet liittyvät tiiviisti konvertoidun aineiston laadun varmistamiseen, mikä on ihmisen suorittamaa manuaalista työtä. Mikäli tiedetään, että konvertoitua aineistoa tullaan muuttamaan paljon tulevaisuudessa, laadun varmistustoimenpiteet voidaan jättää suorittamatta konvertoinnin yhteydessä.

Jälkivarmistus (attrition) on konvertointistrategia, jossa aineisto konvertoidaan SGML-muotoon ja sille suoritetaan tarkastus- ja korjaustoimenpiteet, mutta konvertoidun aineiston laatu varmistetaan vasta varsinaisen käytön yhteydessä (Travis & Waldt, 1995).

Kuvasta 3.2 näemme miten automatisoitu konvertointi ja manuaalisesti tehtävä konvertointityö vaikuttavat kustannuksiin suhteessa koko konvertoinnin valmistumiseen.



Kuva 3.2. Automatisoidun ja manuaalisen konvertoinnin vaikutus kustannuksiin (Travis & Waldt, 1995).

Koska manuaalinen konvertointi kasvattaa kustannuksia huomattavasti automatisoituun konvertointiin nähden, on tietyissä tilanteissa kannattavaa jättää laadun varmistustoimenpiteiden suorittaminen myöhemmäksi. Jälkivarmistus-strategia onkin käyttökelpoinen silloin, kun konvertoitua aineistoa muokataan lähitulevaisuudessa laajasti tai osia siitä korvataan kokonaan uudella aineistolla (Travis & Waldt, 1995). Tällöin ei ole järkevää käyttää resursseja aineiston laadukkuuden varmistamiseen konvertoinnin yhteydessä, vaan laadun varmistamistoimenpiteet kannattaa suorittaa vasta muokatulle aineistolle.

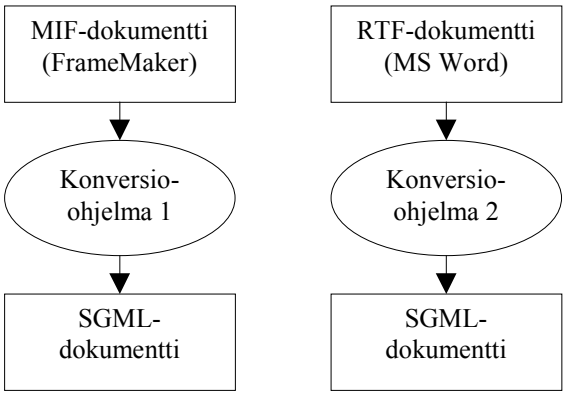
Jälkivarmistus-strategia ei ole käyttökelpoinen, jos konvertoitua aineistoa on tarkoitus jakaa eteenpäin tai hyödyntää eri julkaisuvälineillä (esimerkiksi CD-ROM -levyllä) (Travis & Waldt, 1995). Tällaisissa tilanteissa aineiston laadun varmistusta ei voida jättää

loppukäyttäjän vastuulle, sillä hän ei välttämättä osaa havaita aineistossa olevia sisältövirheitä.

3.5.2 Rainbow DTD

Varsinkin suurissa organisaatiossa dokumentaation tuottamiseen saatetaan käyttää useita tekstinkäsittelyohjelmia ja niiden eri versioita. Tekstinkäsittelyohjelmat tallentavat dokumentit yleensä oman tallennusmuotonsa mukaisesti, jotka eivät ole yhteensopivia keskenään. Tällöin eri tekstinkäsittelyohjelmilla kirjoitettujen dokumenttien konvertoimiseksi SGML-muotoon joudutaan tekemään useita muunnosohjelmia.

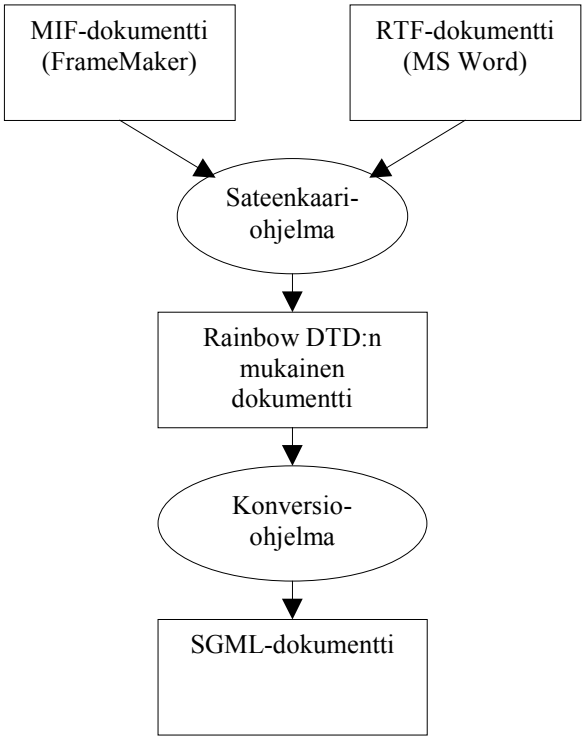
Kuva 3.3 esittää tilannetta, jossa kaksi eri tekstinkäsittelyohjelmalla kirjoitettua dokumenttia konvertoidaan SGML-muotoon. Vasemman puoleinen dokumentti on kirjoitettu FrameMaker-ohjelmalla MIF-muotoon ja oikean puoleinen MS Word-ohjelmalla RTF-muotoon. Koska eri muodoissa tallennetuissa dokumenteissa informaatio esitetään eri tavoin, täytyy kummallekin dokumentille koodata oma konversio-ohjelma.



Kuva 3.3. Kahden eri tallennusmuodossa olevan dokumentin konvertointi.

Muunnosohjelmien tekeminen usealle tallennusmuodolle on kallista ja hyvin työlästä. Ongelman ratkaisee *Rainbow DTD*, jonka tarkoituksena on helpottaa useassa tallennusmuodossa olevien dokumenttien konvertointia ja vähentää tarvittavien

muunnosohjelmien määrää (Electronic Book Technologies, 1993; Sklar, 1995). Rainbow DTD on esitetty liitteessä 4.



Kuva 3.4. Dokumenttien konvertointi SGML-muotoon sateenkaari-strategiaa käyttäen.

Sateenkaari-strategiassa koko konvertoitava aineisto muunnetaan yhteiseen Rainbow DTD:n muotoon, jonka jälkeen tarvitaan vain yksi konversio-ohjelma muuntamaan dokumentit ko. esitysmuodosta halutun dokumenttityyppimäärittelyn mukaiseksi (Travis & Waldt, 1995). Rainbow DTD ei siis ole tarkoitettu dokumenttien lopulliseksi tallennusmuodoksi, vaan ainoastaan konvertoinnin välivaiheeksi.

Alkuperäisen aineiston muuntamiseen on kehitetty *sateenkaari-ohjelmia* (Rainbow Maker), jotka pystyvät muuntamaan yleisimpien tekstinkäsittelyohjelmien tiedostot Rainbow DTD:n muotoon. Koska sateenkaari-ohjelmat muuntavat dokumentit aina Rainbow DTD:n mukaiseksi, luokitellaan ne Ensignin, Goldfarbin ja Pepperin (1998) mukaan erityisiin N-konversio-ohjelmiin.

Sateenkaari-menetelmää käyttämällä konvertoijan ei tarvitse periaatteessa tehdä kuin yksi konversio-ohjelma, koska alkuperäisen aineiston konvertointi Rainbow DTD:n muotoon onnistuu valmiiden sateenkaari-ohjelmien avulla.

Kuvassa 3.4 on esitetty MIF- ja RTF -dokumenttien konvertoiminen SGML-muotoon sateenkaari-strategiaan pohjautuen. Molemmat dokumentit muunnetaan sateenkaari-ohjelmalla yhteiseen Rainbow DTD:n muotoon. Tämän jälkeen dokumenttien muuntaminen halutun dokumenttityyppimäärittelyn mukaiseksi tehdään yhdellä konversio-ohjelmalla.

3.5.3 Esikonvertointi

Vaikka konvertoitavasta aineistosta analysoitaisiin esimerkiksi puolet ennen konversiomäärittelyn tekemistä ja konversio-ohjelman koodaamista, voivat analysoimattomat dokumentit sisältää tapauksia, jotka aiheuttavat ongelmia konvertointiprosessissa. Konvertointiprosessin aikana ilmenevät yllätystilanteet hidastavat prosessin etenemistä ja lisäävät työmäärää. Löydettyjen ongelmien korjaamiseksi dokumenttityppimäärittelyyn ja konversio-ohjelmaan joudutaan pahimmillaan tekemään ehkä suuriakin muutoksia, joiden jälkeen aiemmin konvertoitu aineisto on konvertoitava uudelleen.

Esikonvertointi (pre-conversion testing) on hyödyllinen konvertointistrategia, kun konvertointiprosessin aikana esiintyvien yllätystilanteiden mahdollisuus halutaan minimoida. Esikonvertoinnin avulla pyritään selvittämään aineistossa esiintyvät ongelmat ennen varsinaisen konvertoinnin aloittamista. Täten lopullisen, konvertoidun aineiston laadun varmistaminen on nopeampaa ja helpompaa, koska ongelmia aiheuttavat kohdat on huomioitu ja ratkaistu jo esikonvertointivaiheessa. Esikonvertoinnin avulla pystytään lisäksi arvioimaan konvertointiprosessiin kuluva aika tarkemmin.

Esikonvertointi toteutetaan ennen lopullisen konvertoinnin aloittamista ja sen tarkoituksena on antaa lisätietoa aineistossa mahdollisesti esiintyvistä, dokumenttianalyysissä havaitsemattomista, ongelmakohdista. Esikonvertoinnissa koko aineisto muunnetaan

valitun dokumenttityyppimäärittelyn mukaiseen muotoon ja muunnoksen yhteydessä havaituista ongelmakohdista ja virheistä tehdään tarkat muistiinpanot (Travis & Waldt, 1995).

Ongelmakohtien tunnistamiseksi on hyödyllistä lisätä konversio-ohjelmaan erillinen tunniste, jolla merkataan muunnettavassa dokumentissa esiintyvät ongelmakohdat. Konversio-ohjelma lisää tällöin dokumenttiin esimerkiksi `<ongelma>`-alkutunnisteen, kun kyseiselle kohdalle ei löydy käsittelysääntöä ja `</ongelma>`-lopputunnisteen, kun se pystyy jatkamaan dokumentin muunnosta normaalisti.

Koko aineiston prosessoinnin jälkeen kirjatut ongelmakohdat ratkaistaan ja vaadittavat muutokset ja lisäykset tehdään sekä dokumenttityyppimäärittelyyn että konversio-ohjelmaan. Konvertoitua aineistoa ei tämän jälkeen enää tarvita vaan se voidaan hävittää. Lopullinen lähdeaineiston konvertointi suoritetaan tämän jälkeen käyttäen uutta dokumenttityyppimäärittelyä ja konversio-ohjelmaa.

4 KONVERSIO PROJEKTINA

Organisaation dokumenttiaineiston konvertointi alkuperäisestä esitysmuodosta SGML-merkattuun muotoon on yleensä vaikea ja kallis projekti. Konversioprojektin edetessä kohdataan erilaisia ongelmatilanteita, joita pyritään ennalta ehkäisemään tarkalla suunnittelulla. Varsinkin suurissa konvertointiprojekteissa tavoitteen saavuttaminen määritellyssä aikataulussa ja asetetulla budjetilla vaatii myös tarkkaa projektin seurantaa, ongelmatilanteiden tunnistamista ja niistä toipumista.

Tässä luvussa perehdymme konvertoinnin suunnittelussa huomioitaviin seikkoihin. Tutustumme myös konvertoinnin vaiheistukseen ja lisäksi tarkastelemme suurempien informaatiokokonaisuuksien konvertoinnin hallintaan tarvittavia seurantatoimenpiteitä muutaman esimerkin avulla.

4.1 Suunnittelu

Tarkalla projektin suunnittelulla suurien ja monimutkaisten aineistojen konvertointi on mahdollista suorittaa onnistuneesti, mutta ilman yksityiskohtaista suunnitelmaa hyvin pienenkin aineiston konvertointi voi muodostua hankalaksi (Willner, 1998b). Kuitenkin konvertoinnin suunnittelun merkitystä usein vähätellään, eikä sitä nähdä tärkeänä projektin onnistumisen kannalta.

Hyvin tehdyllä konvertoinnin suunnittelulla pyritään minimoimaan virheiden ja mahdollisten yllätystilanteiden lukumäärä ennen varsinaisen konvertoinnin aloittamista. Myös koko projektiin kuluva työmäärä ja organisaatiolle aiheutuvat kustannukset ovat suunnitelman avulla helpommin arvioitavissa.

Hyvän suunnittelun avulla selviää mitä on tarkoitus konvertoida, missä järjestyksessä ja minkälaisilla resursseilla (Travis & Waldt, 1995). Konvertoinnin suunnittelun tuloksena syntyvä *konversiosuunnitelma* (conversion plan) kattaa koko konversioprojektin.

Jokainen konversioprojekti on jossain määrin yksilöllinen, joten lähtökohtana konvertoinnin suunnittelemiselle voidaankin pitää aineiston pohjalta tapahtuvaa projektin laajuuden hahmottamista. Tutustumme aluksi seikkoihin, jotka tulee huomioida konversioprojektin suunnittelussa.

Projektin aikataulun ja tarvittavien henkilöstöresurssien suunnittelemiseen vaikuttavat suoraan esimerkiksi lähdeaineiston sivumäärä, fyysinen muoto ja rakenteellinen kunto sekä käytettävään dokumenttityyppimäärittelyyn liittyvät kysymykset. Näiden perustietojen pohjalta voidaan suunnitella konvertoinnissa käytettävä konversiometodiikka.

Konvertoitavan dokumentaation *sivumäärä* ja sen *fyysinen muoto* vaikuttaa suoraan tarvittaviin resursseihin ja projektin aikatauluun (Willner, 1998b). Lähdeaineisto saattaa fyysisesti olla joko paperikopioina tai elektronisessa muodossa. Paperidokumenttien sisältämä informaatio täytyy siirtää elektroniseen muotoon joko kirjoittamalla koko teksti uudelleen jollain tekstieditorilla (tai tekstinkäsittelyohjelmalla) tai käyttämällä apuna automaattisia tekstin tunnistusvälineitä (optical character recognition, OCR) (Severson, 1995).

Elektronisessa dokumentaatiossa ongelmia aiheuttavat useat tallennusmuodot. Yleensä on helpompaa ja nopeampaa konvertoida esimerkiksi 100 000 sivua MS Wordilla tuotettua dokumentaatiota, kuin aineistoa joka koostuu 50 000:sta Wordilla ja 50 000:sta Xywritella kirjoitetusta sivusta (Willner, 1998b). Vielä hankalampi on tilanne, jossa vasta konversioprosessin aikana huomataan, että aineistoa on tuotettukin usealla eri työvälineellä.

Konvertoitavan aineiston *rakenteellinen kunto* aiheuttaa joskus lisätyötä, joka osaltaan nostaa konvertoinnin kokonaiskustannuksia. Lähdeaineisto voi joiltakin osin olla vaikeasti muunnettavissa tai rakenteellisesti huonossa kunnossa, jolloin joitain dokumentteja joudutaan mahdollisesti korjaamaan tai pahimmillaan jopa kirjoittamaan ne kokonaan uudestaan. Löydetyt ongelmakohdat täytyy priorisoida ja etsiä ongelmiin ratkaisut (Travis & Waldt, 1995).

Organisaation toimiala saattaa rajoittaa *dokumenttityyppimäärittelyn valintaa*. Mikäli kyseisellä toimialalla on yleisessä käytössä jokin julkinen dokumenttityyppimäärittely, on

organisaation järkevää hyödyntää tätä DTD:ä. Tällöin konvertoidut dokumentit ovat yhdenmuotoisia esimerkiksi mahdollisten yhteistyökumppaneiden dokumentaation kanssa.

Mikäli organisaatiolle soveltuvaa dokumenttityyppimäärittelyä ei ole saatavilla, on jo suunnitteluvaiheessa ratkaistava kuka toteuttaa dokumenttianalyysin, jonka pohjalta organisaation omaan tietotarpeeseen soveltuva dokumenttityyppimäärittely rakennetaan. Dokumenttianalyysi ja dokumenttityyppimäärittelyn kehittäminen voidaan tehdä itse resurssien rajoissa, mutta joskus on järkevää käyttää myös ulkopuolisia ammattilaisia apuna (Willner, 1998b).

Kun aineiston laajuus ja muoto tunnetaan, voidaan suunnitella millaista *konversiometodiikkaa* käytetään. Konvertoitavan aineiston laajuus ja muoto vaikuttavat käytettäviin konvertointitekniikoihin ja -strategioihin sekä valittaviin ohjelmistoihin ja työvälineisiin.

Kuten aiemmin havaitsimme, manuaalinen konvertointi on hidasta ja kallista suhteessa automatisoituun konvertointiin. Mikäli aineisto on epämääräinen ja moniselitteinen, on manuaalinen konvertointi mahdollisesti automatisoitua konvertointia tehokkaampaa. Kuitenkin konversioprosessin automatisoinnilla voidaan tehostaa prosessia huomattavasti, mikäli aineisto on hyvin laaja. Yleensä tehokkainta on konvertoida vaikeat dokumentaation osat manuaalisesti ja käyttää muutoin automatisoitua konvertointia.

Konvertoinnissa käytettävien ohjelmistojen ja työvälineiden määrä ja tyyppi määräytyy valitun konversiometodiikan perusteella. Valittujen työvälineiden soveltuvuus kyseiseen projektiin testataan prototyyppivaiheen aikana (katso luku 4.2). Kun nämä projektiin liittyvät perustiedot on selvitetty, projektin aikataulun ja muiden tarvittavien resurssien suunnittelu on mahdollista.

Projekteilla on aina selvä alkupiste ja loppupiste. Myös konversioprojektissa ajankäytön suunnittelu on tärkeä osa onnistunutta projektia. Lähdeaineiston analysointiin, konversiomäärittelyn ja konversio-ohjelman tekemiseen täytyy varata tarpeeksi aikaa. Ajankäyttöä suunniteltaessa tulee myös varautua eri vaiheissa esiintyviin ongelmiin ja niiden ratkaisemiseen sekä laadun varmistustoimenpiteiden toteuttamiseen (Willner,

1998b). Projektin aikataulun määrittelyyn vaikuttaa ratkaisevasti millainen konvertointimetodiikka on valittu.

Konvertoiva aineisto täytyy myös priorisoida dokumenttien tärkeyden mukaan (Travis & Waldt, 1995). Tärkeimmät dokumentit määritellään kuuluvaksi kriittiseen aineistoon. Lisäksi koko dokumentaatiolle muodostetaan mielekäs konvertointijärjestys.

Konvertointia ei voida suorittaa ilman henkilöitä, joten projektin suunnittelussa täytyy määritellä myös projektiin osallistuvat ihmiset, ja heille vastuut ja velvollisuudet. Projektiin valittavan henkilöstön määrä riippuu käytössä olevista resursseista ja määritellystä aikataulusta. Konversion onnistuminen vaatii yleensä asiantuntemusta dokumenttien analysoinnin, konversio-ohjelman koodaamisen ja konversion toteuttamisen lisäksi myös dokumenttien informaatiosisällön mukaiselta aihealueelta. Lisäksi on arvioitava uuden järjestelmän käyttöönoton vaikutuksia koko organisaatioon, henkilöstöön ja työskentelytapoihin (Travis & Waldt, 1995).

