Lappeenrannan teknillinen yliopisto
Teknistaloudellinen tiedekunta
Tietotekniikan koulutusohjelma

Diplomityö

XML-TEKNIIKOIHIN PERUSTUVA AJURIMALLI LABVIEW-POHJAISEEN TESTAUSYMPÄRISTÖÖN

Työn tarkastajat ovat Professori Jari Porras ja FM Kristian Väisänen
Työn ohjaajat ovat Professori Jari Porras ja FM Kristian Väisänen

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
Teknistaloudellinen tiedekunta
Tietotekniikan koulutusohjelma

Jukka Mäkelä

XML-tekniikoihin perustuva ajurimalli LabView-pohjaiseen testausympäristöön

Diplomityö

2010

80 sivua ja 29 kuvaa

Tarkastajat: Professori Jari Porras
FM Kristian Väisänen

Hakusanat: Ajurimalli, LabView, TestStand, XML, XML-skeema
Keywords: Driver model, LabView, TestStand, XML, XML-schema

XML-muotoista tiedonesitystapaa hyödynnetään yhä enemmän esitettäessä rakenteellista tietoa. Tarkoituksena on antaa yleishyödyllinen ja uudelleenkäytettävä tapa jakaa yleistä tietoa erilaisten rajapintojen yli. XML-tekniikoita käytetään myös korjaamaan aiemmin tehdyissä sovellutuksissa esiintyneitä puutteita ja parantamaan niiden toimintaa.

Tässä diplomityössä esitellään Telestelle LabView-pohjaiseen testaussovellusympäristöön suunniteltava ajuriuudistus. Työssä paranneltiin aiempaa ajurimallia soveltamalla siihen XML-tekniikoita hyödyntäviä toimintoja. Tarkoituksena oli vähentää testaussovelluskehityksessä vaadittavaa ohjelmointityötä korvaamalla sovelluksiin kovakoodatut ominaisuudet XML-pohjaisilla konfiguraatiotiedostoilla.

Järjestelmän pohjana on yleiskäyttöinen ajuri, joka käyttää Telesten omaa EMS-protokollaa kommunikoinnissaan testattavien tuotteiden kanssa. Ajurimalli käyttää XML-pohjaisia konfiguraatiotiedostoja määrittelemään testattavien tuotteiden ominaisuuksia. XML-skeematiedostoilla esitetään ajurin käyttämän kommunikaatioprotokollan viestityypit ja niiden rakenteet.

Työn tuloksena onnistuttiin luomaan uudenlainen XML-tekniikoita hyödyntävä ajurimalli. Yhteen yhteiseen ajuriin perustuva malli yhdenmukaistaa testaussovelluksien toteuttamista ja vähentää tarvittavaa ohjelmointityötä. Ajurin käyttöä helpotettiin toteuttamalla testaussovelluksien kehitysympäristöön erityinen editori, jolla voidaan helposti luoda ajuria käyttäviä toimintoja.

ABSTRACT

Lappeenranta University of Technology
Faculty of Technology Management
Degree Program of Information Technology

Jukka Mäkelä

XML-techniques Utilizing Driver Model for LabView-based Testing Environment

Master's Thesis

2010

80 pages and 29 figures

Examiners: Professor Jari Porras
M.Sc. Kristian Väisänen

Keywords: Driver model, LabView, TestStand, XML, XML-Schema,

XML-based format is more commonly utilized to represent structured data. The aim is to give reusable and compatible way to share common data over different interfaces. Many XML-technologies based solutions have been created for existing applications to fix their known fouls and improve their overall functionality.

This master's thesis introduces a driver development process for Teleste's LabView-based testing software environment. This thesis improves the earlier driver model by applying XML-techniques utilizing functions to it. The purpose was to reduce the amount of programming work needed in testing software's development by replacing hardcoded properties with XML-based configuration files.

The core of the system is broadly utilized driver that communicates with Teleste manufactured devices by using Teleste's own EMS-protocol. Driver model uses XML-based configuration files to define the differences in the properties of the tested products. XML-schema files are used to represent EMS-protocol's message types and packet formations to the driver.

The result of this thesis was the new kind of XML-based techniques utilizing driver model. To one shared driver based model uniforms the creation of the testing applications by reducing additional programming work. In the testing software's development environment the usage of the driver was eased by developing a special editor, which can be used to create driver based functions.

ALKUSANAT

Kiitän Jarno Fahlbomia ja Kristian Väisästä diplomityön aiheesta. Kiitän myös Telesten MET-osaston muuta henkilökuntaa kaikesta avusta ja yhteistyöstä.

Kiitän työn ohjaajina toimineita Professori Jari Porrasta ja Kristian Väisästä diplomityön tekoon saamastani tuesta ja ohjauksesta.

Lisäksi kiitän niitä, joilla riitti kärsivällisyyttä kannustaa ja potkia minua eteenpäin tässä lähes epätoivoisen hitaasti edenneessä projektissa. Tästä suurimman kiitoksen ansaitsee erityisesti avovaimoni Laura.

Lappeenrannassa 21.05.2009

Jukka Mäkelä

SISÄLLYSLUETTELO

1 JOHDANTO........5
1.1 Tausta........5
1.2 Tavoitteet ja rajaukset........6
1.3 Työn rakenne........7
2 XML-TEKNIIKAT JA NIIDEN SOVELLUKSET........8
2.1 Rakenteen perusteet........9
2.2 XML-skeema........12
2.3 XSL-teknologiaperhe........16
2.4 XML-prosessointimallit........17
2.5 XML-protokollaa hyödyntävät sovellukset........19
2.5.1 Protokollien mallinnus ja kapselointi........21
2.5.2 XML-pohjaiset verkonhallintaratkaisut........25
3 TELESTEN NYKYINEN AJURIYMPÄRISTÖ........28
3.1 Ohjelmointirajapinta........28
3.1.1 LabView G........28
3.1.2 TestStand........30
3.2 Telesten elementtienhallintaprotokolla........32
3.2.1 EMS-pakettirakenne........32
3.2.2 EMS-viestityypit........34
3.3 Nykyinen ajuri........36
3.3.1 INI-konfiguraatiotiedostot........37
3.3.2 Ajurikutsu TDriverCall........38
3.3.3 Nykyisen ajurimallin puutteet ja parannuskohteet........38
4 XML-POHJAINEN AJURIMALLI........40
4.1 Ajurin toiminnot........41
4.1.1 EMS-viestin muodostaminen ajurissa........42
4.1.2 Kommunikointi kohdelaitteen kanssa........46
4.1.3 Vastausviestin käsittely........48
4.2 Ajurin tarvitsemat tukitiedostot........50
4.2.1 EMS-viestien kuvaus käyttäen skeematiedostoja........52

4.2.2 Tuotekohtaiset konfiguraatiotiedostot ... 56
4.2.3 Parametritiedostot ... 57
4.3 Ajurin suorituskyvyn optimoiminen ... 58
4.4 XML-ajurimallin TestStand-steppityyppi ... 60
4.4.1 Steppityypin määrittely ... 60
4.4.2 Toteutetut editorit ... 61
4.4.2.1 TestStep-editori ... 62
4.4.2.2 Parametrieditori ... 65
4.4.2.3 Mittausparametrieditori ... 66
5 JOHTOPÄÄTÖKSET ... 68
5.1 Työn tulokset ... 68
5.2 Ajurin jatkokehittäminen ... 69
5.2.1 Tuki GPIB-pohjaisille mittalaiteille ... 70
5.2.2 Uusi LabView-versio ... 71
LÄHTEET ... 72

KÄYTETYT MERKINNÄT JA LYHENTEET

API	Application Programming Interface
ASN.1	Abstract Syntax Notation One
BEEP	Blocks Extensible Exchange Protocol
DOM	Document Object Model
DTD	Document Type Definition
DVX	Telesten laitteissa käytettävä tietoliikenneväylä
EEPROM	Electronically Erasable Programmable Read-Only Memory
EMS	Element Management System
GPIB	General Purpose Interface Bus
HFC	Hybrid Fiber Coax
HMS	Hybrid Management Sub-layer Subcommittee
HTML	Hyper-Text Markup Language
IP	Internet Protocol
MET	Manufacturing Engineering & Technology
MSXML	Microsoftin jäsennin XML-dokumenttien hallintaan
NetPDL	Network Protocol Description Language
OSI	Open Systems Interconnection referenssimalli
RPC	Remote Procedure Call
SAX	Simple API for XML
SGML	Standard Generalized Markup Language
SNMP	Simple Network Management Protocol
StAX	Streaming API for XML
TCP	Transmission Control Protocol
TSEMP	Teleste Element Management Protocol
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USB	Universal Serial Bus
VTD	Virtual Token Descriptor
W3C	World Wide Web Consortium

WWW	World Wide Web
XML	eXtensible Markup Language
XMPF	XML-based Multi Protocol Framework
XMPL	XML-based Multi Protocol Language
XMPL-DS	XMPL Data Structure Specification
XMPL-LS	XMPL Logic Specification
XMPL-MS	XMPL Message-set Specification
XMPP	Extensible Messaging and Presence Protocol
XPath	XML Path Language
XSD	XML Schema Description
XSL	eXtensible Stylesheet Language
XSLT	XSL Transformations
XSL-FO	XSL Formatting Objects

1 JOHDANTO

XML-protokollaa (eXtensible Markup Language) on monin tavoin hyödynnetty erityyppisten tietojen tallentamisessa ja esittämisessä. Näin on pyritty pääsemään riippumattomuuteen ja monipuoliseen käytettävyyteen. XML-tekniikoita on ryhdytty soveltamaan myös sellaisiin tiedon tallennus ja esittämistekniikoihin, joissa alun perin on ollut käytössä muunlainen tapa. Tavoitteena on ollut parantaa vanhoissa tekniikoissa olleita puutteita ja hyödyntää vanhojen tekniikoiden hyviä puolia uusissa käyttöympäristöissä. Useimmissa sovellutuksissa pyritäänkin esittämään jotain yhteistä jaettavaa tietoa XML-muotoilluissa dokumenteissa. Siten ne ovat helposti tulkattavissa ja käytettävissä myös muissa sovelluksissa. Tästä esimerkkinä voidaan mainita pikaviestinsovellusten keskinäisen yhteensopivuuden takaavaksi viestinvälitystekniikaksi kehitelty XMPP (Extensible Messaging and Presence Protocol) [Saint-Andre 2004].

1.1 Tausta

Tässä diplomityössä esitellään ajuriuudistus, jolla Telesten käyttämää testausympäristöä ja siihen liittyvä sovelluskehitystä pyritään parantamaan helppokäyttöisemmäksi ja tehokkaammaksi. Teleste Oyj on kansainvälinen teknologiaryhmittymä, joka on erikoistunut laajakaistaisiin tietoliikennejärjestelmiin ja palveluihin. Yhtiö on perustettu vuonna 1954 ja sen pääpaikka sijaitsee Kaarinassa. Telestellä on tuotantolaitokset Suomessa ja Kiinassa sekä tuotekehitysyksiköt Suomessa ja Puolassa [Teleste 2009, s.5]. Diplomityö tehtiin Telesten MET-osastolla. (Manufacturing Engineering & Technology) Kaarinassa. Osasto on vastuussa tuotannossa käytettävistä testaussovelluksista ja –laitteistoista. Työn tavoitteena oli toteuttaa uudenlainen XML-pohjainen ajurimalli testaussovelluksissa käytettäväksi.

Tuotannon testaussuunnittelijoiden tehtävänä on tuottaa laiteajurit ja mittausohjelmistot testausympäristön sovelluksille. Telesten valmistamia tietoliikennetuot-

teita hallitaan käyttäen Telesten omaa elementtien hallintaprotokollaa. Telesten elementtienhallintaprotokolla sisältää komentoja, joita kaikkien tuotteiden pitää osata käyttää. Tuotteet toteuttavat lisäksi omia tuoteperhekohtaisia komentoja käyttötarkoituksen mukaan. Tuotekohtaisten komentojen rakenne on pääpiirteittäin sama kaikille tuotteille, mutta komentojen parametrit vaihtelevat.

[Raitis 2002] on diplomityössään määritellyt yhdeksi Telesten toimituskykyä parantavaksi osatekijäksi yleiskäyttöisten ohjelmakomponenttien kehittämisen. MET-osaston toteuttamassa sovelluskehityksessä on jo tehty joitakin toimia kohti yleiskäyttöisempää ajurimallia. Ajureita on kuitenkin edelleen useita erilaisia eikä testaussovellusten toteuttaminen ole kovinkaan paljoa helpompaa kuin ennen.

1.2 Tavoitteet ja rajaukset

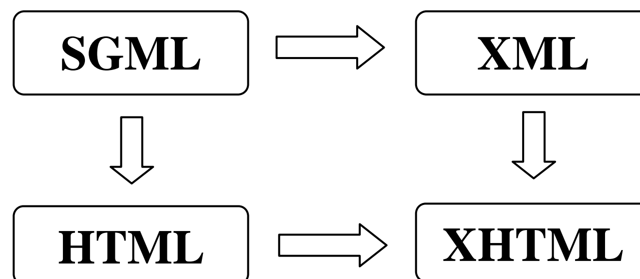
Tämän diplomityön tarkoituksena on helpottaa testaussovellusten toteuttamista kehittämällä yleiskäyttöinen ajurimalli, jonka toiminnallisuuden pohjana on XML-tekniikoita hyödyntävät tukitiedostot. Tavoitteena on, että Telesten testausympäristössä käytettävä laiteajuri olisi kaikille sama ja tuotekohtaiset eroavuudet määriteltäisiin ajurille XML-muotoisilla konfiguraatiotiedostoilla. Ajuria tullaan käyttämään NI:n (National Instruments) valmistamassa testaussovellusten TestStand-kehitysympäristössä. Ajurin hyödyntämistä varten TestStand:n kehitetään erillinen editori, jonka avulla voidaan määritellä testaussovellukseen liitettävän ajurikutsun toiminta esimerkiksi suoritettavan komennon ja noudettavien parametrien osalta. Ajurin käyttökohteena ovat ensisijaisesti Telesten omaa kommunikointiprotokollaa käyttävät tuotteet.

1.3 Työn rakenne

Toisessa luvussa käsitellään XML-protokollan perusteita ja sen käyttöön liittyviä standardeja. Samalla esitellään protokollaa hyödyntäviä menetelmiä ja tekniikoita, joilla on pyritty toteuttamaan diplomityön aiheen puitteissa vastaavanlaisia toimintoja. Lisäksi tutustutaan XML-dokumenttien hallintametodeihin, joilla helpotetaan XML-muotoisen tiedon käsittelyä, niin muotoilun, kuin tiedon rajaamisen yhteydessä. Kolmannessa luvussa esitellään Telesten nykyinen testausympäristö niin testattavien laitteiden kommunikoinnissa käytettävän tiedonsiirtoprotokollan, kuin testaussovellusten pohjana olevien työkalujen ja ohjelmointirajapinnan osalta. Lisäksi tutustutaan nykyiseen ajurimalliin esittelemällä sen puutteita ja kehitysmahdollisuuksia. Neljännessä luvussa käsitellään uutta XML-pohjaista ajurimallia esittelemällä sen ominaisuuksia ja toiminnallisuuden pääperiaatteita. Ajurimallin lisäksi esitellään ajurin käyttöä helpottava editori, jonka avulla testausympäristöön voidaan helposti määritellä ajuria käyttävä testitapahtuma. Lopuksi kootaan johtopäätökset tehdystä työstä ja esitellään uusia mahdollisia käyttötarkoituksia potentiaalisten jatkokehitystarpeiden muodossa.

2 XML-TEKNIIKAT JA NIIDEN SOVELLUKSET

XML (Extensible Markup Language) on W3C (World Wide Web Consortium) standardoima tiedon esittämiseen ja tallentamiseen tarkoitettu tekstiformaatti. Formaatin juuret ovat SGML:ssä (Standard Generalized Markup Language), joka oli aikaisempi standardi määrittelemään laite- ja järjestelmäriippumattomia metodeja tekstimuotoisen informaation esittämiseksi elektronisesti. Käytännönläheisempi johdannainen SGML-standardista on Internetin WWW-sivuilla (World Wide Web) yleisesti käytetty HTML (HyperText Markup Language). Edellä mainitut teknologiat yhdistää XHTML (eXtensible HTML), joka pyrkii olemaan XML-muotoista syntaksia noudattava HTML. Kuvassa Kuva 2-1 esitellään rakenteellisen tiedon esittämiseen käytetyt standardien ja niiden väliset suhteet toisiinsa [Geroimenko et al. 2003, s.5].



Kuva 2-1 Standardien väliset suhteet [Geroimenko et al. 2003, s.5]

[Geroimenko et al. 2003, s.5] pitävät XML-protokollaa mullistavana käyttötapana tulevaisuuden web-teknologioihin ja luettelevat sen hyviksi puoliksi seuraavat ominaisuudet:

- XML on avoin valmistajariippumaton standardi, jonka käyttöä suosivat kaikki suurimmat ohjelmistotalot ja markkinajohtajat.
- XML on tekstimuodossa, jolloin kaikki merkittävät ohjelmistot ja laitteistot pystyvät prosessoimaan sitä. Lisäksi erilaisten merkistökoodauksien käyttömahdollisuus tekee siitä monikäyttöisen yli kansallisten ja kulttuurillisten rajojen.

- XML erittelee sisällön XML-dokumentin rakenteesta, jolloin se on käytettävissä monissa eri formaateissa käytettävästä ympäristöstä riippumatta.
- XML sisältää myös sen sisältöä kuvaavaa informaatiota metadatan muodossa, joten sen sisältö on tulkittavissa automaattisen tietojenkäsittely metodein.
- XML on web-ystävällinen ja dataorientoitunut, jolloin se on helposti yhdistettävissä perinteisiin tietolähteisiin, kuten tietokantoihin, tekstidokumentteihin ja web-sivuihin.

XML-dokumentin rakenne voidaan määritellä sovellettavan teeman mukaan, jolloin eri elementit voidaan nimetä omavaltaisesti kyseisen teeman mukaan ja elementtien väliset suhteet voidaan valita mahdollisimman tehokkaasti. [Brandin 2003, s.4] esittelee kolme ominaisuutta, jotka tekevät XML-protokollasta käytettävyydeltään toimivan teknologian:

- ***Heterogeenisyys***: Jokainen tietoelementti voi sisältää erityyppisiä tietorakenteita. Oikean maailman tapahtumat eivät suoraan ole esitettävissä siisteinä taulukon riveinä ja sarakkeina. Suuri etu saadaan esitettäessä tietoa ilman rajoitteita.
- ***Laajennettavuus***: Uusien tietotyyppien lisääminen ilman ennalta tapahtuvaa määrittelyä, joka enemmänkin kannustaa tekemään uusia muutoksia staattisuuden sijaan.
- ***Joustavuus***: Tietueet voivat erota toisistaan koon ja määrittelyn puolesta instanssista toiseen. XML ei aseta rajoituksia talletettavalle tiedolle, sillä jokainen tietoelementti voi olla niin pitkä tai lyhyt kuin on tarpeellista.

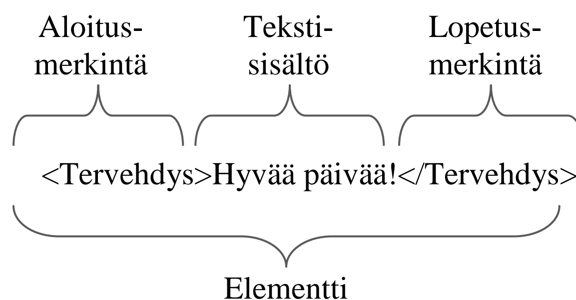
2.1 Rakenteen perusteet

[W3C 2008] mukaan XML-dokumentissa on niin loogisia, kuin fyysisiäkin rakenteita. Fyysisesti dokumentti koostuu entiteeteistä, jotka ovat tiedon tallentamiseen käytettyjä rakenteita. Loogisesti dokumentti koostuu määrittelyistä, elementeistä,

kommenteista, merkintäviittauksista ja prosessointiohjeista, joilla kaikilla on erityinen merkitsemistapansa. Loogisten ja fyysisten rakenteiden pitää asetettua sisäkkäin siten, että ne noudattavat hyvin muodostetun dokumentin ominaisuuksia. XML-dokumentti on hyvin muodostettu seuraavien perusteiden mukaisesti:

- Se sisältää täsmälleen yhden juuri-elementin.
- Se vastaa XML-määrittelyn asettamia laadukkaan dokumentin rajoituksia.
- Sen jokainen jäsennetty itsenäinen kokonaisuus, johon dokumentissa viitataan, on hyvin muodostettu.

XML-dokumentin perusosa on elementti. Elementillä on sitä kuvaava nimi, joka on sijoitettu ”<” ja ”>” -merkkien väliin. Elementeillä voi olla lisäksi attribuutteja, joilla voidaan selventää kyseisen elementin ominaisuuksia kuten esimerkiksi indeksoida elementin järjestysnumeroa. Jokainen elementti vaatii erillisen merkinnän, jolla osoitetaan elementin vaikutusalue päättyneeksi. Lopetuselementti on samanniminen kuin aloituselementti, mutta sen eteen on asetettu sulkemisen määrittelevä /-merkki. Mikäli elementillä ei ole lapsielementtejä, eli hierarkiassa sen alapuolella olevia elementtejä, voidaan /-merkki asettaa aloituselementin nimen perään. Kuvassa Kuva 2-2 on esitelty tarkemmin elementin osat aloitus- ja lopetusmerkintöineen.



