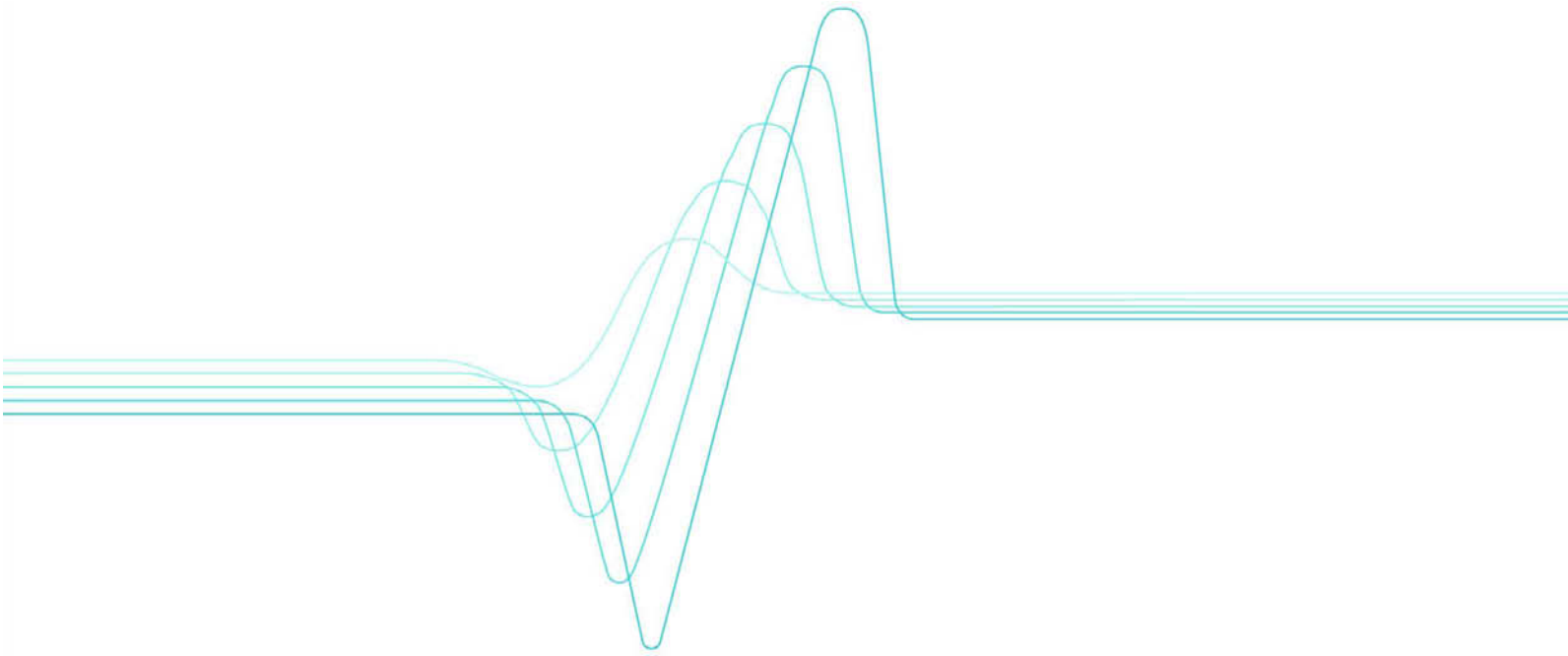
ESPOO 2006 VTT TIEDOTTEITA 2331

Antti Niskanen

Työkalu luotettavuuden mallipohjaiseen analysointiin



VTT TIEDOTTEITA – RESEARCH NOTES 2331

Työkalu luotettavuuden mallipohjaiseen analysointiin

Antti Niskanen

ISBN 951–38–6776–5 (nid.)
ISSN 1235–0605 (nid.)

ISBN 951–38–6777–3 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN 1455–0865 (URL: http://www.vtt.fi/publications/index.jsp)

JULKAISIJA – UTGIVARE – PUBLISHER

VTT, Vuorimiehentie 3, PL 1000, 02044 VTT
puh. vaihde 020 722 111, faksi 020 722 4374

VTT, Bergsmansvägen 3, PB 1000, 02044 VTT
tel. växel 020 722 111, fax 020 722 4374

VTT Technical Research Centre of Finland, Vuorimiehentie 3, P.O.Box 1000, FI-02044 VTT, Finland
phone internat. +358 20 722 111, fax + 358 20 722 4374

VTT, Kaitoväylä 1, PL 1100, 90571 OULU
puh. vaihde 020 722 111, faksi 020 722 2320

VTT, Kaitoväylä 1, PB 1100, 90571 ULEÅBORG
tel. växel 020 722 111, fax 020 722 2320

VTT Technical Research Centre of Finland, Kaitoväylä 1, P.O. Box 1100, FI-90571 OULU, Finland
phone internat. +358 20 722 111, fax +358 20 722 2320

Toimitus Anni Kääriäinen

Otamedia Oy, Espoo 2006

Niskanen, Antti. Työkalu luotettavuuden mallipohjaiseen analysointiin [A tool for model-based reliability analysis]. Espoo 2006. VTT Tiedotteita – Research Notes 2331. 58 s.

Avainsanat model-based analysis, analysis tools, software quality, software systems, reliability, requirements, software architecture, Unified Modelling Language, testing

Tiivistelmä

Mallipohjaisella analysoinnilla tarkoitetaan ohjelmiston laadun arviointia, joka perustuu ohjelmiston arkkitehtuurimalliin, joka kuvaa ohjelmiston rakenteen ja sen käyttäytymisen. Arkkitehtuuritason analysointi tehdään ohjelmistokehityksen alkuvaiheessa, jolloin mahdollisten ongelmakohtien korjaaminen on yksinkertaisempaa ja halvempaa verrattuna toteutetun ohjelmiston korjaamiseen.

Tässä työssä kehitetään ja toteutetaan työkalu ohjelmiston arkkitehtuuritason luotettavuuden analysointiin. Työkalun tarkoituksena on helpottaa ja nopeuttaa olemassa olevan analysointimenetelmän käyttöä, jotta ohjelmistoarkkitehti pystyisi näkemään, toteuttaako kohteena olevan järjestelmän arkkitehtuuri sille asetetut luotettavuusvaatimukset. Työkalun suorittamaa analysointia varten kehitettiin erilliset mallit yksittäisten ohjelmistokomponenttien ja koko järjestelmän luotettavuuden analysointiin hyödyntäen standardia graafista kuvaustapaa. Nämä mallit kuvattiin osana kohdejärjestelmän arkkitehtuurimallia.

Arkkitehtuurimallien kuvaamiseen käytettiin Sparx Systemsin Enterprise Architect -mallinnustyökalua, jonka kautta kehitettävä työkalu pystyy lukemaan ja käsittelemään arkkitehtuurimallia hyödyntämällä mallinnustyökalun tarjoamaa ulkoista ohjelmointirajapintaa. Työkalun kehitysympäristönä käytettiin Microsoftin Visual Studio .NET 2003:a ja työkalu toteutettiin C#-ohjelmointikielellä.

Yhteenvetona voidaan todeta, että teknisesti luotettavuusanalyysi onnistui esimerkkinä käytetystä arkkitehtuurimallista tässä työssä kehitetyn työkalun avulla, mutta tarvitaan huomattavasti jatkokehittelyä, ennen kuin työkalua voidaan käyttää teollisuudessa.

Niskanen, Antti. Työkalu luotettavuuden mallipohjaiseen analysointiin [A tool for model-based reliability analysis]. Espoo 2006. VTT Tiedotteita – Research Notes 2331. 58 p.