4.2 Konvertoinnin vaiheistus

Varsinaisen konvertoinnin toteutus voidaan jakaa kolmeen vaiheeseen, jotka ovat prototyyppivaihe (prototype phase), kriittisen aineiston vaihe (critical-need phase) ja loppuvaihe (follow-up phase) (Travis & Waldt, 1995).

Prototyyppivaiheessa testataan valittujen työvälineiden ja ohjelmistojen soveltuvuus konvertointiprosessiin. Tähän vaiheeseen osallistuvat kaikki konvertoinnissa mukana olevat henkilöt, niin konvertoinnin suunnittelijat kuin ohjelmistojen käyttäjätkin. Työvälineiden käyttökelpoisuuden selvittämisen lisäksi prototyyppivaiheen tarkoituksena on kokeilla eri konvertointitekniikoita ja tehdä havaintoja konvertoitavan aineiston rakenteesta. Prototyyppivaiheessa voidaan myös testata konversioprosessin kulku ja eri vaiheiden seuranta sekä työskentelytavat.

Prototyyppivaiheessa valitaan konvertoitavasta aineistosta yksi tai muutama dokumentti, jotka muunnetaan SGML-muotoon. Muunnettavat dokumentit kannattaa valita niin, että

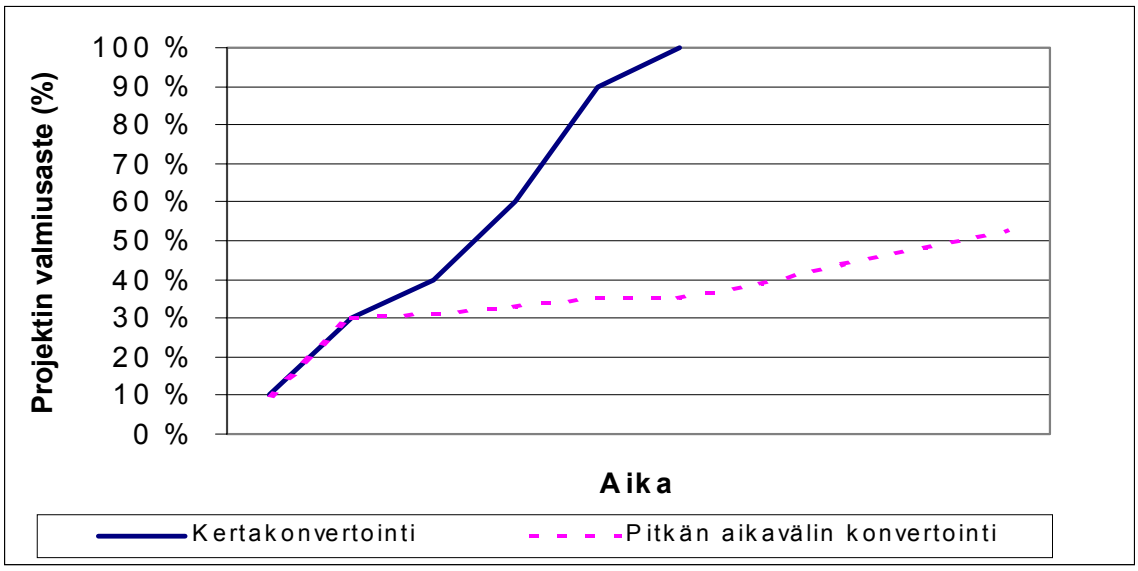
niiden konvertointi ei toisaalta ole liian helppoa eikä toisaalta liian vaikeata. Tällaiset dokumentit edustavat ns. keskiarvo-aineistoa, jonka konvertointi kuvaa parhaiten suuressa konvertointiprojektissa esiintyvät ongelmatilanteet ja konversioprosessissa mahdollisesti olevat puutteet (Travis & Waldt, 1995). Prototyyppivaihe voidaan päättää, kun valittuihin työkaluihin ja ohjelmistoihin sekä itse prosessin kulkuun ollaan tyytyväisiä.

Kriittisen aineiston vaiheessa konvertoidaan aineistosta tärkeimmät kohteet, joiden vuoksi konvertointi alunperin päätettiin toteuttaa, ja joiden konvertoinnista odotetaan saatavan eniten hyötyä. Näiden dokumenttien konvertointi on itse projektin tuloksellisuuden kannalta ensisijaisen tärkeää.

Loppuvaiheessa konvertoidaan aineiston loppuosa eli dokumentit, joita ei määritelty kriittiseen aineistoon. Kun koko aineisto on konvertoitu, voidaan varsinainen konvertointiprojekti päättää. Tämän vaiheen pituus riippuu siitä, valitaanko kertakonvertointi vai työstetäänkö aineistoa pitemmällä aikavälillä. Kummassakin vaihtoehdossa on hyviä ja huonoja puolia.

Kertakonvertoinnin tarkoituksena on konvertoida koko aineisto suoraviivaisesti, niin nopeasti kuin mahdollista. Kertakonvertointi on käyttäjäystävällinen vaihtoehto, koska tällöin käyttäjien ei tarvitse käyttää kahta eri järjestelmää samanaikaisesti. Konvertoinnin jälkeen voidaan vanhaan järjestelmään liittyvät dokumentit hävittää ja siirtyä kerralla uuden järjestelmän käyttöön. Kertakonvertointi voi kuitenkin haitata normaalia tuotantotoimintaa ja tulla kalliimmaksi organisaatiolle.

Toinen vaihtoehto on muuntaa kriittinen aineisto kerralla ja loput dokumentit pitemmällä aikavälillä. Tämän vaihtoehdon etuina ovat kertakonvertointia pienemmät kustannukset ja konvertointiprosessin helpompi hallittavuus. Toisaalta, koska koko konvertointiprojektiin kuluu enemmän aikaa, joutuvat myös käyttäjät käyttämään samanaikaisesti kahta eri järjestelmää. Lisäksi uhkana voidaan pitää tilannetta, jossa kriittisen aineiston konvertoinnin jälkeen loppuaineisto jää ehkä pitkäksikin ajaksi alkuperäiseen muotoonsa.



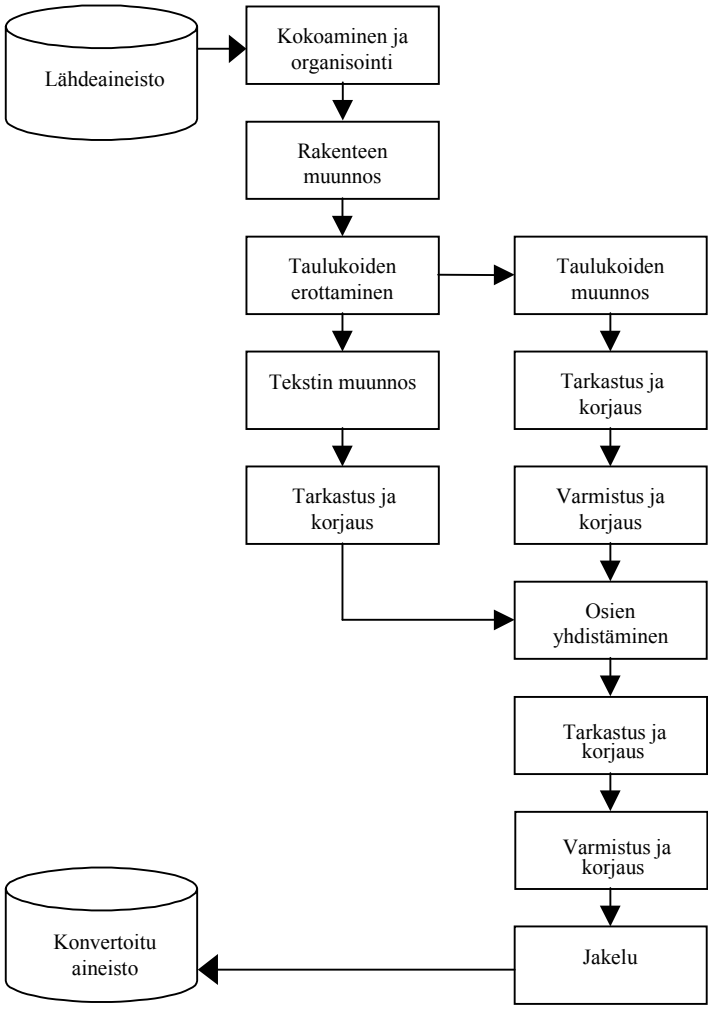
Kuva 4.1. Konvertointitapojen vaikutus konvertoinnin edistymiseen.

Kuvassa 4.1 on esimerkin avulla kuvattu edellä esitetyn kahden konvertointitavan vaikutusta konvertointiprojektin etenemiseen. Yhtenäinen viiva kuvaa kertakonvertoinnin ja katkoviiva pitkällä aikavälillä tapahtuvan konvertoinnin edistymistä suhteessa aikaan. Jälkimmäisessä tapauksessa kriittisen aineistoon kuuluvaksi on määritelty noin 30 % koko aineistosta.

4.3 Konversioprosessin seuranta

Massiivisten aineistojen konvertointi vaatii luvussa 3.1 esitettyjen perusprosessin vaiheiden jakamista pienempiin osiin. Tällöin koko konvertointiprosessin onnistuminen vaatii prosessin etenemisen seurantaa. Konvertointiprosessia johtavan projektipäällikön täytyy tietää, miten konvertointi etenee, missä vaiheessa ollaan menossa ja miten projekti pysyy aikataulussa. Myös konvertoijien on oltava selvillä siitä, mitä konvertoidaan ja miten työstettävä aineisto löydetään. Konversioprosessin seurannassa voidaan apuna käyttää esimerkiksi taulukoita, joissa prosessin alivaiheet ja vaiheiden edistyminen kuvataan.

Jos lähdeaineisto on monimutkainen konvertoitava, muunnosvaihe jaetaan useisiin pienempiin vaiheisiin, jolloin muunnettavaa lähdeaineistoa voidaan konvertoida pala kerrallaan, esimerkiksi taulukot erillään varsinaisesta lähdemateriaalista. Tällöin muunnettua aineistoa saatetaan joutua tarkastamaan jäsentäjällä useaan otteeseen. Lopuksi erillään konvertoidut osat yhdistetään, jolloin muunnetun dokumentin informaatio vastaa lähdemateriaalin informaatiota. Kuvassa 4.2 on esitetty tyypillinen muunnosprosessi (vrt. Travis & Waldt, 1995).



Kuva 4.2. Tyypillisen muunnosprosessin eteneminen.

Pitkä lähdeaineisto voidaan muunnosvaiheessa tarkoituksella jakaa pienempiin osiin, vaikka lukuihin, jolloin usean eri ihmisen on mahdollista työstää eri osia yhtäaikaa. Jokaisen osan muunnosvaihe voi vielä jakautua alivaiheisiin, jolloin osan muunnoksen seuranta on koko konvertointiprosessin etenemisen kannalta tärkeää.

Seuraavaksi tarkastelemme esimerkkien avulla lähdeaineiston jakamista osiin ja eri osien konvertoinnin seurantaa sekä projektipäällikön että konvertoijan näkökulmasta. Kuvassa 4.3 lähdeaineisto on jaettu kolmeen osaan (tässä tapauksessa kolmeen kirjan lukuun) ja kunkin osan muunnosvaihe kolmeen alivaiheeseen (Travis & Waldt, 1995). Myös tarkastusvaihe on jaettu kahteen alivaiheeseen, joista ensimmäinen suoritetaan kahden muunnosalivaiheen jälkeen ja jälkimmäinen viimeisen muunnosalivaiheen jälkeen. Lähdeaineiston eri luvut on esitetty taulukossa riveillä ja konvertointiprosessin vaiheet sarakkeissa. Taulukkoa käyttäen projektipäällikkö pystyy seuraamaan konvertointi-prosessin etenemistä kirjaamalla kunkin vaiheen päättymisen ajankohdan ja tekijän nimikirjaimet sitä mukaa, kun aineiston konvertointiprosessi etenee.

Osa	Muunnos 1	Muunnos 2	Tarkastus 1	Muunnos 3	Tarkastus 2	Varmistus	Jakelu
Luku 1	AA 15.3.99	AA 15.3.99	AA 15.3.99	AA 15.3.99	AA 15.3.99	CC 22.3.99	
Luku 2	AA 17.3.99	AA 17.3.99	AA 17.3.99	AA 19.3.99	AA 19.3.99	CC 23.3.99	
Luku 3	BB 15.3.99	BB 16.3.99	BB 16.3.99	AA 19.3.99	AA 19.3.99		

Kuva 4.3. Eri osien konvertointiprosessin eteneminen.

Konvertoijien täytyy olla selvillä, mistä ja miten kussakin vaiheessa käsiteltävä aineisto löydetään. Edellisen esimerkin tapaan kuvassa 4.4 muunnosvaihe on jaettu kolmeen alivaiheiseen ja tarkastusvaihe kahteen alivaiheeseen. Eri vaiheet esitetään taulukossa riveillä ja kussakin alivaiheessa tarvittavat ohjelmat ja syötetiedostot sekä alivaiheen tuloksena syntyvät tulostiedostot sarakkeissa.

Prosessin vaihe	Ohjelma	Syötetiedosto	Tulostiedosto	Virheloki	Raporttitiedosto
Muunnos 1	STRUCT	.raw	.out	.lg1	
	TEXT	.out	.txt	.lg2	.pct
Muunnos 2	PARTNO	.txt	.prt	.lg3	.pno
	XREFS	.prt	.xrf	.lg4	.xrp
Tarkastus 1		.xrf		.lg5	
Muunnos 3	MRFGN	.xrf	.sgm	.lg6	.fnr
Tarkastus 2		.sgm		.lg7	

Kuva 4.4. Eri vaiheessa tarvittavien tiedostojen tunnistaminen (Travis & Waldt, 1995).

Muunnosvaiheessa käytetään konvertointiin viittä muunnosohjelmaa, jotka konvertoivat aineistosta tietyn osan. Seuraavan alivaiheen syötetiedostona käytetään edellisen alivaiheen tulostiedostoa. Esimerkki on DOS-ympäristöstä: tiedoston nimi pysyy samana, mutta tiedostot tunnistetaan tarkentimen perusteella.

Kahdessa ensimmäisessä muunnosalivaiheessa käytetään yhdessä kahta eri ohjelmaa, jotka voidaan suorittaa peräkkäin esimerkiksi komentojonotiedoston avulla. STRUCT-ohjelma konvertoi raw-tarkenteisesta materiaalitiedostosta rakenne-elementit ja ohjaa tuloksen out-tarkenteiseen tulostiedostoon. Prosessissa mahdollisesti esiintyvät virheet kirjataan lg1-tarkenteiseen lokitiedostoon. TEXT-ohjelma saa syötteenään out-tarkenteisen tiedoston ja konvertoi siitä kappaleet ja muut tekstielementit ohjaten tuloksen txt-tarkenteiseen tiedostoon.

Edellisen muunnosalivaiheen txt-tarkenteista tulostiedostoa käytetään syötetiedostona seuraavassa muunnosalivaiheessa, jossa konvertoidaan osanumerot ja viittaukset. Tämän jälkeen dokumentti jäsennetään ja mahdolliset virheet korjataan esimerkiksi käsityönä. Viimeisessä muunnosalivaiheessa (muunnos 3) materiaalista konvertoidaan alaviitteet ja tulos ohjataan sgm-tarkenteiseen tulostiedostoon. Lopuksi tulostiedosto jäsennetään ja mahdolliset virheet korjataan, jotta konvertoitu materiaali olisi syntaktisesti oikeaa.

Eri vaiheissa syntyy tulostiedostojen lisäksi myös muita konvertoijan kannalta hyödyllisiä tiedostoja. Ohjelmien suorituksen aikana syntyvät virheet kirjataan erillisiin virhelokitiedostoihin virheloki-sarakkeen mukaisin tunnistein. Prosessin etenemisen aikana syntyy useita raportti-tiedostoja (tarkentimet pct, pno, xrp ja fnr), joiden avulla voidaan varmistua muunnoksen laadusta ennen jakelua.

5 TYÖVÄLINEIDEN VERTAILUA

Koska perinteisen dokumentin muuntaminen SGML-muotoon ei ole aivan yksinkertainen tehtävä, on konvertointiprosessin helpottamiseksi ja tehostamiseksi kehitetty erilaisia apuvälineitä. Näiden konversiovälineiden tehtävänä on muuntaa lähdedokumentit tiettyjen sääntöjen mukaan SGML-merkatuiksi tulosdokumenteiksi.

Tässä luvussa analysoimme lyhyen esimerkkidokumentaation, johdamme sen informaation esittämiseen sopivan dokumenttityyppimäärittelyn ja suoritamme kyseisen aineiston konvertoinnin SGML-muotoon muutamaa eri konversiovälinettä käyttäen. Lisäksi vertailemme valittujen työvälineiden käyttökelpoisuutta erityyppisissä konversio-tilanteissa.

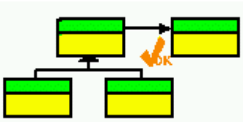
5.1 Esimerkkidokumentaation analysointi

Aiemmin, luvussa 3.2 tarkastelimme dokumenttianalyysia yleisellä tasolla. Dokumenttianalyysin tarkoituksena on määritellä tietyn dokumenttiluokan hierarkinen rakenne, erityisominaisuudet ja erityisvaatimukset (Korhonen & Kuisma, 1999). Dokumenttianalyysin pohjalta voidaan kehittää dokumentaation informaatiorakenteiden esittämiseen soveltuva dokumenttityyppimäärittely.

Seuraavaksi tutustumme dokumenttianalyysiin hieman syvällisemmin esimerkkitapauksen avulla. Esimerkin dokumentaatio koostuu Joensuun yliopiston tietojenkäsittelytieteen laitoksen kurssikohtaisista WWW-sivuista, joita on lyhennetty esimerkkiä varten poistamalla harjoitustehtäviin ja mahdollisiin linkkeihin liittyvät osiot.

Kuvassa 5.1 on esitetty lyhennetty ohjelmistotuotanto-kurssin kurssikuvaus. Lyhennetyt kohdat on merkitty dokumenttiin <listaa lyhennetty> -merkinnöin. Täydelliset, esimerkissä analysoitavat dokumentit löytyvät liitteestä 1.

University of Joensuu
Department of Computer Science



Software Engineering

Ohjelmistotuotanto (173302, 4 ov)

Kurssin sisältö:

- laatujärjestelmät ja ohjelmistotuotanto
- ohjelmistotyön vaiheet
- ohjelmistoprojektin hallinta

Tavoitteet:

- omaksutaan vastuullinen suhde ohjelmiston tuottamiseen
- saadaan tietoa ohjelmistotyön eri vaiheisiin käytettävissä olevista menetelmistä ja välineistä
- ymmärretään laatujärjestelmän merkitys ohjelmistotyölle

Kirjallisuus:

- MSQH - Modelling a Software Quality Handbook. STRI TS2, Icelandic Council for Standardization (STRI), 1991.
- W.S. Humphrey: Managing the Software Process. Addison-Wesley, 1989.

<listaa lyhennetty>

Sisällys:

- 1. Ohjelmistotyön kypsyystasot ja laatujärjestelmä
- 2. Laatujärjestelmä ja sen ylläpito
- 3. Tarkastusmenettelyt

<listaa lyhennetty>

Ohjelmistotuotanto syksyllä 1999

Opetusajankohdat:

- Luennot: 9.9.98 alkaen torstaisin ja perjantaisin klo 10-12 (M4) (Jorma Sajaniemi)
- Harjoitusryhmä: 20.9.99 alkaen maanantaisin klo 14-16, M13
- Välikokeet: tiistaina 2.11.99 ja 14.12.99, 8.00-10.00 M1

Kurssin suorittaminen välikokeilla edellyttää, että 1/3 harjoitustehtävistä on tehty.

