Helsingin yliopisto

Tietojenkäsittelytieteen laitos

Ohjelmistotuotantoprojekti XPerf

Toteutusdokumentti

Tommi Koivula
Antti Levomäki
Juha Mondolin
Timo Suomela

Versio 1.0
11. toukokuuta 2003

Versiohistoria

Versio	Päivämäärä	Kommentti / muutokset	Tekijä
1.0	11.5.2003		Tommi Koivula Timo Suomela Juha Mondolin

Sisällys

1. Johdanto

Tässä dokumenttissa on kuvaus XPerf-ohjelmistotuotantoprojektin toteutusvaiheen kulusta ja sen tuloksena syntyneesta tuotteesta. XPerf-projektin tehtävänä oli toteuttaa MAISA-työkaluun [1] laajennus, jonka avulla MAISA-työkaluun voidaan lukea UML-kaavioita [2] XMI-dokumenteista [3].

Tarkempi kuvaus MAISA-työkaluun tehdyistä uusista ominaisuuksista löytyy määrittelydokumentista [4] ja yksityiskohtainen kuvaus toteutetusta arkkitehtuurista löytyy suunnitteludokumentista [5].

Luvussa 2 esitellään toteutettujen lisäominaisuuksien suhdetta niille asetettuihin vaatimuksiin. Luvussa 3 on esitelty ne vähäiset tarkennukset arkkitehtuuriin, joita toteutusvaiheessa on tehty. Luvussa 4 esitellään lyhyesti kunkin toteutetun osajärjestelmän toteutusperiaatteet, jotka eivät selviä pelkästään JavaDoc-dokumentaatiosta [liite A].

2. Vaatimusten toteutuminen

2.1. Toiminnallisten vaatimusten toteutuminen

XMI.01

Pystyy lukemaan ArgoUML:llä tuotettuja XMI-dokumentteja. Dokumenteista luetaan vain tiedot, joita MAISA-työkalu tarvitsee.

Jos XMI-dokumentissa on viite kielioppiin, tulee viitatun DTD:n olla saatavilla ja dokumentin oikeellisuus tarkastetaan. Jos viitettä kielioppiin ei ole, oikeellisuutta ei myöskään tarkasteta.

Mai.01

MAISA-järjestelmä osaa tiedostotyypin perusteella valita oikean mekanismin tiedostonlukua varten. Jokainen lukumoduuli vastaa itse ”tietoisuudesta”, osaako se lukea annettua tekstivirtaa vai ei. XMI- ja prolog-lukumoduulin kohdalla tunnistaminen tapahtuu tiedostotyypin sisällön perusteella, ei tiedostonnimestä.

Mai.02

MAISA-työkaluun ladataan ajonaikaisesti käytettävissä olevat lukumoduulit, jolloin kaikki tiedostotyypit, joita ladutut lukumoduulit tukevat, voidaan lukea.

Mai.06

Jos XMI-dokumenttia jäsennettäessä kohdataan virheitä, lopetetaan jäsentäminen välittömästi ja kaikki MAISA-työkalun jo täytetyt tietorakenteet tyhjennetään.

2.2. Ei-toiminnallisten vaatimusten toteutuminen

Dok.01

Dokumenteista käyttöohje ja ylläpitodokumentti on kirjoitettu englanniksi. Muut dokumentit on kirjoitettu suomeksi.

Yll.01

MAISA-työkalun eri tiedostotyyppien luku on muutettu modulaariseksi. Ts. jokaista tiedostotyyppiä vastaa oma moduulinsa. Moduulit ladataan MAISA-työkaluun ajonaikaisesti ja kaikki niiden tukemat tiedostotyypit ovat avattavissa MAISA-työkaluun.

XMI- ja prolog-tiedostotyyppejä vastaavien uusien tiedostotyyppien lisääminen jatkossa ei vaadi muutoksia MAISA-järjestelmään.

Koe.01

Kokeellinen osuus korvataan asiakkaan pyynnöstä tutkimuksella XMI-tuen laajentamisesta tukemaan myös XMI-versiota 1.2. Laajentamista tutkitaan sen teknisen toteuttamisen kannalta. Myös laajennuksen työmäärästä tehdään arvio.