Keywords model-based analysis, analysis tools, software quality, software systems, reliability, requirements, software architecture, Unified Modelling Language, testing

Abstract

Model-based analysis is about analysing software quality from the architectural models, which describe the structure and behaviour of a system. Architectural level analysis is realised in the early development phase of the software system, when the changes caused by potential problems of a system are not as complex and expensive to make as correcting the source code.

In this thesis, a tool for analysing reliability of the software system at architectural level is developed and implemented. The purpose of the tool is to support the usage of an existing reliability analysis method and assist an architect to validate whether or not the quality requirements are met in the system architecture. For perform the analysis by the tool, separate models were developed for analysing the reliability of single software components and the whole system using a standard graphical notation. These models were described as part of the system's architectural model.

Sparx System's Enterprise Architect was used as a modelling tool for the representing the architectural models. The analysis tool was able to access the architectural models through Enterprise Architect's external application interface. For the development environment, Microsoft's Visual Studio .NET 2003 was used, and the tool was implemented using C#.

In summary, reliability analysis of the example architecture succeeded technically with the tool developed in this thesis. However, further development is considerably needed before industrial use of the tool.

Alkusanat

Tämä diplomityö tehtiin VTT Elektroniikan sulautettujen ohjelmistojen tutkimusalueella ohjelmistoarkkitehtuurit-ryhmässä. Työ on toteutettu FAMILIES- (FAct-based Maturity through Institutionalisation Lesson-learned and Involved Exploration of System-family engineering) / ITEA-projektissa osana Eureka Σ! 2023 -ohjelmaa.

Ensiksi haluaisin kiittää Anne Immosta (VTT) työn ohjauksesta ja rakentavista kommenteista. Haluaisin kiittää myös tutkimusprofessori Eila Niemelää (VTT) hyödyllisistä neuvoista ja ideoista sekä professori Jukka Riekkiä ja professori Tapio Seppästä Oulun yliopistosta työn valvomisesta ja työn rakenteen tarkastamisesta.

Erityisesti haluaisin kiittää vaimoani Anua hänen osoittamastaan rakkaudesta ja tuesta työn tekemisen aikana sekä tytärtäni Kiiaa hänen iloisuudestansa ja etenkin niistä öistä, jolloin sain nukkua rauhassa.

Oulussa, 20.6.2005

Antti Niskanen

Sisällysluettelo

Tiivistelmä 3

Abstract 4

Alkusanat 5

Symboliluettelo 8

1. Johdanto 9

2. Ohjelmistoarkkitehtuurin kehittäminen 11
   - 2.1 Arkkitehtuurin suunnittelu 11
     - 2.1.1 Arkkitehtonisiin näkymiin perustuvia suunnittelumenetelmiä 12
     - 2.1.2 Laatuohjattu arkkitehtuurisuunnittelu 14
   - 2.2 Arkkitehtuurin mallinnus: Unified Modeling Language 16
     - 2.2.1 Kaaviotyypit 16
     - 2.2.2 Metamalli ja profiilit 19
   - 2.3 Luotettavuusanalyysi 19

3. Vaatimukset 22
   - 3.1 RAP-menetelmän yleiskuvaus 22
   - 3.2 Vaatimusmäärittely 24
     - 3.2.1 Toiminnalliset vaatimukset 25
     - 3.2.2 Tekniset vaatimukset 26
     - 3.2.3 Mallinnustyökalun vaatimukset 26
   - 3.3 Toteutettavat analysointitekniikat 27
     - 3.3.1 Tilapohjainen analysointimalli 27
     - 3.3.2 Polkupohjainen analysointimalli 30

4. Analyysityökalu 34
   - 4.1 Arkkitehtuuri 34
   - 4.2 Toteutus 40
     - 4.2.1 Tekninen toteutus ja rajoitteet 41
     - 4.2.2 Analyysityökalun käyttöliittymä 44
   - 4.3 Testaus 47

5. Tapaustutkimus 49

6. Pohdinta 53

7. Yhteenveto 55

Lähdeluettelo 56

Symboliluettelo

COM	Component Object Model, määritys ohjelmistokomponenttien väliseen kommunikointiin
DiSeP	Distribution Service Platform, hajautettu palvelualusta ohjelmistokomponenteille
GUI	Graphical User Interface, graafinen käyttöliittymä
IEEE	Institute of Electrical and Electronics Engineers, kansainvälinen insinööriyhteisö
ISO	International Standardization Organization, kansainvälinen standardointijärjestö
MTBF	Mean time between failures, keskimääräinen vikaantumisväli
MTTF	Mean time to failure, keskimääräinen vikaantumisaika
MTTR	Mean time to repair, keskimääräinen toipumisaika
OMG	Object Management Group, oliopohjaisten tekniikoiden standardointijärjestö
QADA®	Quality-driven Architecture Design and quality Analysis, arkkitehtuurin suunnittelu- ja analysointimenetelmä
UML	Unified Modeling Language, standardoitu graafinen mallinnuskieli
W3C	World Wide Web Consortium, standardointijärjestö
XML	eXtensible Markup Language, laajennettava tiedon esityskieli

1. Johdanto

Yhä useammat ihmiset ovat päivittäin tekemisissä sovelluksia sisältävien teknisten järjestelmien kanssa, joita käytetään esimerkiksi raha-asioiden hoitamiseen, kommunikointiin, ostosten tekoon tai vaikka tulipalolta suojautumiseen. Tällaisten järjestelmien vikaantuminen voi aiheuttaa taloudellisia menetyksiä, tapaturmia tai jopa kuolemantapauksia. Näin ollen ohjelmistojärjestelmien luotettavuudella on suuri merkitys ihmisten hyvinvoinnille.

Ohjelmiston laadun voidaan ajatella koostuvan useista eri laatuattribuuteista, joita ovat esimerkiksi luotettavuus, suorituskyky, saatavuus ja ylläpidettävyys. Tässä työssä tarkastellaan arkkitehtuurin analysointia luotettavuuden näkökulmasta. Luotettavuus on järjestelmän tai komponentin todennäköisyys vikaantumattomalle toiminnalle tietyn ajan kuluessa tietyssä ympäristössä [1].

Luotettavien ohjelmistojen tekeminen on aina ollut pulmallista, eikä järjestelmien koon kasvaminen ja monimutkaistuminen tuo tähän helpotusta – päinvastoin. Sen takia ohjelmiston luotettavuuteen tulisi kiinnittää enemmän huomiota heti arkkitehtuurin suunnittelusta lähtien. Luotettavuuden arkkitehtuuritason analysointia varten onkin olemassa useita menetelmiä, joita esitellään viitteissä [1–3].

Mallipohjaisella analysoinnilla tarkoitetaan ohjelmistojärjestelmän laadun analysointia sen arkkitehtuurimallista. Arkkitehtuurimallilla puolestaan tarkoitetaan jollain graafisella kuvaustavalla tehtyä esitystä ohjelmiston rakenteesta ja käyttäytymisestä. Mallipohjaista analysointia käytetään silloin, kun halutaan arvioida ohjelmiston laatua aikaisessa kehitysvaiheessa ennen kuin se voidaan mitata toteutetusta ohjelmistosta. Ohjelmiston arkkitehtuurin suunnitteluvaiheessa tehdyillä ratkaisulla ja päätöksillä on merkittävä vaikutus läpi koko ohjelmiston kehityksen. Mallipohjainen analysointi auttaa tunnistamaan mahdollisia ongelmakohtia aikaisessa suunnitteluvaiheessa, jolloin niiden korjaaminen ei ole niin haastavaa ja kallista.