Päivitetty viimeksi: 26.10.1999
saja@cs.joensuu.fi

Kuva 5.1. Eräs esimerkissä analysoitava dokumentti.

Esimerkissä analysoidaan neljä valittua dokumenttia (käyttöliittymät, ohjelmistotuotanto, systeeminsuunnittelu ja tietojärjestelmien dokumentointi) ja kehitetään analyysin perustella dokumenttien informaatiosisällön kuvaamiseen soveltuva dokumenttityyppimäärittely. Koska luennoitavia kursseja ja niihin liittyviä kurssikohtaisia sivuja on useita kymmeniä, on neljän dokumentin otos kattavaan kurssisivujen analysointiin liian suppea, mutta esimerkin kannalta riittävä.

Esimerkkidokumenttien ulkoasut noudattavat hyvin pitkälle samaa muotoa. Analysoitaessa dokumentteja ulkoasun perusteella voimme havaita dokumenttien koostuvan yläreunassa olevasta kuvasta tai kuvista, eri kirjasinkoolla kirjoitetuista otsikoista, listoista ja tekstikappaleista. Lisäksi eräissä dokumenteissa on tekstin tehostekeinona käytetty lihavointia tai kursivointia.

Ulkoasun analysoinnin perusteella voimme määritellä seuraavat dokumenttien merkkauksessa käytettävät tunnisteet:

`<kuva>`	Dokumentin alussa oleva kuva.
`<kk1>`	Vasemmalle tasattu dokumentin pääotsikko, kirjoitettu suurilla kirjaimilla (esim. kurssin nimi).
`<kk2>`	Vasemmalle tasattu pääotsikkoa pienempi otsikko, esiintyy ainoastaan käyttöliittymien -kurssikuvauksessa (esim. "opinto-oppaan mukainen kurssikuvaus" -otsikko).
`<kk3>`	Vasemmalle tasattu kolmas otsikkotaso (esim. "opetusajankohdat" -otsikko).
`<kappale>`	Vasemmalle tasattu kappale.
`<alkio>`	Listan alkio, informaation edessä pallon muotoinen luettelomerkki.
`<lihavointi>`	Lihavoitu teksti.
`<kursivointi>`	Kursivoitu teksti.

Edellä esitetyillä tunnisteilla dokumentille ei synny mitään sisällöllistä rakennetta, vaan niillä kuvataan ainoastaan dokumentin ulkoasu. Dokumenttien rakenteen ja informaatiosisällön selvittämiseksi lähestymme dokumentaatiota myös muista

näkökulmista. Analysoimme seuraavaksi esimerkkidokumentaation informaatiorakenteiden pohjalta.

Dokumentissa on kolmen tasoisia otsikoita. Otsikoille ei ole kiinnitetty järjestystä, joten päätason otsikon jälkeen voi esiintyä myös kolmannen tason otsikko. Otsikot koostuvat tekstistä ja numeroista.

Ensimmäisen tason otsikoita käytetään kurssin nimen, koodin ja laajuuden yhteydessä sekä opetuksen ajankohdan otsikkona. Toisen tason otsikkoa käytetään vain käyttöliittymät-kurssin kuvauksessa. Kolmannen tason otsikkoa käytetään opetusajankohdat-listan otsikkona.

Listat koostuvat pääsääntöisesti listaotsikosta ja yhdestä tai useammasta lista-alkiosta eli kohdasta. Listaotsikko ja listan kohdat koostuvat tekstistä sekä numeroista. Lista voi sisältää myös toisia listoja.

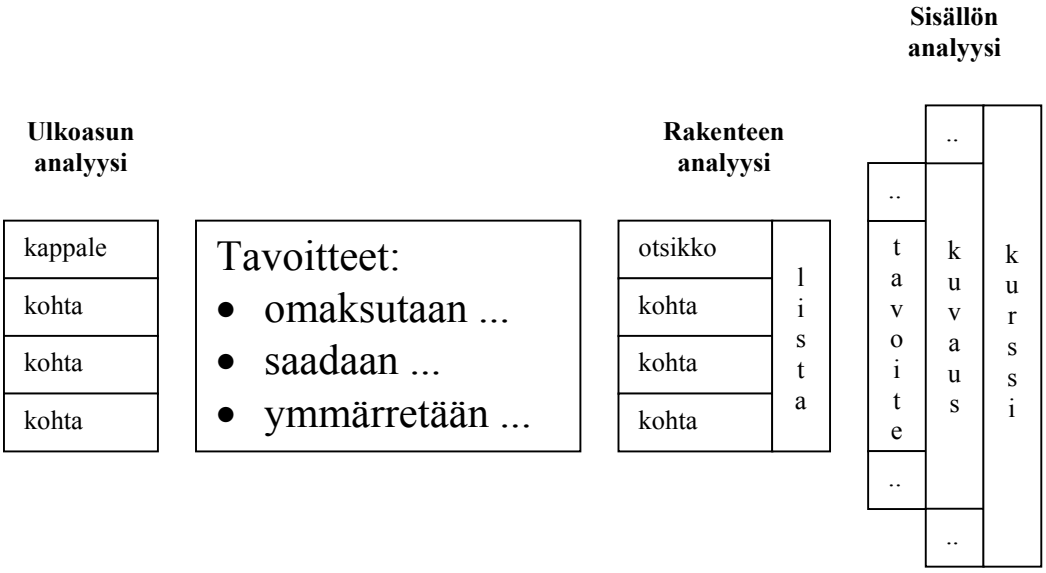
Näiden dokumentin rakenteeseen liittyvien havaintojen perusteella voimme määritellä seuraavat tunnisteet:

`<otsikko1>`	Dokumentin pääotsikot.
`<otsikko2>`	Toisen tason otsikko, esiintyy vain käyttöliittymät-kurssikuvauksessa.
`<otsikko3>`	Kolmannen tason otsikko.
`<lista>`	Tunniste aloittaa listan.
`<otsikko>`	Listan otsikko, esiintyy listassa täsmälleen yhden kerran.
`<kohta>`	Listan kohta, esiintyy listassa vähintään yhden kerran.
`</lista>`	Tunniste päättää listan.

Kuvassa 5.2 on esitetty eri analysointitapojen pohjalta määriteltyjen tunnisteiden suhteet toisiin tunnisteisiin ja esimerkkidokumenttiin.

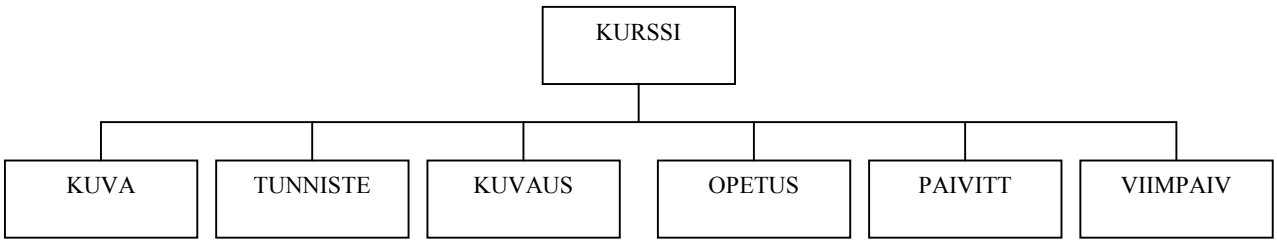
Analysoimme lopuksi dokumentaation sen sisällön mukaan. Jaamme aluksi dokumenttien sisällön neljään erilliseen kokonaisuuteen: kurssin tunnistetietoihin, kurssikuvaukseen,

opetukseen ja dokumentin päivittämiseen liittyviin tietoihin. Tällöin edellä esitellyt <otsikko1>-tunnisteet voidaan korvata kurssin tunnistetietoihin ja opetukseen liittyvillä tunnisteilla. Myös <otsikko3>-tunniste voidaan jättää pois ja määritellä ko. tason otsikot niiden sisällön mukaan opetus-elementin otsikoksi. Lisäksi käyttöliittymät-kurssin kuvauksessa esiintyvät <otsikko2>-tunnisteet korvataan listan <otsikko>-tunnisteilla.



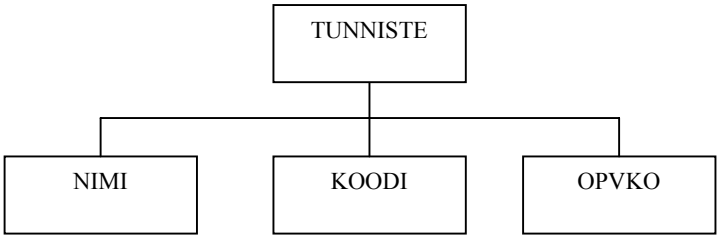
Kuva 5.2. Eri analysointi tapojen pohjalta määriteltyjen elementtien suhteet.

Dokumentin päivittämiseen liittyy päivittäjän nimi tai sähköpostiosoite (paivitt-elementti) ja viimeisen päivityksen päivämäärä (viimpaiv-elementti). Nyt voimme kuvata kurssisivun rakenteen puukaavion avulla kuvan 5.3 mukaisesti.



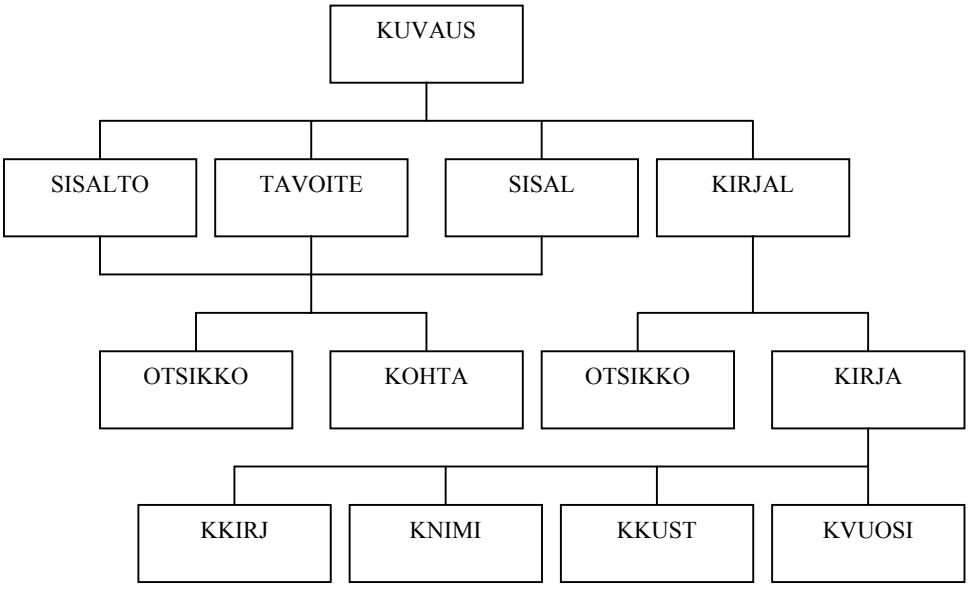
Kuva 5.3. Kurssi-elementin rakenne.

Kurssin tunnistetiedot koostuvat kurssin nimestä, koodista ja laajuudesta. Määrittelemme selvyyden vuoksi `tunniste`-elementin, joka sisältää kurssiin kuuluvat tunnistetiedot kuvan 5.4 mukaisesti.



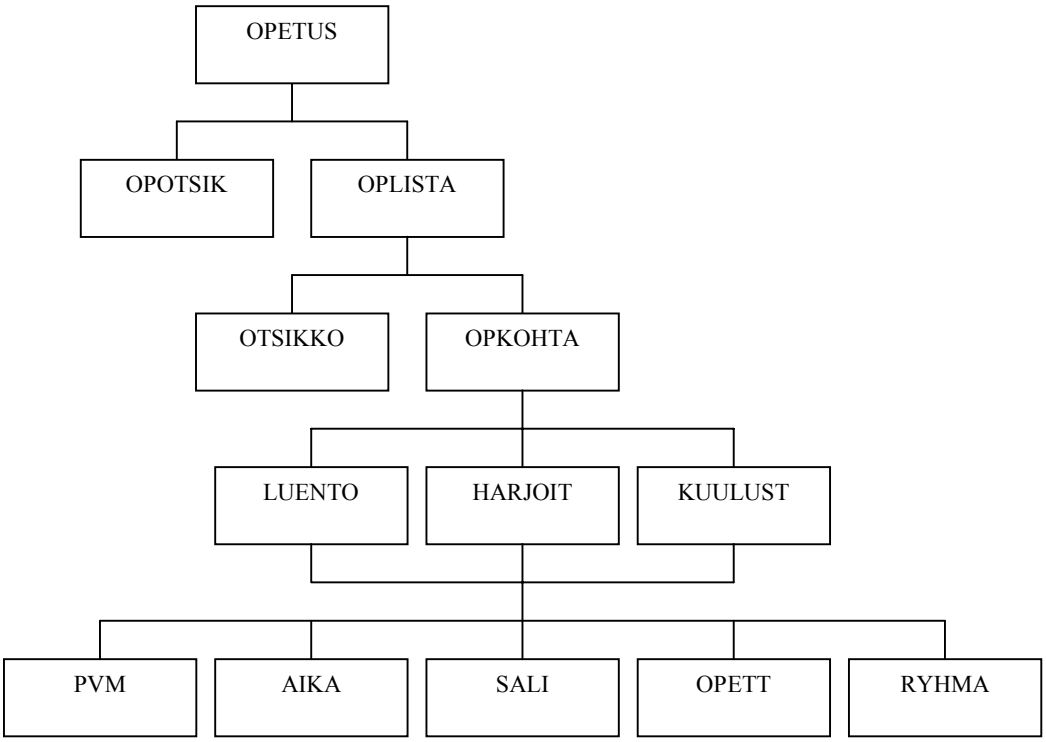
Kuva 5.4. `Tunniste`-elementin rakenne.

Kurssin sisältö, tavoitteet ja sisällys ovat aiemmin esitetyn muotoisia, otsikosta ja listakohdista koostuvia, listoja. Kirjallisuusluettelo on rakenteellisesti myös em. listan muotoinen, mutta kirjallisuuteen liittyvä informaatio on hyödyllistä jakaa tarkasti määriteltyihin osiin, kuten kirjan nimi (`knimi`-elementti), kirjoittaja tai kirjoittajat (`kkirj`-elementti), kustantaja (`kkust`-elementti) ja painovuosi (`kvuosi`-elementti). Kuvassa 5.5 on esitetty `kuvaus`-elementin rakenne (vrt. kuva 5.2).



Kuva 5.5. `Kuvaus`-elementin rakenne.

Opetus-elementti sisältää otsikon (opotsik-elementti) ja yhden tai useampia listoja (vrt. systeeminsuunnittelu ja tietojärjestelmien dokumentointi). Kukin lista koostuu listaotsikosta (otsikko-elementti) ja yhdestä tai useammasta kohdasta (opkohta-elementti). Opkohta-elementti koostuu luento-, harjoit- tai kuulust-elementeistä. Kukin näistä elementeistä voi sisältää pvm-, aika-, sali-, opett- tai ryhma-elementtejä tai merkkitietoa. Kuvassa 5.6 on esitetty opetus-elementin rakenne.



Kuva 5.6. Opetus -elementin rakenne.

Dokumenteissa esiintyvät erilaiset erikoistapaukset huomioidaan omina lisays-, huom- ja poikkeus-elementteinään. Lisays-elementtiä käytetään rakenteeseen sopimattomien kappaleiden merkkaamiseen. Tällaisia kappaleita ovat esimerkiksi kirjallisuuslistan jälkeen esiintyvät luentomoniste-ilmoitukset. Huom- ja poikkeus-elementtejä käytetään esimerkiksi lihavoinnin avulla korostettujen tekstien esittämiseen. Koska dokumentaatiossa esiintyvissä erikoistapauksissa listan sisältämät listat koostuvat poikkeus- ja huom-elementtien sisällöstä, voimme määritellä, että listat eivät sisällä toisia listoja.

Kuvassa 5.7 on edellä esitetyn analysoinnin perusteella kehitetty dokumenttityyppimäärittely.

```
<!-- Julkinen tunniste: PUBLIC "-//JoY//DTD kurssi//FIN"                    -->

<!-- Skandinaaviset merkit                                                  -->
<!ENTITY       Apist          SDATA "[Auml  ]"                               >
<!ENTITY       apist          SDATA "[auml  ]"                               >
<!ENTITY       Opist          SDATA "[Ouml  ]"                               >
<!ENTITY       opist          SDATA "[ouml  ]"                               >


<!-- DTD:ssä käytettävien entiteettien ("makrojen") määrittelyt             -->
<!ENTITY       % ku.tied                  "nimi & koodi? & opvko?"            >
<!ENTITY       % ku.kuv                   "sisalto? & tavoite? & kirjal?
                                           & sisal?"                           >
<!ENTITY       % ki.tied                  "kkirj* & knimi & kkust? & kvuosi?" >
<!ENTITY       % op.tied                  "#PCDATA | pvm | aika | sali |
                                           opettaja | ryhma"                   >

<!-- Dokumentin kokonaisrakenne                                             -->
<!ELEMENT      kurssi         - O         (kuva*, tunniste, kuvaus, opetus*,
                                           (paivitt? & viimpaiv?))
                                           +(lisays | huom | poikkeus)        >

<!-- Kuva (esim. logo) dokumentin alkuun                                    -->
<!ELEMENT      kuva           - O         EMPTY                              >
<!ATTLIST      kuva tied                  CDATA #REQUIRED                    >

<!-- Kurssin tunnistetiedot                                                 -->
<!ELEMENT      tunniste       - O         (%ku.tied;)                        >
<!ELEMENT      (nimi & koodi & opvko)
                              - O         (#PCDATA)                          >


<!-- Kurssikuvaukseen liittyvät elementit                                   -->
<!ELEMENT      kuvaus         - O         (%ku.kuv;)                         >
<!ELEMENT      (sisalto & tavoite & sisal)
                              - -         (otsikko, kohta+)                  >
<!ELEMENT      otsikko        - O         (#PCDATA)                          >
<!ELEMENT      kohta          - O         (#PCDATA)                          >

<!-- Kurssiin kuuluva kirjallisuus                                          -->
<!ELEMENT      kirjal         - -         (otsikko, kirja+)                  >
<!ELEMENT      kirja          - O         (%ki.tied;)                        >
<!ELEMENT      (kkirj & knimi & kkust & kvuosi)
                              - O         (#PCDATA)                          >

<!-- Opetukseen liittyvät elementit                                         -->
<!ELEMENT      opetus         - -         (opotsik, oplista*)                >
<!ELEMENT      opotsik        - O         (#PCDATA)                          >
<!ELEMENT      oplista        - O         (otsikko, opkohta+)                >
<!ELEMENT      opkohta        - O         (luento | harjoit | kuulust)       >
<!ELEMENT      (luento | harjoit | kuulust)
                              - O         (%op.tied;)*                       >
<!ELEMENT      (pvm |aika | sali | opett | ryhma)
                              - O         (#PCDATA)                          >

<!-- Ylläpitäjä ja viimeisen päivityksen ajankohta                          -->
<!ELEMENT      (paivitt & viimpaiv)
                              - O         (#PCDATA)                          >

<!-- Dokumentissa esiintyvät erikoistapaukset                               -->
<!ELEMENT      (lisays | huom | poikkeus)
                              - -         (#PCDATA)                          >
```

Kuva 5.7. Esimerkkidokumentaation analysoinnin perusteella johdettu kurssi-DTD.

Dokumenttityyppimäärittelyssä määritellään 35 elementtiä, yksi `kuva`-elementtiin liittyvä attribuutti, neljä dokumentin merkkauksessa käytettävää entiteettiä (`Apist`, `apist`, `Opist` ja `opist`) ja neljä DTD:ssä käytettävää entiteettiä (`ku.tied`, `ku.kuv`, `ki.tied` ja `op.tied`).