Kuva 2-2 XML-elementin osat

Elementit pitää olla sijoiteltuna sisäkkäin siten, etteivät ne mene toistensa päälle. Toisin sanoen ylemmän tason loppuelementti ei voi olla lapsielementin sisällä. Kuvassa Kuva 2-3 on esitetty virheellinen XML-dokumentin rakenne, jossa elementit ovat väärässä järjestyksessä.

```
<elementti1>
    <elementti2>Teksti</elementti1>
</elementti2>
```

Kuva 2-3 Virheellinen elementtien järjestys

Kuvassa Kuva 2-4 on esitetty samaisilla elementeillä muodostettu oikeanlainen rakenne.

```
<elementti1>
        <elementti2>Teksti</elementti2>
</elementti1>
```

Kuva 2-4 Elementit oikeassa sisäkkäisessä järjestykessä

Seuraavassa kuvassa Kuva 2-5 on esitelty yksinkertainen XML-dokumentti joka pitää sisällään kuvitteellisen kirjahyllyn sisällön.

```
<?xml version=”1.0” encoding=”utf-8” ?>
<Kirjahylly xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=“Kirjahylly.xsd”>
    <Kirjat>
        <Kirja id=1>
            <Nimi>ABC-kirja</Nimi>
            <Kirjoittaja>Mikael Agricola</Kirjoittaja>
            <Julkaisuvuosi>1543</Julkaisuvuosi>
        </Kirja>
        <Kirja id=2>
            <Nimi>1984</Nimi>
            <Kirjoittaja>George Orwell</Kirjoittaja>
            <Julkaisuvuosi>1949</Julkaisuvuosi>
        </Kirja>
    </Kirjat>
</Kirjahylly>
```

Kuva 2-5 Esimerkki XML-dokumentista

Hyvin muodostettu XML-dokumentti alkaa aina XML-märittelyllä, jonka tunnisteena on prosessointiohjeisiin sisällytetty <?xml?>-elementti. XML-määrittelyssä

esitetään attribuutteina käytetty XML-protokollaversion numero ja mahdollisesti käytettävä merkistökoodaus. Lisäksi voidaan välittää XML-dokumentin käsittelijälle ohjeita dokumentin esittämiseksi. Jokaisessa hyvin muodostetussa XML-dokumentissa on prosessointiohjeiden jälkeen yksi ylin elementti, jonka alaisuudessa kaikki muut elementit sijaitsevat. Tätä kutsutaan XML-dokumentin juurielementiksi. Esimerkkikuvassa juurielementtinä toimii Kirjahylly-niminen elementti.

Kirjahyllyssä on kaksi kirjaa, joista on tallennettu niiden nimet, kirjoittajat ja julkaisuvuodet. Tietokanta on yksinkertainen ja selvästi ihmisten luettavissa. Elementtien nimeämisessä on pyritty siihen, että rakenne selittäisi mahdollisimman paljon itseään. Elementit ovat nimetty siten, että ne kuvaavat hyvin omaa sisältöään ja suhdetta muihin elementteihin. Mikäli kuvitteelliseen kirjahyllyyn lisättäisiin esimerkiksi cd-levyjä, muutettaisiin kirjahyllyn sisältöä kuvaavaa XML-dokumenttia siten, että </kirjat>-lopetuselementin jälkeen lisättäisiin cd-levyille samankaltainen elementti kuin kirjoilla on jo olemassa. Tämän elementin sisään voidaan lisätä esimerkiksi levyn nimi, julkaisuvuosi ja tyylilaji omine elementteineen.

2.2 XML-skeema

XML-dokumenttien rakennekuvauksilla määritellään ennalta suunnitellut rakenteet, joita dokumenttien on noudatettava [Walkama & Laakkonen 2004, s.3]. Rakenne ei saa olla omavaltainen eikä satunnaisesti muotoiltu. Walkama & Laakkonen luettelevat myös muutamia toisistaan poikkeavia näkökohtia rakennekuvauksille:

- Rakennekuvauksien avulla voidaan tarkistaa, että ilmentymädokumenttien sisältö vastaa annettuja kuvauksia.
- Skeemakuvaukset voivat sisältää dokumentin rakennekuvauksen lisäksi myös tietoa sen eri osien suhteista toisiinsa.

- XML-dokumentteja voidaan myös pitää eräänlaisina palomuureina, jotka suojelevat dokumentteja käsitteleviä sovelluksia sisällöltä, joita ne eivät osaa käsitellä.
- Erilaisten sanastojen ja uusien kielten määritteleminen on myös tärkeä tehtävä XML-dokumenttien rakennekuvauksille.

Yleisimmin käytettyjä rakennekuvaustekniikoita ovat DTD (Document Type Definition) ja XML-skeema. XML-skeema on kehitetty laajentamaan DTD:n puutteelliseksi havaittuja ominaisuuksia [Walkama & Laakkonen 2004, s.7]. Skeemaa käytetään XML-dokumenttien rakenteen määrittelemiseksi ja muodon oikeellisuuden tarkistamisessa. Skeema kuvaa kohdedokumentin elementtien ja attribuuttien nimet, sekä niiden keskinäiset rakenteelliset suhteet toisiinsa. Samalla voidaan myös määritellä XML-dokumentit sisältämien arvojen tietotyypit erittäin yksityiskohtaisesti.

Vaikka XML-dokumentti noudattaisikin hyvin muodostetun XML-dokumentin yleisiä sääntöjä, se ei kuitenkaan välttämättä ole rakennekuvausten mukainen eli validi. [Walkama & Laakkonen 2004, s.4] esittääkin XML-dokumentin validiudelle seuraavat kriteerit:

- Sen pitää olla hyvin muodostettu
- Siihen on liitetty rakennekuvaus.
- Se noudattaa rakennekuvausta ja muita XML-määrittelyssä asetettuja validiusrajoituksia.