Analysointimenetelmien tehokas soveltaminen edellyttää oikeanlaisen työkalutuen olemassaoloa. Tämän työn tavoitteena on kehittää työkalutuki RAP (Reliability and Availability Prediction) -menetelmälle [4]. RAP-menetelmä on menetelmä ohjelmistojärjestelmän luotettavuuden ja saatavuuden ennustamiseen arkkitehtuuritasolta, ja se on kehitetty erityisesti ohjelmistoperheitä varten, mutta menetelmää voidaan soveltaa myös yksittäisille järjestelmille. Kehitettävän työkalun tarkoitus on helpottaa luotettavuuden analysointia kohdejärjestelmän arkkitehtuurimallista. Työkalulla ei pyritä ennustamaan lopullisen ohjelmistotuotteen luotettavuutta vaan auttamaan arkkitehtiä näkemään, toteuttaako arkkitehtuuri sille asetetut luotettavuusvaatimukset, tai valitsemaan useista eri arkkitehtuurikandidaateista paras luotettavuuden suhteen. Tätä varten evaluoitavan jär-

jestelmän arkkitehtuurimalliin on lisäksi kehitettävä käyttäytymismalli, jonka perusteella työkalu voi simuloida järjestelmän toimintaa. Arkkitehtuurimallin tekoon käytetään kaupallista mallinnustyökalua, jonka ulkoisen rajapinnan kautta analyysityökalu käsittelee mallia. Mallinnustyökaluna käytetään Sparx Systemsin Enterprise Architectia, jonka valitsemiseen vaikuttivat samat kriteerit kuin viitteessä [5] esitettävässä mallinnustyökalujen evaluoinnissa. Tässä työkaluevaluoinnissa soveltuvimmaksi mallinnustyökaluksi sijoitettiin Telelogicin Tau/Developer. Enterprise Architectin valintaa puolsi kuitenkin sen hinta, joka oli murto-osa Tau/Developerin hinnasta. Enterprise Architect ei ole mukana viitteen [5] mallinnustyökalujen evaluoinnissa.

Tämän diplomityön luvut on jaoteltu seuraavasti: aluksi luvussa 2 perehdytään mallipohjaiseen analyysiin liittyviin taustatietoihin. Luvussa 3 tutustutaan RAP-menetelmään ja analyysityökalun vaatimuksiin. Neljännessä luvussa esitellään analyysityökalun suunnittelu ja toteutus. Viidennessä luvussa havainnollistetaan tapaustutkimuksen avulla analyysityökalulla suoritettava luotettavuusanalyysi esimerkkiarkkitehtuurimallista. Lopuksi analysoidaan lyhyesti työn tuloksia.

2. Ohjelmistoarkkitehtuurin kehittäminen

Järjestelmien koon ja monimutkaisuuden kasvaessa ohjelmistoarkkitehtuurin merkitys korostuu. Ohjelmiston kehityksen ja arkkitehtuurin suunnittelun kannalta on tärkeää, että arkkitehtuuri pystytään esittämään selkeästi. Myös laadullisten ominaisuuksien huomioiminen arkkitehtuurissa tulee tulevaisuudessa olemaan yhä tärkeämpää ohjelmistojen kovenevien laatuvaatimusten myötä.

Tässä luvussa luodaan aluksi katsaus arkkitehtuurin suunnitteluperiaatteisiin, sitten esitellään arkkitehtuurin kuvaamiseen yleisesti käytetty mallinnuskieli UML. Lopuksi tutustutaan ohjelmiston luotettavuuden arviointitapoihin arkkitehtuuritasolta.

2.1 Arkkitehtuurin suunnittelu

Ohjelmiston arkkitehtuuri kuvaa ohjelmistojärjestelmän rakenteen, sen komponentit, komponenttien ominaisuudet ja komponenttien väliset suhteet [6]. Ohjelmiston arkkitehtuuri sisältää myös periaatteet ja ohjeistuksen arkkitehtuurin muuttamisesta ja kehittämisestä [7]. Bosch esittelee kolme seikkaa, joiden huomioon ottamista selkeästi määritelty ohjelmistoarkkitehtuuri helpottaa [8]:

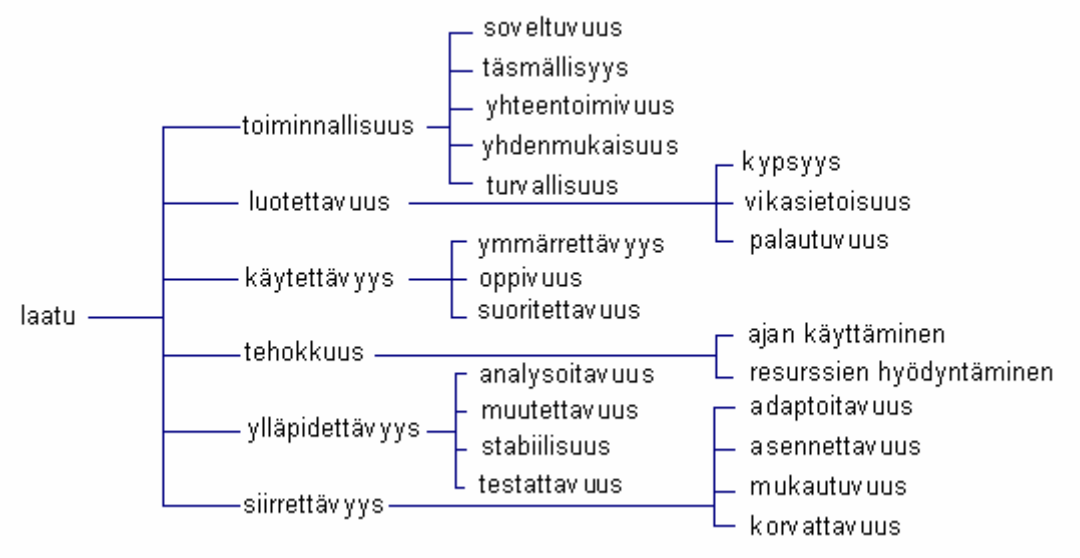
- järjestelmän laatuattribuutit
- kiinnostusryhmien kommunikointi
- tuotelinjan yhteisten komponenttien määrittely.

Arkkitehtuurin laadullisten ominaisuuksien huomioiminen ja arvioiminen mahdollisimman aikaisessa suunnitteluvaiheessa tekee mahdolliseksi laatuohjatun suunnittelun läpi koko ohjelmiston kehitysprosessin. Järjestelmän laatua kuvataan laatuattribuuteilla, joita ovat esim. *toiminnallisuus*, *luotettavuus*, *käytettävyys*, *tehokkuus, ylläpidettävyys*, *siirrettävyys* (kuva 1) [9].

Ohjelmiston kehittämiseen liittyy useita kiinnostusryhmiä (engl. *stakeholder*), joilla tarkoitetaan henkilöitä tai tahoja, jotka ovat jollakin tavoin ohjelmiston kehityksessä mukana, esimerkiksi projektipäällikkö, suunnittelijat, arkkitehdit, asiakkaat ja ylläpitäjät. Kiinnostusryhmien välille pyritään saamaan keskustelua suunnittelun alusta pitäen. Näin voidaan varmistaa, ettei ohjelmiston kehitys etene, ennen kuin kaikki osapuolet ovat hyväksyneet esim. arkkitehtuurisuunnitelman.

Tuotelinja on toisiinsa liittyvä tuotejoukko, joka jakaa yhteisen vaatimusjoukon ja arkkitehtuurin mutta sisältää myös tuotekohtaisia vaatimuksia [8]. Selkeästi määritelty arkkitehtuuri erottelee tuotekohtaiset ja tuotelinjan yhteiset komponentit, jolloin kompo-

nenttien uudelleenkäyttö tehostuu. Ohjelmistotuotteiden lukumäärän kasvaessa tuotelinjaa hyödyntävän ohjelmistokehityksen kustannukset nousevat huomattavasti maltillisemmin kuin perinteisessä ohjelmistokehityksessä, jos tuotteet kuuluvat samaan tuoteperheeseen [10].



Kuva 1. ISO9126-standardin laatumalli ohjelmistojärjestelmille.