2.3. Käytettävyysvaatimusten toteutuminen

Mai.03

Maisa tunnistaa luettavan tiedoston formaatin automaattisesti ilman käyttäjän toimenpiteitä. Tämä toteutuu vaatimuksen Mai.01 yhteydessä kuvatulla tavalla.

Mai.04

Tiedostonlukuoperaation toiminta ilmaistaan käyttäjälle Javan valmiilla käyttöliittymäkomponentilla.

Käyttäjä saa vain tiedon, että lukuoperaatio on käynnissä, mutta ei operaation jäljellöolevasta kestosta tai prosentuaalisesta osuudesta. Tämä siksi, ettei monien tiedostotyyppien (esim. XMI-dokumentit) yhteydessä tämä tieto ole saatavilla.

Mai.05

Jos XMI-dokumentin jäsennyksessä törmätään virheisiin, ilmoitetaan tästä välittömästi käyttäjälle. Virheen sattuessa jäsennys keskeytetään. Virhetilanteiden raportointia käyttäjälle ei muutettu.

3. Tarkennukset suunniteltuun arkkitehtuuriin

Suunnitteluvaiheessa suunniteltuun arkkitehtuuriin ei tehty toteutusvaiheessa enää merkittäviä muutoksia.

Yksityiskohtia, jotka tarkoituksella jätettiin suunnitteludokumentista kirjaamatta, esitellään lyhyesti luvussa 4.

4. Osajärjestelmien toteutuksen kuvaus

Tässä luvussa esitellään lyhyesti kunkin toteutetun osajärjestelmän toteutusperiaatteet, jotka eivät selviä pelkästään JavaDoc-dokumentaatiosta [liite A]. Tällaisia toteutusperiaatteita ovat esimerkiksi

- *yksityiset metodit ja attribuutit, joiden tarkoitus ei ole itsestäänselvä.*
- *perustelut valitulle luokan sisäiselle rakenteelle*

4.1. Lukumoduulin valintajärjestelmä

Lukumodulin valintajärjestelmä (ja itse tiedoston luku) on vaatinut muutoksia *MAISA_tool* ja *MAISA_system* -luokkiin. *MAISA_tool* luokan metodissa *update(Observable, Object)* passivoidaan käyttöliittymä ja ajetaan sopiva *MAISA_tool.ResourceLoader* luokasta perityn luokan ilmentymä, projektikansion kohdalla *MAISA_tool.ProjectLoader*, kaaviotiedoston kohdalla *MAISA_tool.DiagramLoader* -luokan ilmentymä. Resurssit ladataan erillisessä säikeessä, jotta käyttöliittymä ei pysähdy latauksen ajaksi ja *update(…)* metodi palaa heti käynnistettyään säikeen. Valittu lataaja ensimmäisenä käynnistää ajastinsäikeen, joka odotettuaan kolme sekunttia avaa animaation sisältävän

indikaattoridialogin (*IndeterminateProgressMonitor*). Jos lataaminen on valmis ennen ajastimen laukeamista, dialogia ei näytetä. Kun lataus on suoritetty loppuun tai se on käyttäjän (tai virheen) toimesta keskeytetty, aktivoidaan käyttöliittymä. Jos lataus on viety onnistuneesti loppuun asti, lisätään luettu projekti tai kaavio(t) käyttöliittymän projektinäkymään.