Dokumenttityyppimäärittelyssä käytettäviin entiteetteihin viitataan %-merkillä, joka täytyy esiintyä sekä määriteltäessä (välilyönnillä erotettuna entiteetin nimestä) että käytettäessä entiteettiä (välittömästi entiteetin nimen edessä) (Goldfarb, 1990).

Elementit `lisays`, `huom` ja `poikkeus` ovat määritelty *inkluusiolla* (inclusion), joten ne voivat esiintyä `kurssi`-elementin sisällä millä tasolla tahansa.

Tutustumme seuraavaksi muutamaan erityyppiseen konversiovälineeseen ja muunnamme esimerkin dokumentaation kehitetyn dokumenttityyppimäärittelyn mukaiseksi. Erityyppisiä konversiovälineitä ja konversio-tilanteita varten dokumentaatio on tallennettu eri tekstinkäsittelyohjelmien tarjoamissa tallennusmuodoissa sekä puhtaana ASCII-tekstinä.

5.2 OmniMark

OmniMark (OmniMark Technologies Corporation, 1999) on ohjelmointikieli, jonka ympärille on rakennettu laaja dokumenttien konvertointi-ympäristö. OmniMark soveltuu erityisesti ohjelmalliseen tekstin ja tekstitiedostojen muokkaukseen, koska se sisältää tehokkaat *hahmonsovitus*- (pattern-matching) ja *merkkauksen tunnistus*-toiminnot (Chahuneau, 1994). Näiden ominaisuuksien vuoksi OmniMarkia voidaan käyttää sekä tekstitiedostojen konvertoimiseksi SGML-muotoon (N-konversio) että SGML-tiedostojen konvertoimiseksi toisen dokumenttityyppimäärittelyn mukaiseksi (S-konversio).

OmniMark 5 -ohjelmointikielen ympärille on kehitetty OmniMark 5 C/VM, OmniMark Developer IDE 1.0 ja OmniMark Home and School IDE 1.0 -ohjelmistot. Ohjelmistot sisältävät tarkastavan OmniMark SGML Kernel -jäsentäjän.

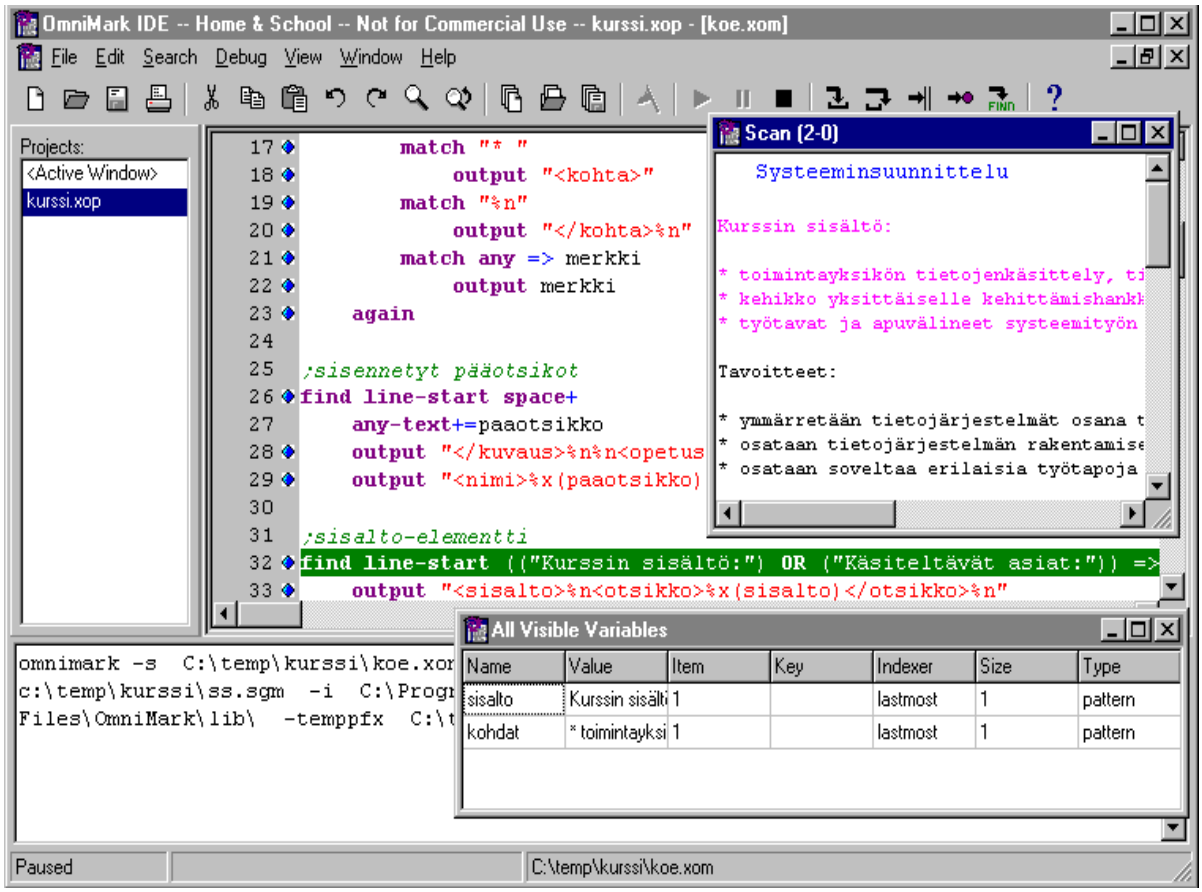
OmniMark 5 C/VM on komentorivipohjainen OmniMark-ohjelmointikielen kääntäjä ja virtuaalikone. Kääntäjä kääntää lähdekieliset ohjelmat *tavukoodiksi* (byte code), jota virtuaalikone suorittaa. OmniMark 5 C/VM tekee käännöksen lähdekoodista suorituksen aikana, eikä se voi tuottaa valmiiksi käännettyä ohjelmakoodia. Ohjelma on ilmainen, ja siitä on versiot Windows-ympäristön lisäksi myös AIX4.3-, HPUX32-, HPUX64-, Linux-, SGI-, Sinix- ja Solaris-ympäristöille.

OmniMark Developer IDE (integrated development environment) 1.0 on kaupallinen ohjelma, jossa OmniMark-ohjelmointikieli, jäsentäjä, kääntäjä ja virtuaalikone on integroitu graafiseen käyttöliittymään. OmniMark Home and School IDE 1.0 on Developer IDE-ohjelman ilmainen, koti- ja opetuskäyttöön tarkoitettu versio. Developer IDE ja Home and School IDE -ohjelmia voidaan käyttää Windows 95/98/NT -ympäristöissä. Kuvassa 5.8 on esitetty Home and School IDE -ohjelman käyttöliittymä.

Tarkoitamme jatkossa termillä OmniMark IDE yleisesti kumpaakin edellä mainittua versiota. IDE-ohjelmistot ja Unix-ympäristöön tarkoitettu OmniMark C pystyvät tuottamaan valmiiksi käännettyä tavukoodia, mutta sitä ei voida siirtää käännösympäristöstä toiseen.

OmniMark IDE-kehitysympäristö sisältää ohjelmoijan avuksi tehtyjä, muista sovelluskehittimistä tuttuja piirteitä kuten ohjelmakoodin virheiden tarkistusominaisuudet (debug). Ohjelmoija voi esimerkiksi tarkastella eri muuttujien arvoja ohjelman suorittamisen aikaan ja määritellä ohjelman suorituksen pysähtymään halutulla rivillä (breakpoint).

IDE-kehitysympäristössä ohjelmakoodin syntaksi esitetään eri väreillä, joka helpottaa ohjelmakoodin lukemista. Lisäksi ohjelma näyttää suoritettavan koodirivin ja prosessoitavan dokumentin käsittelykohdan omassa ikkunassaan konversion aikana. Ohjelman suorituksen aikana syntyvät ilmoitukset tulostuvat oletusarvoisesti alareunan loki-ikkunaan.



Kuva 5.8. OmniMark Home and School IDE 1.0:n käyttöliittymä.

OmniMark IDE:ssa lähde- ja tulosdokumentit sekä erityisvalinnat määritellään projektikohtaisesti. Tulosdokumentti tulostetaan oletusarvoisesti suorituksen yhteydessä loki-ikkunaan. Mikäli tulosdokumentti halutaan tulostaa tiedostoon, täytyy projektitietoihin määritellä tarvittavat parametrit.

Tarkastelemme seuraavaksi lyhyesti OmniMark-ohjelmointikieltä ja etenkin sen tekstitiedostojen konvertointiin soveltuvia piirteitä seuraavan esimerkin avulla.

```
find line-start ("Tavoitteet:") => tavoiteteksti "%n%n" (any-text+ "%n")+
=> kohdat
     output "<tavoite>%n<otsikko>%x(tavoiteteksti)</otsikko>%n"
     listakohdat kohdat
     output "</tavoite>%n%n"
```

Yllä oleva `find`-sääntö toteutuu silloin, kun prosessoitavasta dokumentista luetaan sanalla "Tavoitteet:" alkava rivi, jonka jälkeen seuraa kaksi rivinvaihtomerkkiä. Edelleen dokumentissa on seurattava vähintään yksi tekstirivi, jonka päättää rivinvaihtomerkki. Rivinvaihdot ilmaistaan OmniMarkissa `%n` -merkinnällä.

`Find`-sanan jälkeen seuraava `line-start` määrää, että haettavaa merkkijonoa verrataan käsiteltävän rivin alkuun. Mikäli merkkijonoa etsitään rivin lopusta, käytetään varattua sanaa `line-end`.

Merkkijonot ilmaistaan lainausmerkkien avulla. Etsittävän "Tavoitteet:"-merkkijonon ympärillä olevat sulkeet ilmoittavat, että etsittävä merkkijono sisältyy tulokseen. Etsinnän tulos ohjataan `=>` -merkinnällä seuraavan `tavoitetekst`i-hahmomuuttujan arvoksi. *Hahmomuuttujia* (pattern variable) ei tavallisista muuttujista poiketen tarvitse esitellä ennen arvon sijoittamista.

OmniMark sisältää useita hahmonsovituksen avuksi luotuja *merkkiluokkia* (character class) ja *esiintymäkerta-operaattoreita* (occurrence operator), joiden avulla pystytään täsmällisesti määrittelemään millaisia tekstiesiintymiä haetaan. Merkkiluokka `any-text` vastaa mitä tahansa yksittäistä merkkiä lukuunottamatta rivinvaihto-merkkiä. Esiintymiskerta-operaattorit ovat samat kuin SGML-standardissa (katso luku 2.1). Täten `(any-text+ "%n")` tarkoittaa mitä tahansa merkkijonoa, joka koostuu vähintään yhdestä merkistä jota seuraa rivinvaihtomerkki. Esimerkissä sulkeiden jälkeen seuraava operaattori + ilmoittaa, että haun tulokseen kuuluu yksi tai useampi tällainen rivi. Lopuksi tulos ohjataan hahmomuuttujaan `kohdat`.

Mikäli prosessoitavasta dokumentista löytyy haun ehdot täyttävä kohta, suoritetaan `find`-lausetta seuraavat komennot. `Output`-komennolla kirjoitetaan merkkijono tulosdokumenttiin. `Find`-lauseessa määriteltyihin hahmomuuttujiin viitataan `%x` -merkinnällä. Viitattavan hahmomuuttujan nimi on sijoitettava sulkeiden sisään välittömästi `%x` -merkinnän jälkeen.

Seuraavaksi esimerkissä kutsutaan alla esitettyä `listakohdat`-funktiota.

```
define function listakohdat value stream lkohta as
     repeat scan lkohta
          match "* "
               output "<kohta>"
          match "%n"
               output "</kohta>%n"
          match any => merkki
               output merkki
     again
```

Funktio `listakohdat` saa syötteenään `stream`-tyyppisen parametrin `lkohta`, joka sisältää funktion käsittelemän informaation. Funktio ei palauta mitään arvoa.

OmniMark sisältää myös ehto- ja toistolauserakenteet. Funktiossa esiintyvä `repeat scan - again` -rakenne läpikäy `lkohta`-parametrissa välitetyn merkkijonon merkki kerrallaan. Silmukkaa toistetaan kunnes kaikki merkit ovat käsiteltyjä.

Mikäli läpikäynnin aikana kohdataan *-merkki jota seuraa välilyönti, kirjoitetaan tulosdokumenttiin `<kohta>`-tunniste. Vastaavasti rivinvaihtomerkin esiintyessä tulosdokumenttiin kirjoitetaan `</kohta>`-lopputunniste. Kaikissa muissa tapauksissa luettu merkki sijoitetaan `merkki`-hahmomuuttujan arvoksi ja kirjoitetaan se tulosdokumenttiin. Merkkiluokka `any` vastaa mitä tahansa merkkiä.

Kuvassa 5.9 on osa puhtaana ASCII-tekstinä tallennettua systeeminsuunnittelu-kurssikuvausta. Edellä tarkastelemamme `find`-lauseen mukaisesti "Tavoitteet:"-merkkijonon jälkeen seuraa kaksi rivinvaihtoa ennen *-merkillä alkavia listakohtia. Listakohdat käsitellään erillisessä `listakohdat`-funktiossa, jossa *-merkit korvataan alkutunnisteella ja rivinvaihdot lopputunnisteella.

```
Tavoitteet:


* ymmärretään tietojärjestelmät osana toimintayksikön toimintaa
* osataan tietojärjestelmän rakentamisen eri vaiheet
* osataan soveltaa erilaisia työtapoja ja välineitä
tietojärjestelmän rakentamisessa
```

Kuva 5.9. ASCII-tekstinä tallennettu systeeminsuunnittelu-kurssikuvauksen osa.

.

Kuvassa 5.10 on esimerkkikoodin tuottama osa SGML-merkatusta tulosdokumentista. Listan otsikkona oleva "Tavoitteet:"-merkkijono merkataan `tavoite`-elementtiin sisältyvällä `<otsikko>`-tunnisteella ja listaan kuuluvat kohdat `<kohta>`-tunnisteilla.

```
<tavoite>
<otsikko>Tavoitteet:</otsikko>
<kohta>ymmärretään tietojärjestelmät osana toimintayksikön
toimintaa</kohta>
<kohta>osataan tietojärjestelmän rakentamisen eri vaiheet</kohta>
<kohta>osataan soveltaa erilaisia työtapoja ja välineitä
tietojärjestelmän rakentamisessa</kohta>
</tavoite>
```

Kuva 5.10. Esimerkkikoodin tuottama SGML-merkattu tulosdokumentti.

Liitteessä 2 on esimerkkidokumentaation konvertoinnissa käytetyn OmniMark-ohjelman koko listaus. Ohjelma muuntaa dokumentit edellisessä luvussa kehitetyn dokumenttityyppimäärittelyn mukaiseksi. Ohjelman tuottama SGML-muotoon konvertoitu systeeminsuunnittelu-kurssin kuvaus on liitteessä 3.

Ohjelmakoodin ensimmäisellä rivillä oleva `up-translate` määrittelee, että kyseisen ohjelman tarkoituksena on konvertoida ei-SGML-dokumentteja SMGL-muotoon. Ohjelman alussa esiintyvän `find-start` -lauseen jälkeen seuraavat komennot

suoritetaan ennen dokumentin prosessoinnin aloittamista ja vastaavasti `find-end` -lauseen jälkeiset komennot prosessoinnin loputtua.

Mikäli koko dokumenttityyppimäärittely halutaan sijoittaa SGML-dokumentin prologiin (julkisen tunnisteen sijaan), näyttäisi `find-start` -lause seuraavalta:

```
find-start
     output "<!DOCTYPE kurssi [%n"
     output file "kurssi.dtd"
     output "%n]>%n"
```

Liitteessä esitetty konversio-ohjelma ei ole kuitenkaan täydellinen. Ohjelma ei huomioi kaikkia esimerkkidokumentaatiossa olevia piirteitä, koska kaikkien tapausten sisällyttäminen ohjelmaan kasvattaa sen kokoa kohtuuttomasti. Ohjelma merkkaa esimerkiksi kirjallisuusluettelossa esiintyvät kirjat ainoastaan `<kirja>`-tunnisteella, eikä erittele kirjan nimeä, kirjoittajaa (tai kirjoittajia), kustantajaa tai painovuotta omiksi elementeikseen. Ohjelmassa ei ole myöskään huomioitu skandinaavisten kirjaimien korvaamista entiteettiviittauksella.

5.3 Rainbow Maker

Rainbow Maker (Electronic Book Technologies, 1993) on erityinen N-konversio-ohjelma, joka pohjautuu sateenkaari-strategiaan (katso luku 3.5.2). Ohjelma on ns. freeware-lisenssin alainen, joten sen käyttäminen on ilmaista, eikä ohjelma tarvitse erillistä lisenssiä.

Rainbow Maker on komentorivipohjainen ohjelma, joka muuntaa ohjelman tukemassa tallennusmuodossa olevan dokumentin Rainbow DTD:n mukaiseksi. Ohjelma ei sisällä mitään käyttäjän ohjelmoitavissa olevia piirteitä, joten sen avulla dokumentteja ei voida konvertoida lopulliseen muotoonsa. Rainbow Makeriin ei sisälly myöskään SGML-jäsentäjää.

Rainbow DTD:n mukaan merkatussa dokumentissa eri tekstinkäsittelynohjelmien tyylit esitetään SGML-tunnisteiden avulla (Deseyne, 1995). Dokumenttien konvertoiminen

Rainbow DTD:n esitysmuodosta halutun dokumenttityyppimäärittelyn mukaiseksi voidaan tämän jälkeen tehdä esimerkiksi S-konversio-ohjelmalla. Rainbow DTD (versio 2.5) on esitetty liitteessä 4.

Rainbow Maker tukee FrameMakerin versioita 3, 4 ja 5 sekä Interleafin versioita 5 ja 6 (Ensign, Goldfarb & Pepper, 1998). Lisäksi se pystyy konvertoimaan Windowsin Word 6.0/95 ja Macintoshin Word 5.x -versioilla tuotettuja RTF-tiedostoja sekä WordPerfect 5.x:n tiedostoja. Ohjelmasta on olemassa versiot DOS- ja Solaris-ympäristöille.

Rainbow Makerin komentorivin syntaksi on

```
RBMAKER     -IN <lähdetiedosto> -OUT <tulostiedosto>
            -FIGDIR <kuvahakemisto> -DATADIR <datahakemisto>
            -TEMPDIR <väliaikainen hakemisto>
```

jossa `<lähdetiedosto>` on konvertoitavan tiedoston nimi hakemistopolkuineen ja `<tulostiedosto>` ohjelman tuottaman tulostiedoston nimi hakemistopolkuineen. `<kuvahakemisto>` ja `<väliaikainen hakemisto>` ovat dokumentin konvertoinnissa käytettäviä apuhakemistoja ja `<datahakemisto>` ohjelman tarvitsemien tiedostojen hakemisto (normaalisti hakemisto .\rbmaker.dat\).

Sateenkaari-strategia on erityisen käyttökelpoinen silloin, kun konvertoitava aineisto on tuotettu usealla eri tekstinkäsittelyohjelmalla. Muunnamme seuraavaksi käyttöliittymät- ja tietojärjestelmien dokumentointi -kurssien kuvaukset Rainbow Makerilla Rainbow DTD:n mukaiseksi. Käyttöliittymät-dokumentti on tallennettu MS Word 6.0:lla RTF-muotoon ja tietojärjestelmien dokumentointi -dokumentti FrameMaker 5:lla MIF-muotoon. Lähdedokumentit kl.rtf ja tjd.mif sijaitsevat kiintolevyn kurssi-hakemistossa, jonne myös tulosdokumentit kl.rbw ja tjd.rbw tallennetaan.