2.1.1 Arkkitehtonisiin näkymiin perustuvia suunnittelumenetelmiä

Arkkitehtoninen näkymä (engl. *architectural view*) on kuvaus koko järjestelmästä tietystä ohjelmistoteknisestä perspektiivistä [9]. Arkkitehtoninen näkökulma (engl. *architectural viewpoint*) on puolestaan määritelmä menettelytavoista näkymän käyttämiseen ja sen rakenteen luomiseen [9]. Ohjelmiston arkkitehtuuri kuvataan tyypillisesti useilla näkymillä, jotka ovat siis kuvauksia arkkitehtuurista tietyistä näkökulmista.

Arkkitehtoniset näkymät ovat viime vuosien ajan muodostaneet pohjan useille suunnittelumenetelmille [11]. Suunniteltaessa ohjelmistojärjestelmää arkkitehdin tekemät suunnittelupäätökset sisällytetään näkymiin. Seuraavaksi esitellään kolme tunnettua ja paljon käytettyä näkymiin perustuvaa suunnittelumenetelmää.

Ensimmäinen eri näkökulmia hyödyntävä suunnittelumenetelmä oli Kruchtenin ”4+1”-näkymän malli [12]. Tämä menetelmän käsittämät viisi näkymää ovat seuraavat:

- *Looginen näkymä* kuvaa järjestelmän oliomallin ja olioiden keskinäisen toiminnan.
- *Prosessinäkymä* kuvaa ohjelmiston rinnakkaisuus- ja synkronointiaspektit.

- *Fyysinen näkymä* kuvaa ohjelmiston osien sijoittumista laitteistossa.
- *Kehitysnäkymä* kuvaa ohjelmiston staattista rakennetta.
- ”*+1*”-näkymä koostuu *skenaarioista*, joita käytetään muiden näkymien validointiin ja havainnollistamiseen.

Jaaksi et al. [13] esittelevät ”3+1”-näkymän mallin, joka on hieman muunneltu versio ”4+1”-näkymän mallista. Suurin ero Kructhenin menetelmään verrattuna on se, että tässä näkymät tukevat suoraan UML:n kuvaustapoja. Tässä menetelmässä ohjelmistoarkkitehtuuri kuvataan neljänä näkymänä:

- *Looginen näkymä* havainnollistaa järjestelmän korkean tason osittamisen loogisiin osiin, kuten tuotteisiin ja sovelluksiin.
- *Ajonaikainen näkymä* kuvaa järjestelmän ajonaikaisen konfiguraation, joka koostuu tietokoneresursseista, komponenteista ja niiden riippuvuuksista.
- *Kehitysnäkymä* kuvaa tavan ajattavien ohjelmien rakentamiseen komponenteista.
- *Skenaarionäkymän* avulla voidaan suunnitella ja arvioida muiden näkymien elementtien roolia, vuorovaikutusta, rajapintoja ja vastuita.

Hofmeister et al. [14] esittelevät mallin, joka koostuu neljästä näkymästä:

- *Konseptuaalinen näkymä* kuvaa järjestelmän suurina suunnitteluelementteinä ja niiden välisinä suhteina.
- *Moduulinäkymä* kuvaa konseptuaalisten komponenttien ja linkkien kiinnittämisen alijärjestelmiin ja moduuleihin.
- *Suoritusnäkymä* keskittyy kuvamaan dynaamisia rakenteellisia seikkoja, kuten ohjelmiston allokointia hajautetuissa järjestelmissä ja arkkitehtuuritason synkronointiaspekteja.
- *Koodinäkymä* kuvaa lähdekoodin organisoitumisen suurempiin kokonaisuuksiin aina ajettaviin tiedostoihin asti.

Siitä huolimatta, että jokainen näistä edellä kuvatuista menetelmistä omalla tavallaan tarjoaa perusteellisen tavan kuvata ohjelmiston arkkitehtuuria, yksikään menetelmä ei ota huomioon tuotelinjalähestymistapaa arkkitehtuurin suunnittelussa [11]. Lisäksi nämä menetelmät eivät huomioi eri kiinnostusryhmien näkökulmia vaan tarjoavat vain suunnittelijoiden tarvitsemia koodiorientoituneita näkökohtia [11]. Toisin sanoen nämä menetelmät eivät täytä selkeästi määritellylle ohjelmistoarkkitehtuurille asetettuja vaatimuksia.

2.1.2 Laatuohjattu arkkitehtuurisuunnittelu

Nykyisin ohjelmistojärjestelmien kehittämisessä yksi merkityksellisimmistä asioista on ohjelmiston laatu [15]. Perinteisillä suunnittelumenetelmillä on taipumus keskittyä saavuttamaan järjestelmän toiminnalliset vaatimukset laatuvaatimusten jäädessä usein taka-alalle. Aikaisemmin järjestelmän laadulliset ominaisuudet on mitattu suurelta osin toteutetusta järjestelmästä. Ongelmana tällaisessa lähestymistavassa on, että osa käytettävissä olevista resursseista tuhlataan järjestelmään tai sen osiin, jotka eivät täytä laatuvaatimuksia [8].

Laatuohjatun arkkitehtuurisuunnittelun lähtökohtana on laatuvaatimusten huomioon ottaminen suunnittelun alusta lähtien. Järjestelmän laatuattribuutteja pyritään evaluoimaan koko arkkitehtuurisuunnittelun ajan, jotta lopullinen ohjelmiston toteutus täyttäisi ohjelmistolle asetetut laatuvaatimukset. Laatuattribuutit voidaan jakaa suoritus- ja kehitysaikaisiin laatuattribuutteihin [16]. Suoritusaikaiset laatuattribuutit ovat ajon aikana havaittavia laatuominaisuuksia, jotka liittyvät järjestelmän käyttäytymiseen. Kehitysaikaiset laatuattribuutit kuvaavat järjestelmän staattisia laatuominaisuuksia. Taulukossa 1 esitellään joitakin suoritusaikaisten attribuuttien määritelmiä ja taulukossa 2 kehitysaikaisia attribuutteja.

Taulukko 1. Suoritusaikaisia laatuattribuutteja.

Laatuattribuutti	**Määritelmä**
Luotettavuus	Järjestelmän tai komponentin todennäköisyys vikaantumattomalle toiminnalle tietyn ajan kuluessa tietyssä ympäristössä [1].
Suorituskyky	Järjestelmän käyttämä aika vastata herätteisiin tai prosessoitujen tapahtumien määrä jollakin aikavälillä [15].
Saatavuus	Ajan suhteellinen osuus, jolloin järjestelmä tai komponentti toimii hyväksyttävästi [1].