ResourceLoader luokka toteuttaa **Template Method** -suunnittelumallin vaikkakin abstraktoitu algoritmi on hyvin yksinkertainen ja ainoa abstrakti askel on metodi *loadResource(BitSet)*. Luokan *run()* metodi käynnistää ensimmäisenä indikaattorin ajastimen jonka jälkeen se kutsuu *loadResource(BitSet)* metodia. Metodin parametrin tarkoitus on toimia viestikanavana lataajalle käyttäjän aiheuttaman keskeytyksen tapahtuessa. Keskeytyksen tapahtuessa asetaan välitetyn *BitSet*-ilmentymän bitti *MAISA_system.LOAD_CANCELLED_BIT* pystyyn. Metodi palauttaa onnistuessa *DefaultMutableTreeNode* -olion joka sisältää luetut kohteet. Odotetut poikeukkset *MaisaParseException* ja *IOException* siepataan ja virheilmoitukset otetaan talteen. Viimeisenä *run()* metodissa kutsuttaan *evaluateResult()* metodia, josssa ensimmäisenä pysäytetään indikaattorin laukaisuajastin. Jos lukijassa ei ole tapahtunut virhettä eikä käyttäjä ole keskeyttänyt sitä, liitetään *loadResource(BitSet)* metodin palauttama solmu projektinäkymän puuhun, jonka jälkeen aktivoidaan käyttöliittymä. Jos lukijassa on tapahtunut virhe niin siihen liittyvä virheilmoitus näytetään pääikkunan yläpalkissa.

DiagramLoader luokka lataa yksittäisen kaaviotiedoston ja saa konstruktorissa luettavan tiedoston nimen ja projektinäkymän puun solmun, jonka alle tiedostosta luetut kaaviot lisätään. Solmuviite voi olla null, misää tapauksessa kaaviot luetaan uuden projektin alaisuuteen. Luokan *loadResource(BitSet)* -toteutus kutsuu *MAISA_system* luokan *openDiagram(String, BitSet)* tai *openDiagram(String, DefaultMutablTreeNode, BitSet)* metodia riippuen siitä, onko konstruktorissa annettu solmuviite *null* vai ei ja välittää sen paluuarvon eteenpäin.

ProjectLoader luokka lataa kokonaisen hakemiston kaikki tuetut kaaviotiedostot ja saa konstruktorissa luettavan hakemiston nimen. Luokan *loadResource(BitSet)* -toteutus kutsuu *MAISA_system*-luokan *openProject(String, BitSet)* metodia ja valittää sen paluuarvon eteenpäin.

Projektihakemisto luetaan *MAISA_system*-luokan *openProject(String, BitSet)* metodissa. Metodi käy silmukassa läpi kaikki annetun hakemiston tiedostot jotka eivät ole itse hakemistoja ja kutsuu kunkin kohdalla *parseDiagramFile(File, DefaultMutableTreeNode, String)*-metodia, joka tulkitsee tiedoston sisällön. Silmukan alussa tarkistetaan ensin onko lukuoperaatio keskeytetty, eli onko parametrina annetun *BitSet*-ilmentymän bitti *MAISA_system.LOAD_CANCELLED_BIT* pystyssä. Jos näin on niin metodi palautta heti null arvon. Kun kaikki tiedostot on näin käyty läpi, käydään tiedostot uudestaan läpi ja luetaan kaikki kaavioihin ja projektiin liittyvät resurssitiedostot. Resurssien lukemiseen käytettyyn logiikkaan ei ole tehty muita muutoksia kuin suunnitteludokumentissa mainittu kuvatiedostojen assosiaatio luettuun kaavioon ja kaaviotiedostojen lukusilmukassa käytetty keskeytyksen tarkistus. Jos lukua ei ole keskeytetty niin viimeisenä kutsutaan luodun *Cproject*-luokan ilmentymän *updateGUI()* metodia, johon on siiretty logiikka joka näyttää tarvittaessa luetun sekvenssikaavion viestien aikavaatimusten rajojen määrittelyyn tarkoitetun ikkunan.

Yksittäinen kaaviotiedosto luetaan *MAISA_system*–luokan *openDiagram(String, BitSet)* metodilla, jos sitä ei lueta jo ennestään luetun projektin alaisuuteen. Metodissa luodaan uusi *CProject*-luokan ilmentymä käyttäen luettavan tiedoston hakemiston nimeä projektin nimenä ja kutsutaan metodia *openDiagram(String, DefaultMutableTreeNode, BitSet)*.

Yksittäinen kaaviotiedosto luetaan *openDiagram(String, DefaultMutableTreeNode, BitSet)* -metodilla. Metodi kutsuu *parseDiagramFile(File, DefaultMutableTreeNode, String)* -metodia jonka jälkeen liitetään luettuihin kaavioihin mahdolliset kuvatiedostot samasta

hakemistosta. Logiikka tässä on sama kuin projektihakemistoa lukiessa *openProject(String, BitSet)* -metodissa.