Esimerkkinä käytämme aiemmin esitetyn kirjahyllyn rakenteen määrittelevää skeemaa. Tieto esimerkissä käytetystä skeemasta sijoitetaan XML-dokumentin juurielementin attribuutiksi. Esimerkkiskeema on esitetty kuvassa Kuva 2-6.

```
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema elementFormDefault="qualified"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="Kirjahylly">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:element name="Kirjat">
      <xsd:complexType>
       <xsd:sequence>
        <xsd:element minOccurs="0" maxOccurs="unbounded" name="Kirja">
         <xsd:complexType>
          <xsd:sequence>
           <xsd:element name="Nimi" type="xs:string" />
           <xsd:element name="Kirjoittaja" type="xs:string" />
           <xsd:element name="Julkaisuvuosi" type="xs:gYear" />
          </xsd:sequence>
          <xsd:attribute name="Id" type="xs:ID" />
         </xsd:complexType>
        </xsd:element>
       </xsd:sequence>
      </xsd:complexType>
     </xsd:element>
    </xsd:sequence>
   </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

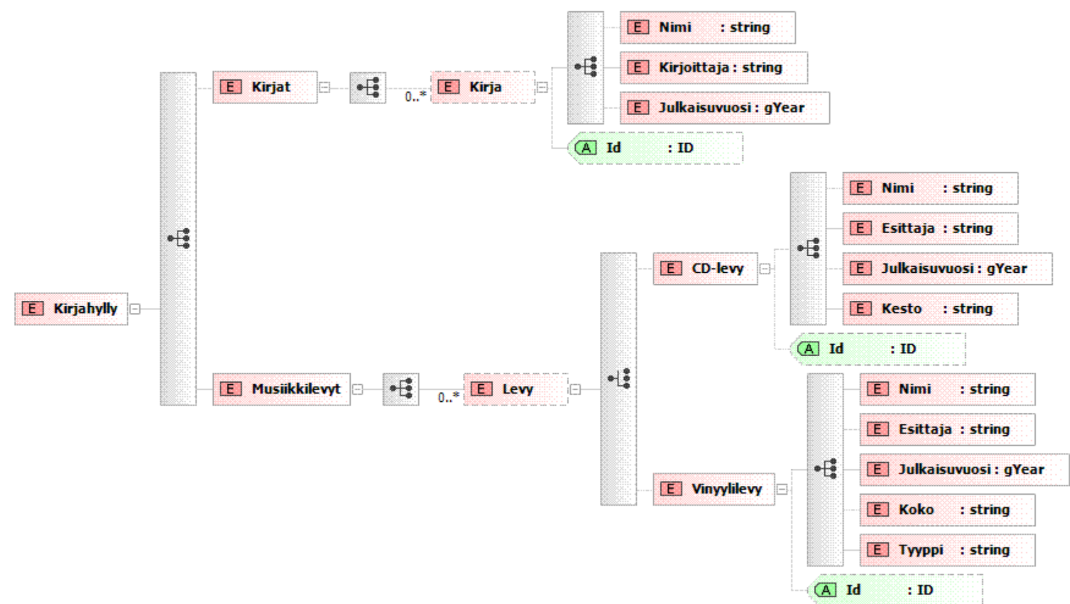
Kuva 2-6 Esimerkki Skeema-dokumentista

Koska skeemadokumentti on normaali XML-tiedosto, se alkaa myös XML-määrittelyllä. Tämän jälkeen skeema määrittelee suoraviivaisesti, että juurielementtinä on Kirjahylly-niminen elementti. Tämän alla voi olla joukko monitasoisen rakenteen omaavia elementtejä, joista nimettynä on Kirjat-niminen elementti. Kyseinen elementti voi skeeman mukaan olla ilman lapsielementtejä tai vaihtoehtoisesti sisältää kirjoja määrittelemättömän lukumäärän. Jokaisella kirjalla on attribuutti Id, joka on yksilöllinen tunnus kirjahyllyn artikkelien erottelemiseksi toisistaan. Jokaisesta kirjasta kerrotaan sen nimi, kirjoittaja ja julkaisuvuosi.

Talletetun datan tietotyyppi on yleensä tekstimuotoista, mutta esimerkiksi vuosiluvulle on määriteltynä oma tietotyyppinsä gYear, jonka avulla varmistetaan, että kyseiseen kohtaan voi laittaa vain vuosiluku-tyyppistä tietoa. Oma erikoistapauksensa on attribuutti ID:n tietotyyppi ID, jonka pitää olla yksilöllinen tunniste. XML-dokumentti ei ole rakenteeltaan oikein muodostettu, jos se sisältää monta samalla ID:llä varustettua samannimistä elementtiä.

Kuten esimerkkiskeemasta ilmeni, complexType-nimisen elementin alaisuudessa olevaa sequence-elementtiä käytetään ryhmittelemään joukko lapsielementtejä toistensa sisarelementeiksi. Vaihtoehtoisesti voitaisiin käyttää choice-nimistä elementtiä määrittelemään valintaa, jossa lopulliseen XML-dokumenttiin voitaisiin valita useammasta vaihtoehdosta yksi elementti ja sen lapsielementit.

Skeemaeditoria apuna käyttäen koko skeeman luomisprosessia pystytään helpottamaan huomattavasti, koska editorissa erilaiset elementit ja niiden suhteet on kuvattu graafisena elementtipuuna. Kuvassa Kuva 2-7 on esitetty aiemmin esitelty skeema graafisen editorin kautta nähtynä.



Kuva 2-7 Kirjahyllyn skeema graafisesti esitettynä

Skeemoja pystytään luomaan ja muokkaamaan tavallisella tekstieditorilla. Skeemadokumentin graafisen esityksen avulla samat tehtävät voidaan tehdä kuitenkin paljon yksinkertaisemmin ja nopeammin. Esimerkiksi elementtien suhteet toisiinsa ovat selvästi näkyvissä ja näkymää voi yksinkertaistaa piilottamalla ylimääräisiä skeemapuun haaroja. Kuvan esimerkkiskeema on tehty Liquid XML Studio -sovelluksen vanhemmalla versiolla, jossa vielä pystyi muodostamaan graafisesti skeemarakenteita. Liquid XML Studion uusimmista kokeiluversioissa kyseinen ominaisuus on kuitenkin jätetty pois ja sen saa käyttöönsä vasta lunastamalla so-

velluksen käyttölisenssin. Maksullisia vaihtoehtoja on paljon ja niistä löytyy paljon hyödyllisiä toimintoja. Myös muita ilmaiseksi käytettäviä sovelluksia on saatavilla, mutta niissä ei välttämättä ole vaadittuja ominaisuuksia sujuvan kehitystyön takaamiseksi.

2.3 XSL-teknologiaperhe

XSL-protokollaperheeseen (Extensible Stylesheet Language) kuuluvia teknologioita käytetään määrittelemään XML-dokumenttien rakenteen muodonmuutosta ja esittämistapaa [W3C 2009]. XSLT-tyylitiedostoja (XSL Transformations) käytetään tiedon muodonmuuttamiseen [W3C 1999a]. XPath-kyselykieltä (XML Path Language) käytetään XML-dokumentin osien osoittamiseen tyylitiedostossa käytettäviä hakulausekkeita hyödyntämällä [W3C 1999b]. Protokollaperheeseen kuuluu vielä lisäksi XSL-FO (XSL Formatting Objects), joka on XML-sanasto määrittelemään muotoilun semantiikkaa ja lopullista XML-dokumentin ulkoasua [W3C 2006].

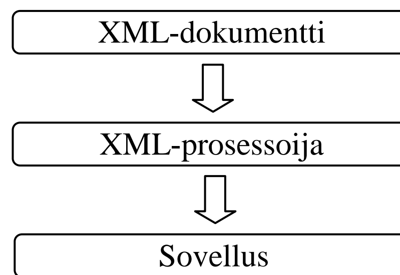
XSLT on tekniikka, jonka avulla voidaan luoda vanhasta XML-dokumentista uusi ilmentymä. Dokumentti muutetaan toiseksi XSLT:n kuvaamilla säännöillä, jotka ovat templaatteja sisältäviä assosioivia malleja. Lähdedokumenttia kuvaavasta puusta löydettyyn malliin sovitetaan muunnoksen tekevä templaatti, jolloin saadaan muodostettua yksi osa muokattua XML-dokumenttia. Templaattien avulla muokatun XML-dokumentin rakenne voi olla täysin poikkeava alkuperäisestä dokumentista. [W3C 1999a]

XPath on melko yksinkertaisia hakulausekkeita käyttävä kyselykieli, jota XSLT käyttää osoittamaan kohdedokumentin osasia. Hakulausekkeet antavat monipuoliset mahdollisuudet löytää erilaisia rakenteita kohdedokumentista. Lisäksi hakulausekkeet antavat perustoiminnot merkkijonojen, numeroiden ja totuusarvomuuttujien manipulointiin. XPathin lauserakenne on syntaksiltaan tiivistä eikä muistuta

XML-tyylistä muotoilua. Sitä voidaan helposti käyttää URI:en (Unicast Resource Identifier) ja XML-attribuuttien yhteydessä. XPath saa nimensä sen tavasta käyttää URL:ien (Unicast Resource Locator) kaltaista hakemistopolkuihin perustuvaa notaatiota osoittaessaan XML-dokumentin hierarkkisen rakenteen eri osia. [W3C 1999b]

2.4 XML-prosessointimallit

[W3C 2008] mukaan odotetaan, että XML-dokumentit käsitellään aina seuraavan mallin mukaisesti, joka on esitetty kuvassa Kuva 2-8. Mallin esittämässä rakenteessa XML-prosessoija välittää XML-dokumentin sitä käyttävälle sovellukselle.



Kuva 2-8 XML-dokumentin prosessointimalli [Jacobs 2006, s.16]

XML:ää prosessoivalla jäsentimellä (parser) voidaan lukea ja manipuloida XML-muotoista tietoa. Jäsennin tulkitsee ja muuntaa dokumentin sellaiseen muotoon, että sitä voidaan käsitellä ohjelmallisesti. Se hyödyntää toiminnassaan API-käyttöliittymiä (Application Programming Interface) päästäkseen käsiksi XML-dokumenttien prosessoinnissa käytettyihin metodeihin ja ominaisuuksiin. API-käyttöliittymissä prosessointityypit voidaan jakaa kahteen eri luokkaan [Jacobs 2006, s.17]: Puurakenteeseen ja tapahtumiin perustuvaan luokkaan.

Puurakenteeseen perustuva ja samalla yleisin käytetyistä rajapinnoista on W3C:n DOM (Document Object Model), joka tarjoaa helposti hyödynnettävän käyttöliittymän HTML- ja XML-dokumenttien käsittelyyn. Toiminta perustuu XML-

dokumenttien esittämiseen erilaisten objektien kautta. Se määrittelee dokumentti-en loogisen rakenteen ja yleiset menetelmät, miten sitä voi käsitellä tai manipuloi-da. DOM esittää XML-dokumentin yhtenä kokonaisena objektipuuna, jonka ra-kenne ja siinä olevien solmujen keskinäiset suhteet ovat selvästi tiedossa ja hel-posti löydettävissä. [W3C 2004]

Suosittu vaihtoehto DOM:lle on SAX (Simple API for XML). Tapahtumiin perus-tuva SAX ei kuitenkaan ole W3C:n suositus standardiksi. Se on esimerkiksi oh-jelmistoyritysten yleisesti käyttämä rajapinta XML-dokumenttien lukemiseksi ja näin ollen vakiinnuttanut asemansa [Jacobs 2006, s.17]. SAX-jäsennin käsittelee sille syötettyä XML-dokumenttia järjestelmällisesti erillisissä osissa ja käynnistää käyttäjän määrittelemän tapahtuman, mikäli haluttu kohta dokumentista löytyy. Jäsennin toimii yksisuuntaisesti eli kerran ohitettuun dokumentin kohtaan ei voida palata aloittamatta jäsennöintiprosessia uudestaan. SAX-jäsentimen rinnalle on kehitetty StAX (Streaming API for XML), joka pyrkii yhdistämään puurakentee-seen ja tapahtumiin perustuvat ominaisuudet. Siinä missä SAX-jäsentimelle työn-netään käsiteltävä XML-data kokonaisuudessaan ja jäsennin testaa jokaisen do-kumentin osan yksitellen, StAX-jäsennin etsii ensin halutun kohdan XML-dokumentistä ja toteuttaa vasta sitten käyttäjän haluaman tapahtuman. StAX:n avulla on mahdollista myös kirjoittaa XML-tiedostoon. toisin kuin SAX:lla.

Aiemmin mainittujen perinteisten menetelmien lisäksi on kehitetty täysin poik-keavia lähestymistapoja XML-dokumenttien käsittelyyn. Esimerkiksi [Zhang et al. 2004] on esitellyt tekniikan nimeltään VDT-XML (Virtual Token Descriptor XML), jonka toiminta perustuu binaariseen XML-dokumentin käsittelyyn. Do-kumentti jaetaan binaarisiin 64 bittisiin osataulukoihin, jotka sisältävät osoittimet dokumentin sisältöön ja sisarelementteihin. Binaarinen XML ei kuitenkaan ole vakiintunut standardi ja siitä jäsennelty tieto ei ole suoraan muiden sovellusten käytettävissä.

[Lam, Ding & Liu 2008, s.34] ovat tehneet vertailua erityyppisten jäsentimien suorituskyvystä. He suorittivat valituille XML-tiedostoille joukon yksinkertaisia

operaatioita ja mittasivat suorituskykyä kuvaavia suureita. Siinä osoitettiin, että jäsennintyypeistä DOM oli toiminnaltaan kaikista raskain käyttäen eniten prosessoriaikaa ja muistia, kun taas SAX/StAX ja VTD-XML olivat resurssienkulutukseltaan säästäväisempiä. Pienellä tiedostokoolla (1-15 kilotavua) VTD-XML oli kapasiteetiltaan viisi kertaa nopeampi kuin DOM ja suurella tiedostokoolla (1-15 megatavua) ero oli jo kahdeksankertainen. SAX/StAX:n kapasiteetti ei ollut riippuvainen käytetystä tiedostokoosta, joten esimerkiksi suuria tiedostoja käsiteltäessä se oli kaikista tehokkain.

XML-jäsentimiä on hyödynnettävissä useampia erilaisia riippuen käytettävästä kehitysympäristöstä. Monissa ohjelmointikielissä on käytettävissä valmiit mekanismit XML-dokumenttien hallintaan joko DOM tai SAX-pohjaisesti. Esimerkiksi Microsoft tarjoaa käyttäväksi MS Windows ympäristöönsä oman monikäyttöisen MSXML-rajapinnan [Microsoft 2010]. Vaikka edellä mainituista jäsentimistä nopein oli VDT-XML, sen käyttöä hankaloittaa esimerkiksi puuttuva rajapinta LabView-ohjelmointiympäristöön, johon tässä työssä käsiteltävä ajurimalli tullaan kehittämään. Myös SAX-pohjaiset jäsentimet olivat tehokkaita, mutta niiden puutteina oli rajoittunut toiminnallisuus. DOM-pohjaisten jäsentimien käyttöä puolustavat niiden laajat käyttömahdollisuudet eri ympäristöissä ja kattavat metodien joukot XML-dokumenttien kokonaisvaltaiseen hallintaan.

2.5 XML-protokollaa hyödyntävät sovellukset

Protokollan mukautuvien ominaisuuksia kautta XML-rakenteina esitettyä tietoa pystytään käyttämään hyvinkin erilaisissa käyttötarkoituksissa. Esimerkiksi [Bagnasco et al. 2002, s.578] ovat esittäneet LabView-pohjaiseen sovellusympäristöön mallin, jolla testitapaukset pystytään esittämään XML-muotoisina konfiguraatiotiedostoina. Testitapaus luodaan valmiista komponenteista WWW-käyttöliittymän avulla. Komponentit ovat LabView -ohjelmointikielellä toteutettuja sovelluksia, jotka käyttöliittymän avulla ryhmitellään yhdeksi kokonaisuudeksi. XML-

tekniikoita hyödyntävän järjestelmän avulla on pyritty saavuttamaan alustariippumattomuutta sekä helpottamaan laajennettavuutta ja kunnossapitoa [Bagnasco et al. 2002, s.584].

[Anderson 2005] on tutkinut XML-pohjaisen tiedon prosessointiin tarvittavaa suorituskykyä erilaisten vertailujen kautta. Lopputuloksena hän on todennut, että XML-muotoisen tiedon käsittelyyn vaaditaan yleensä enemmän prosessointiaikaa. Sen etuna on kuitenkin laaja yhteensopivuus etenkin, jos sitä käytetään erityistoiminnallisten sovelluksien välillä tapahtuvaan pienimuotoiseen tiedonsiirtoon [Anderson 2005, s.200].

Paljon käytetty XML-muotoisen datan käyttökohde on kommunikaatiosovellusten välittämien protokollakohtaisten viestien rakenteen esittäminen, kuten esimerkkinä aiemmin käytetty XMPP tekee. Kommunikaatioprotokollissa on tärkeää, että tiedonsiirrossa välitettävä tieto noudattaa aina yhtenäistä esitystapaa (encoding). Yhteisten pelisääntöjen kautta viestinvälitys on sujuvaa, tehokasta ja sisältää hyvin vähän tai ei ollenkaan yllätyksiä. [Sharp 2008, s.241] määrittelee kolme pääasiallista tapaa esittää protokollan datapaketin sisältö:

1. **Simple binary**– tai ad-hoc–esitystapa, jossa protokollan datapaketissa sijaitsevat kiinteästi määritetyt bittijonot ovat suoraan välitettävien kenttien lukuarvoja omavaltaisessa järjestyksessä.
2. **Type-Length-Value (TLV)** -esitystapa, jossa jokainen lähetettävä lukuarvo on kuvailtu kolmella alikentällä. Ensimmäinen arvo määrittelee välitettävän datan tyypin, toinen arvo määrittelee sen koon ja kolmas arvo on välitettävä lukuarvo itse.
3. **Matched tag** -esitystapa, jossa välitettävä data sisältää sisältöä kuvaavan aloitusmerkinnän ja vastaavan lopetusmerkinnän varsinaisen datan jäädessä merkintöjen keskelle.

Edellä mainitut tavat asettavat kukin erilaiset vaatimukset siirrettävien pakettien sisällön selvittämiseksi ja kommunikoinnin onnistumiseksi. Kiinteästi sijoitettujen bittijonojen tapauksessa paketteja välitettävien tahojen pitää tietää aina, mitä tie-

toa ollaan vastaanottamassa tai lähettämässä. Vain näin voidaan soveltaa staattisia rakennemäärityksiä aina sopivissa yhteyksissä. TLV helpottaa hieman enemmän paketteja käsitelevän tahon toimintaa. Silloin ei välttämättä tarvitse määritellä jokaiselle mahdolliselle pakettirakenteelle omaa tietorakennetta. Käsittelijä pystyy dynaamisesti prosessoimaan siirrettävän paketin, kun se jo oletuksena tietää, että paketin hyötykuormassa on tietyn rakenteen mukaisesti jaoteltua dataa.

Viimeinen vaihtoehto tiedon esitystavaksi vie dynaamisuuden vielä pidemmälle. Siinä paketin rakenne perustuu ennalta sovitulla tavalla nimettyihin elementteihin, joilla erotellaan eri osat toisistaan. Nimetyn elementin perusteella paketin käsittelijä pystyy suorittamaan sille haluamansa toimenpiteet. Esitystavan vaatimuksena on, että käsittelijään on toteutettu toimintalogiikka jokaiselle mahdolliselle elementtityypille.

XML-tekniikoita hyödyntävien protokollasovellusten voidaan katsoa toimivan kahdella tavalla. Ensimmäisenä ovat sovellustasolla toimivat protokollat, jotka esittävät protokollien pakettirakenteet XML-muotoisena. Toisena ovat sovellukset, jotka toimivat kehysrakenteena suoraan yhteystasolla välittäen ylemmän tason protokollia XML-muotoisissa kehyksissä. Useimmissa sovellutuksissa esitetään alemman tason protokollan pakettirakenteet käyttäen XML-tekniikoita. Tällöin toteutuksessa on usein välityspalvelu joka tulkitsee heksadesimaalimuotoiset pakettirakenteet XML-muotoisiksi ja päinvastoin.

2.5.1 Protokollien mallinnus ja kapselointi

Erilaisten verkkoliikennettä seuraavien sovellusten, joiden tehtävinä voi esimerkiksi olla liikenteen reitittäminen, luokittelu tai rajoittaminen, pitää jotenkin tunnistaa välittämänsä pakettiliikenteen sisältö. Olemassa olevat ratkaisut perustuvat kuitenkin ohjelmoinnissa käytettyihin rakenteisiin ja tekniikoihin sekä vaativat aina ohjelmointityötä esimerkiksi uusien protokollien ilmaantuessa. Näin ollen olisikin tarve yleiskäyttöisille sovelluksille, jotka käyttävät toiminnassa jotain

muita menetelmiä, kuin kovakoodattuja pakettirakennemäärittelyjä. Tähän tarpeeseen on pyritty toteuttamaan XML-tekniikoita hyväksi käyttäviä ratkaisuja.

[Risso & Baldi 2006] ovat tutkimustyössään esitelleen NetPDL-merkintäkielen (Network Protocol Description Language), jolla voidaan esittää eri protokollien otsakekenttien rakenteita XML-pohjaisella mallilla. Kieli on sovellusriippumaton ja se sisältää universaalin tietokannan käytetyimpien verkkoprotokollien otsakekenttien rakenteista. NetPDL käyttää pakettirakenteita kuvaavia XML-tiedostoja tunnistaakseen välitettävän paketin osat, jolloin saatuja tietoja voidaan käyttää esimerkiksi verkkoliikenteen suodattamiseen tai mallinnukseen. Toteutuksessa on pyritty yksinkertaisuuteen, joten se ei esimerkiksi sovellu täydellisiin protokollamäärityksiin, jossa kuvaillaan kommunikoinnin tilasta riippuvaiset erimuotoiset pakettirakenteet. Esitetty malli sopiikin enemmän vain protokollien otsakekenttien rakenteen kuvaamisiin ja erilaisten protokollien kapselointiin.

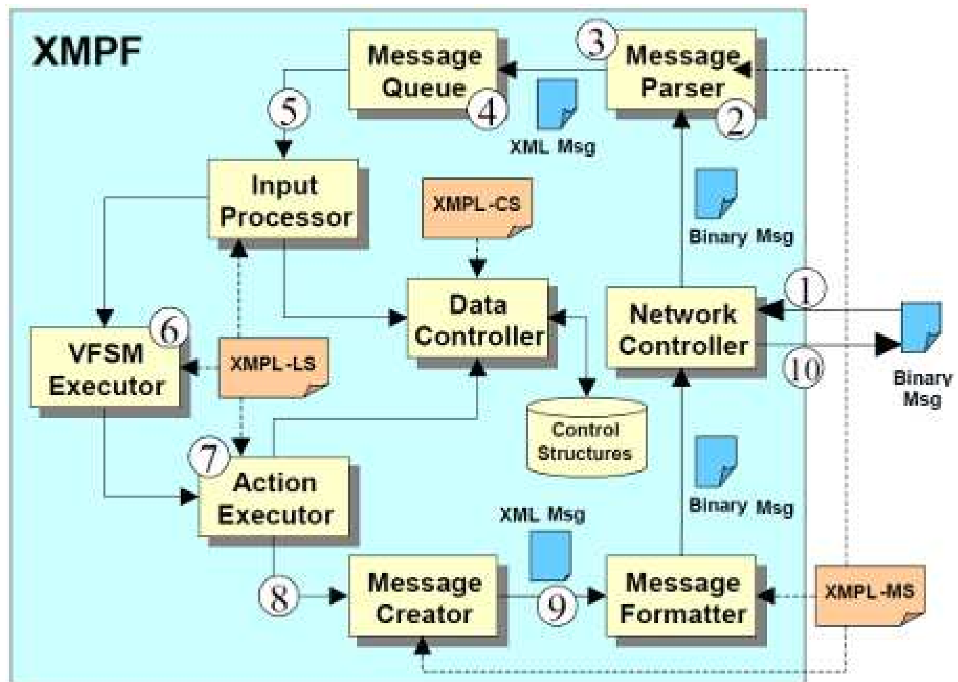
[Baroncelli, Martini & Castoldi 2006, s.139] ovat tutkimustyössään menneet paljon pidemmälle. Myös heidän esittämänsä järjestelmä pyrkii esittämään protokollien pakettirakenteen XML-muotoisina, mutta he mallintavat myös protokollien toimintalogiikat. XML-pohjainen mallinnuskieli on nimeltään XMPL (XML-based Multi Protocol Language). Protokollan tueksi on myös kehitetty kehysympäristö XMPF (XML-based Multi Protocol Framewok), jonka tehtävänä on toimia protokollan määritykset tulkitsevana ympäristönä. Tiedonsiirrossa noudatetaan edelleen binaarimuotoista formaattia.

XMPL pystyy esittämään yleisen kuvauksen mistä tahansa TLV:n perustuvasta protokollasta, mikäli sille on määritelty kyseisen protokollan ominaisuuksista kootut seuraavat kolme spesifikaatiota [Baroncelli, Martini & Castoldi 2006, s.140]:

- Viestijoukot kuvaava XMPL-MS-spesifikaatio (XMPL Message-set Specification)
- Logiikkaa kuvaava XMPL-LS-spesifikaatio (XMPL Logic Specification)

- Tietorakenteita kuvaava XMPL-DS-spesifikaatio (XMPL Data Structure Specification)

Jokaiselle spesifikaatiolle on omat skeemamäärittelynsä, jonka perusteella määritellään XMPF:n käyttämien XML-dokumenttien rakenteet, sisällöt ja semantiikat. Kuvassa Kuva 2-9 on esitetty XMPF-kehysmallin lohkokaavio, josta ilmenee XML-pohjaisen pakettikäsittelijän toiminta.



Kuva 2-9 XMPF-kehysmallin lohkokaavio [Baroncelli et al. 2006, s.140]

[Baroncelli, Martini & Castoldi 2006, s.141] kuvailee yleisellä tasolla lohkokaavion toiminnan vastaanotettu viestiä käsiteltäessä seuraavasti:

1. Verkkokontrolleri (Network Controller) poimii verkosta tulevat viestit ja välittää ne käsittelijälle.
2. Viestien käsittelijä(Message Parser) muuntaa binaarimuotoisen paketin XML-muotoiseksi.
3. Viestin käsittelijä tarkistaa XML-muotoisen viestin rakenteen oikeellisuuden.
4. Viestijono (Message Queue) säilöö vastaanotetun viestin odottamaan vuoroansa.

5. Viesti prosessoija (Input Processor) noutaa viestin jonosta ja poimii siitä määritellyn protokollalogiikan (XMPL-LS) mukaisesti hyödylliset osat.
6. Prosessoijan poimimat osat välitetään ne yksityiskohtaisemmin käsiteleválle tilakoneelle (VFSM Executor (Virtual Finite-State Machine)), joka yhdistää saadut osat niitä kuvaaviin toimintoihin.
7. Toimintojen suorittaja (Action Executor) tekee saatujen komentojen perusteella vaaditut operaatiot ja välittää ne protokollalogiikan mukaisesti joko tiedon kontrolloijalle tai vastausviestin luojalle.
8. Jos saatu viesti edellyttää vastausviestin luomista, toimintosuorittaja välittää tiedot viestinluojalle, joka muodostaa XML-muotoisen vastausviestin.
9. Saatu XML-dokumentti välitetään viestien muokkaajalle (Message Formatter), joka muuntaa XML-muotoiset viestit binaarimuotoisiksi XMPL-MS-spesifikaation määrittelyjen mukaisesti.
10. Verkkokontrolleri lähettää viestin eteenpäin verkkorajapinnalle.

Toteutuksen eduiksi [Baroncelli, Martini & Castoldi 2006, s.142] mainitsevat muun muassa pienentyneet implementointi- ja käyttöönottoajat sekä parantuneen vakauden ja helpomman laajennettavuuden muihin ympäristöihin. Toteutuksen huolenaiheiksi he nostivat XML-dokumenttien hallintaan kuluvan liian suuren prosessointiajan ja muistinkäytön.

Erääksi kehysmekanismiksi on kehitetty BEEP (Blocks Extensible Exchange Protocol), joka mahdollistaa samanaikaiset itsenäiset viestien välitykset verkkopäätteiden välillä. Kehyksissä voidaan välittää kehittäjien omia protokollia. Välitetyn datan muoto koodaustapana käytetään omaa MIME-tyyppiä (Multipurpose Internet Mail Extensions). Sisällöltään lähetetyt viestit ovat pääasiallisesti tekstimuotoisia XML-dokumentteja. Viestinvälitykset tapahtuvat yhdyspisteiden välille muodostetun kanavan kautta. Jokaiselle kanavalle on määritelty oma profiilinsa, joka määrittelee välitettävien viestien syntaksin ja semantiikan. [Rose, 2001]