Taulukko 2. Kehitysaikaisia laatuattribuutteja

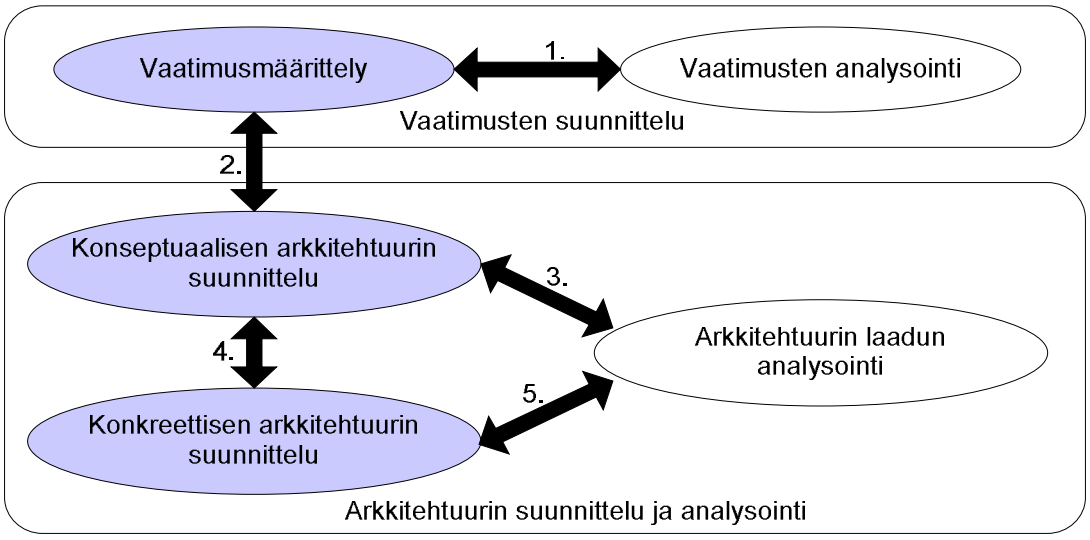
Laatuattribuutti	**Määritelmä**
Ylläpidettävyys	Järjestelmän tai komponentin muuttamisen tai mukauttamisen vaivattomuus muuttuneeseen ympäristöön [15].
Laajennettavuus	Järjestelmän ominaisuus omaksua uusia komponentteja [15].
Integroitavuus	Erillään kehitettyjen komponenttien ominaisuus virheettömään yhteistoimintaan järjestelmässä [15].

Esimerkkinä laatuohjatun arkkitehtuurin suunnittelumenetelmästä mainittakoon QADA®[1] (Quality-driven Architecture Design and quality Analysis) [17]. QADA® on laatuohjattu ohjelmistoarkkitehtuurin suunnittelumenetelmä, jossa ohjelmiston laatuvaatimusten saavuttaminen kulkee rinnan toiminnallisten vaatimusten kanssa. QADA®-metodologiassa arkkitehtuurin suunnittelu on yhdistetty laadun analysointiin, jolloin arkkitehtuurin kehitystä voidaan tarkkailla laatunäkökulmasta suunnittelun alusta lähtien [18]. Metodologia on kehitetty pitäen silmällä tuotelinja-arkkitehtuurin tarpeita, mutta sitä voidaan soveltaa myös yksittäisten tuotteiden kehittämiseen.

QADA®-metodologiassa arkkitehtuurin suunnittelu on jaettu konseptuaaliseen ja konkreettiseen abstraktiotasoon. Molemmilla abstraktiotasoilla arkkitehtuuri kuvataan neljästä eri näkökulmasta, jotka ovat *rakenne*-, *käyttäytymis*-, *sijoittumis*- ja *kehittämisnäkökulma*. Rakennenäkökulma vastaa komponenttien kokoonpanosta, siinä missä käyttäytymisnäkökulma tarkastelee järjestelmää käyttäytymisen kannalta. Sijoittumisnäkökulma ohjaa komponenttien sulauttamista ja kohdistamista eri laiteympäristöihin. [17] Kehittämisnäkökulma esittelee komponentit ja niiden keskinäiset suhteet sekä toiminnalliset vastuut niiden kehittämisestä [11].

Kuva 2 [17] esittää pääpiirteissään QADA®-metodologian vaiheet. Vaatimusten suunnitteluvaiheessa järjestelmän laadulliset ominaisuudet määritetään ja analysoidaan järjestelmän kontekstin ja teknisten ominaisuuksien suhteen. Konseptuaalisen arkkitehtuurin suunnitteluvaiheessa mallinnetaan ja dokumentoidaan järjestelmän rakenne, käyttäytyminen ja sijoittuminen abstraktilla tasolla. Arkkitehtuurin laadun analysoinnissa arvioidaan konseptuaalisen tai konkreettisen arkkitehtuurin laatua esim. skenaariopohjaisilla menetelmillä. Lisäksi siinä evaluoidaan arkkitehtuurikandidaatteja vertaamalla niiden laatuominaisuuksia toisiinsa tai vaatimusten suunnitteluvaiheessa saatuihin laatuvaatimuksiin. Laadun analysoinnin jälkeen arkkitehtuurimalleja päivitetään tarvittaessa. Konkreettisen arkkitehtuurin suunnitteluvaiheessa määritellään järjestelmän rakenne, käyttäytyminen ja sijoittuminen yksityiskohtaisemmin konseptuaalisen arkkitehtuurin suunnittelusta saadun arkkitehtuurikuvauksen perusteella.

[1] QADA® on VTT:n rekisteröity tavaramerkki, http://www.vtt.fi/ele/research/soh/projects/qada/.



Kuva 2. QADA®-menetelmän vaiheet.

2.2 Arkkitehtuurin mallinnus: Unified Modeling Language

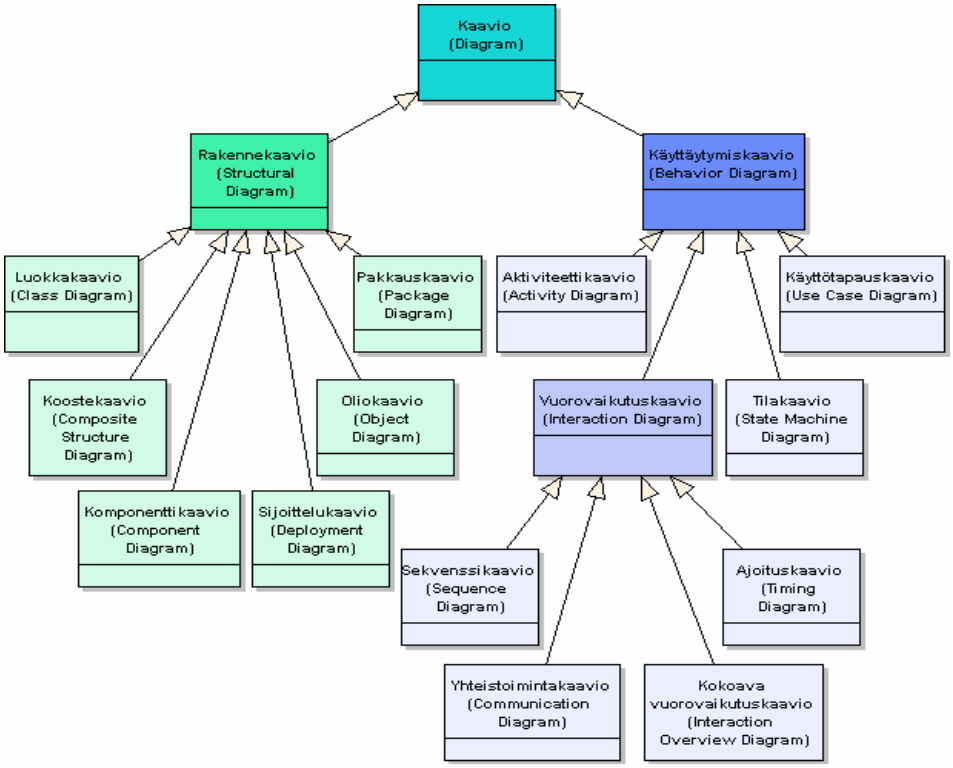
Unified Modeling Language (UML) on standardoitu graafinen mallinnuskieli, joka tarjoaa oliosuuntautuneille ohjelmistokehittäjille yhden yhtenäisen tavan järjestelmien määrittelyyn, visualisointiin, rakenteen luomiseen ja dokumentointiin [19]. Vaikka UML on standardin mukaan tarkoitettu edellisten lisäksi myös liiketoiminnan ja muiden ohjelmistotekniikan ulkopuolisten järjestelmien mallintamiseen [19], tässä kohdassa tarkastellaan UML-kieltä kuitenkin vain ohjelmistokehittäjän näkökulmasta.