Jokainen kaaviotiedosto luetaan *parseDiagramFile(File, DefaultMutableTreeNode, String)* -metodissa. Metodissa pyydetään ensin *DocumentReaderFactory*-luokalta sopivaa lukijaa parametrina saadulle tiedostolle. Jos *DocumentReaderFactory*-luokan *getDocumentReader(File)* -metodi palauttaa jonkin *DocumentReader*-luokan ilmentymän, kutsutaan sen *read(BufferedReader, PrintWriter, CProject)* -metodia. Seuraavaksi käydään läpi kaikki *CProject*-olion kaaviot tekemällä kaaviolistasta kopio ja poistamalla kopiosta kaikki ne kaaviot, joita jo vastaa yksi projektia vastaavan puusolmun lapsista. Jäljellejääneille kaavioille asetetaan projekti omistajaksi ja lisätään projektia vastaavaan puusolmuun yksi lapsi kutakin kaaviota kohden.

DocumentReaderFactory-luokka toteuttaa **Singleton** ja **Factory Method** -sunnittelumallit. Luokalta pyydetään sopivaa lukijaa *getDocumentReader(File)* -metodilla. Metodi iteroi lukijalistan ylitse ja tarjoaa tiedostoa luettavaksi jokaiselle lukijalle vuorollaan kutsumalla tämän *canRead(BufferedReader)* -metodia kunnes jokin näistä palauttaa arvon true. *DocumentReaderFactory*-luokan konstruktorissa kutsutaan *loadReaderList()* -metodia, joka lataa käytettävissä olevat lukijaluokat. Lukijaluokat luetaan javan *Properties*-formaatissa olevast konfiguraatiotiedostosta jonka kullakin rivillä määritellään yksi lukijaluokka siten, että attribuutin nimi on symboolinen nimi lukijalle ja attribuutin arvo on lukijaluokan täydellinen nimi. Konfiguraatiotiedoston sijainti selvitetään seuraavalla tavalla:

- Jos ympäristömuuttuja ***maisareaders.configuration*** on määritelty, sen arvo yritään tulkita konfiguraatiotiedoston URL-viitteenä.

- Jos yllä mainittua ympäristömuuttujaa ei ole asetettu tai sen arvoa ei voida tulkita URL- viitteenä, yritetään ladata ***maisareaders.properties*** niminen tiedosto luokkapolun varrelta käyttäen *Class.getResourceAsStream(String)* -metodia.

- Jos yllä mainitut yritykset epäonnistuvat, yritetään avata ***maisareaders.properties*** niminen tiedosto nykyisestä työhakemistosta.

4.2. Prolog-lukumoduuli

Tähän luokkaan on suunnitellusti siirretty prolog-muotoisten kaavioiden lukuoperaatiot, jotka aiemmin sijaitsivat CDiagram-luokan laajennoksissa sekä CProject-luokan read()-metodi että DocumentReader-rajapintaan kuuluvat operaatiot.

Jokaista kaaviota vastaava lukuoperaatio on tehty omaksi julkiseksi metodikseen. Alkuperäisissä lukumetodeissa olleet viittaukset kaavioluokan sisäisiin tietorakenteisiin on muutettu tarvittaessa set- ja get-metodien kutsuiksi. Osa näistä metodeista oli implementoitu luokkiin jo aiemmin, osa lisättiin tarpeen mukaan.

Prolog-lukumoduuliin siirrettyä entistä CProject-luokan read()-metodia on muutettu siten, että kuvatiedostojen lukeminen ym. toiminnallisuus on siirretty MAISA_system-luokkaan .