Suoritamme konversiot komennoilla:

```
C:\rainbow>rbmaker -in \kurssi\kl.rtf -out \kurssi\kl.rbw
-figdir .\figdir -datadir .\rbmaker.dat -tempdir c:\temp
```

```
C:\rainbow>rbmaker -in \kurssi\tjd.mif -out \kurssi\tjd.rbw
-figdir .\figdir -datadir .\rbmaker.dat -tempdir c:\temp
```

Kuvassa 5.11 on ensimmäiset rivit konversion tuloksena syntyneestä kl.rbw -tiedostosta. Konvertoidut tiedostot löytyvät kokonaisuudessaan liitteestä 5.

Tiedoston alussa määritellään, mihin dokumenttityyppimäärittelyyn dokumentin rakenne perustuu. Lisäksi dokumentin prologissa esitellään entiteetti `fgraf0000.wmf` dokumentissa olevalle kuvalle. Elementin `styinfo` sisältönä on dokumentissa esiintyvien tyylien kuvaukset (Sklar, 1995).

```
<!DOCTYPE rainbow PUBLIC "-//EBT//DTD Rainbow 2.5//EN" [
<!ENTITY fgraf0000.wmf SYSTEM "graf0000.wmf" NDATA wmf >
]>
<RAINBOW><FILEINFO ORIGIN="WinWord-RTF1" DTDVER="2.5"
><STYINFO><PARATYPE FONT-FAMILY="Times New Roman" FONT-SIZE="10" LINE-
SPACING="12" CHARSET="ISO-8859" JUSTIFICATION="LeftJust" FIRST-INDENT="0"
LEFT-INDENT="0" RIGHT-INDENT="0" SPACE-BEFORE="0" SPACE-AFTER="0" FONT-
WEIGHT="Medium" FONT-SLANT="Roman" NAME="Normal"
><PARATYPE FONT-FAMILY="Arial" FONT-SIZE="14" LINE-SPACING="16"
CHARSET="ISO-8859" JUSTIFICATION="LeftJust" FIRST-INDENT="0" LEFT-
INDENT="0" RIGHT-INDENT="0" SPACE-BEFORE="12" SPACE-AFTER="3" KEEP-WITH-
NEXT=1 FONT-WEIGHT="Bold" FONT-SLANT="Roman" NAME="heading 1"
><PARATYPE FONT-FAMILY="Arial" FONT-SIZE="12" LINE-SPACING="14"
CHARSET="ISO-8859" JUSTIFICATION="LeftJust" FIRST-INDENT="0" LEFT-
INDENT="0" RIGHT-INDENT="0" SPACE-BEFORE="12" SPACE-AFTER="3" KEEP-WITH-

...
```

Kuva 5.11. Rainbow-DTD:n mukainen käyttöliittymät-kurssisivun alku.

Varsinainen dokumentin informaatiosisältö on esitetty `<doc>`- ja `</doc>`-tunnisteiden välissä. Kuvassa 5.12 on osa konvertoidusta tjd.rbw -tiedostosta.

Kappaleen alku merkataan `<para>`-tunnisteella, jonka attribuutissa `paratype` määritellään, mitä tyyliä kappale noudatti alkuperäisessä dokumentissa. Kappale päättyy `</para>`-lopputunnisteeseen. Kappaleen sisältämä informaatio on `paracont`-elementin alku- ja lopputunnisteiden välissä. Koska alkuperäistä dokumenttia kirjoitettaessa on

käytetty tyhjiä kappaleita eli kappaleita jotka sisältävät vain rivinvaihtomerkin, esiintyy tiedostossa tyhjiä `paracont`-elementtejä.

Rainbow Maker lisää `<autogen>`-tunnisteen tulostiedostoon generoidessaan tekstinkäsittelyohjelmien erikoismerkkien tilalle omat tunnisteensa (Sklar, 1995). Lista-alkioiden edessä oleva luettelomerkki korvataan `&bull;` -entiteettiviittauksella ja sisennys `tab`-elementillä.

```
<para paratype="Heading1"><paracont>
Tietoj&auml;rjestelmien dokumentoiminen (2 ov) 173207
</paracont></para>
<para paratype="Body"><paracont>
</paracont></para>
<para paratype="Body"><paracont>
Kurssi on pakollinen kurssi ohjelmistotuotannon linjalla, muilla
linjoilla kurssi k&auml;y valinnaiseksi kurssiksi cum laude
opintoihin. </paracont></para>
<para paratype="Body"><paracont>
</paracont></para>
<para paratype="Body"><paracont>
Kurssin sis&auml;lt&ouml;: </paracont></para>
<para paratype="Body"><paracont>
</paracont></para>
<para paratype="Bulleted"><paracont>
<autogen>&bull;<tab></autogen>Dokumenttien rakenteeseen
vaikuttavat tekij&auml;t </paracont></para>

...
```

Kuva 5.12. Osa Rainbow-DTD:n mukaista tietojärjestelmien dokumentointi -kurssin sivua.

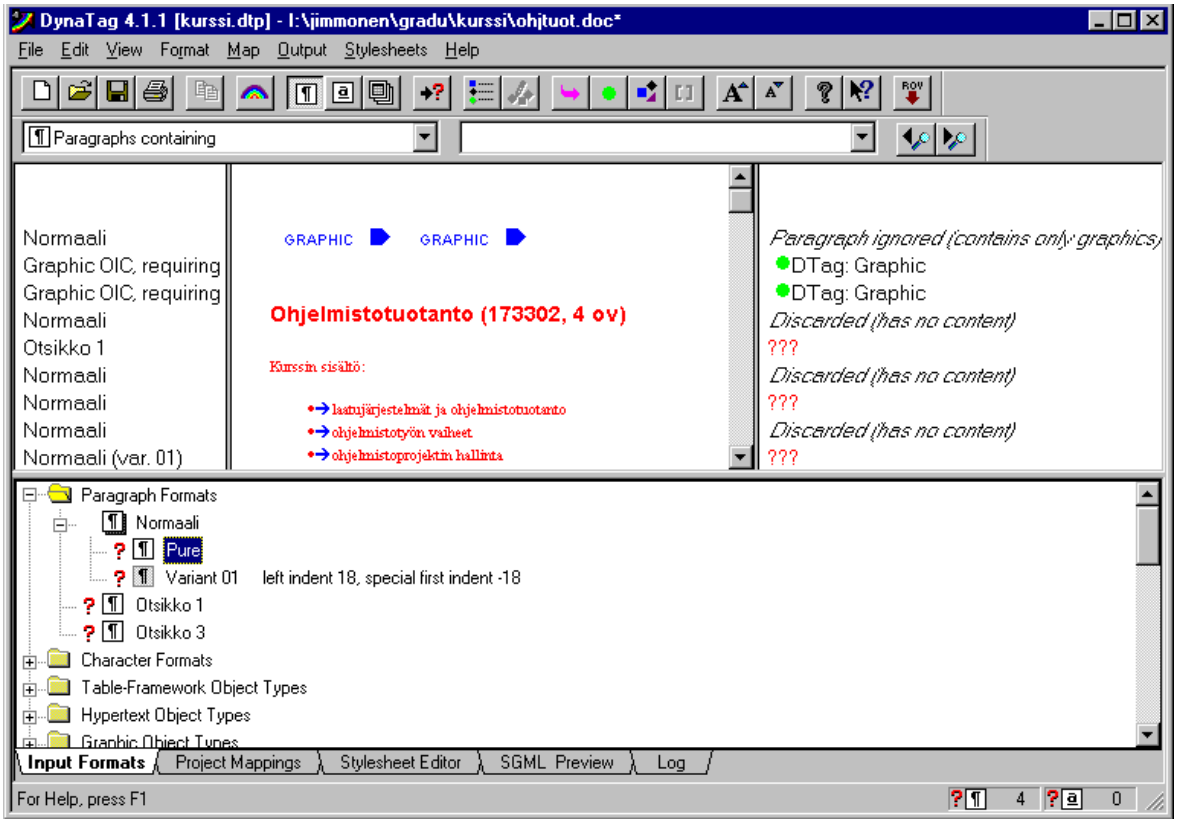
Rainbow Maker ei sisällä mitään työvälineitä sen tuottaman Rainbow DTD:n muotoisen dokumentaation jatkokonvertointiin. Täten ohjelman tuottama väliaikainen aineisto täytyy konvertoida vielä halutun dokumenttityyppimäärittelyn mukaiseksi käyttäen jotain S-konversio-ohjelmaa.

5.4 DynaTag

DynaTag (Inso Corporation, 1999) on graafinen konversioväline, joka hyödyntää sateenkaari-strategiaa konversion aikana. DynaTag konvertoi eri tekstinkäsittelyohjelmilla tuotetut dokumentit Rainbow DTD:n mukaiseksi, jonka jälkeen käyttäjä voi määritellä haluamansa tunnisteet dokumentissa esiintyville tyyleille. Ohjelma sisältää oman SGML-jäsentimen.

DynaTag on kaupallinen ohjelma, joka vaatii online-dokumentaation näyttämiseksi DynaText-ohjelmiston. DynaTag on saatavissa Windows-ympäristön lisäksi myös Solaris-, RS/6000 AIX- ja HP/UX -ympäristöille (Ensign, Goldfarb & Pepper, 1998).

DynaTagin graafinen käyttöliittymä sisältää useita käyttäjälle hyödyllisiä näkymiä. Sovellusikkunassa voi olla samanaikaisesti näkymät esimerkiksi prosessoitavaan dokumenttiin, siinä käytettyihin tyyleihin ja niiden ominaisuuksiin, sekä tulosdokumentin rakenteeseen. Kuvassa 5.13 on DynaTag 4.1.1 -ohjelman käyttöliittymä.



Kuva 5.12. DynaTag 4.1.1:n käyttöliittymä.

DynaTagissa konversioon liittyvät ominaisuudet määritellään projektikohtaisesti. Yksi projekti voi sisältää useita, mahdollisesti eri tekstinkäsittelyohjelmien tuottamia, dokumentteja joihin sovelletaan käyttäjän määrittelemiä muunnossääntöjä. Ohjelma konvertoi dokumentteihin sisältyvät kuvat automaattisesti JPEG-, TIFF- tai CGM-muotoon.

DynaTag pystyy käsittelemään MS Wordilla, Frame Makerilla ja Interleafilla tuotettuja dokumentteja. Se muuntaa prosessoitavan dokumentin tai kaikki projektin dokumentit Rainbow DTD:n mukaiseksi ennen prosessoitavan dokumentin näyttämistä.

Tämän jälkeen käyttäjä voi muodostaa lähdedokumentissa esiintyville tyyleille haluamansa tunnisteet ns. mapping-toimintojen avulla. DynaTag sisältää useita erilaisia mapping-toimintoja, joiden avulla käyttäjä voi määritellä esimerkiksi kullekin tyylille oman tunnisteen (base mapping) tai tyyliin liittyvät ehdot, joiden perusteella tunniste merkataan dokumenttiin (conditional mapping). Lisäksi ohjelma sisältää mapping-toiminnot listojen, taulukoiden ja erikoismerkkien käsittelyyn.

DynaTagin avulla voidaan luoda SGML-dokumenttien lisäksi DynaText-kirjoja (DynaText Book), jolloin kullekin tulosdokumentin tunnisteelle täytyy määritellä vielä haluttu ulkoasu. DynaText-kirjat ovat sidottuja DynaText-ohjelmaan, eikä niitä voida hyödyntää muilla työvälineillä.

Emme tutustu DynaTagiin tämän laajemmin, koska se perustuu jo edellä esitetyn Rainbow Makerin kanssa Rainbow DTD:n hyödyntämiseen.

5.5 Konversiovälineiden vertailua

Seuraavaksi arvioimme edellä käsiteltyjen konversiovälineiden käyttökelpoisuutta erilaisissa konvertointitilanteihin liittyen. Jokainen konvertointitilanne on pääsääntöisesti erilainen ja vaatii kyseiseen tilanteeseen soveltuvat välineet. Millainen väline kuhunkin

tilanteeseen valitaan riippuu pitkälti siitä missä muodossa ja miten laaja lähdeaineisto on. Tässä esitettävä konversiovälineiden vertailu perustuu kirjoittajan subjektiivisiin havaintoihin ja mielipiteisiin.

Käsittelemämme kolme konversiovälinettä ovat yleisessä käytössä ja soveltuvat hyvin erityyppisiin konvertointitilanteisiin. Ne poikkeavat toisistaan huomattavasti sekä ominaisuuksien että käyttötarkoituksen osalta. Näiden perustelujen lisäksi konversiovälineiden valintaan on vaikuttanut myös ohjelmien saatavuus: Rainbow Maker on täysin ilmainen, OmniMark on ilmainen koti- ja opetuskäytössä ja DynaTagista on saatavilla 30 päivän kokeiluversio.

OmniMarkia voidaan välineenä verrata perinteisten ohjelmointikielten kehitys-ympäristöihin. Se sisältää tehokkaan ohjelmointikielen ja kattavan kehitysympäristön. OmniMark sisältää tarkastavan SGML-jäsentimen, joka helpottaa konversio-ohjelmien kehittämistä.

OmniMark-ohjelmointikieli on selkeä ja helposti opittava. Se sisältää tehokkaat tekstinmuokkaus-ominaisuudet, joiden avulla lähdedokumentissa esiintyvä rakenne ja informaatio voidaan havaita. Lisäksi OmniMark on ohjelmointikielenä tarpeeksi laaja monimutkaisten konversio-ohjelmien laatimiseen.

OmniMark soveltuu tehokkaan ohjelmointikielensä ansiosta lähes kaikkiin konvertointi-tilanteisiin. Se on käsitellyistä välineistä ainoa, joka pystyy prosessoimaan puhtaasta ASCII-tekstistä koostuvaa aineistoa tehokkaasti. OmniMarkia voidaan hyödyntää niin jälkivarmistus- kuin esikonvertointi-strategioita käytettäessä.

Graafisen OmniMark IDE -kehitysympäristön heikkoutena voidaan pitää sen hitautta. Laajoja aineistoja konvertoitaessa OmniMark-ohjelmat on järkevä ajaa komento-rivipohjaisella OmniMark C/VM -sovelluksella IDE:n sijasta.

Rainbow Maker soveltuu erinomaisesti konvertointitilanteiseen, jossa lähdeaineisto on tuotettu usealla tekstinkäsittelyvälineellä. Tällaiset konvertointitilanteet ovat yleisiä esimerkiksi suurissa, useita osastoja käsittävissä organisaatioissa. Sateenkaari-strategian

mukaisesti lähdeaineisto muunnetaan ohjelmalla Rainbow DTD:n mukaiseksi, jonka jälkeen ei tarvita kuin yksi konversio-ohjelma muunnetun aineiston jatkokonvertointiin.

On selvää, että Rainbow Makerin käyttäminen dokumenttien esitysmuodon yhdenmukaistamiseen on halvempaa ja nopeampaa kuin erillisen konversio-ohjelman laatiminen jokaiselle esitysmuodolle. Rainbow Maker on nopea komentorivipohjainen konversioväline, joten sen avulla voidaan konvertoida sujuvasti useita dokumentteja komentojonotiedostoa käyttäen.

Rainbow Makerilla konversioprosessia ei kuitenkaan voida suorittaa loppuun asti, koska se ei sisällä Rainbow DTD:n mukaisen aineiston jatkokonvertointiin tarvittavia ominaisuuksia. Täten Rainbow Makerin tuoma hyöty on vain eri tallennusmuodoissa tallennetun lähdeaineiston saattamisessa yhdenmuotoiseksi. Rainbow Makerilla ei voida vaikuttaa lähdeaineiston rakenteeseen. Se muuntaa samalla tavoin rakenteellisesti sekä hyvät että huonot dokumentit, jolloin mahdolliset lähdeaineistoon liittyvät ongelmat täytyy huomioida jatkokonvertointia tehdessä.

Sateenkaari-dokumentti sisältää yleensä paljon SGML-tunnisteita informaatiosisältöön verrattuna, joten sen jatkokonvertointi manuaalisesti on hyvin työlästä. Mikäli Rainbow Makeria käytetään konvertointiprosessissa, on varauduttava siihen, että jatkokonvertointia varten on käytössä jokin muu konversioväline. Luonnollisesti Rainbow Maker ei sovellu jälkivarmistus- tai esikonvertointi-strategoihin.

DynaTag on Rainbow Makeria laajempi konversioväline, joka hyödyntää sateenkaari-strategiaa dokumenttien konvertoinnissa. DynaTag sisältää Rainbow DTD:n mukaisten dokumenttien jatkokonvertointiin tarvittavat ominaisuudet, joten sen avulla voidaan tuottaa halutun muotoisia SGML-dokumentteja.

DynaTagissa tulosdokumenttien tuottamiseen ei OmniMarkista poiketen tarvita ohjelmointitaitoa. DynaTagissa SGML-tunnisteet liitetään lähdedokumentissa esiintyviin tyyleihin. Jokaista tyyliesiintymää ei välttämättä automaattisesti korvata määritellyllä SGML-tunnisteella, vaan tunnisteiden esiintymiselle voidaan asettaa myös ehtoja. Tämä

onkin perusvaatimus automaattisesti tapahtuvalle dokumenttien sisällön mukaiselle merkkaamiselle.

DynaTagin heikkoutena voidaan pitää kuitenkin tyylisidonnaisuutta (sama väite pätee myös Rainbow Makeriin). Dokumenttien konvertointi DynaTagilla on hankalaa, mikäli dokumentti ei sisällä mitään tyylimuotoilua. DynaTag soveltuukin sellaisen eri tekstinkäsittelyvälineillä luodun aineiston konvertointiin, jonka tuottamisessa on käytetty yhdenmukaisesti tekstinkäsittelyohjelmien tyyli-ominaisuuksia. Kuitenkin jos konvertoitava aineisto on hyvin laaja, on mahdollisesti nopeampaa ja tehokkaampaa käyttää muunnokseen Rainbow Makeria ja jotain S-konversio-ohjelmaa kuin DynaTagia.

Vaikka esimerkkidokumentaatio on lyhyt ja rakenteellisesti yksinkertainen, sen ohjelmallinen konvertointi kurssi-DTD:n mukaiseksi ei ole aivan helppoa. Dokumentaatio sisältää muutamia erikoistapauksia ja poikkeuksia, joiden käsittely ohjelmallisesti on hankalaa. Käsitellyistä konversiovälineistä esimerkkidokumentaation konvertointiin sopivin on OmniMark. OmniMarkin tehokkaasta ohjelmointikielestä huolimatta eräät dokumentaation osat on helpompi ja nopeampi konvertoida manuaalisesti kuin luoda konversio-ohjelmaan säännöt kaikille eri mahdollisuuksille.

6 YHTEENVETO

Usein elektronisten dokumenttien käyttöikä on sidottu tekstinkäsittelyohjelmiin, joilla ne on tuotettu. Kuitenkin tekniikan ja ohjelmistojen kehittyessä myös vanha dokumentaatio on siirrettävä uuteen ympäristöön, jolloin ongelmaksi muodostuu uusien ohjelmien ja vanhan dokumentaation yhteensopivuusongelmat. Lisäksi suurissa organisaatioissa dokumentaation tuottamiseen käytetään useita eri tekstinkäsittelyvälineitä, joiden esitysmuodot eivät ole yhteensopivia toistensa kanssa.