UML-kielen rakenteesta, määrittelystä ja standardoinnista vastaa Object Management Group (OMG) [20]. Nykyisin käytössä oleva UML-standardi 1.5 on peräisin vuodelta 1997. Käytännössä kuitenkin monet ohjelmistokehittäjät ovat siirtyneet uudempaan UML 2.0 -versioon aikaisemman version puutteiden takia [21]. UML 2.0 saanee OMG:n standardin aseman vuoden 2005 aikana.

2.2.1 Kaaviotyypit

Monet UML:n kaaviotyypit ovat itse asiassa kauan tunnettuja graafisia mallinnuskieliä. UML toisin sanoen kokoaa yhteen joukon myös mallinnuskielinä tunnettuja esitystapoja ja spesifioi niiden esitysmuodon. Näitä esitystapoja kutsutaan kaaviotyypeiksi. UML:n spesifikaatio ei kuitenkaan määrittele kaaviotyyppejä riittävän tarkasti, jotta voitaisiin puhua graafisista osakielistä. UML:n ajattelutavan mukaan kukin kaavio antaa eri näkökulman koko järjestelmää kuvaavaan abstraktiin malliin. [22]

UML 2.0 sisältää 13 kaaviotyyppiä, jotka esitetään hierarkkisesti kuvassa 3. UML:n kaaviotyypit on jaettu rakennekaavioihin ja käyttäytymiskaavioihin. Rakennekaaviolla kuvataan järjestelmän staattista ajasta riippumatonta rakennetta ja järjestelmän ajonaikaista dynaamista toimintaa kuvataan puolestaan käyttäytymiskaavioilla. Sekä rakennetta että käyttäytymistä voidaan tarkastella kaavioiden avulla eri näkökulmista ja abstraktiotasoilta. On huomattava, että järjestelmän mallintamiseen ei vaadita kaikkien kaaviotyyppien käyttöä eikä se ole usein tarkoituksenmukaistakaan.



Kuva 3. UML 2.0 -kaaviotyypit.

Rakenteen kuvaamiseen voidaan käyttää kuutta kaaviotyyppiä [19, 22]:

- *Luokkakaavio* on tärkein järjestelmän staattista rakennetta kuvaava kaavio. Siinä määritellään järjestelmän luokkien väliset suhteet ja kaikki mahdolliset ajonaikaiset oliokokoelmat. Kuvassa 3 esitetty kaaviotyyppien hierarkkinen rakenne on toteutettu luokkakaavion elementeillä, joissa suorakulmiot ovat luokkia ja nuolet osoittavat periytymisen.
- *Koostekaavio* on luokkakaavion tapainen kaavio, mutta siinä mallinnetaan luokkarakenteen tietty spesifinen käyttö. Koostekaaviolla voidaan kuvata myös rajapintojen ja komponenttien välisiä toiminnallisia riippuvuuksia.

- *Komponenttikaaviolla* kuvataan komponenttien välisiä suhteita, ja siinä komponentti on itsenäinen tietyn rajapinnan toteuttava ohjelmistoyksikkö. Siinä missä luokkakaavio on yksityiskohtainen kuvaus luokkien välisistä suhteista, komponenttikaavio on korkeamman arkkitehtuuritason kuvaus järjestelmän staattisesta rakenteesta.
- *Oliokaavio* kuvaa yhden mahdollisen järjestelmässä ajon aikana esiintyvän oliokonfiguraation. Oliokaavio on siten luokkakaavion yksi ilmentymä.
- *Sijoittelukaavio* on korkean abstraktiotason kuvaus järjestelmän laitearkkitehtuurista ja ohjelmiston sijoittumisesta laitteistoon.
- *Pakkauskaaviolla* parannetaan mallin hallittavuutta organisoimalla alijärjestelmät pakkauksiin ja määrittämällä niiden riippuvuussuhteet.

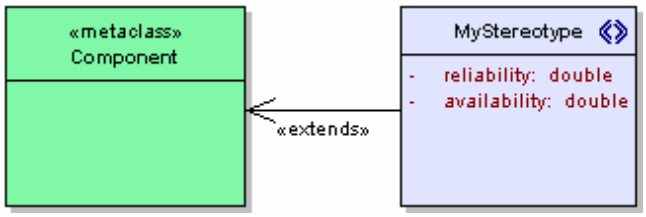
Käyttäytymisen kuvaamiseen UML 2.0 tarjoaa seitsemän kaaviotyyppiä [19, 22]:

- *Käyttötapauskaavio* on helposti ymmärrettävä malli koko järjestelmän toiminnasta. Usein käyttötapauskaavio on lähtökohtana koko järjestelmän suunnittelulle, sillä sen perusteella nähdään järjestelmän toiminnalliset vaatimukset. Käyttötapauskaavio koostuu käyttötapauksista, järjestelmän ulkopuolisista toimijoista (esim. käyttäjä) ja niiden välisistä riippuvuuksista.
- *Tilakaavio* on esimerkiksi olion tai prosessin käyttäytymisen kuvaus tilakoneena. Tilakaaviota voidaan käyttää sekä abstraktiin että yksityiskohtaiseen käyttäytymisen kuvaamiseen. Tilakaaviot mahdollistavat järjestelmän tai sen osan toiminnan simuloimisen ilman lopullista toteutustason informaatiota.
- *Aktiviteettikaaviolla* voidaan kuvata koko järjestelmän sisäistä sekä tieto- että kontrollivuota. Solmuina voidaan käyttää esimerkiksi olioita, jolloin saadaan tietovuomainen kuvaus.
- *Sekvenssikaavio* on tyypillisesti kuvaus olioiden välisestä viestien vaihdosta tietyissä suorituspoluissa, jotka esitetään aikajärjestyksessä. Jotkin mallinnustyökalut osaavat tuottaa sekvenssikaavioita tilakaavioiden simuloinnin tuloksena.
- *Yhteistoimintakaavio* esittää sekvenssikaavion tapaan olioiden välistä vuorovaikutusta ajon aikana, mutta samalla se visualisoi sekvenssikaavioita paremmin olioiden väliset vuorovaikutussuhteet, kun taas sekvenssikaavio esittää olioiden ajasta riippuvan toiminnan paremmin.
- *Kokoava vuorovaikutuskaavio* kuvaa nimensä mukaisesti muiden vuorovaikutuskaavioiden yhteistoimintaa. Tällä tavoin käyttäytymistä voidaan kuvata laajempina kokonaisuuksina yhdessä kaaviossa.
- *Ajoituskaavio* kuvaa olioiden käyttäytymisen ja tarjoaa visuaalisen esityksen olioiden tilavaihdoksista ja keskinäisistä vuorovaikutuksista reaaliajan suhteen.

2.2.2 Metamalli ja profiilit

UML:n metamalli on korkeamman tason kuvaus UML-kielessä sallittujen mallien abstrakteista rakenteista. Metamallilla määritellään siis kieli mallin ilmaisuun. Metamalli puolestaan rakentuu metaluokista, joiden ilmentymiä mallitason elementit ovat. Joustava piirre UML-kielessä on, että se sallii metamallin rakentamisen metaluokista tavallisen luokkakaavion tapaan, eli UML-kieltä itseään voidaan käyttää määrittelemään UML:n abstrakti rakenne.

UML-kielen monipuolisista mahdollisuuksista huolimatta syntyy usein tarve lisätä kieleen omia rajoituksia tai lisämäärittelyjä. Tämä voitaisiin tehdä suoraan UML:n metamallia muokkaamalla. Yleensä tällainen menettely ei ole kuitenkaan järkevää, koska UML:n standardin mukaisen metamallin modifiointi johtaisi siihen, että pian eri ohjelmistokehittäjillä olisi omat epästandardit versiot UML:stä. Sen sijaan UML:n *profiilit* tarjoavat käyttökelpoisen mekanismin laajennustarpeita varten. Profiilien avulla UML:n metamallia voidaan näennäisesti laajentaa *stereotyyppien* avulla, jotka ovat metaluokkien erikoistuksia. Kuvassa 4 UML:n komponenttielementin metaluokkaa laajennetaan kahdella laatuattribuutilla stereotyypin avulla. Tällainen laajennus on mahdollista tallentaa profiilina, joka voidaan ottaa käyttöön tarpeen mukaan.