BEEP helpottaa omien verkkoyhteyksiä hyödyntävien sovellusten toteuttamista. Se pitää jo valmiiksi sisällään TCP/IP-yhteyksissä tarvittavat perusmekanismit.

Tällöin sovelluskehittäjän ei tarvitse esimerkiksi huolehtia kommunikointiin kuuluvista yhteydenmuodostuksesta, autentikoinnista, viestien välittämisestä ja virheiden raportoinnista. [Dumbill 2001]

2.5.2 XML-pohjaiset verkonhallintaratkaisut

XML-pohjaiset verkonhallintaratkaisut ovat yksi erityinen sovellusryhmä niiden tekniikoiden joukossa, jotka perustuvat joko hallintaprotokollan kapselointiin tai sen esittämiseen XML-muotoisena tietorakenteena. Diplomityön aiheen puitteissa käsiteltävään Telesten elementtienhallintaprotokollaan viitaten voidaan työssä soveltaa yleisiä hallintaprotokolliin liittyviä XML-tekniikoita ja niitä hyödyntäviä ratkaisuja.

[Menten 2004, s99] kokemuksiensa perusteella toteaa, että XML-pohjaisten esitystapojen ja työkalujen omaksuminen laitehallintaan liittyvissä infrastruktuureissa parantaa dramaattisesti niiden eheyttä, joustavuutta ja toiminnallista kestävyyttä. Lisäksi [Klie & Strauβ 2003, s.506] esittelevät XML-protokollien käyttömahdollisuuksia verkonhallintaprotokollissa seuraavasti:

- **Hallittava data on helposti esitettävissä XML-dokumentteina.** Esimerkiksi, jos suurien tietomäärien tiedonsiirron yhteydessä vertaillaan SNMP:n ASCII merkkijonojen käyttöä XML-dokumentteina siirrettävään, XML:n kanssa selvitään paremmin.
- **XML-dokumentit voidaan sijoittaa monenlaisten tiedonsiirtoprotokollien päälle.**
- **DOM- ja SAX-sovellusrajapintoja voidaan käyttää hallintadatan käsittelyyn.** Monet XML-jäsentimet tukevat näitä yleisesti käytettyjä rajapintoja.
- **XPath-kyselykieltä voidaan käyttää erillisten rakenteiden poimimiseen hallintadatasta**
- **XSLT-metodeja voidaan käyttää hallintadatan suodattamiseen ja muokkaamiseen.**

- **Hallintadatan rakenne voidaan esittää XML-skeemakielellä.** Näin varmistetaan käytettävien XML-dokumenttien eheys.

XML-pohjaisessa verkonhallinnassa on kaksi pääsuuntaa: puhtaasti XML-pohjaiseen viestintään tukeutuvat ratkaisut, sekä ratkaisut, joissa XML-sovellus voi kommunikoida SNMP-pohjaisen sovelluksen kanssa. Ensin mainittu toimii suoraan ilman välityspalvelimia. Siitä esimerkkeinä voidaan mainita Juniperin tuotteissaan käyttämä Junoscript [Juniper 2010] ja yleinen standardi Netconf [Enns 2006].

Junoscript on Juniperin toteuttama XML-pohjainen sovellusrajapinta Juniperin valmistamien palvelimien hallintaan. Rajapinnan avulla kohdelaitteilta voidaan pyytää sen ylläpitämiä konfiguraatiotietoja tai niitä voidaan muokata niissä. Netconf:n tavoitteena on tarjota standardoitu XML-tekniikoihin perustuva protokolla verkkolaitteiden konfiguraatioiden hallintaan. Protokolla määrittelee käytettävät konfiguraatioiden esitystavat ja menetelmät niiden operoimiseksi käyttämällä RPC-pohjaista (Remote Procedure Call) yhteysmuotoa XML-muotoisten metodien kautta. Esimerkiksi [Mi-Jung et al. 2004] ovat Netconf:n pohjalta kehittäneet XML-pohjaisen konfiguraationhallintajärjestelmän, jossa kommunikointi toteutetaan web-palvelujen kautta. Junoscript ja Netconf eivät kuitenkaan ole samanlaisia: Siinä missä Netconf pyrkii käyttämään yleisen tason operaatioita, Junoscript:n operaatiot ovat käsiteltävästä datasta riippuvaisia [Cridlig et al. 2005].

[Alexopoulos & Soldatos, 2005, s.451] esittävät XML-pohjaiseen verkonhallintaan omaa XMLNet-nimistä arkkitehtuuria, joka toteuttaa ajonaikaisen ympäristön verkonhallintaoperaatioiden välittämiseksi verkon laitteille. Verkonhallintaoperaatiot voidaan esittää XML-pohjaisella komentokielellä. Näin pystytään käyttämään perinteisiä ohjelmointikielien mekanismeja, kuten esimerkiksi ehtolausekkeita ja silmukoita haastavampien tehtävien suorittamiseksi. Toimintoja voidaan toteuttaa omaehtoisesti ilman varsinaista ohjelmointityötä. Rakenteet kuvataan käyttäen XML-skeematiedostoja. Kuten esimerkiksi [Cridlig et al. 2005], [Alexopoulos & Soldatos 2005, s.470] ovat myös käyttäneet kehysympäristössään

XPath-hakuja erottelemaan lähteenä toimivasta XML-dokumentista haluttuja tietoja.

SNMP-protokollaa käyttävien laitteiden kanssa kommunikoivat XML-sovellukset vaativat välityspalvelimen tulkitsemaan siirrettyä dataa. [Zhixia, Ziheng, Yu & Debao 2007, s.5192] ovat kehittäneet SNMP:n tehokkuuteen ja skaalautuvuuteen liittyvien ongelmien ratkaisuksi XML-pohjaisen yhdyskäytävän. XML-yhdyskäytävä kommunikoi samanaikaisesti usean verkonhallintaan käytetyn agentin kanssa, joita hallitaan web-pohjaisella käyttöliittymällä. Agentit kommunikoivat SNMP-protokollalla hallittavien verkkolaitteidensa kanssa. Yhdyskäytävä muuntaa SNMP-muotoiset vastaukset ja komennot XML-muotoisiksi vastineiksi, jolloin ne ovat paremmin sovellettavissa. Samankaltaiseen toteutukseen ovat pyrkineet myös [Klie & Strauβ 2004]. He esittävät SNMP:n mukaiset tietorakenteet skeematiedostoilla, joiden perusteella muodostetaan varsinaiset hallintaan käytetyt komennot. Esitystavan toimivuuden testaamiseksi, he ovat kehittäneet erityisen välityspalvelimen, joka luo SNMP-agenteille lähettävät komennot ja tulkitsee agenteilta vastaanotetut tiedot [Klie & Strauβ 2004, s.81].

3 TELESTEN NYKYINEN AJURIYMPÄRISTÖ

Tässä luvussa esitellään testausympäristö, jossa tuotannon testaussovelluksia ja mittalaitteita käytetään. Aiemmin testaussovellukset olivat melko yksinkertaisia. Tekniikan kehittyessä ja laadullisten vaatimusten kiristyessä testaussovelluksien toiminnallekin on asetettu kovemmat vaatimukset. Telestellä on tuotannossaan ollut jo useamman vuoden käytössä yhtenäinen testausympäristö, joka koostuu yleiskäytettävistä sovelluksista ja sovituista käytännöistä.

3.1 Ohjelmointirajapinta

Telesten testausjärjestelmät perustuvat National Instrumentsin kehittämään LabView-pohjaiseen kehitysympäristöön. Itse ajuri ohjelmoitiin käyttämällä graafisen LabView-ohjelmointiympäristön 8.6 versiota. LabView ohjelmointikielellä toteutettua ajuria käytetään testisekvensseissä, jotka ovat joukko toimintoja tuotteille kehitettyjen proseduraalisten testien suorittamiseksi. Testisekvenssit luodaan NI:n TestStandilla, josta on käytössä 4.1 versio.

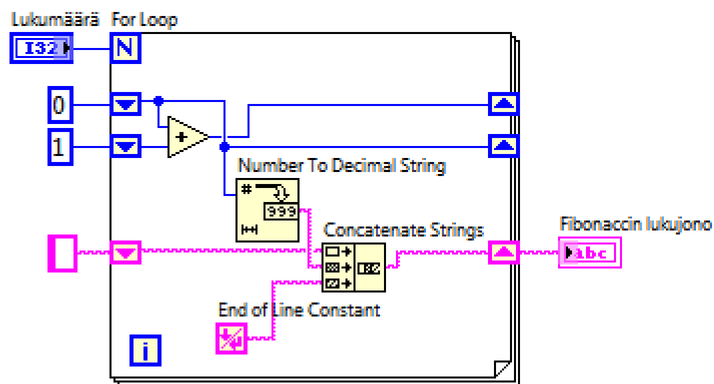
3.1.1 LabView G

LabView G -kieli eroaa tavanomaisista ohjelmointikielistä juuri graafisen ohjelmointiympäristönsä takia. Ohjelmointikielen eri tietorakenteet ja toiminnot ovat esitetty toiminnallisuutta havainnollistavilla kuvakkeilla. Kuvakkeiden vasemmalla puolella on liitäntäkohta syötteelle ja oikealla puolella sijaitsee liitäntäkohta tulostukselle. Toiminnoista toiseen tieto välitetään käyttämällä liitäntäkohtia yhdistäviä viivoja, joiden väri ja täytekuvio kertovat välitettävän arvon tietotyypin. Esimerkiksi kokonaisluvuille on valittu sininen väri ja tekstitietoa välittävät viivat ovat esitetty vaaleanpunaisena. Yksittäisiä arvoja välittävä viiva on ohut ja yh-

denvärinen. Yksi ja useampiulotteiset taulukot erotetaan yksittäisistä arvoista viivan paksuuden ja kuvioinnin perusteella. Tietotyyppiä kuvaava perusväri pysyy tällöin kuitenkin samana.

Perinteisissä ohjelmointikielissä ohjelmaa suoritetaan kontrollivuona (control flow), jossa ohjelmalohkojen peräkkäinen järjestys määrää lopullisen suoritusjärjestyksen [Sumathi & Surekha, 2007, s.61]. LabView suorittaa toimintonsa tietovuona (dataflow), joka noudattaa vedettyjä viivoja vasemmalta oikealle. Virtuaali-instrumentiksi luotu lohko suoritetaan, kun sen kaikissa syötekohdissa on vaaditut arvot. Ohjelman suoritus on valmis, kun kaikissa sen tulostuskohdissa on palautetut arvot. Näin ollen ohjelmalohkojen syötteet määräävät suoritusjärjestyksen.

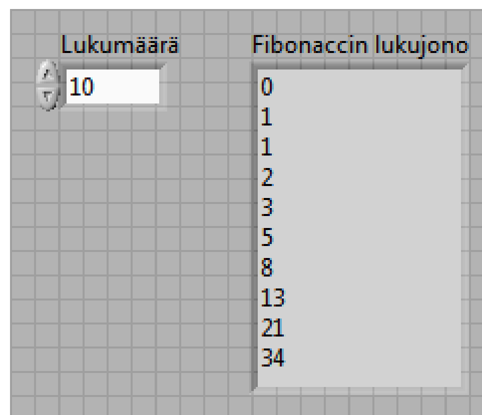
Kuvassa Kuva 3-1 on esitetty LabView G –ohjelmointikielellä esitetty lohkokaavio Fibonaccin lukujonon tulostamiseksi.



Kuva 3-1 Lohkokaavio Fibonaccin lukujonon tulostamiseksi

Syötteenä annetaan Fibonaccin lukujonosta tulostettavien numeroarvojen määrä. Lukumäärä on 32 bittinen kokonaisluku, jonka myös ilmaisee sininen väri. Syötetty numeroarvo on yhdistetty viivalla for-silmukan käsittävään rakenteeseen. Silmukassa N on kierrosten lukumäärä ja i kierroksen järjestysnumero. Lähtöarvoina silmukalle annetaan luvut 0 ja 1. Nämä luvut lasketaan yhteen ja tulos yhdistetään siirtorekisteriä kuvaavaan elementtiin. Suoritusjärjestyksen ollessa vasemmalta oikealle siirtorekistereistä luetaan silmukan vasemmalta puolelta ja niihin kirjoitetaan silmukan oikealle puolelle. Ylöspäin osoittava tasasivuinen kolmio on siirto-

rekisteriin tallentamista ja alaspäin osoittava lukemista varten. Jokaisen silmukan kierroksen alussa siirtorekisteristä luetaan kahden edellisen kierroksen arvot. Purppuralla värillä esitetään tekstimuotoista dataa sisältävät objektit. Esimerkkikuvan yhteydessä ennen silmukkaa alustetaan tyhjä tekstikenttä, joka myöhemmin tulee pitämään sisällään rivinvaihtomerkein erotellun listan Fibonaccin luvuista. Jokaisella kierroksella tekstijonon perään lisätään laskettu tulos, joka on ensin muutettu numerosta tekstimuotoiseksi. Jokaisen tuloksen jälkeen lisätään rivinvaihtomerkki tulostuksen selventämiseksi. Kuvassa Kuva 3-2 on esitetty ohjelman suoritukseen käytettävä käyttöliittymä.



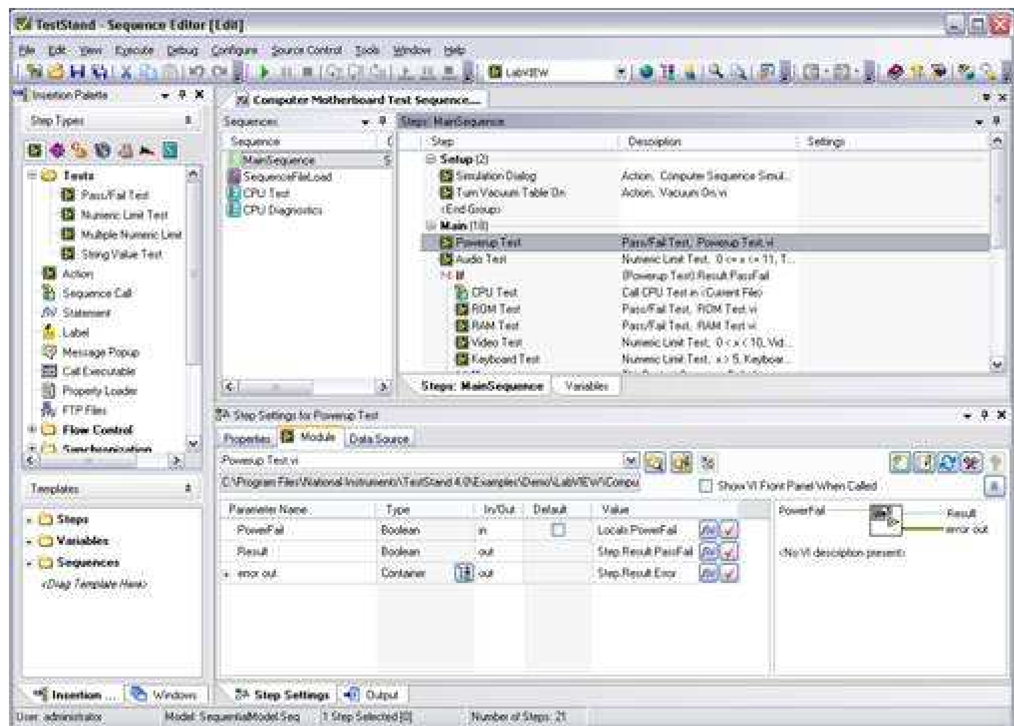
Kuva 3-2 Käyttöliittymä Fibonaccin lukujonon laskemiseksi

Kuvassa vasemmalla puolella on käyttäjän määrittelemä lukumäärä, jonka perusteella lasketaan lukuja lukujonosta. Oikealla puolella on rivinvaihtomerkein eroteltu lista lasketuista arvoista virtuaali-instrumentin suorituksen jälkeen.

3.1.2 TestStand

TestStand on testisarjojen suunnitteluohjelma, joka mahdollistaa testausympäristön hyvin monipuolisen mukauttamisen. Se myös tarjoaa yhtenäisen ympäristön ja käyttöliittymän automaattisille testeille. Yksinkertaisimmalla tasolla se tarjoaa hallintatyökalut ja työtilan mille tahansa itse tehdylle testaussovellukselle. Vaativammalla tasolla se kokoaa useat testisovelluksien osaset yhdeksi kokonaisuudek-

si, joka voidaan käydä läpi toistuvissa sykleissä. Sille voidaan myös asettaa suorittamisen ehdoiksi kompleksisia ehtolausekkeita. TestStand erottelee yleiset testin suorittamiseen liittyvät taustatoimenpiteet, kuten esimerkiksi raporttien teot, varsinaisesta testaustapahtumasta. Se voidaan myös liittää yritysten tietojärjestelmiin, kuten esimerkiksi tuoteinformaatiota käsitteleviin hallintajärjestelmiin. Kuvassa Kuva 3-3 on esitetty TestStand:n käyttöliittymä.



Kuva 3-3 TestStand:n käyttöliittymä

Käyttöliittymässä ylin iso työskentelyalue on testisekvenssieditori, johon muodostetaan järjestetty lista peräkkäin suoritettavista toiminnoista. Toiminnot voivat olla esimerkiksi kutsuja toisiin testisekvensseihin, itsenäisiä virtuaali-instrumentteja tai testisekvenssin suorittamisjärjestyksen hallintaa käytettävää ohjausinformaatiota. Alemmassa osassa on testiaskeleiden ominaisuuksia näyttävä työskentelytila, jossa voidaan määritellä valittujen askeleiden toimintaa tarkemmin. Vasemmassa laidassa on joukko erilaisia työkaluja toimintoja, joita voidaan käyttää apuna. Suunnittelutyökalulla määriteltyjä testisarjoja suoritetaan erillisissä ajonaikaisissa ympäristöissään, jotka voidaan ulkonäöltään ja toiminnaltaan räätälöidä täysin käyttäjän tarpeiden mukaisiksi.

3.2 Telesten elementtienhallintaprotokolla

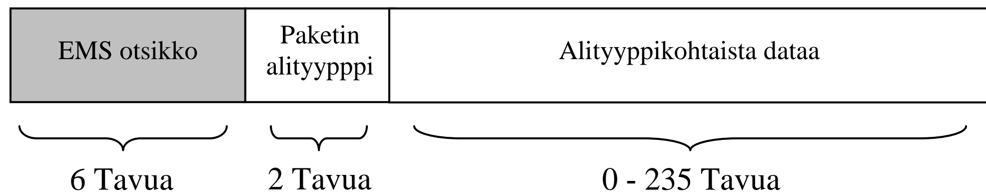
Telestellä on käytössään oma elementtienhallintaprotokolla TSEMP (Teleste Simple Element Management Protocol). Sitä kutsutaan yleisesti myös EMS-protokollaksi (Element Management System), ja se perustuu toiminnallisuudeltaan TMN-kehykseen (Telecommunication Management Network) [ITU-T 2000]. EMS-protokollaa käytetään yleisesti Telesten valmistamissa tuotteissa. Näin varmistetaan samanlainen rajapinta kaikille tuotteille ja samalla kannustetaan asiakkaita investoimaan myös Telesten valmistamiin tuotehallintasovelluksiin.

Telesten elementtien hallintaprotokolla on joukko sääntöjä, joilla määritellään miten EMS-viestit muodostetaan ja miten sovellukset toteuttavat kyseisen protokollan mukaisen kommunikoinnin. Toiminnaltaan protokolla on yksinkertainen ja johdonmukainen. Tietoturvallisuus- ja salausominaisuuksiin protokollassa ei ole juurikaan panostettu.

EMS-protokolla toimii useamman eri tiedonsiirtoväylän kautta. Aiemmin EMS-protokollaa käytettiin paljon kommunikoitaessa Telesten omaa RS-485:een perustuvaa DVX-väylää pitkin [Susi 2002]. Uudemmissa tuotteissa on jo tuki USB-väylän (Universal Serial Bus) kautta tapahtuvaan tiedonsiirtoon, mutta myös IP-protokollan (Internet Protocol) päällä toimivaa UDP-pohjaista (User Datagram Protocol) kommunikointimallia tuetaan edelleen.

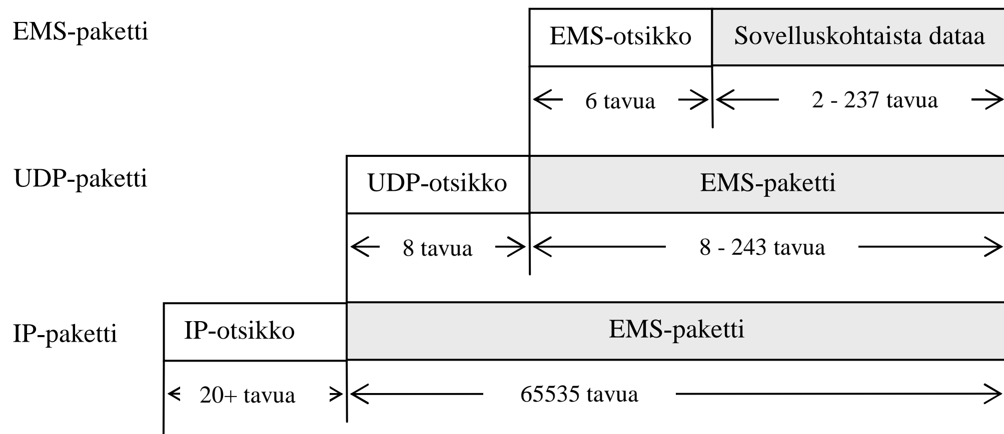
3.2.1 EMS-pakettirakenne

EMS-protokollan mukainen paketti on 243 tavun kokoinen ja sitä voidaan välittää useamman erilaisen kommunikointiprotokollan päällä. EMS-paketti muodostuu kuuden tavun mittaisesta otsikko- ja 2-237 tavun mittaisesta hyötykuormaosuudesta. Kuvassa Kuva 3-4 on esitetty tarkemmin EMS-paketin rakenne.



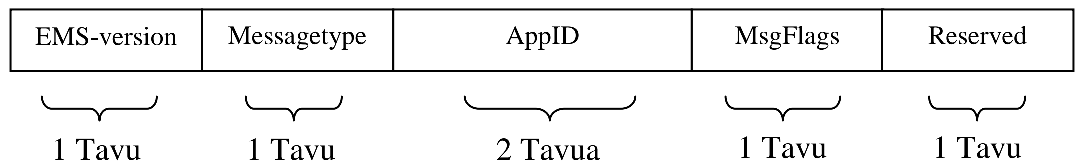
Kuva 3-4 EMS-paketin rakenne

Protokollan viestit ovat helposti kapseloitavissa eri siirtoprotokollien sisään. Kuvassa Kuva 3-5 on esitetty EMS-viestin rakenne IP-protokollan päällä välitettynä [Kuusisto 2002, s.56].



Kuva 3-5 EMS-viestin rakenne IP-protokollan päällä [Kuusisto 2002, s.56]

Viestin otsikko-osan rakenne on aina sama. Muu rakenne vaihtelee käytettävän viestityypin mukaisten toimintojen perusteella. Otsikon rakenne ja sen osien koot ovat esitetty kuvassa Kuva 3-6.



Kuva 3-6 EMS-paketin otsikko-osan rakenne

Otsikko-osassa EMS-version-kenttää käytetään kuvaamaan viestien käyttämän EMS-protokollan versionumeroa. Viestityyppi esitetään toisella tavulla. Sen pe-

rusteella vastauspaketin käsittelijälle pystytään määrittelemään, minkä viestityypin mukaisia toimintoja ja pakettirakenteita käsitellään. AppId-kenttää käytetään yksilöimään eri EMS-viestit toisistaan, jotta voidaan yhdistää kysely-vastausviestiparit keskenään suuremmista viestijoukoista. MsgFlags-kenttä määrittelee kahdeksan bitin kokoisella binäärikuviolla viestin prioriteetin sekä viestityypin jaottelun kysely- ja vastausviesteihin. Edellä mainittujen ominaisuuksien lisäksi saman binäärikuvion avulla voidaan kertoa, että vastausviesti on jouduttu paloittelemaan useampaan eri viestiin. Näin tiedetään odottaa lisää vastausviestejä samaan kyselyyn ennen käsittelyn aloittamista. Viimeinen otsikko-osion kenttä on jätetty tyhjäksi, mutta se on käytettävissä tulevaisuudessa tarpeiden varalle.

3.2.2 EMS-viestityypit

Erilaisia EMS-viestityyppejä ovat seuraavat: tila eli status, yleiset eli general, verkkokohtaiset eli Network-, laitekohtaiset eli Element-, laitteiden toiminnallisuuteen perustuvat eli Func Based-, varoituslippuviestit eli Flags ja ohjelmiston päivitykseen käytettävät Swd Upd-viestit. Viestityypit määrittelevät viestille ominaisen pyynnön (request) ja pyynnölle ominaisen vastauksen (reply).

Viestityypeistä viisi ensimmäistä ovat perustoiminnallisuuden takaavia ja yhtenäisiä kaikille Telesten valmistamille EMS-protokollalla kommunikoiville laitteille. Elementtikohtaisilla viesteillä päästään käsiksi tuotekohtaisiin ominaisuuksiin. Viestit ovat yleensä yhteensopivia pelkästään saman tuoteperheen tuotteiden kesken [Virtanen, Susi & Salminen 2003].

- **Status-viestit:** Viestit muodostuvat pelkästään vastausviesteistä. Niitä käytetään yleisesti ilmaisemaan EMS-protokollatasolla ilmeneviä virhetilanteita. Tällaisia virhetilanteita voivat esimerkiksi olla, että kyselyviesti on väärin muodostettu tai jotakin komentoa ei ole implementoitu.
- **SwdUpd-viestit:** Viestityypin komentoja käytetään päivitettäessä Teleste-laitteiden EEPROM:ia (Electronically Erasable Programmable Read-Only-

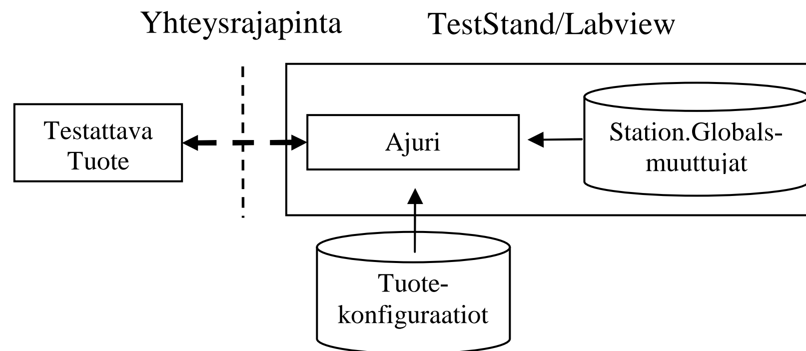
Memory). Komennot ovat kaikille laitteille samat. Komennon avulla kohdelaiteelle välitetään verrattain suuria määriä dataa.

- **General-viestityyppi:** Näitä yleisviestejä käytetään kuvaamaan laitteen yleisempiä ominaisuuksia ja toiminnallisuuksia, jotka ovat laitteista toiseen samantyyppisiä. Näitä tietoja ovat esimerkiksi sarjanumero, tuotenimi ja malli. Kaikkien Telesten valmistamien laitteiden pitää toteuttaa jossain määrin kyseisen viestityypin viestejä. Esimerkiksi kohdelaitteen uudelleenkäynnistykseen käytettävä komento on kaikille täysin sama.
- **Network-viestityyppi:** Kyseisellä viestityypillä voidaan kohdelaitteelta noutaa tai siihen asettaa verkkotekniikasta riippuvia ominaisuuksia.
- **Flags-viestit:** Lippuviestejä käytetään virhetiloista kertovien hälytysten ja varoitusten noutamiseen kohdelaitteelta. Vakavampien hälytysten ja varoitusten lisäksi laiteet voivat ilmoittaa lievemmistä tilanteista, jotka eivät varsinaisesti ole virheellisiä, mutta kannattaisi kuitenkin ottaa huomioon. Tällaisia tilanteita ovat esimerkiksi suljetuksi tarkoitetun kannen aukiolo tai joku muu ei niin kriittinen tilanne.
- **Elementtikohtaiset viestit** vaihtelevat laitteesta toiseen. Kuitenkin saman tuoteperheen eri laitteet käyttävät pääsääntöisesti samankaltaisia viestejä. Elementtikohtaisten viestien erikoistyyppi on alun perin DBM100-tuotteessa käytetyt GET- ja SET-viestit. Näitä viestejä on ryhdytty käyttämään eri tuoteperheissä laajemmin, koska kyseessä on melko tehokas tapa noutaa tai asettaa parametreja. Tällä tavoin on haluttu toteuttaa edes yksi yhtenäinen tapa parametrien käsittelemiseksi, jota voidaan käyttää jokaisessa Telesten valmistamissa laitteessa.

Telesten laitteisiin on alettu implementoimaan monimutkaisempia toimintoja. Tämän saavuttamiseksi on muodostettu mukautuva Functional based-viestityyppi, jonka sisällä pystytään välittämään näitä erikoistoiminnallisia viestejä. Kyseisen viestityypin avulla voidaan esimerkiksi käyttää EMS:n päälle sovitettuja SNMP-viestejä. Viestityypin kaikki komennot ja niiden rakenteet ovat kaikille laitteille samat. Laitekohtaisia eroja kuitenkin löytyy tuettujen viestien määrässä.

3.3 Nykyinen ajuri

Tuotannon testisekvensseissä käytettävät ajurit ovat jo pitkään olleet tekijänsä näköisiä. Suunnittelijoilla on ollut pientä pyrkimystä kohti yhdenmukaisempaa ajurisuunnittelua, jonka hallitsevana ominaisuutena ovat olleet INI-tiedostoja toiminnassaan hyödyntävät ajurit. INI-muotoisiin konfiguraatiotiedostoihin on talletettu laitekohtaisia parametreja, joilla tuotteen tietoihin päästään käsiksi. Jokaiselle eri tuotteelle on kuitenkin vielä oma ajurinsa, jonka kautta testaussovellukset toteuttavat kommunikointinsa. Näin ollen ajureita on yhä useampi erilainen, joka osaltaan monimutkaistaa testausympäristön toimintaa ja ylläpitoa. Kuvassa Kuva 3-7 on esitetty nykyisen ajurin sijoittuminen testausympäristöön:



Kuva 3-7 Vanhan ajurimallin arkkitehtuuri

Testisekvenssien käyttämät ajurit ja tukitiedostot ovat pääsääntöisesti ryhmitelty yhteen työskentelykansioon, jonka polku pidetään aina vakiona tuotannon tietokoneesta toiseen. Tuotannon tietokoneiden toimiessa lähes poikkeuksetta MS Windows-ympäristössä, oletuspolku on muotoa ”C:\Data\Teleste\”. Kyseiseen hakemistopolkuun sijoitetaan testisekvenssit ja niiden tukitiedostot. Ajurit sijaitsevat Driver-kansiossa. Ajurit ovat vielä luokiteltu selvyyden vuoksi kohdelaitteensa mukaisen käyttötarkoituksen perusteella alikansioihin. Näitä luokkia ovat esimerkiksi vahvistimet, kontrollerit, optiset vastaanottimet ja lähettimet.

3.3.1 INI-konfiguraatiotiedostot

Konfiguraatiotiedostot noudattavat Microsoftin kehittämää INI-formaattia [Microsoft 1995]. Tiedostoilla kuvataan vanhalle ajurille noudettavan parametrin ominaisuuksia. Kyseisillä ominaisuuksilla kerrotaan ajurille esimerkiksi mistä osoitteesta kohdeparametri löytyy ja miten vastaanotettua dataa pitää käsitellä. Esimerkki ajurin INI-muotoisesta konfiguraatiotiedostosta on esitetty kuvassa Kuva 3-8.