SGML on kansainvälinen dokumenttirakenteiden määrittelyyn tarkoitettu standardi, joka erottaa dokumentin loogisen rakenteen sen typografisesta esitystavasta. Koska SGML-dokumentit ovat täysin laitteisto- ja ohjelmistoriippumattomia, niitä on mahdollista käsitellä useilla eri välineillä ja eri ympäristöissä.

Vanhan dokumentaation konvertointi SGML-muotoon ei kuitenkaan ole aivan yksinkertainen tehtävä. Koska SGML:n tarkoituksena on antaa dokumenteille yhteisen muodon lisäksi myös tietty hierarkinen rakenne, täytyy vanha dokumentaatio muuntaa halutun dokumenttityyppimäärittelyn mukaiseksi.

Konvertointi voidaan suorittaa manuaalisesti tai ohjelmallisesti. Yleensä on järkevää pyrkiä automatisoimaan konvertointiprosessia mahdollisimman paljon, koska tietokone on huomattavasti ihmistä nopeampi ja halvempi työntekijä. Kuitenkin lähdeaineistot sisältävät usein osia, joiden automaattinen konvertointi on vaikeaa tai mahdotonta. Tällaiset osat kannattaa konvertoida manuaalisesti.

Jokainen konvertointitilanne on pääsääntöisesti erilainen, eikä ole olemassa yhtä oikeaa tapaa suorittaa konvertointi. Millainen strategia kuhunkin tilanteeseen valitaan riippuu pitkälti lähdeaineiston muodosta, tulosdokumenttien käyttötarkoituksesta ja konversioprojektille asetetuista vaatimuksista.

Tässä tutkielmassa lähestyimme SGML-konvertointia teoreettiselta pohjalta ja vertailimme näiden yleisten periaatteiden kautta muutamaa markkinoilla olevaa konversiovälinettä.

Emme vertailleet varsinaisesti välineitä keskenään, vaan pyrimme selvittämään niiden käyttökelpoisuuden erityyppisissä konvertointitilanteissa. Mikäli olisimme halunneet vertailla välineitä toisiinsa, olisi valittavien konversiovälineiden täytynyt olla ominaisuuksiltaan samankaltaisia.

Tutkielman aihealuetta olisi voitu laajentaa ulottamalla vertailu myös S-konversio-ohjelmiin. Tällöin olisimme voineet vertailla esimerkiksi konversiovälineitä, joilla muunnetaan sateenkaari-muotoiset dokumentit halutun dokumenttityyppimäärittelyn mukaiseksi.

VIITELUETTELO

Chahuneau F. (1994). *Current Approaches to SGML Up-Translation.* http://www.bim.be/BeLuxweb/94/6sgmltr.html. 29.12.1998.

Chesnutt D.R., Lowry C.B. (1998). Managing Technology SGML and the Digital Libraries of Tomorrow. *Journal of Academic Librarianship*, **24** (3), ss. 232-237.

Deseyne J. (1995). *Authoring SGML Documents With Word Processors*. http://www.vtt.fi/inf/nordep/travel/belux95/jdeseyne.htm. 7.7.1999.

El Andaloussi J., Maler E. (1996). *Developing SGML DTDs - From Text to Model to Markup.* Prentice Hall, New Jersey.

Electronic Book Technologies (1993). *SGML '93: Rainbow "SGML Enabler"*. http://www.w3.mag.keio.ac.jp/Tools/RainbowAnnouncement.html. 18.08.1999.

Ensign C., Goldfarb C.E., Pepper S. (1998). *SGML Buyer's Guide*. Prentice Hall, New Jersey.

Goldfarb C.E. (1990). *SGML Handbook.* Clarendon Press, Oxford.

Inso Corporation (1999). *WWW-sivusto.* http://www.inso.com/. 1.12.1999.

International Organization for Standardization (1986). *International Standard 8879: Information Prosessing-Text and Office Systems -Standard Generalized Markup Language (SGML)*, first edition. ISO, Geneva.

Karjalainen A., Salminen A. (1999). Tallenteiden saartamana, dokumenttien hallinta organisaatiossa. *Tietoyhteys*, 3/99, s 28-29.

Korhonen J., Kuisma R. (1999). *DTD:n rakentaminen*. http://www.jyu.fi/~rkuisma/dtdhtml.html. 26.8.1999.

Kuikka E., Nikunen E. (1997). Trends in SGML software. SGML Finland 1997, SGML Users' Group Finland, ss. 131-141.

Kunnari R., Lyytikäinen V., Pietinen T. (1995). *Digitaalinen julkaiseminen - sovellusalueena yliopiston julkaisutuotanto.* http://www.cs.jyu.fi/~gelpo/sgmlspl/node1.html. 22.7.1999.

Kuronen T. (1997). *Hajautettu dokumenttien hallinta*. Oulun yliopiston kirjasto.

Leinonen P. (1996). *Syntaksin ohjaama dokumenttien muuntaminen*. Joensuun yliopisto, Tietojenkäsittelytieteen laitos, Pro Gradu -tutkielma.

McGrath S. (1998). *Parseme.1st SGML for Software Developers*. Prentice Hall, New Jersey.

OmniMark Technologies Corporation (1999). *WWW-sivusto.* http://www.omnimark.com/. 29.11.1999.

Salminen A. (1992). *Rakenteisen tekstin hallinta*. Jyväskylän yliopisto, Tietojenkäsittelytieteen laitos, opetusmoniste.

Salminen A. (1995). Elektroninen teksti - mitä se on ?. Teoksessa Kangas S., Karjalainen L. (Toim.): *SGML-seminaari*. Painatuskeskus Oy, ss. 1-18.

Severson E. (1999). *The Art of SGML Conversion: Eating Your Vegetables and Enjoying Dessert.* http://www.ileaf.com/avhome/veggies.html. 23.3.1999.

Sklar D. (1995). *The Annotated Rainbow DTD*. ftp://ftp.ebt.com/nv/nv/dtd/rainbow/rbow2-5.ps. 24.11.1999.

Travis B.E. (1997). *OmniMark at Work*. SGML University Press, Denver.

Travis B.E., Waldt D.C. (1995). *The SGML Implementation Guide. A Blueprint for SGML Migration.* Springer, Berliini.

Tyrväinen P. (1998). *SGML - Standard Generalized Markup Language.* http:// www.infoma.jyu.fi/digimedi/Pasi/eds/sgml.htm. 23.3.1999.

Virta H. (1995). Näkökulmia SGML:n soveltamiseen. Teoksessa Kangas S., Karjalainen L. (Toim.): *SGML-seminaari*. Painatuskeskus Oy, ss. 46-59.

Willner E. (1998). Preparing Data for the Web with SGML/XML. *Information Today*, **15** (5), s. 54.

Willner E. (1998). Preparing Data with SGML/XML: Part II. *Information Today*, **15** (6), ss. 52-54.

University of Joensuu
Department of Computer Science

Käyttöliittymät

Opinto-oppaan mukainen kurssikuvaus

Käyttöliittymät (3 ov) 173222

- Luennot 40 t, Harjoitukset 20 t
- Kuva tietokoneen ja sen käyttäjän välisen yhteyden toteuttamisesta ja ihmisen ominaisuuksien tälle yhteydelle asettamista vaatimuksista. Kyky suunnitella nykyaikainen sovelluksen käyttöliittymä. Kirjallisuus: Shneiderman: Designing the User Interface: Strategies for the Effective Human-Computer Interaction. Sutcliffe: Human-Computer Interface Design. Brown: Human-Computer Interface Design Guidelines. Esitiedot: Ohjelmoinnin peruskurssi.

Kurssin sisältö:

Käsiteltävät asiat:

- käyttäjä ja käyttöliittymä
- käyttöliittymien suunnittelu ja toteutus
- käyttöliittymien standardointi

Tavoitteet:

- omaksutaan vastuullinen suhde käyttöliittymien toteuttamiseen
- saadaan tietoa käyttöliittymien suunnittelun ja toteutuksen eri vaiheisiin käytettävissä olevista menetelmistä ja välineistä
- ymmärretään standardien merkitys käyttöliittymien suunnittelussa

Kirjallisuus:

- C.M. Brown: Human-Computer Interface Design Guidelines. Ablex Publishing Corporation, 1988.
- J. Nielsen: Usability Engineering. Academic Press, 1993.
- Sutcliffe: Human-Computer Interface Design. Macmillan Education Ltd, 1988.
- B. Shneiderman: Designing the User Interface: Strategies for the Effective Human-Computer Interaction. Addison-Wesley, 1992.
- Vähäisessä määrin myös muita lähteitä.

Kurssista on luentorunko paperimonisteena.

Sisällys:

- 8. Käyttöliittymästandardit

Käyttöliittymät syksyllä 1999

Opetusajankohdat:

- Luennot: 7.9.99 alkaen tiistaisin (M3) ja torstaisin (M2) klo 16-18 (Markku Tukiainen)
  Poikkeukset:
  - Tiistaina 5.10. ei ole luentoa liikuntapäivän vuoksi
  - Torstaina 16.9. ei ole luentoa. Sen tilalla pidetään luento tiistaina 16.11. klo 16-18, M3
- Harjoitusryhmät: keskiviikkoisin klo 16-18 (ryhmä 1) ja perjantaisin klo 12-14 (ryhmä 2) M13
  Poikkeukset:
  - --

- Välikokeet:

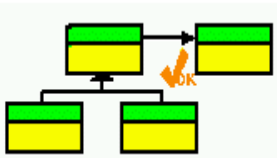
  HUOM! HUOM! Välikoeaika ja paikka muuttuneet: TORSTAI 14.10.99 16:00 - 18:00 sali M2

  ja tiistaina 23.11.99 klo 8-10, M1
  - Ensimmäiseen välikokeeseen tulee luentorungon luvut 1-4 sekä harjoitukset 1-4. Pääpaino on luentorungolla.

Kurssin suorittaminen välikokeilla edellyttää, että 1/3 harjoitustehtävistä on tehty (max 50, tehty 17).

Päivitetty viimeksi: 15.10.1999
mtuki@cs.joensuu.fi

University of Joensuu
Department of Computer Science



Software Engineering

Ohjelmistotuotanto (173302, 4 ov)

Kurssin sisältö:

- laatujärjestelmät ja ohjelmistotuotanto
- ohjelmistotyön vaiheet
- ohjelmistoprojektin hallinta

Tavoitteet:

- omaksutaan vastuullinen suhde ohjelmiston tuottamiseen
- saadaan tietoa ohjelmistotyön eri vaiheisiin käytettävissä olevista menetelmistä ja välineistä
- ymmärretään laatujärjestelmän merkitys ohjelmistotyölle

Kirjallisuus:

- MSQH - Modelling a Software Quality Handbook. STRI TS2, Icelandic Council for Standardization (STRI), 1991.
- W.S. Humphrey: Managing the Software Process. Addison-Wesley, 1989.
- P. Rook: Software Reliability Handbook. 1990.
- P. Yli-Olli, T. Hokkanen: Softa 9000. Mecrastor Oy, 1991.
- Vähäisessä määrin myös muita lähteitä.

Luentomoniste tilattavissa luentojen yhteydessä ensimmäisellä luentoviikolla. Luentomoniste on tarkoitettu luentojen tueksi. Kurssin sisältö ei ole opiskeltavissa pelkästään monisteen avulla. Moniste on ilmestynyt (hinta 20 mk) ja sitä on saatavana luentojen väliajoilla.

Sisällys:

Ohjelmistotuotanto syksyllä 1999

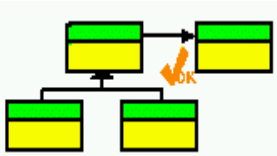
Opetusajankohdat:

- Luennot: 9.9.98 alkaen torstaisin ja perjantaisin klo 10-12 (M4) (Jorma Sajaniemi)
- Harjoitusryhmä: 20.9.99 alkaen maanantaisin klo 14-16, M13
- Välikokeet: tiistaina 2.11.99 ja 14.12.99, 8.00-10.00 M1

Kurssin suorittaminen välikokeilla edellyttää, että 1/3 harjoitustehtävistä on tehty.

Päivitetty viimeksi: 26.10.1999
saja@cs.joensuu.fi

University of Joensuu
Department of Computer Science



Software Engineering

Tietojärjestelmien dokumentoiminen (2 ov) 173207

Kurssi on pakollinen kurssi ohjelmistotuotannon linjalla, muilla linjoilla kurssi käy valinnaiseksi kurssiksi cum laude opintoihin.

Kurssin sisältö:

- Dokumenttien rakenteeseen vaikuttavat tekijät
- Eri dokumenttien sisällöt
- Dokumenttien laatiminen

Tavoitteet:

- Erilaisiin dokumentteihin tarvittavat asiat
- Dokumentoinnin oleellisten asioiden ymmärtäminen

Kirjallisuus:

- J. T. Hackos: Managing Your Documentation Projects, 1994
- M. Thirlway : Writing Software Manuals - a Practical Guide, 1994
- L. Denton, J. Kelly : Designing , writing and producing computer documentation, 1992

Sisällys:

- Dokumentoinnin tarpeellisuus
- Erilaiset dokumentit
- Dokumentin kirjoittaminen
- Dokumentointivälineet

Opetus syksyllä 1999

Opetusajankohdat:

Luennot: (**18.10**. - 2.12.1999, ei viikolla 44)

- ma 10 - 12 M3 ja
- to 12 - 14 M4

Harjoitukset (25.10. - 8.12.1999, ei viikolla 44)

- Ryhmä 1: ti 8 - 10 M13
- Ryhmä 2: ke 12 - 14 M13
- Muuntokoulutus Ryhmä 3: ma 16 - 18 M10
- Muuntokoulutus Ryhmä 4: to 8 - 10 M10

Kertauskuulustelu:
Kertauskuulusteluun saa osallistua, kun on tehnyt vähintään kolmanneksen kurssin harjoitustehtävistä. Harjoitustehtävistä saa ylimääräisiä pisteitä

2 - 6, kun kurssikuulustelun maksimi on 60 pistettä (arvosteluperusteet).

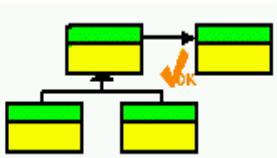
Kurssikuulustelu **torstaina 16.12.1999 klo 12 - 14 sali M1**

Loppukuulustelut:

- 21.1.2000,
- 31.3.2000,
- 14.6.2000

Päivitetty viimeksi: 20.10.1999
vouti@cs.joensuu.fi

University of Joensuu
Department of Computer Science



Software Engineering

Systeeminsuunnittelu

Kurssin sisältö:

- toimintayksikön tietojenkäsittely, tietojärjestelmät ja niiden kehittäminen
- kehikko yksittäiselle kehittämishankkeelle
- työtavat ja apuvälineet systeemityön suorittamiseksi

Tavoitteet:

- ymmärretään tietojärjestelmät osana toimintayksikön toimintaa
- osataan tietojärjestelmän rakentamisen eri vaiheet
- osataan soveltaa erilaisia työtapoja ja välineitä tietojärjestelmän rakentamisessa

Kirjallisuutta:

- luentomoniste
- Bennett et al.1999: Object-oriented system analysis and design.
- Hoffer et al. 1999: Modern systems analysis and design.
- Karvinen et al. 1994: Tietotekniikkainvestoinnit.
- SFS-ISO 5807 Tietojenkäsittely.Dokumentointi. Symbolit. 1989-05-15.
- Valtioneuvoston päätös näyttöpäätetyöskentelystä, N:o 1405, 1993.
- Dreger 1989: Function Point Analysis.
- EDIFACT ABC-kirja 1990.
- Longhurst et al. 1995: How to make light of HTML.

Sisällys:

Systeeminsuunnittelu syksyllä 1999

Opetusajankohdat:

- Luennot: 7.9. alkaen tiistaisin klo 10-12 ja perjantaisin klo 8-10, M3 (Raimo Rask)
- Harjoitusryhmät: Ryhmä1: torstaisin klo 10-12 M13, Ryhmä 2: torstaisin klo 16-18 M13
- Kertauskuulustelut: 1. tiistai 5.10. klo 8-10 M1, 2. tiistai 7.12. klo 8-10 M1.

Kurssin suorittaminen kertauskuulusteluilla edellyttää, että 1/3 harjoitustehtävistä on tehty.

Päivitetty viimeksi: 13.8.1999
rask@cs.joensuu.fi

Esimerkkidokumentaation konvertointiin tarkoitettu OmniMark-ohjelma

```
up-translate

;dokumentin alku
find-start
    output "<!DOCTYPE kurssi PUBLIC %"-//JoY//DTD kurssi//FIN%">%n"
    output "<kurssi>%n<tunniste>%n"

;dokumentin loppu
find-end
    output "%n</kurssi>%n"

;funktio listakohtien käsittelyyn
define function listakohdat value stream lkohta as
     repeat scan lkohta
        match "* "
            output "<kohta>"
        match "%n"
            output "</kohta>%n"
        match any => merkki
            output merkki
    again

;sisennetyt pääotsikot
find line-start space+
    any-text+=paaotsikko
    output "</kuvaus>%n%n<opetus>%n<opotsik>%x(paaotsikko)</opotsik)%n"
           when element is kuvaus
    output "<nimi>%x(paaotsikko)</nimi>%n</tunniste>%n<kuvaus>%n"
           when element is tunniste

;sisalto-elementti
find line-start (("Kurssin sisältö:") OR ("Käsiteltävät asiat:")) =>
sisaltoteksti "%n%n" (any-text+ "%n")+ => kohdat
    output "<sisalto>%n<otsikko>%x(sisaltoteksti)</otsikko>%n"
    listakohdat kohdat
    output "</sisalto>%n%n"

;tavoite-elementti
find line-start ("Tavoitteet:") => tavoiteteksti "%n%n" (any-text+ "%n")+
=> kohdat
    output "<tavoite>%n<otsikko>%x(tavoiteteksti)</otsikko>%n"
    listakohdat kohdat
    output "</tavoite>%n%n"

;kirjal-elementti
find line-start (("Kirjallisuus:") OR ("Kirjallisuutta")) => kirjateksti
"%n%n" (any-text+ "%n")+ => kohdat
    output "<kirjal>%n<otsikko>%x(kirjateksti)</otsikko>%n"
    repeat scan kohdat
        match "* "
            output "<kirja>"
        match "%n"
            output "</kirja>%n"
        match any => merkki
            output merkki
    again
    output "</kirjallisuus>%n%n"
```

```
;sisal-elementti
find line-start ("Sisällys:") => sisallysteksti "%n%n" (any-text+ "%n")+
=> kohdat
    output "<sisal>%n<otsikko>%x(sisallysteksti)</otsikko>%n"
    listakohdat kohdat
    output "</sisal>%n%n"

;oplista-elementti
find line-start ("Opetusajankohdat:") => opetusaikateksti "%n%n" (any-
text+ "%n")+ => kohdat
    output "<oplista>%n<opotsik>%x(opetusaikateksti)</opotsik>%n"
    repeat scan kohdat
        match space{3} "* "
            output ""
        match "* "
            output "</poikkeus>%n<kohta>" when element is poikkeus
            output "</huom>%n<kohta>" when element is huom
            output "<kohta>" when element is oplista
        match "%n"
            output "</kohta>%n" when element is kohta
        match "POIKKEUS:" OR "POIKKEUKSET:" OR "Poikkeus:" OR
        "Poikkeukset:"
            output "<poikkeus>"
        match "HUOM! HUOM!"
            output "<huom>"
        match any => merkki
            output merkki
    again
    output "</poikkeus>" when element is poikkeus
    output "</huom>" when element is huom
    output "</oplista>%n%n"

;viimpaiv- ja paivitt-elementit
find line-start "Päivitetty viimeksi: " any-text+ => viimpaivteksti "%n"
(any-text+ => paivittajateksti)
    output "<viimpaiv>%x(viimpaivteksti)</viimpaiv>%n"
    output "<paivitt>%x(paivittajateksti)</paivitt>%n"

;muut tekstit lisays-elementin sisällöksi
find any-text+ => lisaysteksti
    output "<lisays>%x(lisaysteksti)</lisays>"
```