4.3. XMI-lukumoduuli

XMIDocumentReader

XMIDocumentReader-luokan *canRead()*-metodi lukee SAX-jäsennintä hyväksi käyttäen annettua virtaa, kunnes se löytää XMI-elementin tai virheellisen kohdan dokumentissa (kohdan, joka ei ole XML:n mukainen). Jos XMI-version on 1.0, palauttaa metodi *true,* muuten *false*. Koska SAX-jäsennin keskeyttää dokumentin jäsentämisen heti kun se huomaa virheitä dokumentissa, ei virtaa lueta pitkään binääri- ja ei-XML-dokumenttien

kohdalla. XML-dokumenttien, jotka eivät ole XMI:tä, kohdalla lukeminen jatkuu koko dokumentin läpi. Tämä tietysti, varsinkin isojen tiedostojen kohdalla, voi hidastaa projektien avaamista Maisa-järjestelmään. Näin saavutetaan kuitenkin toimivuus kaikilla mahdollisilla XMI-dokumenteilla.

canRead-metodin toiminta epäonnistuu, jos SAX-jäsennintä ei saada alustettua. Tämä voi tapahtua, koska SAX-jäsennin ladataan ajonaikaisesti. Jos alustus ei onnistu, heittää metodi *IOException*-poikkeuksen, jonka viesti on *"Cannot initialize XML-parser"*.

XMIDocumentReader-luokan *read*-metodi alustaa DOM-jäsentimen ja jäsentää annetusta virrasta dokumentin *Document*-olioksi, jonka avulla dokumentti käsitellään hierarkisesti. Metodi luo myös *TreeWalker*-olion ilmentymän, joka laitetaan osoittamaan *Foundation.Core.Model*-elementtiin ja ko. elementin kiinostavasta sisällöstä tehdään malli käyttäen hyväksi *ModelNode*–luokkaa. Näiden avulla alustetaan *XMIParser*, jolla kunkin löydetyn hierarkian ylimmän tason elementti välitetään sitä vastaavalle lukijaluokalle, Oikea lukijaluokka, *XMIDiagramReader*-rajapinnan ilmentymä, valitaan käyttäen *XMIDiagramReaderFactory*-luokkaa.

Multiplicity-elementtien luku lisättiin *read*-metodiin johtuen argoUML:n oudosta tavasta tallentaa niitä XMI-dokumenttiin. Koko dokumenttin käydään läpi SAX-jäsennintä käyttäen ja *Multiplicity*-elementit luetaan staattiseen taulukkoon, josta kaavioiden lukijat voivat tarvittaessa lukea niitä. SAX-jäsennyksessä käytetään hyväksi *DefaultHandler*-luokan ilmentymää *MAISA.reader.xmi.MultiplicityHandler*.

Jotta yllä mainittu *Multiplicity*-elementtien lukeminen pystyttiin käytännössä lisäämään, jouduttiin *read*-metodin parametrina saatu tiedostovirta tallentamaan ensin levylle väliaikaistiedostoon, josta se voidaan lukea useammassa otteessa. Tämä tapahtuu *XMIDiagramReader*-luokan metodien *saveStreamToDisk*, *getTempStreamFromDisk* ja *deleteStreamTempFile* avulla.

XMIParser ja ModelNode

*XMIParser*in luomiseksi tarvitaan *org.w3c.dom.traversal.TreeWalker* –olio, joka osoittaa dokumentin johonkin elementtiin. XMIParser tarvitsee myös mallin. Malli annetaan *XMIParser*ille antamalla mallin juurisolmu (*ModelElement)*. Mallin juurisolmun elementin on oltava sama kuin annetun *TreeWalker*-olion osoittama dokumentin elementti. Jos näin ei ole *XMIParser*-olion *parse()*-metodin kutsumisen hetkellä, heittää *XMIParser XMIParseException* poikkeuksen.

Jäsentämisen mallia tehtäessä *ModelNode*-olioilla, on huomioitava, että jokaiselle mallin solmuista on syytä antaa yksilöllinen nimi, jotta solmut voidaan käsittelijöissä tunnistaa toisistaan nimien perusteella.