```
[STATION]
_SUBADDRESS                       = "0x01;"
TEMPERATURE                       = "0x03;0.01*[WORD]"
VOLTAGE_12_V                      = "0x05;0.01*[WORD]"
VOLTAGE_24_V                      = "0x06;0.01*[WORD]"
VOLTAGE_24_V_2                    = "0x07;0.01*[WORD]"
RETURN_PATH_OPERATION_MODE        = "0x11;1*[BYTE]"
```

Kuva 3-8 Esimerkki nykyisen ajurin INI-konfiguraatiotiedostosta

Hakasuluilla rajattuja tunnuksia käytetään erottelemaan INI-tiedoston osia toisistaan. Esimerkin tapauksessa STATION-arvo kuvaa kohdelaitteen parametrien ryhmää, jotka voivat sijaita esimerkiksi yhdessä yhteisessä moduulissa. Lohkotunnuksen jälkeen tulee lista muuttujia ja muuttujien arvoja. Muuttujat kuvaavat ajuriin toteutettua komentoa ja muuttujien arvot sijoitetaan yhtäläisyysmerkin oikealle puolen lainausmerkkeihin rajattuna. Telesten INI-ajurimallin mukaan lainausmerkkien väliin on sijoitettu useampi ominaisuus, joita käytetään parametrien käsittelyssä. Määreitä ovat tässä tapauksessa muuttujan kuvaaman ominaisuuden osoite, osoitteen perusteella palautettavan tai noudettavan arvon muunnoskerroin sekä kyseisen arvon kokoa kuvaavan tekstiarvo.

3.3.2 Ajurikutsu TDriverCall

Ajureiden perustana on TDriverCall-niminen virtuaali-instrumentti, joka toimii rajapintana testisekvenssistä ajureihin. Ajurikutsu saa parametrina ladattavaksi halutun ajurin luokan (Class) ja yksilöllisen määritteen (Specifier). Näiden tietojen perusteella käydään läpi TestStand:n testauspaikkakohtaiset Station.Globals-muuttujat. Näissä muuttujissa määritellään tiedot kaikista koneella olevista ajureista. Ajurille pitää kertoa myös sen käyttämä yhteysmuoto ja siihen liittyvät parametrit. Ajurikutsu lataa dynaamisesti halutun ajurin ja alustaa ajuriin esimerkiksi yhteyden muodostamiseen tarkoitetut parametrit Station.Globals.muuttujista noudetuilla arvoilla. TDriverCall tarjoaa yleiskäyttöiset rajapinnat niin syötteen antamiseksi ajurille, kuin ajurilta saatujen tietojen välittämiseksi testaussovellukselle. Nykyisille ajureille on yhteistä se, että kaikki tarjoavat samankaltaiset liitynnät testausympäristöön.

3.3.3 Nykyisen ajurimallin puutteet ja parannuskohteet

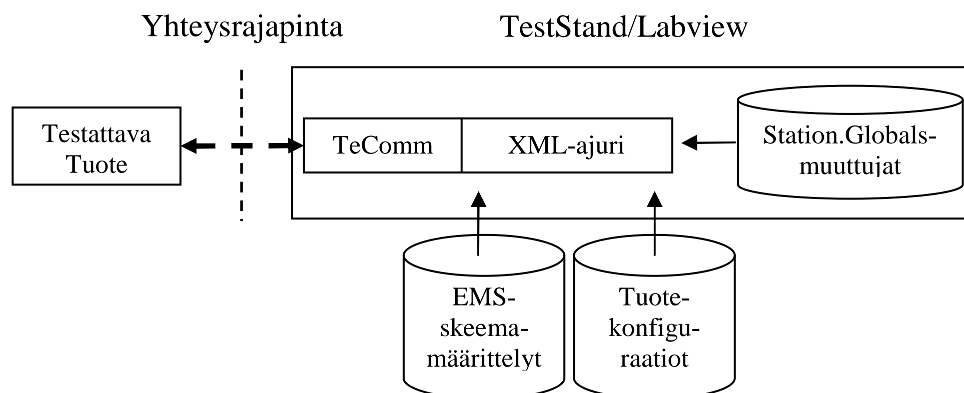
Ajurin pääasiallisin puute on sen sovellettavuus käyttökohteen muuttuessa. Ajurit sisältävät hyvin vähän sellaista muokattavuutta, jota voitaisiin tehdä ilman ohjelmointityötä. Muokattavuutta on hieman saavutettu INI-konfiguraatiotiedostomallilla, mutta edelleen jokaiselle tuotteelle joudutaan tekemään oma ajurinsa. Jonkin verran pystytään kuitenkin soveltamaan saman tuoteperheen ajureita keskenään. Eri tuoteperheillä on lisäksi omat vastaavat testaussuunnittelijansa. Näin ollen ajurien kehitystyötä tehdään melko itsenäisesti ja ajurit ovat melko paljon tekijöidensä näköisiä. Lisäksi ajurit tukevat EMS-protokollasta pelkästään ne toiminnot, jotka ovat pakollisia testatessa, joka edelleen lisää eri tuotteiden ajureiden eroavuuksia toisistaan. Myös ajurien tukemat yhteysmuodot ovat puutteellisia. Ajureihin on yleensä kehitetty vain yksi yhteysmuoto juuri ajurin toteuttamishetkellä vaaditun tarpeen mukaan, joka rajaa ajurin käyttömahdollisuuksia tulevaisuudessa.

Kehitettäessä TestStand:ssä testisekvenssejä, joudutaan tekemään kohtuullisen paljon työtä yksinkertaisten toimintojen toteuttamisessa. Esimerkiksi testattavien tuotteiden hallintaan liittyvissä toiminnoissa, kun luodaan yksittäinen ajurikutsu, joudutaan ensin erikseen tarkastamaan tuotekohtaisesta dokumentaatiosta halutun parametrin ominaisuudet kyseisessä tuotteessa. Tuotekehityksen tuottama dokumentaatio on yleensä melko niukkaa ja vähäsanaista, mutta se sisältää kuitenkin vaadittavat tiedot. Ajurikutsu määritellään konfiguroimalla oikeat komennot kyseisen testiaskeleen ominaisuuksia kuvaaviin sarakkeisiin. Tätä prosessia pitäisi helpottaa, koska ajurikutsun täydentämisprosessi kohtuuttoman työläs. Puutteena voidaan myös pitää tuotekohtaisten dokumentaatiokäytäntöjen monimuotoisuutta, joka vaatisi yhteisiä pelisääntöjä.

4 XML-POHJAINEN AJURIMALLI

Uudella ajurimallilla pyritään siihen, että ohjelmointityön ja erilaisten ajurien määrä vähenisi. Samalla pyritään myös helpottamaan testaussovellusten kehitystyötä tarjoamalla helppokäyttöiset työkalut ajurikutsujen luomiseksi. Esimerkiksi uudet testaussovellukset olisivat muokattavissa vanhasta pelkästään konfiguraatiotiedostoja muuttaen.

Ajurin avulla kommunikoidaan kohdelaitteen kanssa asettaen tai noutaen parametreja. Ajuria kutsutaan XML-ajuriksi, koska sen toimintaan kuuluvat olennaisena osana XML-pohjaiset tekniikat. Kuvassa Kuva 4-1 on esitetty uuden XML-ajurimallin arkkitehtuuri testausympäristössä.



Kuva 4-1 XML-ajurin arkkitehtuuri testausympäristössä

Ajurin pohjana ovat XML-muotoiset tukitiedostot. Toimintaa tukevien tiedostojen avulla päästään eroon ajureihin kovakoodatuista ominaisuuksista ja lisätään ajurien käytettävyyttä. Tukitiedostojen rakenteen suunnittelussa pitää ottaa huomioon tallennettavan datan ominaisuudet. Tietotyyppien perusteella pitää päättää, kuinka elementtejä ja attribuutteja käytetään tiedon tallentamiseen, sekä käytetäänkö elementti- vai tyyppimäärityksiä rakenteen kuvaamiseen. Näiden lisäksi pitää varautua mahdollisiin tulevaisuuden muuttuviin vaatimuksiin.

Ajurin tarvitsemat parametrit esitetään laitekohtaisissa konfiguraatiotiedostoissa ja ajurin käyttämä EMS-kommunikaatioprotokolla määritellään skeematiedostoja käyttäen. Tällä tavoin ajuriin jää mahdollisimman vähän staattista tietoa. Uudella ajurimallilla protokollassa tapahtuvat muutokset voidaan implementoida pelkästään muokkaamalla tarvittavia tukitiedostoja.

4.1 Ajurin toiminnot

Ajuria käytetään kommunikointiin kohdelaitteen kanssa. Ajurin suorittaminen voidaan jakaa neljään yksittäiseen päävaiheeseen:

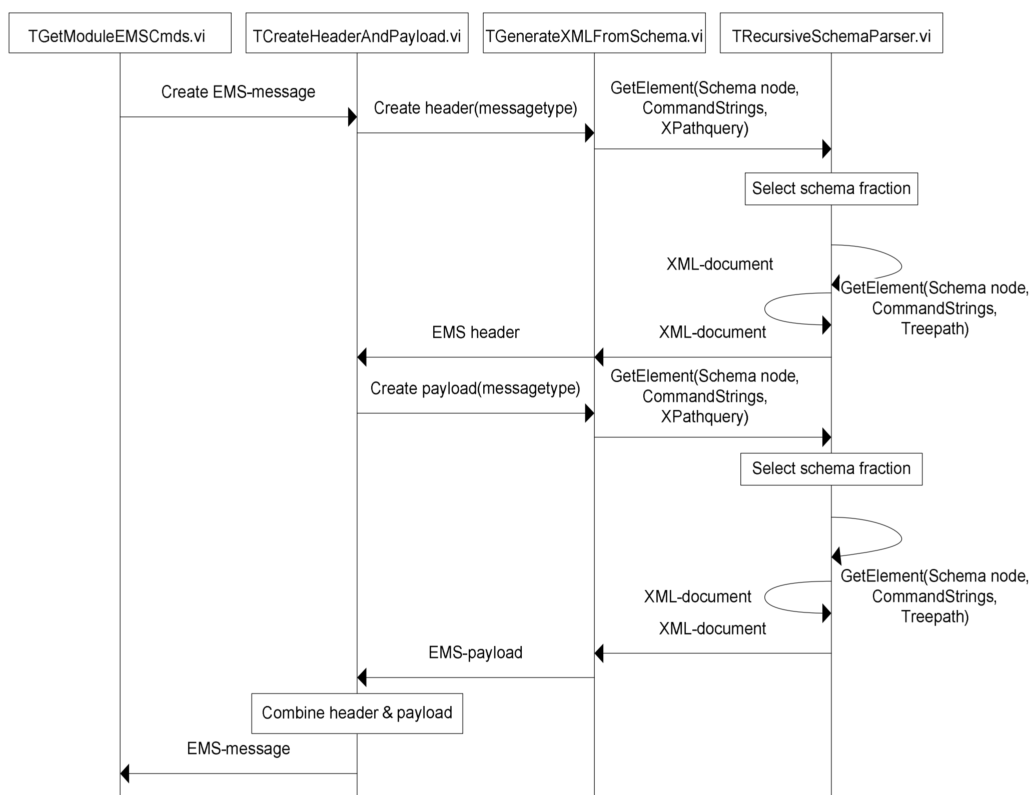
1. Testipaikkakohtaisen ajuri-informaation nouto Station.Globals-muuttujista
2. EMS-paketin muodostaminen
3. kommunikaatio kohdelaitteeseen
4. Vastausviestin prosessointi.

TestStand ylläpitää yleisiä muuttujia, joihin voidaan tallettaa esimerkiksi testausympäristöön liittyviä tietoja. Station.Globals-muuttujista noudetaan ajurin toimintaan parametrit ennen varsinaisen EMS-paketin muodostamista. Paketti muodostetaan komentoparametrien mukaan käyttämällä pakettirakenteita kuvaavia skeematiedostoja. Kommunikaatiovaihe käsittää varsinaisen yhteydenpidon kohdelaitteen kanssa. Käytetty tiedonsiirtomuoto määritellään Station.Globals-muuttujien avulla ja EMS-protokollan ominaisuuksien mukaisesti pakettirakenteet eivät ole riippuvaisia käytetystä tiedonsiirtoväylästä. Viimeisenä vaiheena on vastausviestin käsittely, jossa lähetetyn viestin perusteella välitetään saatu vastaus selkokielisenä käyttäjälle.

4.1.1 EMS-viestin muodostaminen ajurissa

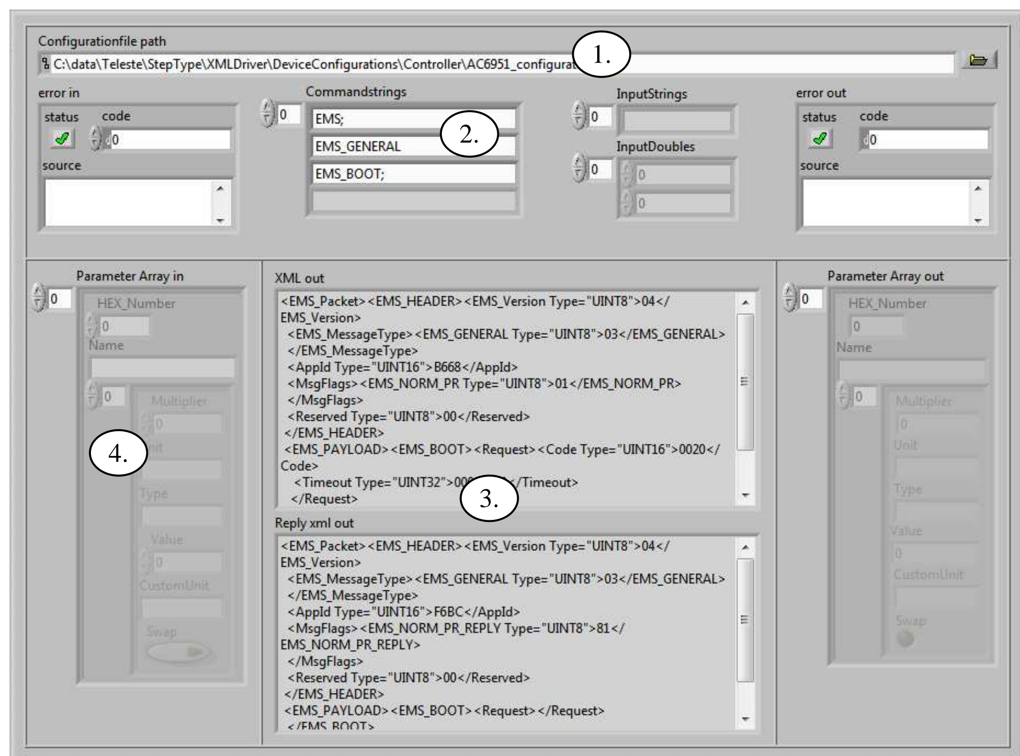
Ajuri muodostaa EMS-paketin määriteltyjen parametrien perusteella. EMS-komentopaketin lisäksi muodostetaan komentoa vastaava vastauspaketin templaatti, jota käytetään kohteelta saadun vastauksen käsittelyyn. Templaatin avulla pystytään poimimaan heksadesimaalijonosta merkitsevät arvot ja välittämään ne käyttäjälle. Toteuttamisessa käytetään mallina [Baroncelli, Martini & Castoldi 2006] esittämän kehysympäristöä, koska EMS-paketin käsittelyssä on paljon samoja vaiheita.

Toiminnallisuudeltaan merkittävät virtuaali-instrumentit sijaitseva TXMLDrv.vi:n alaisuudessa. TXMLDrv.vi sisältää kaikki toiminnot EMS-protokollaa hyödyntävää kommunikointia varten. Näitä toimintoja ovat XML-muotoisen EMS-paketin luominen, yhteydenhallinta kohdelaitteeseen ja vastauspaketin käsittely. Paketinmuodostamisessa hallitsevissa rooleissa toimivien virtuaali-instrumenttien hierarkia ja riippuvaisuussuhteet toisiinsa nähden ovat esitetty kuvassa Kuva 4-2.



Kuva 4-2 Ajurin lohkokaavio ajonaikaisten tiedostokutsujen perusteella

TGetModuleEMSCmds.vi muodostaa XML-muotoisen EMS-kysymyspaketin aliprosesseilta saatujen palautusarvojen perusteella. Sama instrumentti luo templaatin oletetulle vastauspaketille, jonka perusteella voidaan erotella heksadesimaalimuotoisesta vastauspaketista vaaditut arvot vastaaviin sarakkeisiinsa. Virtuaali-instrumenttia voidaan käyttää myös itsenäisesti skeematiedostojen parseroinnin testaamiseen. Käyttäjän pitää manuaalisesti syöttää konfiguraatiotiedostojen sijainnit ja halutun viestityypin mukaiset komennot. XML-muotoinen paketti muodostetaan etukäteen annettujen komentojen perusteella, jotka kattavat kaikki mahdolliset valinnat skeematiedoston tarjoamien vaihtoehtoisten rakenteiden selvittämiseksi. Kuvassa Kuva 4-3 on esitetty kyseisen instrumentin käyttöliittymä.



Kuva 4-3 Käyttöliittymä paketinmuodostuksen testaukseen

Käyttöliittymän yläosassa (1.) määritellään ohjattavan tuotteen konfiguraatiotiedoston sijainti. Konfiguraatiotiedostossa määritellään muun muassa tuotekohtaisten skeematiedostojen sijainnit. Keskellä yläosassa (2.) on nelikenttäinen komentoparametreille tarkoitettu alue. Ylintä kenttää käytetään määrittelemään käytettävä protokolla, toisessa kentässä määritellään luotavan viestin tyyppi ja kolmannessa kentässä valitaan viestityypin mukainen komento. Neljättä kenttää käytetään

antamaan ylimääräisiä tarkennuksia lopulliseen pakettirakenteeseen. Alhaalla keskellä (3.) on kaksi isoa kenttää: Ylimmässä esitetään varsinainen XML-muotoinen EMS-paketti ja alemmassa esitetään komennon mukaisen vastauspaketin templaatti. Käyttöliittymän vasemmalla puolella (4.) olevalla tietorakenteella määritellään kohteeseen asetettavat tai noudettavat parametrit. Annettujen parametreja käytetään täydentämään lopulliset EMS-pakettitemplaatit.

TCreateHeaderAndPayload.vi muodostaa EMS-pakettien otsikko- ja hyötykuormaosat annettujen komentoparametrien perusteella. Virtuaali-instrumentti kutsuu molempien osien muodostamisen yhteydessä TGenerateXMLFromSchema.vi:tä, jolle annetaan komentoargumenttina muodostettavien osien tyypit. Muut komentoparametrit välitetään sellaisenaan ylemmältä tasolta. Kyseinen virtuaali-instrumentti käynnistää varsinaiset skeemoja läpikäyvät prosessit.

Skeematiedosto käydään läpi rekursiivisesti TRecursiveSchemaParser-virtuaali-instrumentin avustuksella. Ensimmäisellä kierroksella funktio saa komentoparametrien lisäksi syötteenä koko skeematiedoston ja XPath-muotoisen haun ”/”. Näin rekursio aloittaa skeeman juurielementistä ja valitsee uudella ”/*”-kyselyllä sen lapsielementit. Jokaiselle lapsielementille käynnistetään oma rekursionsa samalla XPath-haulla. Tätä jatketaan niin kauan kunnes saavutetaan skeemapuun alimmat tasot. Tämän jälkeen palautetaan siihen mennessä muodostettu XML-rakenne rekursion aiemmalle tasolle ja siirrytään käymään läpi mahdollista sisarsolmua kohdasta, josta aiempi rekursio aloitettiin. Näin jatketaan kunnes on käsitelty puun jokainen haara.

Skeematiedostossa tärkeimpiä elementtejä ovat Element, Choice, Sequence ja Attribute. Rekursio tarkastaa jokaisessa solmussa kyseisen elementin nimen ja toteuttaa ennalta määritellyt toiminnot sen perusteella. Perinteisen elementin attribuutista muodostetaan yksi muodostettavan EMS-paketin elementeistä. Choice-elementin tapauksessa EMS-paketin rakenteessa on useampi vaihtoehto. Valinta tehdään käyttäjän antamien komentoparametrien perusteella. Jos käyttäjältä ei tule sopivaa vaihtoehtoa, pysäytetään rekursio siihen ja palautetaan senhetkinen XML-

puu. Sequence-elementin tullessa kohdalle, käydään läpi sen lapsielementtejä. Jos käsiteltävä elementti on nimeltään ”xs:attribute”, täydennetään EMS-pakettiin lisättävälle elementille skeeman ”Name”-attribuutin määrittelemä attribuutti. Attribuutin arvo täydennetään joko skeemasta poimitulla vakioarvolla tai käyttäjän määritysten mukaisesti.

Rekursiossa löydetyt elementit joudutaan kapseloimaan väliaikaisten elementtien sisään, jotta ne säilyttävät XML-dokumenttien vaatimukset oikeasta muodosta ja ovat välitettävissä rekursion syvyyksistä kutsujilleen. Tällöin EMS-paketti koostuu useasta turhasta elementistä, jotka pitää saada poistettua. Kuvassa Kuva 4-4 on esitetty XSL-tyylitiedosto, jota käytetään poistamaan väliaikaisesti lisätyt elementit. Lopputuloksena on varsinainen XML-muotoinen EMS-paketti.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" version="1.0" indent="yes"/>
    <xsl:template match="*">
        <xsl:copy>
            <xsl:copy-of select="@*"/>
            <xsl:apply-templates/>
        </xsl:copy>
    </xsl:template>
    <xsl:template match="temp">
        <xsl:apply-templates select="*"/>
    </xsl:template>
</xsl:stylesheet>
```