Esimerkki OmniMark-ohjelman konvertoimasta SGML-tiedostosta

```
<!DOCTYPE kurssi PUBLIC "-//JoY//DTD kurssi//FIN">
<kurssi>
<tunniste>
<nimi>Systeeminsuunnittelu</nimi>
</tunniste>
<kuvaus>
<sisalto>
<otsikko>Kurssin sisältö:</otsikko>
<kohta>toimintayksikön tietojenkäsittely, tietojärjestelmät ja niiden
kehittäminen</kohta>
<kohta>kehikko yksittäiselle kehittämishankkeelle</kohta>
<kohta>työtavat ja apuvälineet systeemityön suorittamiseksi</kohta>
</sisalto>


<tavoite>
<otsikko>Tavoitteet:</otsikko>
<kohta>ymmärretään tietojärjestelmät osana toimintayksikön
toimintaa</kohta>
<kohta>osataan tietojärjestelmän rakentamisen eri vaiheet</kohta>
<kohta>osataan soveltaa erilaisia työtapoja ja välineitä
tietojärjestelmän rakentamisessa</kohta>
</tavoite>


<kirjal>
<otsikko>Kirjallisuutta:</otsikko>
<kirja>luentomoniste</kirja>
<kirja>Bennett et al.1999: Object-oriented system analysis and
design.</kirja>
<kirja>Hoffer et al. 1999: Modern systems analysis and design.</kirja>
<kirja>Karvinen et al. 1994: Tietotekniikkainvestoinnit.</kirja>
<kirja>SFS-ISO 5807 Tietojenkäsittely.Dokumentointi. Symbolit. 1989-05-
15.</kirja>
<kirja>Valtioneuvoston päätös näyttöpäätetyöskentelystä, N:o 1405,
1993.</kirja>
<kirja>Dreger 1989: Function Point Analysis.</kirja>
<kirja>EDIFACT ABC-kirja 1990.</kirja>
<kirja>Longhurst et al. 1995: How to make light of HTML.</kirja>
</kirjallisuus>


<sisal>
<otsikko>Sisällys:</otsikko>
<kohta>1. Systeemit ja systeemityö</kohta>
<kohta>2. Kehikko yksittäiselle kehittämishankkeelle</kohta>
<kohta>3. Ylläpito</kohta>
<kohta>4. Työtavat ja apuvälineet</kohta>
</sisal>


</kuvaus>

<opetus>
<opotsik>Systeeminsuunnittelu syksyllä 1999</opotsik)


<oplista>
```

```
<opotsik>Opetusajankohdat:</opotsik>
<kohta>Luennot: 7.9. alkaen tiistaisin klo 10-12 ja perjantaisin klo 8-
10, M3 (Raimo Rask)</kohta>
<kohta>Harjoitusryhmät: Ryhmä1: torstaisin klo 10-12 M13, Ryhmä 2:
torstaisin klo 16-18 M13</kohta>
<kohta>Kertauskuulustelut: 1. tiistai 5.10. klo 8-10 M1, 2. tiistai 7.12.
klo 8-10 M1.</kohta>
</oplista>


<lisays>Kurssin suorittaminen kertauskuulusteluilla edellyttää, että 1/3
harjoitustehtävistä on tehty.</lisays>

<viimpaiv>13.8.1999</viimpaiv>
<paivitt>rask@cs.joensuu.fi</paivitt>


</kurssi>
```

Sateenkaari -dokumenttityyppimäärittely

```
<!--

Rainbow DTD: VERSION
   Current version: 2.5
   The exact RCS version stamp is stored just after the end of the SGML decl.


Rainbow DTD: TYPICAL USAGE
            <!DOCTYPE rainbow PUBLIC "-//EBT//DTD Rainbow 2.5//EN" [
            ]>


Rainbow DTD: SGML DECLARATION
            The SGML declaration for Rainbow documents lies in "rainbow.dcl".


Rainbow DTD: LEGAL NOTES

   Copyright 1993-1995 Electronic Book Technologies, Inc.

   Permission to use, copy, modify and distribute the Rainbow DTD and
     its accompanying documentation for any purpose and without fee is
     hereby granted, provided that this copyright notice appears in all
     copies.  If you modify the Rainbow DTD, rename your modified
     version and make it clear that your version is not the "official"
     version maintained by EBT.

   EBT makes no representation about the suitability of the DTD
     for any purpose. It is provided "as is" without expressed or
     implied warranty.

   The Rainbow DTD is maintained by EBT, Inc.  Please
     direct all questions, bug reports, or suggestions for changes to:
                 dfs@ebt.com

   Public Identifier:
      "-//EBT//DTD Rainbow 2.5//EN"

-->



<!-- Rainbow DTD: CHANGE LOG

   version 2.5 by alb:
            *) 1994/12/28 proforma attribute on CLF, permitting clf with no
overrides
            *) 1994/12/23 SYSATTR on TABLE, ROW, and ENTRY
            *) 1994/11/29 ANCHOR may be a sibling of PARA (for RTF)
            *) 1994/11/21 XREF may contain TABs
            *) 1994/11/03 arch-type, arch-type-info added
   1994/06/02 (RCS version 2.4) by alb:
            *) NAMEDCLFs nested, continue attribute
            *) NDATA NOTATIONS & entities for graphics and equations
               Notation names are 3 letters in length to facilitate their
               acting as DOS filename suffixes.  (That means our notation names
               are not compatible with those recommended by ISO/IEC/JTC1/WG8)
            *) Fix for AUTOGEN content model: can contain TAB

   1994/05/10 (RCS version 2.3) by dfs:
            *) Major new release.  For full details, see the "Annotated DTD"
               document, available in PostScript form via the FTP server.
            *) DTD is now auto-generated from the WinWord "Annotated DTD"
document.
```

```
            This allows dfs to maintain the DTD by editing a single
            document, instead of doing error-prone multiple-file editing.
            The side-effect: the rainbow.dtd file is not as pretty as it once
            was, for vertical spacing is not provided.

   1994/02/21 (RCS version 2.2) by dfs:
            *) HEAD is now merely a wrapper around one or more PARAs.
            *) STRUCLVL now supports the ID attribute.
            *) SGML declaration now fully supports ISO 8859-1 character set, and
               is stored in a separate file (rainbow.dcl) for convenience.
            *) CLFTYPE now supported; NAMEDCLF elements can be used to refer to
               a CLFTYPE.
            *) NAMEDCLFs can contain CLFs, thus providing limited nesting of
               character-level-formatting objects.
            *) Definition of an SDATA entity for representation of tabchars.
            *) ANCHOR, IDXTERM, and XREF: content model repaired to match the
               original intended semantics.  The content models are now
               repeatable OR groups.
                 NOTE: This leaves AUTOGEN and CLF as the only element types
                       with controversial "mixed content" models.  We hope
                           to resolve these controversies at the SGML/Open
                           Rainbow review.

   1994/01/24 (RCS version 2.1) by dfs:
            *) Set SHORTTAG to YES in SGML decl (recommended use: only to
               allow omission of attribute-value delimiters)
            *) Various additions to the set of element types supporting
               end-tag omission.
            *) This change log was moved out of the SGML decl.

   1994/01/10 (RCS version 2.0) by dfs:
            ** Very large number of changes - -  see the Annotated DTD document
               for complete details. **

   1993/12/06 (RCS version 1.9) by dfs:
            *) The SGML declaration minimization portion now allows OMITTAG.

   1993/12/06 (RCS version 1.8) by dfs:
            *) Font size is no longer a NUMBER (to support fractionals)

   1993/12/05 (RCS version 1.7) by dfs:
            *) Deletion of "continue" attribute on PARA/HEAD.  That attribute
               never really existed; it was a mega-typo.
            *) Addition of "dtdver" attribute allowing a document instance
               to specify the exact version of Rainbow DTD used by its
               generator/author.
            *) Addition of copyright notice to DTD.

   1993/11/30 (RCS version 1.6) by dfs:
            *) MARK now has same content model as PARACONT.

   1993/11/29 (RCS version 1.5) by dfs:
            *) Typo fix: TXTFLD changed to TEXTFLD

   1993/11/29 (RCS version 1.4) by dfs:
            *) All PLF/CLF attributes are #IMPLIED.
            *) NUMBER type now used very sparingly, because of its inability
               to support negative numbers.  NUMBER now used only in cases for
               which a negative number is absolutely impossible.
            *) All ID attributes are of type CDATA, because word processors
               allow "bookmarks" to have names that do not meet the SGML
               requirements for ID-type attribute values.
            *) GI change: INDEXENTRY is now IDXTERM
               Motivation: it was the only GI that was > 8 chars in length
            *) IDXTERM was an "orphan" in the previous version; now it is
               included in the DOC content model as an inclusion exception.
            *) The PARACONT attributes are now #IMPLIED.
-->
```

```
<!-- NOTE: THIS DTD IS NOT MEANT FOR HUMAN CONSUMPTION.
               THIS IS FOR MACHINE PROCESSING ONLY.
               HUMANS SHOULD READ THE "Annotated DTD", AVAILABLE IN PostScript
               FORM VIA THE FTP SERVER (pub/nv/dtd/rainbow/rbow2-x.ps)
  -->


<!-- NOTE: THIS DTD IS NOT MEANT FOR HUMAN CONSUMPTION.
               THIS IS FOR MACHINE PROCESSING ONLY.
               HUMANS SHOULD READ THE "Annotated DTD", AVAILABLE IN PostScript
               FORM VIA THE FTP SERVER (pub/nv/dtd/rainbow/rbow2-x.ps)
  -->


<!-- NOTE: THIS DTD IS NOT MEANT FOR HUMAN CONSUMPTION.
               THIS IS FOR MACHINE PROCESSING ONLY.
               HUMANS SHOULD READ THE "Annotated DTD", AVAILABLE IN PostScript
               FORM VIA THE FTP SERVER (pub/nv/dtd/rainbow/rbow2-x.ps)
  -->



<!ELEMENT rainbow  - - (FILEINFO?,STYINFO,DOC)>
<!ELEMENT fileinfo - O EMPTY>
<!ATTLIST fileinfo     origin   CDATA #IMPLIED
                       dtdver   CDATA #REQUIRED  -- Rev. number of Rainbow DTD
                                                    used for this instance -->
<!ENTITY % boolean "NUMBER"  -- 1 means yes, 0 means no -->
<!ENTITY % plf-att -- Paragraph Level Formatting --
"left-indent CDATA #IMPLIED
       -- measured in points; absolute number; do not use += notation --
 right-indent CDATA #IMPLIED
       -- measured in points; absolute number; do not use += notation --
 first-indent CDATA #IMPLIED
       -- same as in DynaText: offset from the left-indentation --
 justification (LeftJust|CenterJust|RightJust|FullJust|InJust|OutJust) #IMPLIED
       -- FullJust means both left and right justification
          InJust/OutJust means justify towards binding/perimeter --
 space-before CDATA #IMPLIED
 space-after CDATA #IMPLIED
       -- both of the above are as in DT: measured in pts --
 keep-with-next %boolean  #IMPLIED
 keep-with-prev %boolean  #IMPLIED
       -- keep with (next/prev) paragraph --
 tab-stops CDATA #IMPLIED
       -- space-separated list of tabstop specifiers
          each specifier must be of one of the following forms:
                   num                left-aligned tabstop
                   >num               right-aligned tabstop
                   |num               center-aligned tabstop
                   C.num              US-decimal-point-aligned tabstop
                   C,num              Euro-decimal-point-aligned tabstop, etc.
          The 'num' is measured in points from the left margin --
 keep-together %boolean; #IMPLIED
       -- does author demand that the paragraph not cross page boundaries? --
 pg-brk-before %boolean; #IMPLIED       -- page-break before --
 col-brk-before %boolean; #IMPLIED      -- column-break before --
 border NUMBER #IMPLIED
       -- value is sum of the appropriate members of this list:
                1 for top
                2 for right
                4 for bottom
                8 for left   --
">
```

```
<!ENTITY % clf-att -- Character Level Formatting --
"charset CDATA #IMPLIED
       -- standard values for this attribute will be announced later --
 font-family CDATA #IMPLIED
 font-size CDATA #IMPLIED
 font-weight (Medium|Bold) #IMPLIED
 font-slant (Roman|Ital)   #IMPLIED    -- Ital for both italic and oblique --
 line-spacing CDATA #IMPLIED
       -- same as in DT: distance between baselines --
 score-location (Under|Over|Through) #IMPLIED
 score-type (Single|Double|Dotted) #IMPLIED
       -- score-type is ignored if score-location is not set --
 vertical-offset CDATA #IMPLIED
       -- as in DT: offset from the baseline, in points, - for sub, + for sup
--
 foreground CDATA #IMPLIED
 background CDATA #IMPLIED
       -- If the generator can do so, it must specify color as a RGB
             hexadecimal numbers in this format:  #rrggbb
          If the generator is unable to do so, it can use an arbitrary
             representation (e.g. name, index number).
          Note that if the latter representation is used, it will not
             be possible for the Rainbow consumer to reproduce colors
             in its rendering facilities. --
       -- Note: if the text is not in a special color, don't set
             the foreground attribute. --
       -- Note: if the text is not on a special background, don't set
             the background attribute. --
       -- Example: use #888888 for background in the typical case of shaded
                   background for black-on-white printing --
 lowercase-display (SmallCaps|FullCaps)  #IMPLIED
       -- Note that this specifies how lowercase data is displayed;
          this does not describe the data itself. --
 outline %boolean; #IMPLIED
       -- Is each letter being displayed as a hollow outline? --
 change-bar (BarLeft|BarRight|BarIn|BarOut) #IMPLIED
 hidden %boolean; #IMPLIED
">
<!ENTITY % com-att -- Common attributes that are not style dependent --
"arch-type CDATA #IMPLIED
       --
          recognized values:
            TOC - semantic of paragraph is 'table of content' element
                - arch-type value is the level number, starting at 1
                  for the book title, 0 means that the level number
                  cannot be determined
       --
 arch-type-info CDATA #IMPLIED
">
<!ELEMENT styinfo  - - (PARATYPE|CLFTYPE)+>
<!ELEMENT paratype - O EMPTY>
<!ATTLIST paratype     name     CDATA #REQUIRED
                       %plf-att; %clf-att; %com-att;>
<!ELEMENT clftype - O EMPTY>
<!ATTLIST clftype     name     CDATA #REQUIRED
                      %clf-att; %com-att;>
<!ELEMENT doc      - - ( HEAD?,
                         (((ANCHOR|PARA|ILLUS|TABLE|SYSOBJ)+,
                           STRUCLVL*)
                          |(STRUCLVL+ ) ) )
                        -- inclusion exceptions --
                        +(WPLOC|IDXTERM)          >
<!ELEMENT struclvl - -
(SYSATTRS?,HEAD,(ANCHOR|PARA|ILLUS|TABLE|SYSOBJ)*,STRUCLVL*)>
<!ELEMENT head     - - (SYSATTRS?,(ANCHOR|PARA)+)>
<!ATTLIST (struclvl|head)
            id CDATA #IMPLIED>
<!ELEMENT para  - O (SYSATTRS?,PARACONT)>
```

```
<!ATTLIST para
                      paratype CDATA    #REQUIRED
                      %plf-att; %clf-att; %com-att;
      -- If the para is a child of a head, the ID attribute should go
         on the head, not the para --
                      id       CDATA    #IMPLIED
      -- remaining attrs are set only if this element
         represents a "continuation" of a para in orig WP doc --
                      continue %boolean #IMPLIED
                      preced-hard-returns NUMBER #IMPLIED >
<!ENTITY % par-cont
         "#PCDATA|SYSOBJ|ANCHOR|XREF|AUTOGEN|GRAPHIC|GRAPHGRP|EQN|NOTE|TAB">
<!ELEMENT paracont - O
          (%par-cont;|NAMEDCLF|CLF)+ >


<!-- The DTD referenced by the following public ID can be found at
     ftp.ebt.com:pub/nv/dtd/rainbow/rbw-all.ent -->
<!ENTITY % rbwents  PUBLIC
   "-//EBT//ENTITIES Concatenated Rainbow 2.5//EN">   %rbwents;


<!ELEMENT tab - O EMPTY>
<!ELEMENT illus    - - (SYSATTRS?,(GRAPHIC|GRAPHGRP|EQN))>
<!ATTLIST illus       %plf-att; %com-att;
                      id       CDATA #IMPLIED>
<!ELEMENT graphgrp - - (GRAPHIC+)>
<!ELEMENT graphic  - - (SYSATTRS?)>
<!ATTLIST graphic     filename ENTITY #REQUIRED>
<!ELEMENT eqn      - - (SYSATTRS?,EQNCONT?)>
<!ELEMENT eqncont   - - RCDATA>
<!ATTLIST eqn         format NOTATION (rtf|ieq|feq) #IMPLIED
                      filename ENTITY #IMPLIED>
<!ELEMENT namedclf  - -  (SYSATTRS?,(%par-cont;|CLF|NAMEDCLF)+) >
<!ELEMENT clf       - -  (SYSATTRS?,(%par-cont;)+ )>
<!ATTLIST namedclf    clftype CDATA #REQUIRED
                      %clf-att; %com-att;
                      ID CDATA #IMPLIED
      -- remaining attrs are set only if this element
         represents a "continuation" of a para in orig WP doc --
                      continue %boolean #IMPLIED
                      preced-hard-returns NUMBER #IMPLIED >
<!ATTLIST clf         %clf-att; %com-att;
                      ID CDATA #IMPLIED
      -- proforma has value 1 if this clf has no (other) attributes --
                      proforma CDATA #IMPLIED >
<!ELEMENT note - - (SYSATTRS?,(ANCHOR|PARA|ILLUS|TABLE|SYSOBJ)*)    -(NOTE) >

<!ELEMENT idxterm   - - (#PCDATA|CLF|NAMEDCLF)+           -(IDXTERM) >
<!ATTLIST idxterm      term1 CDATA #IMPLIED
                       term2 CDATA #IMPLIED
                       term3 CDATA #IMPLIED
                       see CDATA #IMPLIED
                       see-also CDATA #IMPLIED>

<!ELEMENT anchor    - - (#PCDATA|CLF|NAMEDCLF)+>
<!ATTLIST anchor       ID CDATA #REQUIRED>
<!ELEMENT xref      - - (#PCDATA|CLF|NAMEDCLF|TAB)+>
<!ATTLIST xref         REFID CDATA #REQUIRED
                       TYPE (xref-textcopy|xref-pagenum|xref-link) #REQUIRED>
<!ELEMENT autogen      - - (SYSATTRS?,(#PCDATA|TAB|NAMEDCLF|CLF)*)>
<!ELEMENT wploc      - O EMPTY>
<!ATTLIST wploc      wp-addr    CDATA #REQUIRED
                     human-addr  CDATA #IMPLIED>
<!ELEMENT sysobj   - - (datafld|textfld)+>
<!ATTLIST sysobj       type CDATA #REQUIRED
                       id       CDATA #IMPLIED>
<!ELEMENT datafld  - - RCDATA>
```

```
<!ELEMENT textfld  - - (#PCDATA|CLF|NAMEDCLF)+>
<!ATTLIST (datafld|textfld)   type CDATA #REQUIRED>
<!ELEMENT sysattrs   - - (sysattr*)>
<!ELEMENT sysattr    - - RCDATA>
<!ATTLIST sysattr       attrname CDATA #REQUIRED>
<!ELEMENT table    - - (sysattrs?,tgroup+)>
<!ATTLIST table        %plf-att; %com-att;
                       id CDATA #IMPLIED>
<!ELEMENT tgroup   - - (colspec*,spanspec*,thead?,tfoot?,tbody?)>
<!ATTLIST tgroup       cols NUMBER #REQUIRED>
<!ELEMENT colspec  - O EMPTY>
<!ATTLIST colspec      colnum NUMBER #IMPLIED
                       colname NMTOKEN #IMPLIED
                       align (left|right|center) #IMPLIED
                       colsep %boolean; #IMPLIED
                       rowsep %boolean; #IMPLIED
                       colwidth CDATA #IMPLIED>
<!ELEMENT spanspec - O EMPTY>
<!ATTLIST spanspec     namest NMTOKEN #REQUIRED
                       nameend NMTOKEN #REQUIRED
                       spanname NMTOKEN #REQUIRED
                       colsep %boolean; #IMPLIED
                       rowsep %boolean; #IMPLIED
                       align (left|right|center) #IMPLIED>
<!ELEMENT (thead|tfoot|tbody)
                   - - (row+)>
<!ELEMENT row      - - (sysattrs?,entry*)>
<!ATTLIST row          rowsep %boolean; #IMPLIED >
<!ELEMENT entry    - - (sysattrs?,(ANCHOR|PARA|ILLUS|SYSOBJ)*)>
<!ATTLIST entry        colname NMTOKEN #IMPLIED
                       spanname NMTOKEN #IMPLIED
                       colsep %boolean; #IMPLIED
                       rowsep %boolean; #IMPLIED

               morerows NUMBER       '0'

               valign (top|middle|bottom) 'top'
                       align (left|right|center) #IMPLIED>

<!NOTATION cgm              PUBLIC "ISO 8632-4:1987//NOTATION Information
                           processing systems - Computer Graphics -
                           Metafile for the storage and transfer of
                           picture description information -
                           Part 4: Clear text encoding//EN"        >
<!NOTATION igs      PUBLIC "NBS IR 88-3813//NOTATION Initial
                           Graphics Exchange Specification (IGES)
                           //EN"                                    >
<!NOTATION fax      PUBLIC "CCITT VII.3 T 6//NOTATION Blue book -
                           Terminal equipment and protocols for
                           telematic services - Facsimile encoding
                           schemes and coding control functions for
                           group 4 facsimile apparatus//EN"        >

<!NOTATION eps      PUBLIC "+//ISBN 0-201-18127-4::Adobe//NOTATION
                           PostScript Language Reference Manual//EN">
<!NOTATION tif      PUBLIC "-//Aldus//NOTATION
                           Tagged Image File Format//EN">
<!NOTATION g4       PUBLIC "-//Aldus//NOTATION
                           Group 4 Compression//EN">
<!NOTATION wmf      PUBLIC "-//Microsoft Corporation//NOTATION
                           Windows Metafile//EN">
<!NOTATION pmm      PUBLIC "+//ISBN 0-933186::IBM//NOTATION
                           OS/2 Metafile Format//EN">
<!NOTATION ilg      PUBLIC "-//Interleaf Inc//NOTATION
                           ASCII Format for Graphic Objects//EN">
<!NOTATION fri              PUBLIC "-//Frame Technology Corporation//NOTATION
                            Frame Image//EN">
<!NOTATION frg      PUBLIC "-//Frame Technology Corporation//NOTATION
```

```
                        MIF Graphic Objects//EN">
<!NOTATION bmp      PUBLIC "-//Microsoft Corporation//NOTATION
                        Windows Bit Map//EN">
<!NOTATION dib      PUBLIC "-//Microsoft Corporation//NOTATION
                        Device Independent Bit Map//EN">
<!NOTATION wpg      PUBLIC "-//WordPerfect Corporation//NOTATION
                        WordPerfect Graphic//EN">
<!NOTATION pcx      PUBLIC "-//Corel//NOTATION
                        Corel PhotoPaint Bitmap//EN">
<!NOTATION pct      PUBLIC "-//Apple//NOTATION
                        Mac Pict//EN">
<!NOTATION rtf      PUBLIC "-//Microsoft Corporation//NOTATION
                        Rich Text Format//EN">
<!NOTATION ieq      PUBLIC "-//Interleaf Inc//NOTATION
                        ASCII Format for Equation Objects//EN">
<!NOTATION feq      PUBLIC "-//Frame Technology Corporation//NOTATION
                        MIF Equation Objects//EN">


<!NOTATION unknown        SYSTEM "">
<!NOTATION unk            SYSTEM ""> <!-- unk is here for backward compat -->
```