Kuva 4. Metaluokan laajentaminen stereotyypillä.

2.3 Luotettavuusanalyysi

Luotettavuus on yksi tärkeimmistä, ellei tärkein, ohjelmiston laadun mittareista. Etenkin kriittisissä ohjelmistojärjestelmissä, kuten ydinvoimaloissa, lentokoneissa ja sairaaloissa, luotettavuuden maksimoiminen on välttämätöntä. Luotettavuuden analysoinnin tarkoitus on mahdollisten luotettavuuteen vaikuttavien riskitekijöiden tunnistaminen, luotettavuuden arviointi ja luotettavuusvaatimusten toteutumisen todentaminen [23].

Yksinkertainen ja usein käytetty luotettavuuden mittari on *keskimääräinen vikaantumisväli* MTBF (mean time between failures). Keskimääräinen vikaantumisväli määritellään kaavan 1 mukaisesti:

$$MTBF = MTTF + MTTR, \quad (1)$$

missä

MTTF on *keskimääräinen vikaantumisaika* (mean time to failure)
MTTR on *keskimääräinen toipumisaika* (mean time to repair).

Perinteisesti ohjelmiston luotettavuutta mitataan ohjelmiston toteutuksen jälkeen testaamalla. Ideaalisessa tapauksessa ohjelmiston luotettavuus voitaisiin mitata jo arkkitehtuuritasolla, mikä säästäisi merkittävästi ohjelmistokehityksen kustannuksia. Käytännössä kuitenkin arkkitehtuurimallista ei ole mahdollista tehdä tarkkoja mittauksia lopullisen ohjelmistotuotteen luotettavuuden määrittämiseksi [24], koska arkkitehtuurista eivät ilmene kaikki ohjelmiston luotettavuuteen vaikuttavat tekijät, kuten ohjelmoijan taito. Tarkkojen laskelmien sijaan arkkitehtuuritason luotettavuuden analysoinnin tavoitteena onkin arvioida järjestelmän luotettavuutta ennen sen toteuttamista [25].

Ohjelmistojärjestelmän monimutkaisuus on yleisesti todettu merkittävimmäksi luotettavuuteen vaikuttavista tekijöistä. Arkkitehtuurin suunnittelulla on selkeä yhteys lopullisen ohjelmiston monimutkaisuuteen, sillä esimerkiksi rakenteellisuus, modulaarisuus ja komponenttien väliset riippuvuudet ovat monimutkaisuuteen vaikuttavia tekijöitä [26]. Monimutkaisuuden mittaamiseen on kehitetty lukuisia mittausvälineitä [27], joista tunnetumpia ovat McCaben [28] ja Halestadin [29] metriikat. Myös monimutkaisuuden vähentämiseen tähtääviä suunnittelumenetelmiä on olemassa, esimerkiksi Jaaksin et al. [13] esittelemä tekniikka syklisten riippuvuuksien eliminoimiseksi. Monimutkaisuuden käyttäminen luotettavuuden mittana on usein vaikeaa, sillä hyvin monimutkainenkin ohjelma voi toimia luotettavasti.

Luotettavuus on suoritusaikainen laatuattribuutti, mistä johtuen luotettavuuden analysoinnissa tulee aina ottaa huomioon järjestelmän ajonaikainen käyttäytyminen. Järjestelmän toiminnallisuuden huomioon ottavia tapoja luotettavuuden arvioimiseksi arkkitehtuuritasolla ovat [8]

- skenaariopohjainen arviointi
- simulaatio
- matemaattinen mallinnus
- kokemukseen perustuva arviointi.

Skenaarioilla suunnittelija kuvaa tiettyjä ohjelman toimintaa mallintavia käyttötapauksia, jotka esiintyvät valmiissa ohjelmassa vastaavassa käyttötapauksessa. Skenaarioita voidaan tehdä varta vasten tiettyä laatuattribuuttia, esim. luotettavuutta, varten. Skenaariopohjaisen arvioinnin tehokkuus riippuu paljolti siitä, kuinka tarkasti valmiin ohjelman käyttäytyminen on onnistuttu mallintamaan skenaarion avulla. Komponenttipohjaisissa järjestelmissä skenaarioiden avulla voidaan selvittää tietyn komponentin käyttötaa-

juutta ja komponenttien välisiä suorituspolkuja, joiden avulla voidaan arvioida luotettavuutta. [8]

Simulaation tarkoituksena on rakentaa arkkitehtuurimallista ajettava malli ilman kaikkien järjestelmään kuuluvien komponenttien toteuttamista. Simulaation toteuttamiseksi järjestelmän käyttäytymisen ja komponenttien vuorovaikutuksen täytyy olla hyvin tarkasti tiedossa jo arkkitehtuurin suunnitteluvaiheessa. Skenaariopohjaisen luotettavuuden arvioinnin tapaan simulaatiolla saadaan selvitettyä komponenttien suoritussekvenssejä eri käyttötilanteissa. Erona skenaariopohjaiseen arviointiin on, että simulaatio mahdollistaa huomattavasti laajemman analyysin järjestelmän eri käyttötilanteista, kun taas skenaariopohjainen arviointi ottaa kantaa vain ennalta määriteltyihin suunnittelijan mielestä tärkeisiin käyttötilanteisiin. Toinen simulaatiota muistuttava tapa, jota käytetään ohjelmiston laadun arvioimiseen, on *prototypointi*. Prototypoinnissa ei kuitenkaan pysytä arkkitehtuuritasolla, kuten simulaatiota tehtäessä, vaan osa järjestelmästä on konkreettisesti toteutettu. [8]

Useille ohjelmistotekniikkaa hyödyntäville aloille, kuten suurteholaskenta [30], luotettavat järjestelmät ja reaaliaikaiset järjestelmät [31], on kehitetty *matemaattisia malleja*, joita voidaan käyttää ohjelmiston laadun arvioimiseen. Nämä mallit soveltuvat erityisesti suoritusaikaisten laatuattribuuttien, kuten luotettavuuden, arvioimiseen. Matemaattinen mallintaminen on usein vaihtoehto simuloinnille, mutta toisinaan sitä käytetään täydennyksenä. Esimerkiksi matemaattista mallia voidaan käyttää arvioimaan komponentin luotettavuutta sen tilakoneesta tehdyn Markovin ketjun [32] avulla ja koko järjestelmän luotettavuuden arviointi tehdään simulaation avulla.

Kokeneet ohjelmistoarkkitehdit osaavat usein välttää huonoja suunnittelupäätöksiä ja intuitiivisesti arvioida suunnitelman ”hyvyyttä” tai ”huonoutta” [8]. *Kokemukseen perustuvalla* arvioinnilla parhaimmillaankin saadaan vain hyvin karkeita arvioita ohjelmiston luotettavuudesta.

Lähteissä [1–3] esitetään menetelmiä arkkitehtuuritason luotettavuuden analysointiin. Kaikki nämä menetelmät hyödyntävät analyysissä kohdejärjestelmän skenaariopohjaista käyttäytymisen kuvausta.