Kuva 4-4 Tyylitiedosto väliaikaisten elemenettien poistamiseksi

Tyylitiedosto käy läpi XML-dokumentin jokaisen elementin. Jos elementin nimi on temp, sitä ei huomioida eikä lisätä eteenpäin välitettävään varsinaiseen EMS-pakettiin. Rekursiivisten skeemankäsittelyoperaatioiden ja ylimääräisten elementtien poistamisen jälkeen lopputuloksena on EMS-paketin XML-muotoinen ilmentymä, joka on esitetty kuvassa Kuva 4-5.

```
<EMS_Packet>
    <EMS_HEADER>
        <EMS_Version Type="UINT8">04</EMS_Version>
        <EMS_MessageType>
            <EMS_GENERAL Type="UINT8">03</EMS_GENERAL>
        </EMS_MessageType>
        <AppId Type="UINT16">5BE4</AppId>
        <MsgFlags>
            <EMS_NORM_PR Type="UINT8">01</EMS_NORM_PR>
        </MsgFlags>
        <Reserved Type="UINT8">00</Reserved>
    </EMS_HEADER>
    <EMS_GENERAL>
        <EMS_BOOT>
            <Request>
                <Code Type="UINT16">0020</Code>
                <Timeout Type="UINT32">00000000</Timeout>
            </Request>
        </EMS_BOOT>
    </EMS_GENERAL>
</EMS_Packet>
```

Kuva 4-5 Ajurin muodostama väliaikainen EMS-paketti

Esimerkkikuvan tapauksessa väliaikainen EMS-paketti määrittelee pakettirakenteen komennolle, jolla kohdelaite käynnistetään uudestaan. Luodun XML-dokumentin rakenteessa on pyritty selkeyteen ja yksinkertaisuuteen. Paketissa olevien heksadesimaalimuotoisten tekstiarvojen, kuten esimerkiksi AppId-arvo ”5BE4”, ja niiden kokoa kuvaavien Type-attribuuttien (UINT16) perusteella voidaan luoda varsinainen heksadesimaalimuotoinen tieto lähetettäväksi.

4.1.2 Kommunikointi kohdelaitteen kanssa

Ajurissa kommunikointiin liittyvät toiminnot jaotellaan omaan toiminnalliseen lohkoonsa. Toiminnot voidaan jakaa neljään seuraavanlaiseen osa-alueeseen:

- yhteyden muodostus.
- viestin lähettäminen kohdelaitteeseen,
- vastausviestin vastaanottaminen kohdelaitteelta
- yhteyden sulkeminen.