Käyttöliittymät-kurssin sivu sateenkaari-DTD:n mukaisesti merkattuna

```
<!DOCTYPE rainbow PUBLIC "-//EBT//DTD Rainbow 2.5//EN" [
<!ENTITY fgraf0000.wmf SYSTEM "graf0000.wmf" NDATA wmf >
]>
<RAINBOW><FILEINFO ORIGIN="WinWord-RTF1" DTDVER="2.5"
><STYINFO><PARATYPE FONT-FAMILY="Times New Roman" FONT-SIZE="10" LINE-
SPACING="12" CHARSET="ISO-8859" JUSTIFICATION="LeftJust" FIRST-INDENT="0"
LEFT-INDENT="0" RIGHT-INDENT="0" SPACE-BEFORE="0" SPACE-AFTER="0" FONT-
WEIGHT="Medium" FONT-SLANT="Roman" NAME="Normal"
><PARATYPE FONT-FAMILY="Arial" FONT-SIZE="14" LINE-SPACING="16"
CHARSET="ISO-8859" JUSTIFICATION="LeftJust" FIRST-INDENT="0" LEFT-
INDENT="0" RIGHT-INDENT="0" SPACE-BEFORE="12" SPACE-AFTER="3" KEEP-WITH-
NEXT=1 FONT-WEIGHT="Bold" FONT-SLANT="Roman" NAME="heading 1"
>

...

</STYINFO
><DOC><STRUCLVL><HEAD><WPLOC wp-addr="1">
<PARA SPACE-BEFORE="12" SPACE-AFTER="3" KEEP-WITH-NEXT=1 FONT-
WEIGHT="Bold" FONT-FAMILY="Arial" PARATYPE="heading 2" ><PARACONT><CLF
FONT-SIZE="10" ></CLF
><GRAPHIC FILENAME="fgraf0000.wmf"
></GRAPHIC></PARACONT
></PARA
></HEAD
></STRUCLVL
><STRUCLVL><HEAD><WPLOC wp-addr="2">
<PARA SPACE-BEFORE="12" SPACE-AFTER="3" KEEP-WITH-NEXT=1 FONT-
WEIGHT="Bold" FONT-FAMILY="Arial" PARATYPE="heading 2"
><PARACONT></PARACONT
></PARA
></HEAD
><WPLOC wp-addr="3">
<PARA FONT-SIZE="10" LINE-SPACING="12" PARATYPE="Normal"
><PARACONT></PARACONT
></PARA
><STRUCLVL><HEAD><WPLOC wp-addr="4">
<PARA SPACE-BEFORE="12" SPACE-AFTER="3" KEEP-WITH-NEXT=1 FONT-
WEIGHT="Bold" FONT-FAMILY="Arial" FONT-SIZE="14" LINE-SPACING="16"
PARATYPE="heading 1" ><PARACONT>K&auml;ytt&ouml;liittym&auml;t</PARACONT
></PARA
></HEAD
></STRUCLVL
></STRUCLVL
><STRUCLVL><HEAD><WPLOC wp-addr="5">
<PARA SPACE-BEFORE="12" SPACE-AFTER="3" KEEP-WITH-NEXT=1 FONT-
WEIGHT="Bold" FONT-FAMILY="Arial" PARATYPE="heading 2" ><PARACONT>Opinto-
oppaan mukainen kurssikuvaus</PARACONT
></PARA
></HEAD
><WPLOC wp-addr="6">
<PARA FONT-SIZE="10" LINE-SPACING="12" PARATYPE="Normal" ><PARACONT><CLF
FONT-WEIGHT="Bold" >K&auml;ytt&ouml;liittym&auml;t (3 ov) 173222</CLF
></PARACONT
></PARA
><WPLOC wp-addr="7">
<PARA FONT-SIZE="10" LINE-SPACING="12" PARATYPE="Normal" ><PARACONT><CLF
FONT-WEIGHT="Bold" ></CLF
```

```
></PARACONT
></PARA
><WPLOC wp-addr="8">
<PARA FONT-SIZE="10" LINE-SPACING="12" PARATYPE="Normal" ><PARACONT><CLF
FONT-WEIGHT="Bold" ></CLF
></PARACONT
></PARA
><WPLOC wp-addr="9">
<PARA FIRST-INDENT="-18" LEFT-INDENT="18" PARATYPE="Normal"
><PARACONT><AUTOGEN>&bull;<tab
></AUTOGEN>Luennot 40 t, Harjoitukset 20 t</PARACONT
></PARA
><WPLOC wp-addr="10">
<PARA FIRST-INDENT="-18" LEFT-INDENT="18" PARATYPE="Normal"
><PARACONT><AUTOGEN>&bull;<tab
></AUTOGEN>Kuva tietokoneen ja sen k&auml;ytt&auml;j&auml;n v&auml;lisen
yhteyden toteuttamisesta ja ihmisen ominaisuuksien t&auml;lle yhteydelle
asettamista vaatimuksista. Kyky suunnitella nykyaikainen sovelluksen
k&auml;ytt&ouml;liittym&auml;. Kirjallisuus: Shneiderman: Designing the
User Interface: Strategies for the Effective Human-Computer Interaction.
Sutcliffe: Human-Computer Interface Design. Brown: Human-Computer
Interface Design Guidelines. Esitiedot: Ohjelmoinnin
peruskurssi.</PARACONT
></PARA
>

...

<STRUCLVL><HEAD><WPLOC wp-addr="49">
<PARA SPACE-BEFORE="12" SPACE-AFTER="3" KEEP-WITH-NEXT=1 FONT-
FAMILY="Arial" PARATYPE="heading 3"
><PARACONT>Opetusajankohdat:</PARACONT
></PARA
></HEAD
><WPLOC wp-addr="50">
<PARA FONT-SIZE="10" LINE-SPACING="12" PARATYPE="Normal"
><PARACONT></PARACONT
></PARA
><WPLOC wp-addr="51">
<PARA FIRST-INDENT="-18" LEFT-INDENT="18" PARATYPE="Normal"
><PARACONT><AUTOGEN>&bull;<tab
></AUTOGEN>Luennot: 7.9.99 alkaen tiistaisin (M3) ja torstaisin (M2) klo
16-18 (Markku Tukiainen)<CLF FONT-WEIGHT="Bold" ></CLF
></PARACONT></PARA><PARA CONTINUE="1" PRECED-HARD-RETURNS="1" FIRST-
INDENT="-18" LEFT-INDENT="18" PARATYPE="Normal"  ><PARACONT><CLF FONT-
WEIGHT="Bold" >Poikkeukset:</CLF
></PARACONT
></PARA
><WPLOC wp-addr="52">
<PARA FIRST-INDENT="-18" LEFT-INDENT="83" PARATYPE="Normal"
><PARACONT><AUTOGEN>&bull;<tab
></AUTOGEN>Tiistaina 5.10. ei ole luentoa liikuntap&auml;iv&auml;n
vuoksi</PARACONT
></PARA
><WPLOC wp-addr="53">
<PARA FIRST-INDENT="-18" LEFT-INDENT="83" PARATYPE="Normal"
><PARACONT><AUTOGEN>&bull;<tab
></AUTOGEN>Torstaina 16.9. ei ole luentoa. Sen tilalla pidet&auml;&auml;n
luento tiistaina 16.11. klo 16-18, M3</PARACONT
></PARA
><WPLOC wp-addr="54">
```

```
<PARA FIRST-INDENT="-18" LEFT-INDENT="18" PARATYPE="Normal"
><PARACONT><AUTOGEN>&bull;<tab
></AUTOGEN>Harjoitusryhm&auml;t: keskiviikkoisin klo 16-18 (ryhm&auml; 1)
ja perjantaisin klo 12-14 (ryhm&auml; 2) M13</PARACONT></PARA><PARA
CONTINUE="1" PRECED-HARD-RETURNS="1" FIRST-INDENT="-18" LEFT-INDENT="18"
PARATYPE="Normal"  ><PARACONT>Poikkeukset:</PARACONT
></PARA
><WPLOC wp-addr="55">
<PARA FIRST-INDENT="-18" LEFT-INDENT="83" PARATYPE="Normal"
><PARACONT><AUTOGEN>&bull;<tab
></AUTOGEN>--</PARACONT
></PARA
><WPLOC wp-addr="56">
<PARA FIRST-INDENT="-18" LEFT-INDENT="18" PARATYPE="Normal"
><PARACONT><AUTOGEN>&bull;<tab
></AUTOGEN>V&auml;likokeet:</PARACONT
></PARA
><WPLOC wp-addr="57">
<PARA FIRST-INDENT="18" PARATYPE="Normal" ><PARACONT><CLF FONT-
WEIGHT="Bold" >HUOM! HUOM! V&auml;likoeaika ja paikka muuttuneet: TORSTAI
14.10.99 16:00 - 18:00 sali M2</CLF
></PARACONT
></PARA
><WPLOC wp-addr="58">
<PARA FIRST-INDENT="18" PARATYPE="Normal" ><PARACONT>ja tiistaina
23.11.99 klo 8-10, M1</PARACONT
></PARA
><WPLOC wp-addr="59">
<PARA FIRST-INDENT="-18" LEFT-INDENT="83" PARATYPE="Normal"
><PARACONT><AUTOGEN>&bull;<tab
></AUTOGEN>Ensimm&auml;iseen v&auml;likokeeseen tulee luentorungon luvut
1-4 sek&auml; harjoitukset 1-4. P&auml;&auml;paino on
luentorungolla.</PARACONT
></PARA
><WPLOC wp-addr="60">
<PARA PARATYPE="Normal" ><PARACONT></PARACONT
></PARA
><WPLOC wp-addr="61">
<PARA PARATYPE="Normal" ><PARACONT>Kurssin suorittaminen
v&auml;likokeilla edellytt&auml;&auml;, ett&auml; 1/3
harjoitusteht&auml;vist&auml; on tehty (max 50, tehty 17).</PARACONT
></PARA
><WPLOC wp-addr="62">
<PARA PARATYPE="Normal" ><PARACONT></PARACONT
></PARA
><WPLOC wp-addr="63">
<PARA PARATYPE="Normal" ><PARACONT>P&auml;ivitetty viimeksi:
15.10.1999</PARACONT
></PARA
><WPLOC wp-addr="64">
<PARA PARATYPE="Normal" ><PARACONT>mtuki@cs.joensuu.fi</PARACONT
></PARA
><WPLOC wp-addr="65">
<PARA PARATYPE="Normal" ><PARACONT></PARACONT
></PARA
></STRUCLVL
></STRUCLVL
></STRUCLVL
></DOC
></RAINBOW
>
```

Tietojärjestelmien dokumentointi -kurssin sivu sateenkaari-DTD:n mukaisesti merkattuna

```
<!DOCTYPE rainbow PUBLIC "-//EBT//DTD Rainbow 2.5//EN" [
 <!ENTITY fgraph001.fri SYSTEM "graph001.fri" NDATA fri>
]>
<rainbow>
<fileinfo origin="MIF-4.0" dtdver="2.5">
<styinfo>
<paratype name="#DEFAULT-PARATYPE"
      left-indent="0"
      first-indent="0"
      right-indent="0"
      justification="FullJust"
      line-spacing="13"
      space-before="0"
      space-after="10"
      pg-brk-before="0"
      col-brk-before="0"
      charset="iso8859-1"
      font-family="Courier"
      font-size="10"
      font-weight="Medium"
      font-slant="Roman"
      vertical-offset="0"
      foreground="0"
      arch-type=""
      arch-type-info=""
>
<paratype name="Body"
      left-indent="0"
      first-indent="0"
      right-indent="0"
      justification="FullJust"
      line-spacing="14"
      space-before="0"
      space-after="0"
      pg-brk-before="0"
      col-brk-before="0"
      charset="MIFSTANDARD"
      font-family="Times"
      font-size="12"
      font-weight="Medium"
      font-slant="Roman"
      vertical-offset="0"
      foreground="0"
      arch-type=""
      arch-type-info=""
>

...

</styinfo>
<doc>
<para paratype="Body"      left-indent="0"
      first-indent="0"
      right-indent="0"
      justification="FullJust"
      line-spacing="14"
      space-before="0"
```

```
      space-after="0"
><paracont>
</paracont></para>
<para paratype="Body"><paracont>
</paracont></para>
<para paratype="Heading1"><paracont>
<graphic filename="fgraph001.fri"></graphic>
</paracont></para>
<para paratype="Heading1"><paracont>
Tietoj&auml;rjestelmien dokumentoiminen (2 ov) 173207 </paracont></para>
<para paratype="Body"><paracont>
</paracont></para>
<para paratype="Body"><paracont>
Kurssi on pakollinen kurssi ohjelmistotuotannon linjalla, muilla
linjoilla kurssi k&auml;y valinnaiseksi kurssiksi cum laude opintoihin.
</paracont></para>
<para paratype="Body"><paracont>
</paracont></para>
<para paratype="Body"><paracont>
Kurssin sis&auml;lt&ouml;: </paracont></para>
<para paratype="Body"><paracont>
</paracont></para>
<para paratype="Bulleted"><paracont>
<autogen>&bull;<tab></autogen>Dokumenttien rakenteeseen vaikuttavat
tekij&auml;t </paracont></para>
<para paratype="Bulleted"><paracont>
<autogen>&bull;<tab></autogen>Eri dokumenttien sis&auml;ll&ouml;t
</paracont></para>
<para paratype="Bulleted"><paracont>
<autogen>&bull;<tab></autogen>Dokumenttien laatiminen </paracont></para>
<para paratype="Body"><paracont>
</paracont></para>
<para paratype="Body"><paracont>
Tavoitteet: </paracont></para>
<para paratype="Body"><paracont>
</paracont></para>
<para paratype="Bulleted"><paracont>
<autogen>&bull;<tab></autogen>Erilaisiin dokumentteihin tarvittavat asiat
</paracont></para>
<para paratype="Bulleted"><paracont>
<autogen>&bull;<tab></autogen>Dokumentoinnin oleellisten asioiden
ymm&auml;rt&auml;minen </paracont></para>
<para paratype="Body"><paracont>
</paracont></para>

...

<para paratype="Body"><paracont>
Kurssikuulustelu <clf      font-weight="Bold"
>torstaina 16.12.1999 klo 12 &truehy; 14 sali M1 </clf></paracont></para>
<para paratype="Body"><paracont>
</paracont></para>
<para paratype="Body"><paracont>
Loppukuulustelut: </paracont></para>
<para paratype="Body"><paracont>
</paracont></para>
<para paratype="Bulleted"><paracont>
<autogen>&bull;<tab></autogen>21.1.2000, </paracont></para>
<para paratype="Bulleted"><paracont>
<autogen>&bull;<tab></autogen>31.3.2000, </paracont></para>
<para paratype="Bulleted"><paracont>
```

```
<autogen>&bull;<tab></autogen>14.6.2000 </paracont></para>
<para paratype="Body"><paracont>
</paracont></para>
<para paratype="Body"><paracont>
P&auml;ivitetty viimeksi: 17.11.1999</paracont></para>
<para paratype="Body"><paracont>
vouti@cs.joensuu.fi </paracont></para>
<para paratype="Body"><paracont>
</paracont>
</para>
</doc>
</rainbow>
```