Mallia ei voida enää muuttaa, kun XMI-jäsentimen *parse()*-metodia on kutsuttu. Tämä siksi, että jäsentimen toimintaa muuttuu ennustamattomaksi, jos mallin tehdään sen toiminnan aikana muutoksia. Jos *parse()*-metodin kutsumisen jälkeen yritetään käyttää *ModelNode*-olion metodeita jotka muuttavat sen tilaa, heitetään ajonaikainen poikkeus *UnmodifiableException*. Tätä ajonaikaista poikkeusta ei tarvitse ottaa kiinni. Jos mallia halutaan käyttää useampien dokumenttien jäsentämiseen, on malli luotava aina uudelleen jäsentämisen jälkeen.

Malli voi sisältää myös tiettyjä rekursiivisia solmuja, ts. solmuja, joiden lapsina on solmu itse. Tällöin dokumentista voidaan lukea rajoittamaton määrä sisäkkäisiä samoja elementtejä. Sisäkkäisten elementtien määrää ei tällöin kuitenkaan voi eri tasoilla määritellä, vaan rajat elementtien lukumäärille ovat samat jokaisella tasolla. Esimerkki sallitusta rekursiivisesta mallista:

```
ModelNode root = new ModelNode(”root”,”rootElem”);
ModelNode child = new ModelNode(”child”,”childElem”);
```

```
root.addChild( child, Limit.ONE );
child.addChild( child, Limit.ZERO_TO_FINITE );
```

Rekursiiviset solmut, jossa solmu ei ole itsensä välitön lapsi, **eivät ole** sallittuja. Esimerkiksi seuraava rakenne ei ole sallittu:

```
ModelNode root = new ModelNode("root","rootElem");
ModelNode child = new ModelNode("child","childElem");
root.addChild( child, Limit.ZERO_TO_ONE );
child.addChild( root, Limit.ZERO_TO_FINITE );
```

Jos yo. mukaisia ei-sallittuja malleja tehdään, *XMIParser* ei huomaa rekursiota vaan solmun lapsissa vierailevat *ModelNode*-luokan metodit jäävät ikuisen silmukkaan.

XMIParser-luokalla on myös *getTaggedValueXXX* –metodit toteutettuna kuten suunnitteludokumentissa mainitaan, mutta niitä laajennettiin palauttamaan kaikki haettua avainta vastaavat arvot taulukkona, koska elementillä voi olla samaa taggedValue-avainta useita.

Toisin kuin suunnitteludokumentissa mainitaan [5 s.10], *XMIParser* ei käytä *NodeFilter*-rajapintaa elementtien poisrajaukseen, koska *XMIParser*in rajapinta DOM-jäsentimeen tapahtuu jo olemassaolevan *TreeWalker*-olion kautta ja *NodeFilter* on rekisteröitävä jo *TreeWalker*-olion konstruktorissa. Käytännössä *NodeFilterin* alustaminen tapahtuu siis *XMIDocumentReader*-luokasta, jossa *TreeWalker*-olio luodaan. Toiminnallisuus on kuitenkin tässä täsmälleen sama kuin suunnitteludokumentissa esitetyssä.

NodeActionListener ja XMlNode

NodeActionListener-rajapinta on toteutettu kuten suunnitteludokumentissa mainitaan [5 s.10], eikä siihen ole tarkennettavaa.

XMICollaborationReader

Ei muutoksia suunniteltuun toteutukseen nähden.

XMIClassDiagramReader

Luokkakaavion *actionEvent()*-metodi käyttää hyväkseen XML-jäsentimen *TreeWalker*-oliota dokumentissa liikkumiseen, vaikka se ei olekaan suotavaa. Tämä on kuitenkin välttämätöntä muutamissa kohdinä XML:n rakenteellinen monimutkaisuuden takia. *TreeWalker*-oliota käytetään suoraan seuraavissa kohdissa:

- Sisäkkäiset pakkaukset
- Luokan liittäminen omistavaan pakkaukseen
- Pakkauksen sisällä olevat muut kaaviot (esim. tilakaavio tai aktiviteettikaavio)

ArgoUML tallentaa omituisesti XMI-dokumenttiin *Multiplicity*-elementit. Niiden tallentaminen tapahtuu siten, että ensimmäinen esiintymiskerta jotakin tiettyä monikertaa tallennetaan sellaisenaan ja lopuissa käytetään vain viitettä ensimmäiseen. Näin ollen assosiaatioiden monikerrat voivat viitata ihan mihin tahansa dokumentissa, jolloin niitä ei välttämättä pystytä lukemaan. Esimerkiksi Rational Rosessa tätä ongelmaa ei ole.