Yhteyden muodostus ja sulkeminen ovat valinnaisia toimintoja, jotka ovat riippuvaisia esimerkiksi käytetystä tiedonsiirtoväylästä. Nämä toiminnot ovat käytössä esimerkiksi sarjaporttimuotoisessa tiedonsiirrossa. Viesti lähetään kommunikointiväylään heksadesimaalimuotoisena merkkijonona., joka kapseloidaan alemman tason tiedonsiirtoprotokollan sisään. Koska testattavat laitteet eivät suoraan tue XML-pohjaista kommunikointimallia, kuten esimerkiksi [Juniper 2010] tapauksessa, pitää XML-viestit ajurissa tulkata käytetylle kommunikointiväylälle sopivaan muotoon. Vastaavaa mallia käytettiin esimerkiksi [Baroncelli, Martini & Castoldi 2006] esittelemässä kehysympäristössä. Edellä muodostettu paketti muutetaan ennen lähetystä heksadesimaalilukujonoksi yksinkertaisella XML-tyylitiedostolla, joka on esitelty kuvassa Kuva 4-6.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform";>
  <xsl:output method="xml" version="1.0" indent="yes"/>
  <xsl:strip-space elements="*"/>
  <xsl:template match="*">
    <xsl:copy>
      <xsl:copy-of select="@*"/>
      <xsl:apply-templates/>
    </xsl:copy>
  </xsl:template>
  <xsl:template match="output">
    <xsl:apply-templates select="*"/>
  </xsl:template>
</xsl:stylesheet>
```

Kuva 4-6 Tyylitiedosto tekstiarvojen poimimiseen

Tyylitiedosto käy koko dokumentin läpi ja poimii ensin kaikki elementit, joille on asetettu attribuutti. Ainoat attribuutilliset elementit ovat niitä, joiden sisältämät arvot kuvaavat varsinaisen EMS-paketin sisällön. Tyylitiedosto käy poimitut elementit läpi ja valitsee niistä niiden lapsina olevat tekstiarvot. Lopputuloksena on heksadesimaalimuotoinen lukujono, joka voidaan seuraavaksi välittää kommunikoinnista huolehtivalle TeComm-komponentille.

Uuden ajurin haluttiin olevan käytettävyydeltään parempi. Sen haluttiin toteuttavan kaikki tarpeelliset yhteysmuodot. Tähän tarpeeseen löydettiin Telesten omissa elementtienhallintasovelluksissa käytössä oleva TeComm-komponentti. TeComm on Telesten itse kehittämä kommunikointirajapinta, jota käytetään pääasiallisesti Telesten elementtienhallintaohjelmiston CatVisor:n osana. Komponentti tarjoaa läpinäkymättömän kommunikointirajapinnan UDP-, TCP-, DVX- ja sarjaporttiyhteyksiin. Komponenttia voidaan käyttää myös USB-portin kautta tapahtuvaan tiedonsiirtoon, mutta tällöin sovelletaan DVX-muotoista yhteyttä USB-väylään. Suoraa tukea USB-yhteyksille ollaan kuitenkin kehittämässä. Kyseistä komponenttia voidaan käyttää ActiveX-rajapinnan kautta, joten käyttö on melko vaivatonta myös National Instrumentsin työkaluilla. [Colliander & Vilola 2004]

TeComm:n antama rajapinta toimii siltä osin erinomaisesti, että testaussovelluksen ei tarvitse pitää huolta yhteyden muodostamisesta ja ylläpidosta. Komponentin käytössä on kuitenkin viivettä lisääviä ongelmia. TeComm lataa dynaamisesti taustalle oman käsittelijän hallitsemaan tarvittavia yhteyksiä fyysisesti. Dynaamisesti ladattu käsittelijä myös suljetaan automaattisesti, mikäli sitä ei hetkeen tarvita. TeComm pitää yllä listaa muodostetuista yhteyksistä pelkästään objektireferensseinä, jotka aktivoidaan tarpeen mukaan käsittelijän kautta. Käsittelijän käynnistyksessä saattaa yhteysmuodosta riippuen kulua jopa useita sekunteja, sillä se joutuu muodostamaan fyysiset yhteydet uudestaan. Testaussovelluksissa jotkin testauksen prosessit saattavat kestää useita minuutteja. Tällöin TeComm:n lataama komponentti ehtii välillä sulkeutumaan ja testaussovellus joutuu toimettomana odottelemaan yhteyden muodostamista. Vaikka kyse on sekunneista, tuotantoympäristöissä sekin hukattu aika on liikaa.

4.1.3 Vastausviestin käsittely

Vastausviestin käsittelyssä täytyy selvittää, mikä lukuarvo vastaanotetussa heksadesimaalitaulukossa vastaa mitäkin merkitsevää arvoa. Kohdelaitteelta saatu vastausviesti käsitellään EMS-viestin muodostamisen yhteydessä luodun templaatin

perusteella, jotta voidaan edes jollain tasolla olla tietoisia, minkä tyyppisestä vastausviestin rakenteesta on kyse. Yleisten ja kaikkien tuotteiden tukemien EMS-viestityyppien vastausviestien voidaan olettaa olevan samanlaisia eri tuotteista toiseen. Elementtikohtaiset viestit sen sijaan saattavat olla rakenteeltaan hieman poikkeavia tuoteperheiden kesken, joten niille tarkoitetun käsittelijän toiminnallisuuteen on asetettu enemmän vaatimuksia. SwdUpd-tyypin kyselyviesteihin vastataan aina Status-viestityypillä, joten se ei tarvitse erillistä käsittelijää.

Elementtikohtaisten ja Functional based-tyyppisten vastausviestien monimuotoisuuden takia käsittelijään on kovakoodattu paljon kiinteitä muuttujia, jotta rakenteeltaan erilaiset viestit pystyttäisiin käsittelemään. Muuttujia käytetään määrittelemään kiinteitä tiedonsiirrossa käytettäviä rakenteita, joista esimerkkinä mainittakoon aiemmin käsitelty TLV. Näin tiedetään mitä tietoa on missäkin osassa vastausviestiä ja mikä osa tiedosta lopulta pitää palauttaa käyttäjälle.

Vastausviestiä ryhdytään käsittelemään tutkimalla ensin vastausviestin toinen tavu, joka määrittelee viestin tyypin. Viestityypin perusteella siirrytään käyttämään viestityyppikohtaista vastausviestinkäsittelijää rakenteen selvittämiseksi. Käsittelijät toimivat siten, että ne etsivät ensin paketin muodostamisen yhteydessä luodusta vastausviestitemplaatista tuttujen XML-elementtien nimiä. Elementtien nimet määrittelevät tarkemmin varsinaisen vastausviestin rakennetta. Hakuihin käytetään XPath-lausekkeita.

Status-viestityypin vastaukset ovat yksinkertaisia ja helposti tulkittavissa. Käsittelijä poimii vastauksesta 16 bittisen heksadesimaaliarvon ja noutaa sen perusteella pakettirakenteen määrittelevästä skeematiedostosta tekstimuotoisen selityksen vastaanotetulle tilaviestille. Hälytys-, varoitus- ja ilmoitus-lippuihin liittyvän informaation välittämiseen tarkoitetun Flags-viestityypin vastausviestit ovat myös melko helposti käsiteltävissä. Tässä tapauksessa pitää vain selvittää templaatista vastausviestin tyyppi ja vastausviesti osaa sen mukaan prosessoida vastaanotetun heksadesimaalilukujonon.

General- ja Network-viestityyppien mukaiset vastaukset noudattavat samaa rakennetta, joten ne voidaan tulkita samassa käsittelijässä. Oletuksena käsittelijä muuntaa saadun vastauksen suoraan desimaaliluvuksi. Mikäli templaatin rakenteesta löydetään Length-niminen elementti, tiedetään vastausviestin rakenteen muodostuvan määrätyllä tavalla. Vastauksessa on ensin välitetyn arvon pituus kahdeksan bittisenä heksadesimaalilukuna ja tämän jälkeen seuraa varsinainen käyttäjälle välitettävä tulos.

Vaikka tuoteperheiden elementtikohtaisissa viesteissä on paljon eroavaisuuksia esimerkiksi yksilöllisten vastausviestirakenteidensa osalta, kaikki käyttävät isompien parametrijoukkojen palauttamiseksi samankaltaista ryhmittelyä. Näin ollen elementtikohtaisten viestien käsittelijään on määritelty muuttuja DataArray, jonka käyttötarve tarkastetaan vastausviestitemplaatista. Tämän perusteella tiedetään, että vastausviestissä on joukko parametreja, joiden lukumäärä ja koot bitteinä esitetään viestin alkuosassa.

Funktionaalisen viestityypin vastausviestien käsittelijä on toiminnaltaan kaikkein monimutkaisin. Siinä joudutaan käsittelemään lukumääräisesti kaikista eniten erilaisen rakenteen omaavia vastausviestivaihtoehtoja. Tämän vuoksi on hyvä, että kaikki tuoteperheet pyrkivät noudattamaan samanlaisesti kyseisen viestityypin mukaista kommunikointia. Näin voidaan käyttää kyseiselle viestityypille myös yhteistä käsittelijää.

4.2 Ajurin tarvitsemat tukitiedostot

Ajuri tarvitsee toimiakseen kahden tyyppisiä tukitiedostoja: XML- ja skeematiedostoja. XML-tiedostot määrittelevät laitteen ominaisuuksia. Skeematiedostot kuvaavat EMS-protokollan toteuttamia komentoja ja niiden pakettirakenteita. Ajurin tarvitsee toiminnassa seuraavia XML-tiedostoja:

- Laitekohtainen konfiguraatiotiedosto kuvaa tuotteen ominaisuuksia.

- Parametritiedosto määrittelee kaikki tuotteen muuttujat komentoineen.
- Laitteen varoituksia, virheitä tai huomioitavia ominaisuuksia määrittelevät lippuparametrit talletetaan omaan tiedostoonsa.

Skeemalla määritellään XML-pohjaisen tiedoston rakenne. Kohdedatan pitää skeeman määritysten mukaan koostua tietyistä elementeistä noudattaen niille valittuja tietotyyppejä. Skeeman avulla voidaan kertoa, ovatko elementit valinnaisia tai pakollisia. Luotaessa skeeman määrityksiin perustuvaa XML-dokumentin ilmentymää, XML-tiedoston sisällön rakenteen oikeinmuodostuneisuus tarkastetaan samaisen skeeman perusteella. Ajurin tarvitsemia skeematiedostoja on kolmea eri tyyppiä:

- EMS-viestin otsakerakennetta kuvaava skeema, joka on kaikille tuotteille yhteinen.
- Yleisten EMS-viestityyppien hyötykuormaa kuvaavat skeemat, jotka ovat kaikille yhteisiä.
- Tuotekohtaiset EMS-viestityyppien hyötykuorman rakennetta kuvaavat skeemat.

Edellä mainittujen skeematiedostojen lisäksi ajurin tarvitsemille XML-tiedostoille on omat skeematiedostonsa, mutta ne ovat lähinnä viitteellisiä määrityksiä, eikä niitä käytetä ajurin suorituksen yhteydessä. Näitä skeemoja voidaan hyödyntää muun muassa silloin, kun muokataan XML-tiedostoja käsin ja halutaan varmistua rakenteen oikeellisuudesta.

EMS-protokollan mukaiset viestityypit on jaettu tyyppinsä perusteella omiin tiedostoihinsa. Skeematiedostot yksinkertaistuvat ja niistä tulee pienempi kokoisia. Näin helpottuu skeematiedostoa läpikäyvä prosessi ja ajurin suorituskyky paranee. Eri skeemat on jaoteltu siten, että yhteiset skeematiedosto sijaitsevat kansiohierarkiassa laitekohtaisten skeemojen yläpuolella. Tuotekohtaiset skeemat sijoitetaan tuotteen omaan kansion, jossa ovat muutkin ajurin käyttämät tuotekohtaiset tiedostot.

Tukitiedostot ryhmitellään testaussuunnittelijoiden käyttämiin työskentelykansioihin siten, ettei niitä sekoiteta vanhan ajurimallin mukaisiin tiedostoihin. Näin voidaan pitää vanha ajurimalli käytössä uuden rinnalla. Kansiot nimetään saman käytännön mukaan, kuin vanhassa ajurimallissa, jolloin esimerkiksi vahvistimet sijaitsivat luokittelunsa perusteella kaikki omassa kansiossaan ja lähettimet omassaan.

4.2.1 EMS-viestien kuvaus käyttäen skeematiedostoja

Telesten vanhoille laiteajureille oli ominaista, että ne sisälsivät paljon tekijäkohtaisia ominaisuuksia. Täysimittaisia yhtenäisiä suunnittelukäytäntöjä ei INI-muotoisten konfiguraatiotiedostojen lisäksi juurikaan ollut. Uuden ajurimallin yhteydessä haluttiin minimoida suunnittelijakohtaiset ominaispiirteet ja rakentaa ajurin käyttämän EMS-kommunikaatioprotokollan pakettirakenteet ajuriin uudella tavalla. Vanhassa ajurimallissa EMS-protokollan mukaiset pakettirakenteet olivat aina kovakoodattuina, jolloin eri paketit toisistaan erottava 16 bittinen AppId-heksadesimaalitunnus oli kaikissa sama. Uudessa ajurissa pakettirakenteet luodaan aina ajurikutsun yhteydessä uudestaan eivätkä ne ole enää ajuriin kovakoodattuina viestityypin mukaan.

Sharp esittää protokollasuunnittelussa merkistökoodauksen osalta huomioitaviksi seikoiksi seuraavia ominaisuuksia [Sharp 2008, s.241]:

- ***Tehokkuus:*** Protokollan pakettirakenteen pitää olla kooltaan niin tiivis kuin mahdollista.
- ***Rajaus:*** Vastaanottajan pitää olla helposti kykenevä tunnistamaan vastaanotetun paketin alku- ja loppukohdat.
- ***Enkoodauksen helppous:*** Vastaanottajan pitää helposti pystyä selvittämään tarkalleen mitä tietoa se vastaanotti.

- ***Tiedon läpinäkyvyys:*** Tiedon esitystavan pitää olla sellainen, että mielivaltaiset bittijonot voidaan lähettää sellaisenaan protokollatietueen mukana.

Sharp kuitenkin toteaa, että edellä mainitut säännöt voivat olla konfliktissa keskenään ja niiden soveltamisessa joudutaan tekemään kompromisseja. Tällöin pitää etsiä mahdollisimman tasapainoinen ratkaisu kyseisten ominaisuuksien suhteen.

Skeemoja käytetään määrittelemään rakenteelliselle tiedolle sääntöjä, joiden mukaan voidaan todentaa esitystavan oikeellisuus. Skeemoja voidaan myös soveltaa muun muassa määriteltäessä protokollien käyttämiä pakettirakenteita. Esimerkiksi [Bertolino, Gao, Marchetti, & Polini 2007] käsittelevät tutkimustyössään XML-instanssien automaattista luomista skeemadokumentin perusteella. [Mazumdar 2007] on taas tutkinut XML-skeemadokumenttien käyttöä muunnettaessa vastaanotettuja SNMP:n ASN.1-muotoisia (Abstract Syntax Notation One) tapahtumaviestejä automaattisesti XML-muotoisiksi raporteiksi. Edellä mainittuja menetelmiä voidaan soveltaa esimerkiksi paketinmuodostuksen toteuttamiseksi.

[Mostéfaoui 2008] on esimerkiksi esittänyt mallin, jossa XML-skeema muodostetaan kolmitasoisen ryhmittelyn välityksellä, joka jaetaan käsitteelliseen, loogiseen ja fyysiseen tasoon. Käsitteellisellä tasolla olevat tosielämän asiayhteydet jaotellaan ensin sisällöltään yhteisiin loogisiin joukkoihin ja relaatioihin. Tämän ryhmittelyn perusteella voidaan luoda fyysinen skeematiedosto, jota käytetään esitettävien asiayhteyksien validoimiseen.

Skeema-dokumentteja voidaan kuitenkin muodostaa myös väärin. Esimerkiksi [Emer, Vergilio & Jino 2005] tutkimustyössään esittävät XML-skeemoille omaa testausmallia, jonka avulla voidaan selvittää skeemojen toteuttamisessa syntyviä yleisempiä virheitä. Heidän mukaansa XML-skeemojen virheet voidaan luokitella kolmeen eri luokkaan: Elementteihin, attribuutteihin ja tyyppimäärityksiin liittyviin virheluokkiin. Heidän testausprosessinsa perustuu skeemojen virheitä esitteleviin mutantoituneisiin XML-dokumentteihin ja niiden perusteella muodostettui-

hin kyselyihin, joita sovelletaan testattaviin skeemadokumentteihin. Kyselyjen perusteella saatuja tuloksia verrataan määritysten mukaisiin oletettuihin tuloksiin ja etsitään niistä poikkeavuuksia skeemapohjaisten virheiden löytämiseksi.

[Sharp 2008, s.241] esittämiä tavoitteita protokollan esitystavaksi (encoding) voidaan hyödyntää myös EMS-protokollan pakettirakenteiden mallintamisessa skeema-tiedostoiksi. Pakettirakenteet muodostetaan niin tiiviiksi kuin mahdollista, välttäen ylimääräisiä elementtejä. Skeematiedostojen määrittelyjen avulla voidaan erotella vastaanotetun paketin sisältö siten, että tiedetään mitä mikin vastaanotettu heksadesimaaliarvo tarkoittaa. Vaikka skeematiedostoilla määritettään melko kiinteästi pakettirakenteet, säilytetään mahdollisuus mielivaltaisten bittijonojen lähettämiseen. Silloinkin ollaan edelleen tietoisia vastaanotetun paketin rakenteesta, alusta ja lopusta.

Telesten elementtien hallintaprotokollan mukaisen rakenteen perusteella EMS-paketti jaetaan kahteen osaan: Header- ja Payload-osaan. Header-osan rakenne on kaikille yhtenäinen. Payload-osuus vaihtelee käytetyn viestityypin perusteella. Näin ollen Header-osuuteen voidaan käyttää yhtä skeematiedostoa. Payload-osuudessa käytetään jokaiselle viestityypille ominaista skeematiedostoa. Skeemoja käyttäen kaikki kyseiset EMS-viestin osat voitaisiin sijoittaa myös yhteen yhteiseen tiedostoon. Tällöin vältyttäisiin useammalta tiedoston avaus- ja lukuoperaatiolta, mutta yhteisestä skeematiedostosta tulisi suurikokoinen ja vaikeasti hallittava.

XML-skeemojen elementtejä ja tyyppejä voidaan määritellä joko globaalisti tai paikallisesti. Skeeman rakenteen suunnitteluun on olemassa kolme lähestymistapaa [Walkama & Laakkonen 2004, s.106]:

1. Maatuskasuunnittelu (Russian Doll Design)
2. Salamisiivusuunnittelu (Salami Slice Design)
3. Sälekaihdinsuunnittelu (Venetian Blind Design)

Skeeman perusteella luodaan XML-dokumentti ohjelmallisesti, joten skeeman olisi hyvä olla rakenteeltaan erittäin yksinkertainen. Salamisiivusuunnittelu tekee skeeman rakenteesta lyhyemmän ja se olisi ylläpidollisesti helpompi käsitellä, koska esimerkiksi toistuvat rakenteet olisi määritelty vain kertaalleen. Tämä kuitenkin vaikeuttaa skeema läpikäyvän komponentin toimintaa, koska erillään määriteltyjen skeemakomponenttien löytämiseksi toiminnalliseen komponenttiin pitää ohjelmoida monimutkaisempi toimintalogiikka. Sälekaihdinsuunnittelu voi myös sisältää salamisiivusuunnittelun tapaan globaaleja komponentteja, joten se ei myöskään ole suositeltava vaihtoehto. Jäljelle jää maatuskasuunnittelu, jossa elementit ovat paikallisia. Rakenteet ovat loogisesti jaoteltu ja niiden läpikäymiseen riittää rekursiivisesti toimivat operaatiot ilman hyppimistä dokumentin osista toisiin Maatuskasuunnittelu kuitenkin monimutkaistaa skeemadokumenttien luontia, koska toistuvia rakenteita joudutaan kopioimaan useaan eri paikkaan. [Walkama & Laakkonen 2004, s.107-111]

[Walkama & Laakkonen 2004, s. 119] esittelevät perusteet nimiavaruuksien piilottamiselle seuraavasti:

- Dokumentin selkeys ja luettavuus on erityisen tärkeää.
- Nimiavaruuksien esittäminen ei tarjoa mitään tarpeellista lisätietoa.
- Jos halutaan antaa mahdollisuus muuttaa skeemaa ilman, että se vaikuttaa ilmentymädokumentteihin.

Näin ollen skeemadokumentin nimiavaruuksille käytölle ei katsota olevan tarvetta, koska dokumentin selkeyden ja luettavuuden katsotaan olevan erityisen tärkeää. Skeemoja soveltavaa käyttäjäkuntaa voidaan pitää kokemattomana XML-tekniikoiden käytössä, joten näin ollen ei haluta vaikeuttaa entisestään skeemadokumenttien luontia ja käsittelyä.

Skeematiedosto luotiin manuaalisesti käymällä graafista skeemanluontityökalua. Rakenteiden määrittelyissä käytettiin hyväksi EMS-protokollan ja ennalta valittujen tuotteiden viestirakenteita määritteleviä spesifikaatioita. Esimerkkituotteiksi valittiin AC6951 transponderit ja DVO302 lähettimet, koska ne olivat melko uu-

sia tuotteita ja ne edustivat tuoteperheitä, joiden muut tuoteversiot toimivat samankaltaisesti. Skeemaan määriteltävien elementtien nimeämisessä noudatettiin kyseisissä spesifikaatioissa käytettyjä nimeämiskäytäntöjä.

4.2.2 Tuotekohtaiset konfiguraatiotiedostot

Ajurin kanssa käytettävien laitteiden ominaisuuksien hallintaan muodostetaan räätälöity XML-muotoinen konfiguraatiotiedosto. Tiedostoon pyritään tallentamaan kaikki laitteen ominaisuuksia ja toiminnallisuutta kuvaavat tiedot. Tällaisia tietoja ovat esimerkiksi luettelo laitteen tukemista EMS-komennoista heksadesimaaliosoitteineen. Laitekohtaisessa konfiguraatiotiedostossa määritellään myös laitteen parametreja kuvaavien parametritiedostojen sijainnit, jonka perusteella ajuri pääsee niihin käsiksi.

Tuotekohtaisessa konfiguraatiotiedostossa voidaan myös asettaa pakettiliikenteen monitorointi päälle. Tällöin voidaan esimerkiksi selvittää virhetilanteita. Kuvassa Kuva 4-7 esitetään osa tuotekohtaista konfiguraatiotiedostoa, jonka perusteella voidaan asettaa ajuri debug-tilaan ja tallentaa siirrettyjä paketteja lokitiedostoon.

```
<Debug>
    <Log>0</Log>
    <TargetSubAddressHex>00</TargetSubAddressHex>
</Debug>
```

Kuva 4-7 Asetukset tiedonsiirron monitoroinnin kontrolloimiseen

Ajuri asetetaan debug-tilaan muuntamalla Log-elementin arvoksi 1. Debug-tilaa voidaan vielä kohdentaa rajoittamalla siirrettyjen pakettien tallentamisen koskemaan pelkästään yhtä aliosoitetta, jolloin voidaan saada tarkempaa tietoa esimerkiksi jonkun tietyn moduulin kommunikoinnista. Lokitiedostot tallennetaan samaan paikkaan, jossa sijaitsevat muutkin tuotekohtaiset tiedostot.

4.2.3 Parametritiedostot

Parametritiedostoihin tallennetaan tuotteelle ominaiset parametrit. Tiedostojen avulla tiedetään, mitä parametreja kyseisessä tuotteessa on asetettavissa ja noudettavissa sekä minkä tyyppisestä datasta kyse. Ajurin toiminnalle olennaisinta tietoa ovat parametrien heksadesimaaliosoitteet, joilla niihin pääsee kohdetuotteessa käsiksi. Parametrit vaihtelevat yleensä vähän tuoteperheen sisällä. Parametritiedostot jaetaan laitteen omiin parametreihin sekä useiden laitteiden käyttämiin Modem- ja Tuner-moduulien parametreihin. Modeemi- ja viritintoiminnalliset moduulit ovat samoja useissa eri tuotteissa ja näin ollen ne on parempi erotella tuotekohtaisista parametreista. Kuvassa Kuva 4-8 on esitetty esimerkkitapaus erään tuotteen yhdestä parametrista.

```
<Parameter Name="STATION_TEMPERATURE">
   <Param_Number>3</Param_Number>
   <HEX_Number>03</HEX_Number>
   <Access_Type>R</Access_Type>
   <Property_Table />
   <Init_Type />
   <Multiplier>0.01</Multiplier>
   <Unit>C</Unit>
   <Type>INT16</Type>
   <Swap>0</Swap>
   <Description>Temperature</Description>
   <SNMP/>
</Parameter>
```

Kuva 4-8 Esimerkki Parametritiedoston yhdestä parametristä

Kyseessä on lämpötilasta kertova parametri, joka löytyy kohdetuotteen osoitteesta 0x03 ja on pelkästään luettavissa eli R. Tuotteesta saatu arvo on 16 bittinen kokonaisluku, joka pitää kertoa 0,01:llä, jotta saadaan tuotteen mittaama lämpötila celsiusasteina. Parametrille voidaan määritellä vielä SNMP-muotoinen komento, mutta se ei ole tässä ajurimallissa käytössä.

Parametritiedostojen erikoistapaus on hälytys-, varoitus- ja neutraalimpien ilmoituslippujen määrittelyyn tarkoitettu flags-konfiguraatiotiedosto. Kyseistä tiedostoa ei niinkään käytetä kertomaan tuotteen ominaisuuksia, vaan sitä käytettään tulkitsemaan tuotteen tilasta kertovia ilmoituksia. Erillisellä komennolla kysyttäessä

tuote ilmoittaa yksinkertaisesti koodatulla viestirakenteella kaikki päällä olevat hälytys-, varoitus- ja ilmoituslippunsa. Konfiguraatiotiedoston perusteella kyseisille heksadesimaalimuotoisille lippuarvoille voidaan antaa tekstimuotoinen kuvaus.

Parametritiedostojen luontia varten kehitettiin LabView-pohjainen työkalu. Tarkoitukseen kehitetyn editorin tehtävänä oli helpottaa tuotekohtaisten parametritiedostojen luontia. Telesten oman elementtienhallintajärjestelmän yhteydessä käytettävät laiteominaisuudet ovat määriteltyinä C++-ohjelmointikielessä käytettävissä header-tiedostoissa. Editori toimii yksinkertaisesti ”kopioi ja liitä”-periaatteella valikoiden kaikki tiettyä rakennetta noudattavat parametrimääritykset lähteestä. Kopio ja liitä -menetelmää käytetään, koska muuten parametreja kuvaavat tyyppiluokitukset olisivat liian vaikeasti ryhmiteltävissä ja parametrien poimintaa ei voitaisi toteuttaa automaattisesti.

4.3 Ajurin suorituskyvyn optimoiminen

Ajuria testatessa TestStand testisekvenssieditorilla huomattiin, että jossain ajurin ajon vaiheessa varataan huomattavat määrät muistia. TestStand-ympäristö pyrkii yleensä toimimaan tehokkaasti käyttämiensä resurssien suhteen, mutta varattua muistia ei kuitenkaan palauteta missään vaiheessa ja järjestelmämuistin varaus kasvaa merkittävän suureksi.

Muistiylivuodon aiheutti ActiveX:n kautta käytettävä MSXML. Kyseinen DOM on muistinhallinnaltaan tehoton etenkin nopeasti suoritettavissa operaatioissa. XML-tiedostoja käsiteltäessä tallennetaan koko tiedosto muistiin kerralla. Jos kyseisellä tiedostolle tehdään operaatioita, jokainen operaation osanen varaa oman osansa muistia ja näin varaus kertautuu. LabView:n toiminnallisuuksiin kuuluu muistinhallintaa tehostavat toiminnot. Huolimatta perusteellisesta ohjelmoidusta muistinvapautuksesta, MSXML jättää jälkeensä pieniä vapauttamattomia muisti-

varauksia, joiden yhteenlaskettu koko on useamman peräkkäisen ajon jälkeen sadoissa megatavuissa. MSXML huolehtii kuitenkin lopulta roskienkeruusta, mutta viive muodostuu suureksi ja on näin ollen tehoton. Ongelmat korostuivat etenkin ajurin rekursiivisessa osuudessa, jossa muodostetaan useita uusia XML-objekteja nopeassa tahdissa. Kyseiset objektit varaavat suuremman muistimäärän, kuin pelkän vastaavan tekstipohjaisen tietomäärän tallentaminen vaatisi.

Ratkaisuksi muistivuotoon määriteltiin testisekvenssin ajurikutsuun asetus, jolla ajuri ja siihen liittyvät tukitiedostojen referenssit vapautetaan heti testiaskeleen suorituksen jälkeen. Oletuksena TestStand säilyttää kaikki virtuaali-instrumentit muistissaan, jotta ne olisivat nopeasti ja vaivattomasti saatavilla. Vapautettaessa varattu muisti, menetetään joitakin suoritusta nopeuttavia ominaisuuksia. Esimerkiksi jokaisen ajurikutsun yhteydessä joudutaan muodostamaan uudestaan yhteys testiympäristön muuttujiin ja noutamaan laitekohtaiset parametrit. Tämän lisäksi kaikki ajurin käyttämä tukitiedostot pitää lukea uudestaan levyltä jokaisen ajurikutsun yhteydessä.

Suurin yksittäinen tekijä suorituskyvyssä ovat ajurin tukitiedostojen käsittelyihin liittyvät toiminnot. Jos vertaillaan pienintä (EMS_General, 54 kilotavua) ja suurinta (EMS_Func_Based, 323 kilotavua) skeematiedostoa käyttävää paketinmuodostusoperaatiota keskenään, ajurin nopeudessa on noin 10 millisekunnin ero. Pienellä tiedostokoolla koko operaation suoritusaika oli noin 10 millisekuntia, joten kasvatetulla tiedostokoolla oli suhteellisen iso merkitys suorituskykyyn. FuncBased-viestityypin, jota tullaan tulevaisuudessa kommunikoinnissa käyttämään enemmän, mukaisen pakettirakenteen määrittelevän skeematiedoston koko oli kaikista suurin. Näin ollen tiedostokokoon joudutaan puuttumaan tulevaisuudessa jakamalla viestityypin skeematiedostoa pienempiin rakenteellisiin osiin.

4.4 XML-ajurimallin TestStand-steppityyppi

TestStand:n testisekvenssi on joukko testiaskelia, joita kutsutaan stepeiksi. Yksi askel voi suorittaa joko yksittäisiä tai suuriakin määriä toimenpiteitä. Testisekvenssien luonnin helpottamista varten TestStand:ssä on jo valmiina joukko perustoimintoja suorittavia steppityyppejä. Yksi tällainen valmis steppityyppi on ”Numeric Limit Step”-tyyppi. Käytettäessä kyseistä steppityyppiä, sille määritellään alussa raja-arvot. Testisekvenssiä ajettaessa kyseinen steppityyppi saa aina kutsuttaessa parametrina numeerisen vertailuarvon, jota verrataan alussa määriteltyihin rajoihin. Jos annettu arvo on määriteltyjen rajojen sisällä, paluuarvona annetaan PASS testin läpäisyn merkiksi. Jos ei olla rajoissa, annetaan paluuarvona FAIL testin epäonnistumisen merkiksi. Kyseessä on siis yksinkertainen apuohjelma, joka on helposti uudelleenkäytettävissä. Etuna on lisäksi testisekvenssiä luodessa mahdollisuus asettaa nämä raja-arvot erityisen käyttöliittymän kautta, joka helpottaa steppityypin käyttöä.

Telesten testausympäristössä ajuria käytettäessä ollaan vastaavassa tilanteessa. Tarvitaan uudelleenkäytettävä komponentti, joka toteuttaa automaattisesti joukon toimintoja annettujen parametrien mukaan. Aiempaa ajurikutsua käytettäessä jouduttiin määrittelemään useampi muuttuja käsivaraisesti testiaskeleen ominaisuuksiin. Näin ollen muodostetaan oma steppityyppi, jonka tarkoitus on helpottaa testisekvenssiin liitettävän ajurikutsun luontia.

4.4.1 Steppityypin määrittely

Steppityyppiä varten pitää määritellä kiinteitä tietotyyppejä, joita hyödynnetään niin testisteppiä editoitaessa, kuin ajuria käytettäessä. Kuvassa Kuva 4-9 on esitetty XMLDriver-tietotyyppi, jonka alaisuuteen talletetaan tiedot kaikista implementoiduista ajureista.

Name	Value	Type
XMLDriver		Array of XMLDriver[0..9]
[0]		XMLDriver (Container)
Class	"Amplifier"	String
Model	"AC3000"	String
ConfigurationFile	"AC3000_configuration.xml"	String
Buss		Buss (Container)
channel_address	"0.0.3.255"	String
channel_port	"0003"	String
channel_protocol	"DVX"	String
serial_baudrate	"19200"	String
callback_id	""	String
host_address	""	String
packet_window_size	""	String
post_buffer_size	""	String
trycount	""	String
timeout	""	String
Specifier	"AC3000_DVX3"	String
ConnectionReference	Nothing	Object Reference
TIVI	""	String
Manufacturer	""	String
SerialNumber	""	String

Kuva 4-9 Station.Globals-tietotyypit TestStand:ssa

Ajurien tiedot luetteloidaan Stations.Globals-tiedostoon, jota käytettiin jo aiemman ajurin aikana. Muuttujat ovat yleensä testipaikkakohtaisia ja niiden tehtävänä onkin tarjota yleinen rajapinta, joiden kautta voidaan määritellä yhteyskohtaisia ominaisuuksia.

Omia toimintoja toteuttavan steppityypin luominen onnistuu TestStand:n käyttöliittymän kautta. Jotta uusi steppityyppi voitaisiin ottaa käyttöön asennetussa testausympäristössä, pitää steppityypin määrittelevä INI-tiedosto sijoittaa TestStand:n muita steppityyppejä sisältävään kansioon. Muuta toimenpiteitä ei tarvita ja steppityyppi on TestStand:n työkaluissa käytettävissä.

4.4.2 Toteutetut editorit

Steppityypin editoreilla pyritään helpottamaan testaussuunnittelijoiden työtä tarjoamalla käyttöliittymät ajurikutsuun liittyvien toimintojen toteuttamiseksi. TestStand:n integroitu testiaskeleditori voidaan suorittaa, kun testisekvenssiin lisätään

määrittelyä vaativa ajurikutsu. Parametrieditoria käytetään testiaskeleditorin yhteydessä täydentäen sen parametreihin liittyviä toimintoja. Testiaskeleditorissa on kaksi muuta välilehteä joilla voidaan luoda mittausparametreja tulostavan ja yhteyden sulkemiseen käytetyn ajurikutsun.

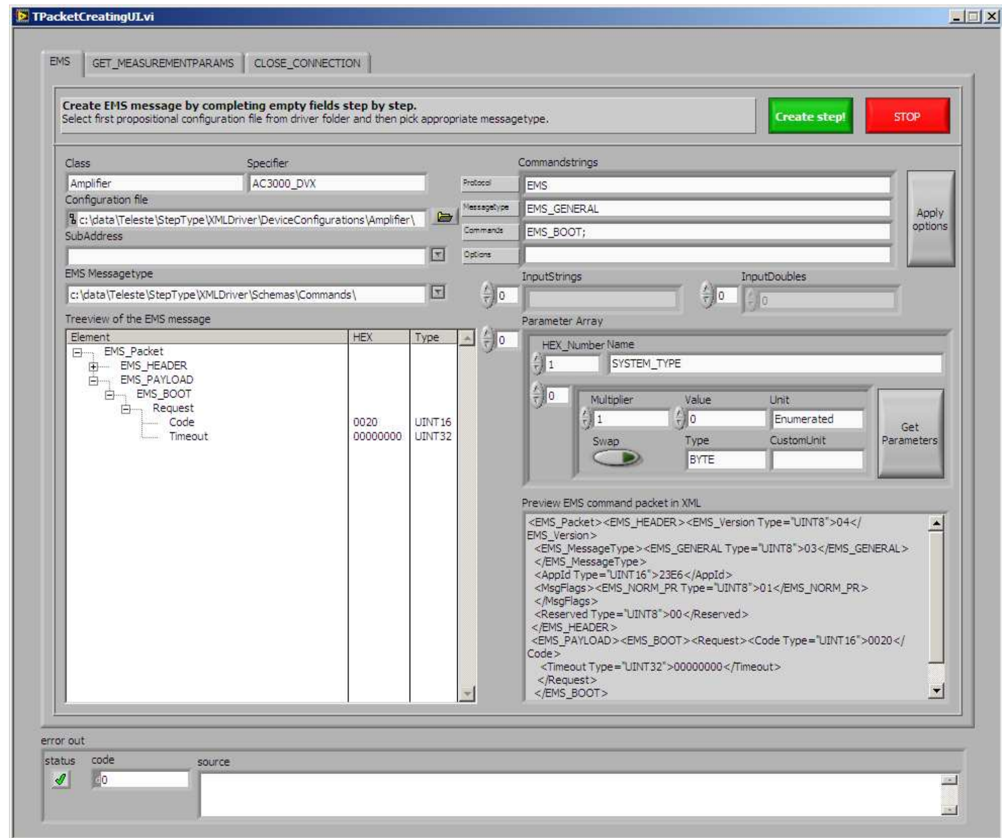
Käyttöliittymän toteuttamiseen National Instruments tarjoaa valmiita toiminnallisia komponentteja, joiden avulla voidaan luoda helposti yksinkertaisia hiirellä ja näppäimistöllä operoitavia toimintoja. Eri käyttöliittymäkomponentteja yhdistelemällä voidaankin toteuttaa ulkoiselta olemukseltaan melko näyttäviä käyttöliittymiä. Pinnan alla kuitenkin kuohuu, sillä työläin osuus on toteuttaa toiminnallisuus käyttöliittymän takana, eikä graafinen ohjelmointiympäristö ole siihen kaikkein sopivin. Perinteisesti LabView:ssa tarvittava käyttäjäinteraktio kestää vain yhden ohjelman ajon ajan, jonka jälkeen ohjelma palaa alkutilaansa. Toteutettaessa monimutkaisempaa toimintaa vaativaa käyttöliittymää välilehtineen, graafinen lohkokaavio täyttyy sisäkkäisistä silmukoista ja välilehtikohtaisista muuttujista.

Käyttöliittymät omaavat monimutkaisia toimintoja, jotka asettavat korkeat vaatimukset käyttöliittymän toiminnalle ja informatiivisuudelle. [Cooper & Reiman 2003, s.436] kuitenkin toteavat sovelluksien esittämien virheviestien olevan käyttäjilleen pääasiallisesti vihantunteita aiheuttavaa informaatiota. Käyttäjä haluaa edetä ohjelman suorittamissa omaa tahtiansa ja ylimääräiset virheviestit haittaavat sitä. [Cooper & Reiman 2003, s.439] esittävätkin, että virheviestien sijaan pitäisi keskittyä siihen, että käyttäjä ei voisi tehdä virhettä. Editorit tulisikin suunnitella siten, että niissä virheen mahdollisuus minimoidaan.

4.4.2.1 TestStep-editori

TestStand:n testisekvenssieditorissa toimivan editorin avulla määritellään XML-ajuria käyttävä testiaskel kehitettävään testisekvenssiin. Graafisen käyttöliittymän avulla määritellään tarvittavat muuttujat ja komennot, joiden avulla ajuri muodos-

taa tarvittavan EMS-viestin kohdelaitteen kanssa kommunikointia varten. Kuvassa Kuva 4-10 on esitetty testistepin luomiseen käytetty käyttöliittymä.



Kuva 4-10 Testisteppieditorin käyttöliittymä

Kun editori käynnistetään, se noutaa automaattisesti Station.Globals-muuttujista joukon testisteppikohtaisia tietoja, jotka on voitu määritellä ennen editorin käynnistämistä. Näillä tiedoilla voidaan täydentää valmiiksi määrättyjä kenttiä editorissa. Tärkeimpiä tietoja editorin toiminnalle ovat luokalle (Class) ja yksilöintitunnukselle (Specifier) varatut kentät, jotka sijaitsevat vasemmalla yläreunassa. Näiden tietojen perusteella esimerkiksi erotellaan testipaikkakohtaiset muuttujajoukot toisistaan.

Luotaessa testiaskeleelle EMS-viestiä, valitaan ensimmäiseksi viestityypin perusteella komennot ja pakettirakenteet kuvaileva skeematiedosto. Avatun skeematiedoston perusteella päivittyy vasemmassa alalaidassa oleva puurakenne niin pitkälle, kuin ennalta annettujen komentoparametrien perusteella päästään. Käyttäjä voi

puurakenteesta valita sopivanmuotoisen pakettirakenteen tuplaklikkaamalla jotain vaihtoehdoksi tarjottua riviä.

Oikeassa yläreunassa on CommandStrings-niminen neljän tekstikentän joukko. Kolme ylintä kenttää päivittyvät käyttäjän viereisessä puurakenteessa tekemien valintojen perusteella. Ylin kenttä kuvaa käytettävää protokolla, toiseksi ylin valitun viestityypin ja kolmanteen kenttään sijoitetaan valitun viestityypin perusteella saatavilla olevat komennot. Kyseisillä kentillä voidaan antaa myös manuaalisesti komentoja, mutta pääsääntöisesti nämä kentät täydentyvät automaattisesti. Neljännellä tekstikentällä voidaan täydentää muodostettua puurakennetta vapaavalintaisilla arvoilla. Käyttäjä voi esimerkiksi määritellä omia heksadesimaaliarvoja luotavaan EMS-pakettiin. Tehdyt muutokset komentoparametreissa täydennetään kenttien oikealta puolelta löytyvällä ”Apply options”-napilla.

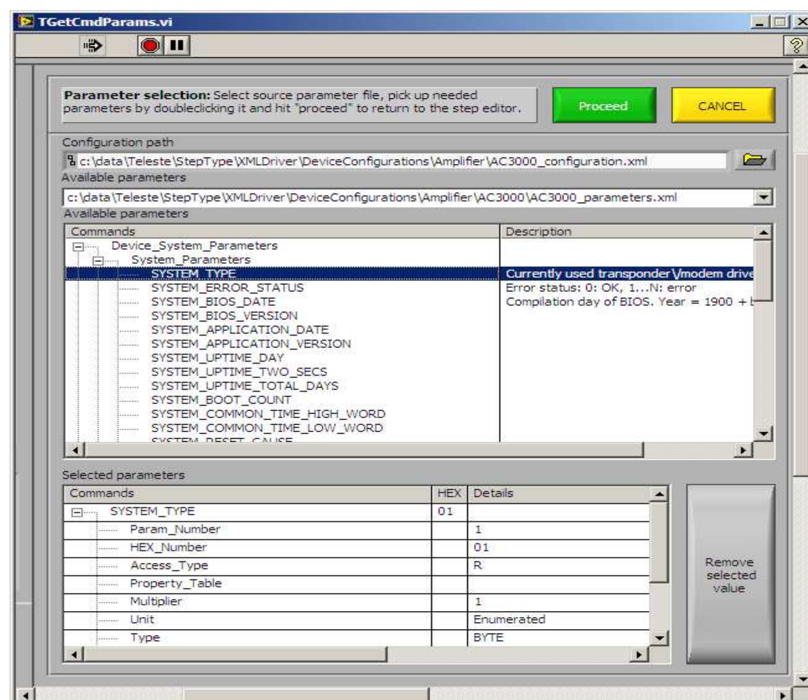
Komentoparametrien alapuolella on tuotekohtaisille parametreille varattu tietorakenne. Parametrit voidaan täyttää joko manuaalisesti tai apuna voi käyttää sille kehitettyä editoria. Parametreista määritellään sen heksadesimaalinen komento tai selkokielinen tunnus, jonka perusteella parametrien tarkemmat tiedot noudetaan tuotekohtaisten parametrien joukosta. Parametreille on vielä lisäksi määritelty seuraavat ominaisuudet joita käytetään apuna niiden noutamisessa ja asettamisessa: arvo (value), kerroin (multiplier), yksikkö (unit), vaihto (swap), tyyppi (type) ja omavalintainen yksikkö (custom unit). Arvo-nimistä kenttää käytetään itse parametrien arvojen asettamiseen kohdetuotteessa. Kerrointa käytetään muuntamaan asetettavan tai noudettavan parametrin suuruusluokkaa. Yksikköä käytetään määrittelemään asettavaa arvoa esimerkiksi tuloksia käsiteltäessä. Vaihtoa käytetään silloin, kun jossain tuotteissa asetettava tai noudettava parametri on tavallisesta poiketen talletettuna Little-Endian -muotoisena bittijonona. Nämä luvut muunnetaan ajurissa Big-Endian-järjestykseen. Tyyppi kuvaa kyseisen parametrin kokoa tavuina ja tunnuksina ovat tekstimuotoiset vastineet, kuten esimerkiksi BYTE ja UINT16. Käyttäjän määrittelemää yksikköä käytetään silloin, jos halutaan esimerkiksi muuntaa vastaanotettu celsius-asteina oleva arvo fahrenheit-asteikolle tai muuntaa saatu arvo megahertseistä kilohertseiksi. Edellä mainitut parametrien

muuttujat kannattaa kuitenkin täyttää käyttämällä siihen tarkoitettu parametriedi-toria, jonka tiedot perustuvat tuotekohtaisiin parametritiedostoihin.

Editori ei ota kantaa syötteiden oikeellisuuteen, mutta viitteellistä tietoa määritys-ten onnistumisesta antaa vasemmassa alareunassa oleva EMS-paketin esikatse-luikkuna. Siinä on esitetty XML-muotoinen ilmentymä luodusta EMS-paketista, ja sen perusteella voidaan tulkita, onko paketti määritelty tarpeeksi hyvin. Määri-tellyn testiaskeleen voi luoda testisekvenssiin painamalla vihreää Proceed-nappia. Editorin suorituksen voi keskeyttää painamalla punaista Stop-nappia.

4.4.2.2 Parametrieditori

Parametrieditoria käytetään apuna määriteltäessä vaadittuja parametreja noudetta-vaksi tai asetetavaksi. Graafisen käyttöliittymän avulla määritellään parametritie-dosto, josta haetaan kaikki parametrit. Parametrieditorin käyttöliittymä on esitetty kuvassa Kuva 4-11.



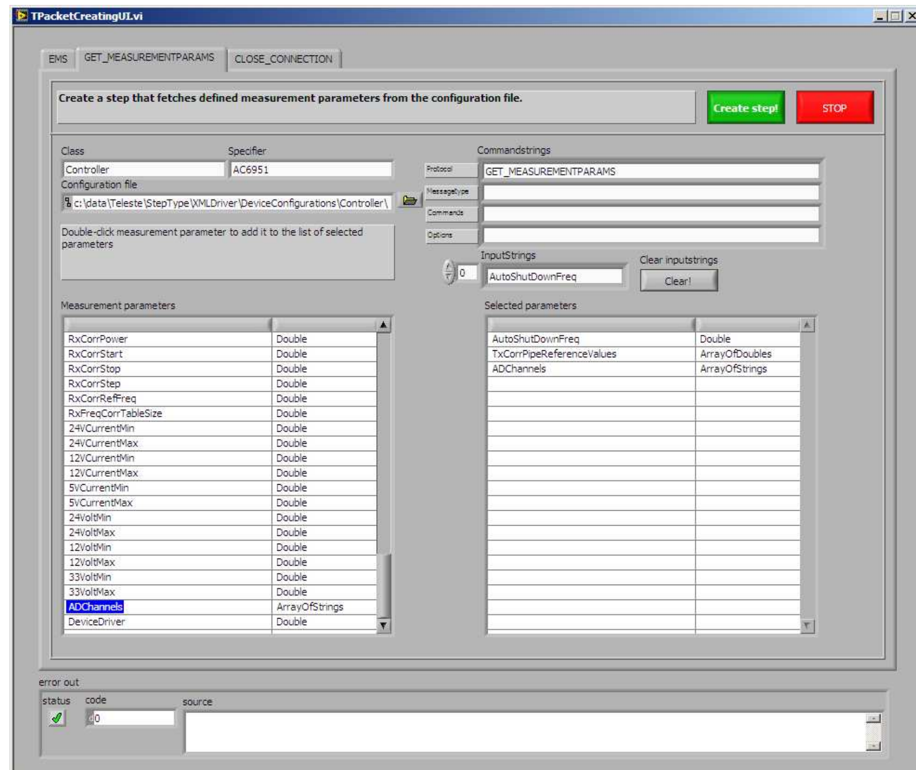
Kuva 4-11 Parametrieditorin käyttöliittymä

Kun editori käynnistetään, on konfiguraatiotiedoston sijainti jo valmiina täytettynä. Sen perusteella voidaan tarjota käyttäjälle lista mahdollisista parametritiedostoista, joista voi valita yhden alapuolella sijaitsevan pudotusvalikon kautta. Valitun parametritiedoston perusteella käyttäjälle luodaan puurakenteinen lista mahdollisista parametreista.

Perusnäkymässä valittavissa olevista parametreista näkyy vain nimi. Puun haaran avaamalla parametreista näytetään parametrikohtaiseen konfiguraatiotiedostoon talletetut muut ominaisuudet. Tuplaklikkaamalla yksittäistä parametria, ilmestyy vastaavan parametrin tiedot sivun alaosassa olevaan parametrilistaan. Parametreja voidaan valita useampia. Valittuja parametreja voidaan myös poistaa aktivoimalla haluttu kenttä listasta sekä painamalla oikealla sijaitsevaa isoa ”Remove selected value”-nappia. Editori lopetetaan valitsemalla joko vihreä Proceed-nappi, jolloin valitut parametrit välitetään testisteppieditorille, tai keskeyttämällä prosessi painamalla keltaista Cancel-nappia.

4.4.2.3 Mittausparametrieditori

Eräät tuotteet omaavat mittausparametreja, joita käytetään tuotteen fyysisten ominaisuuksien kuvaamiseen. Käyttöliittymässä kyseisen testiaskeleen luomiseen käytetyt toiminnot ovat sijoitettu omalle välilehdelleen. Itse laite ei näitä tietoja millään tavalla säilytä, vaan ne on määritelty esimerkiksi tuotekehityksen tai moduulivalmistajan toimesta. Kyseisiä mittausparametreja käytetään erilaisissa muunnoksissa, joissa voidaan korjata tuotteelta mitattuja tuloksia tai säätää tuotteen ominaisuuksia. Kuvassa Kuva 4-12 on esitetty mittausparametrieditorin käyttöliittymä.



Kuva 4-12 Mittausparametrieditorin käyttöliittymä

Käyttöliittymässä on samoja toiminnallisia elementtejä kuin EMS-muotoisen ajurikutsun luomisessa käytetyssä editorissa. Alhaalla vasemmalla on lista kaikista mahdollisista mittausparametreista. Tuplaklikkaamalla yhtä, se ilmestyy valittuja parametreja ilmaisevaan listaan käyttöliittymän oikealle puolelle. Määritellyn testiaskeleen voi luoda painamalla vihreää Proceed-nappia. Editorin suorituksen voi keskeyttää painamalla punaista Stop-nappia. Kyseinen askeltyyppi ei siis muodosta yhteyttä kohdetuotteeseen, vaan sitä käytetään pelkästään arvojen noutamiseen konfiguraatiotiedostosta.

5 JOHTOPÄÄTÖKSET

Ajurin kehitystyöllä pyrittiin tekemään pohjatyö kohti älykkäitä ajuriratkaisuja, joiden kehittäminen ja käyttöönotto sujuisivat vaivattomasti. Testaussuunnittelijoiden tietotasoa XML-tekniikoista voitiin pitää melko alhaisena, joten tämän dokumentin toivotaankin toimivan ohjaavana tietolähteenä tulevaisuuden kehitystyölle älykkäiden ajurimallien parissa. Seuraavissa kappaleissa käsitellään työn tulokset ja pohditaan ajurimallin jatkokehitystarpeita.

5.1 Työn tulokset

Ajuri pyrittiin kehittämään siten, että se ei vaatisi uudelleenohjelmointia aina, kun uudelle tuotteelle luodaan testaussovellus. Ohjelmointityö toteutettiin käyttäen graafista LabView G –ohjelmointikieltä. Ajurin käyttöä helpotettiin kehittämällä TestStand:n omat hallintatyökalunsa.

Ensimmäiseksi luotiin ajuri, jonka kautta kommunikoidaan EMS-protokollaa käyttävien tuotteiden kanssa. Ajuri muodostaa XML-muotoisen EMS-paketin soveltaen viestityypin mukaisia skeematiedostoja, joissa määritellään EMS-protokollan mukaiset pakettirakenteet kysymys- ja vastausviestille. Ajuri käsittelee tuotteilta saadut vastausviestit ja välittää tulokset sitä kutsuneelle taholle. XML-muotoisilla konfiguraatiotiedostoilla pyrittiin siihen, että itse ajuriin sisällytetään mahdollisimman vähän tietoja sen tukemista yhteysmuodoista ja tuotteista. Näin vähennetään ylimääräistä ohjelmointityötä, sillä tarvittavat muutokset voidaan pääasiallisesti tehdä nyt XML-muotoisiin tukitiedostoihin. Ajurin käyttämän kommunikointiprotokollan oletetaan pysyvän toiminnaltaan samanlaisena tulevaisuudessakin. Ajurin toimintalogiikka on kuitenkin rakennettu siten, että se on laajennettavissa käyttämään uusia viestityyppejä pelkästään XML-skeematiedostoja muuttamalla.

Testisekvenssien suunnitellun helpottamiseksi luotiin erityinen testiaskeleditori, jolla voidaan luoda ajuria hyödyntävä testiaskel testaussovellukseen. Sovelluskehittäjän ei enää tarvitse käydä lävitse erillisiä spesifikaatioita kohdelaitteen toiminnasta ja valita niiden perusteella noudettavia ominaisuuksia. Kaikki tiedot ovat nyt lueteltuina joko XML-skeema- tai XML-konfiguraatiotiedostoissa. Testiaskeleditorin avulla voidaan myös muokata noudettavien tai asetettavien parametrien ominaisuuksia, kuten palautettavan tuloksen yksikköä tai pyöristystä.

Asetetut tavoitteet saavutettiin pääpiirteittäin hyvin. Testausympäristöön luotiin uudenlainen ajurimalli, jonka toiminta perustuu XML-muotoisiin konfiguraatiotiedostoihin. Ajurille luotiin oma testiaskeltyyppinsä editoreineen, jonka avulla voidaan helposti tehdä ajuria hyödyntäviä kutsuja testaussovelluksiin. Ajurimallissa määriteltiin myös Telesten käytännöistä poikkeava uudenlainen tapa esittää Telesten käyttämä EMS-protokolla viestityyppeineen ja pakettirakenteineen. Rakennemäärittelyt tehtiin XML-pohjaisilla skeematiedostoilla, joiden perusteella EMS-paketit muodostettiin ajonaikaisesti.

Ajurimalliin sisältyi kuitenkin joitakin ongelmia, jotka vaativat lisää kehitystyötä. Esimerkiksi kommunikointiin käytettävän TeComm-komponentin kanssa oli viiveitä muodostavia ongelmia, jotka pitäisi korjata. Lisäksi ajurimallin toimintaa pitäisi vielä testata enemmän varsinaisten testaussovellusten yhteydessä, jotta saadaan parempi varmuus ajurin toiminnasta ja ylipäätään sen käytettävyydestä testausympäristössä. Myös Telesten tarpeita ajatellen joitakin tärkeitä ominaisuuksia jäi toteuttamatta, joita käsitellään seuraavassa.

5.2 Ajurin jatkokehittäminen

FuncBased-viestityyppi sisältää jo pakettirakenteet SNMP-muotoisen datan hallintaa EMS-protokollan välityksellä. Ajuria pitäisikin jatkokehittää siten, että se voisi hyödyntää kommunikoinnissa konfiguraatiotiedostoissa määriteltyjä SNMP

objektitunnisteita. Ajurimallin laajentamista tukemaan suoraan SNMP-kommunikaatiota voidaan myös harkita.

Poikkeuksellisesta EMS-protokollan määrittelystä johtuen myös Telesten tuotekehityksen suunnasta on osoitettu kiinnostusta ajurimallin käyttömahdollisuuksista muissa Telesten käyttämissä ympäristöissä. Hyödyntämistä edesauttaa se, että Labview-komponentteja voidaan käyttää myös ActiveX-rajapinnan kautta. Tämä mahdollistaa hyvinkin paljon erilaisia käyttöskenaarioita. Yksi tällainen vaihtoehto voisi esimerkiksi olla EMS-protokollan mukaisten viestien käyttö XML-muotoisina web-palvelujen yhteydessä.

Tärkein yksittäinen kaivattu ominaisuus testausympäristön ja siinä käytettävien sovellusten kannalta on kuitenkin mittalaitteiden hallintaan liittyvien toimintojen puuttuminen nyt kehitetystä ajurimallista. Seuraavassa esitellään ajurimallin jatkokehitystarpeita mittalaitteiden asettamien vaatimusten perusteella. Lisäksi käsitellään uuden LabView-version tuomia mahdollisuuksia ajurimallin parantamiseksi.

5.2.1 Tuki GPIB-pohjaisille mittalaiteille

Teleste käyttää tuotteidensa testaamisessa usean eri valmistajan kehittämiä mittalaitteita, joita ohjataan automaattisesti LabView-pohjaisilla testaussovelluksilla. Mittalaitteisiin ollaan yhteydessä GPIB-väylän (General Purpose Interface Bus) kautta. Laitteiden hallinta onnistuu myös Ethernet- ja USB-väylän kautta käyttäen samaa GPIB-käskykantaa. Uudempia mittalaitteita tullaan käyttämään näiden suorituskykyisempien väylien kautta. Eri valmistajien samaa tarkoitusta varten kehitetyt mittalaitteet toimivat testaussovelluksissa hyvin paljon samankaltaisesti toisiinsa verrattuna. Ne saattavat kuitenkin erota toisistaan esimerkiksi komennoissa käytettävien parametrien osalta tai useampia peräkkäisiä toimintoja vaativien operaatioiden suhteen. Ajurimallin yksi laajennuskohde olisikin mittalaitteiden hallinta XML-konfiguraatiotiedostoihin perustuvalla ajurimallilla.

Mittalaiteiden operoinnissa testisekvenssien yhteydessä pyritään siihen, että testisekvenssin tarvitsee tietää testilaitteiston yhteydessä olevista mittalaitteista vain niiden tyypit. Näin voidaan valita mittalaitetyypille sopiva käskykanta, ja ajuri tulkitsee mittalaitekohtaiset erot esimerkiksi yksilöivien komentoparametrien suhteen. Ongelmia ilmenee käytettäessä usean eri valmistajan mittalaitteita, koska laitteiden toimintatavat saman operaation suorittamiseksi saattavat erota toisistaan. Tästä johtuen olisikin erittäin tärkeää kehittää myös XML-pohjainen skriptikieli, jonka avulla pystyttäisiin tekemään yksinkertaisia makroja useampien toimintojen ketjuttamiseen. Tällainen ketju voisi esimerkiksi olla kohdelaitteen säätö mitattujen tulosten perusteella. Tällöin säätökierrosten lukumäärää ei ennalta tunnettaisi ja ajuri toimisi adaptiivisesti säätäen joka kierroksella, kunnes säädöt ovat kohdillaan. XML-skripteihin perustuva toimintalogiikka mahdollistaisi myös sen, että erillistä ohjelmointityötä ei tarvittaisi, vaan muutokset tehtäisiin XML-tiedostoihin. Esimerkiksi NetPDL-tekniikassa käytetään XML-tiedostoissa elementtien niminä ohjelmoinnista tuttuja rakenteita, kuten esimerkiksi if, else, tai switch-case. Tätä tekniikka hyödyntäen voitaisiin kehitellä erillinen käsittelijä, joka muodostaisi makron kaltaisten XML-toimintojen kautta vaaditut operaatiot ajurikutsuina.

5.2.2 Uusi LabView-versio

Työ tehtiin LabView-versiolla 8.6. Myöhemmin on kuitenkin julkaistu uusi versio LabView 2009, jossa on paranneltu useita aiheeseen liittyviä toimintoja. Esimerkiksi rekursio voidaan siinä toteuttaa paljon helpommin kuin vanhemmissa versioissa. Lisäksi uudessa versiossa on panostettu monipuolisemmin XML-työkaluihin, jolloin ei välttämättä tarvitse käyttää ActiveX:n kautta MSXML-metodeja. Myös olio-ohjelmoinnin tekniikoihin on uudemmassa versiossa keskitetty enemmän. Versionmuutos tulee varmasti olemaan tulevaisuudessa ajankohtainen, joten siinä yhteydessä voidaan myös parantaa ajurimallin toimintaa.

LÄHTEET

Alexopoulos, Dimitris & Soldatos, John, 2005. XMLNet: An Architecture for Cost Effective Network Management Based on XML Technologies. Journal of Network and Systems Management, Volume 14, Number 4, December 2005, p.451-477. Springer New York.

Anderson, Paul, 2005. The Performance Penalty of XML for Program Intermediate Representations. Proceedings of the fifth IEEE International Workshop on Source Code Analysis and Manipulation, p.193-201, 2005.

Bagnasco, A., Chirico, M., Scapolla, A.M & Amodei, E., 2002. XML Data Representation for Testing Automation. IEEE AUTOTESTCON Proceedings, p.577-584, 2002.

Baroncelli, Fabio, Martini, Barbara & Castoldi, Piero. 2006. A Robust XML-based Approach for Network Protocols Implementation. International Conference on Transparent Optical Networks 2006 ,Volume 3, November 2006, p.139-142.

Brandin, Chris, 2003. Information Modeling with XML. teoksessa: Chaudhri, Rashid & Zicari. XML Data Management, Chapter 1. Addison Wesley. ISBN 0-201-84452-4.

Bertolino, Antonia, Gao, Jinghua, Marchetti, Eda & Polini, Andrea, 2007. Automatic Test Data Generation for XML Schema-based Partition Testing. Second International Workshop on Automation of Software Test (AST’07) At 29th International Conference on Software Engineering (ICSE’07), Minneapolis, MN, USA, May 20-26, 2007.

Colliander, Janne & Vilola, Marko, 2004. TeComm Interface Specification Version 0.9. Teleste Oyj.

Cooper, Alan & Reimann, Robert, 2003. About Face: The Essentials of Interaction Design. John Wiley & Sons. NewYork. ISBN 0-7645-26413.

Cridlig, Vincent, Abdelnur, H., Bourdellon, J. & State, Radu, 2005. a NetConf Network Management Suite: ENSUITE. IPOM 2005: 5th IEEE International Workshop on IP Operations and Management, Barcelona, Spain, 26-28 October 2005, Volume title: Operations and Management in IP-Based Networks.

Dumbill, Edd, 2001. XML Watch: Bird's-eye BEEP.
Saatavilla: http://www.ibm.com/developerworks/xml/library/x-beep/index.html (Viitattu 10.09.2009).

Emer, Maria Claudia F. P, Vergilio, Silvia Regina & Jino, Mario, 2005. A Testing Approach for XML Schemas. Proceedings of the 29th Annual International Computer Software and Applications Conference, Volume 2, p.57-62, 2005. COMPSAC 2005.

Enns, Rob, 2006. NETCONF Configuration protocol. RFC4741.

Geroimenko, Vladimir & Chen, Chaomei, 2003. Visualizing the Semantic Web. Springer-Verlag. London Berlin Heidelberg. ISBN 1-85233-576-9.

ITU-T, 2000, Principles of Telecommunications Management Network. ITU-T Recommendation M.3010.

Jacobs, Sas, 2006. Beginning XML with DOM and Ajax – From Novice to Professional. Apress. ISBN 978-1-59059-676-0.

Juniper Networks, 2010. JUNOScript API Guide, Release 10.1. Saatavilla: http://www.juniper.net/techpubs/software/junos/junos101/junoscript-guide/junoscript-guide.pdf (Viitattu 22.03.2010).

Klie, Torsten & Strauβ, Frank, 2003. Towards XML Oriented Internet Management. Eight International Symposium on Integrated Network Management, 2003, p.505-518. IFIP/IEEE.

Klie, Torsten & Strauβ, Frank, 2004. Integrating SNMP-agents with XML-based Management Systems. IEEE Communications Magazine, Volume 42, Issue 7, p.76-83, 2004.

Kuusisto, Sami, 2002. Kaapelitelevisioverkon transponderin digitaaliosan suunnittelu ja toteutus. Pro gradu –tutkielma. Turun yliopisto.

Lam, Tak Cheung, Ding, Jianxun Jason & Liu, Jyh-Charn, 2008. XML Document Parsing: Operational and Performance Characteristics. Computer, volume 44, issue 9, s.30-37.

Mazumdar, Subrata, 2007. Architecture and Implementation of Automatic Conversion of SNMP Event Data to XML Based Document.
Saatavilla: http://pubs.research.avayalabs.com/pdfs/ALR-2007-007.pdf (Viitattu 22.03.2010).

Menten, Lawrence E., 2004. Experiences in the Application of XML for Device Management. IEEE Communications Magazine, Volume 42, Issue 7, p.92-100, 2004.

Microsoft, 1995. MS Windows NT Workstation 4.0 Resource Guide: Initialization Files and Registry. Microsoft Corporation. Saatavilla:
http://technet.microsoft.com/en-ie/library/cc722567(en-us).aspx (Viitattu 24.04.2010).

Microsoft, 2010. MSXML SDK Overview. Microsoft Corporation. Saatavilla:
http://msdn.microsoft.com/en-us/library/ms760399%28v=VS.85%29.aspx (Viitattu 24.04.2010).

Mi-Jung, Choi, Hyoun-Mi, Choi, Hong, James W. & Hong-Taek, Ju, 2004. XML-based Configuration Management for IP Network Devices. Communications Magazine, IEEE, Volume 42, Issue 7, p.84-91. July 2004.

Mostéfaoui, Soraya Kouadri, 2008. A Context Model Based on UML and XML Schema Representations. IEEE/ACS International Conference on Computer Systems and Applications, p.810-814, 2008.

Raitis, Teemu, 2002. Testausohjelmisto tilausohjautuvassa toimitusprosessissa. Diplomityö. Teknillinen korkeakoulu.

Risso, Fulvio & Baldi, Mario, 2006. NetPDL: An Extensible XML-based Language for Packet Header Description. Computer Networks: The International Journal of Computer and Telecommunications Networking, Volume 50, Issue 5, April 2006, s.668-706. Elsevier, Amsterdam 2006.

Rose, M., 2001. The Blocks Extensible Exchange Protocol Core. RFC3080.

Saint-Andre, P., 2004. Extensible Messaging and Presence Protocol. RFC3920.

Sharp, Robin, 2008. Principles of Protocol Design. Springer-Verlag. Berlin Heidelberg. ISBN 978-3-540-77540-9.

Sumathi, Sai & Surekha, Paneerselvam, 2007. Labview Based Advanced Instrumentation Systems. Springer Berlin Heidelberg. ISBN 978-3-540-48500-1.

Susi, Matti, 2002. DVX Bus Specification Version 1.7. Teleste Oyj.

Teleste Oyj, 2009. Teleste Presentation. Teleste Oyj. Saatavilla: http://www.teleste.com/dm/file.phtml?id=2298 (Viitattu: 18.03.2010).

Virtanen, Petri, Susi, Matti & Salminen, Rainer, 2003. EMS Protocol Description v1.23. Teleste Oyj.

W3C, 1999a. XSL Transformations (XSLT) Version 1.0. W3C Recommendation. Saatavilla: http://www.w3.org/TR/xslt.

W3C, 1999b. XML Path Language (XPath) version 1.0. W3C Recommendation. Saatavilla: http://www.w3.org/TR/xpath/.

W3C, 2004. Document Object Model (DOM) Level 3 Core Specification Version 1.0. W3C Recommendation. Saatavissa http://www.w3.org/TR/DOM-Level-3-Core/.

W3C, 2006. Extensible Stylesheet Language (XLS) Version 1.1. W3C Recommendation. Saatavilla: http://www.w3.org/TR/xsl/.

W3C, 2008. Extensible Mark-up Language Version 1.0 (Fifth edition). W3C Recommendation. Saatavilla: http://www.w3.org/TR/xml/.

W3C, 2009. The Extensible Stylesheet Language Family (XLS). Saatavilla: http://www.w3.org/Style/XSL/

Walkama, Pekka & Laakkonen, Aapo, 2004. Inside XML-skeema. Edita Prima Oy. Helsinki. ISBN 951-826-665-4.

Zhixia, Zhao, Ziheng, Liu, Yu, Bai & Debao, Xiao, 2007. Design and Implementation of Universal Gateway for XML-based Network Management. International Conference on Wireless Communications, Networking and Mobile Computing, p.5192-5195, 2007, WICOM 2007.

Zhang et al., 2004. VTD-XML: The Future of XML Processing. Saatavilla: http://vtd-xml.sourceforge.net (Viitattu 24.04.2010).