Ongelma korjattiin siten, että koko dokumentin kaikki *Multiplicity*-elementit luetaan jo *XMIDiagramReader*-luokan *read*-metodissa staattiseen hajautustauluun. Luokkakaavion assosiaatioiden lukemisessa viitatut monikerrat luetaan tästä hajautustaulusta. Näin kaikki dokumentit pystytään varmasti lukemaan.

XMIActivityDiagramReader

XMIActivityDiagramReader-luokka on toteutettu kuten suunnitteludokumentissa mainitaan. Koska tuettavalla CASE-väline ArgoUMLla ei voida tuottaa kaikkia laajennetun UML:n ominaisuuksia, kuten esimerkiksi sisäkkäisiä aktiviteettikaavioita, on näiden tuki jätetty pois. Nämä osiot ovat kuitenkin helposti lisättävissä mikäli se on tarpeen.

XMIStateMachineReader

XMIStateMachineReader-luokan *actionEvent(XMINode)* -metodin toteutus perustuu tilakoneeseen, jolla on neljä mahdollista tilaa, perustila (*STATE_DEFAULT*), ja yksi tila jokaista luettavaa elementtiä kohden: *STATE_IN_MACHINE* (itse tilakaavio), *STATE_IN_VERTEX* (yksittäinen kaavion tila) ja *STATE_IN_TRANSITION* (kaavion tilojen välinen siirtymä). Jokaisella kutsulla *actionEvent(XMINode)* lukee työstettävän *XMINode*-arvon talteen aiemmin *buildModel()* -metodin avulla rakennettua mallia tulkiten. Lopuksi metodissa kutsutaan *shiftState(int)* -metodia joka tulkitsee annetun parametrin avulla onko luku siirtynyt kahden elementin välisen rajan yli. Jos näin on, yritetään luoda viimeisen tilan mukainen elementti ja siirrytään uuteen tilaan.

XMIDiagramReaderFactory

Ei muutoksia suunniteltuun toteutukseen nähden.

4.4. Tiedostonluvun käyttöliittymä

Käyttäjälle näkyvä tiedostonlukuindikaattori on toteutettu luokassa *IndeterminateProgressMonitor*. Suunnitteludokumentissa mainitusta Java 1.4 ympäristön mukana tulevasta *ProgressMonitor*-luokan käytöstä luovuttiin, sillä se ei tukenut indikaattoria tehtäville, joiden kokoa ei tiedetä. *IndeterminateProgressMonitor*-luokkalla ei ole mitään omia metodeja. Konstruktorissa luokka (joka perii *JDialog*- luokasta) lisää

itseensä *JProgressBar* olion ja sen alle keskeytysnapin. Nappiin liitetään konstruktorissa annettava kuuntelia.

Lähteet

[1] MAISA, Metrics for Analysis and Improvement of Software Architectures http://www.cs.helsinki.fi/group/maisa/

[2] Unified Modelling Language (UML) version 1.3 spesifikaatio, Copyright (C) 1997-2003 Object Management Group, Inc., [http://www.omg.org/cgi-bin/doc?formal/00-03-01].

[3] XML Metadata Interchange (XMI) version 1.0, Copyright © 1997-2003 Object Management Group, Inc., [http://www.omg.org/cgi-bin/doc?formal/00-06-01]

[4] Xperf, Xperf –ohjelmistotuotantoprojektin määrittelydokumentti, 2003 [http://www.cs.helsinki.fi/group/maisa/xperf/doc/Maarittelydokumentti_1.1.6.PDF]

[5] Xperf, Xperf –ohjelmistotuotantoprojektin suunnitteludokumentti, 2003 [http://www.cs.helsinki.fi/group/maisa/xperf/doc/Suunnitteludokumentti_1.0.4.PDF]

Liite A – JavaDoc-kuvaukset toteutetuista luokista