3. Vaatimukset

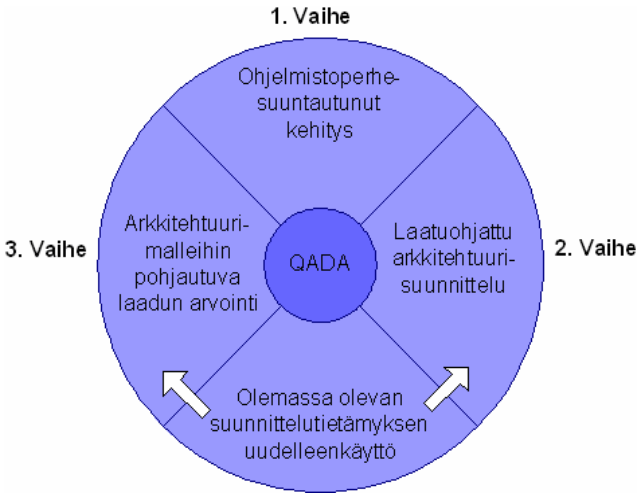
Tässä työssä esiteltävä analyysityökalu on kehitetty tukemaan ja nopeuttamaan RAP-menetelmän [4] soveltamista ohjelmistojärjestelmän luotettavuuden analysointiin. Täten RAP-menetelmä eli menetelmä luotettavuuden ja saatavuuden ennustamiseen arkkitehtuuritasolla asettaa lähtökohdat analyysityökalun vaatimusten suunnittelulle.

Tässä luvussa tutustutaan aluksi RAP-menetelmään ja sen kolmeen päävaiheeseen, minkä jälkeen esitetään analyysityökalun vaatimusmäärittely ja ulkopuoliselta mallinnustyökalulta vaaditut ominaisuudet. Lopuksi käydään läpi RAP-menetelmästä johdetut ja analyysityökalulta vaaditut luotettavuuden analysointitekniikat esimerkeillä havainnollistaen.

3.1 RAP-menetelmän yleiskuvaus

RAP (Reliability and Availability Prediction) -menetelmä [4] on menetelmä ohjelmistojärjestelmän luotettavuuden ja saatavuuden ennustamiseen arkkitehtuuritasolla. RAP-menetelmä on suunnattu ensisijaisesti ohjelmistotuoteperheitä varten, mutta menetelmää voidaan soveltaa myös yksittäisiin järjestelmiin. RAP-menetelmän soveltaminen koko laajuudessaan pieniin ja yksinkertaisiin järjestelmiin ei ole kuitenkaan aina tarkoituksenmukaista. RAP-menetelmä on olennainen osa QADA®-metodologiaa ja noudattaa sen ideaa, jossa arkkitehtuurikandidaateista etsitään parhaiten laatuvaatimukset täyttävä ratkaisu arvioimalla eri kandidaattien laatua ja vertaamalla niiden ominaisuuksia toisiinsa. RAP-menetelmän tapauksessa keskitytään siis luotettavuuden ja saatavuuden evaluointiin arkkitehtuurikandidaateista.

RAP-menetelmä koostuu kolmesta päävaiheesta, jotka liittyvät QADA®:n vaiheisiin kuvan 5 [4] mukaisesti.



Kuva 5. RAP-menetelmän vaiheet.

1. vaihe sisältää viisi kohtaa, joissa määritellään luotettavuuden ja saatavuuden tavoitteet järjestelmälle. Nämä kohdat ovat

- kiinnostusryhmien ja niiden tarpeiden tunnistaminen
- luotettavuus- ja saatavuusvaatimusten jalostaminen
- luotettavuus- ja saatavuusvaatimusten kiinnittäminen toiminnallisuuteen
- arkkitehtuurityylin valinta ja eri kompromissien analysointi
- kriteerien määrittäminen luotettavuuden ja saatavuuden arvioimiseksi.

2. vaihe sisältää kolme kohtaa, jotka liittyvät luotettavuuden ja saatavuuden esittämiseen arkkitehtuurimalleissa. Nämä kohdat ovat

- luotettavuus- ja saatavuusvaatimusten esittäminen arkkitehtuurimalleissa erotellen ohjelmistoperhe- ja järjestelmäkohtaiset vaatimukset
- konseptuaalisen arkkitehtuurin kohdentaminen konkreettiseen arkkitehtuuriin
- järjestelmän tarjoaman luotettavuuden ja saatavuuden esittäminen arkkitehtuurimalleissa.

3. vaiheessa suoritetaan luotettavuuden ja saatavuuden evaluointi, ja se sisältää kolme pääkohtaa, jotka ovat seuraavat:

- Kvantitatiivinen analyysi
  - Arvioidaan komponenttien luotettavuus.
  - Arvioidaan ohjelmistojärjestelmän luotettavuus ja saatavuus.
  - Arvioidaan järjestelmän luotettavuus ja saatavuus kehitysympäristössä.
- Kvalitatiivinen analyysi
  - Tehdään vaatimusten jäljitys ja analysoidaan luotettavuus- ja saatavuusvaatimusten toteutuminen arkkitehtuurissa.
  - Tunnistetaan ongelmat, joita täyttämättä jääneet vaatimukset voivat aiheuttaa.
- Analyysiin pohjautuva päätöksenteko

3.2 Vaatimusmäärittely

RAP-menetelmässä arkkitehtuurikandidaateista evaluoidaan niiden luotettavuutta ja saatavuutta. Evaluoitavat arkkitehtuurikandidaatit toteuttavat saman toiminnallisuuden mutta käyttävät erilaisia arkkitehtuurityylejä ja suunnittelumalleja, mikä vaikuttaa niiden luotettavuuteen ja saatavuuteen. Analyysityökalua tarvitaan helpottamaan ja nopeuttamaan RAP-menetelmän evaluointiprosessia, joka ilman työkalutukea voisi olla hyvin työläs, etenkin jos arkkitehtuurikandidaatteja on paljon.

Analyysityökalun tehtävänä on erityisesti tukea RAP-menetelmän 3. vaiheen kvantitatiivista analyysiä. Kvantitatiivisessa analyysissä lasketaan virhetodennäköisyys analysoitavasta järjestelmästä sen rakenteen ja käyttäytymisen perusteella. Analyysin tekemiseksi järjestelmä täytyy kuvata sekä staattisesta että dynaamisesta aspektista. Tässä staattinen aspekti ilmaisee järjestelmän sisältämät komponentit ja niiden keskinäisen vuorovaikutuksen, ja dynaamisesta aspektista ilmenevät järjestelmän ajonaikainen käyttäytyminen ja komponenttien virhekäyttäytyminen.

Kvantitatiivisen analyysin aktiviteetit hyödyntävät sekä tilapohjaista että polkupohjaista analyysiä. RAP-menetelmässä tilapohjaista analyysiä käytetään komponenttien luotettavuuden määrittämiseen, jota varten arkkitehtuurimalliin tehdään komponenttien virhekäyttäytymistä esittävät tilapohjaiset vikatilamallit. Polkupohjaisessa analyysissä evaluoidaan koko järjestelmän luotettavuutta. Tila- ja polkupohjaiset analyysit kuvataan tarkemmin kohdassa 3.3. Koska luotettavuus on suoritusaikainen laatuattribuutti, järjestelmän ajonaikainen toiminta on tunnettava sen analysoimiseksi. Arkkitehtuuritason luotettavuusanalyysin tapauksessa järjestelmää ei ole vielä toteutettu, minkä takia ajonaikaista käyttäytymistä mallinnetaan simulaation avulla. Simuloimista varten arkkitehtuurimallia täydennetään simulointimallilla, jota hyödynnetään polkupohjaisessa analyysissä.

Simulointi- ja vikatilamallien määrittämisen jälkeen luotettavuuden analysointi on puhtaasti laskentaa, joka jää analyysityökalun tehtäväksi. Analysointiin liittyy monta välivaihetta, jotka selitetään tarkemmin kohdassa 3.3.

RAP-menetelmä asettaa vaatimuksia niin analyysityökalulle kuin analyysityökalun käyttämälle UML-mallinnustyökalulle. Analyysityökalun vaatimukset on jaettu toiminnallisiin ja teknisiin vaatimuksiin. Lisäksi koska arkkitehtuurikandidaatit mallinnetaan kaupallisella UML-mallinnustyökalulla, johon analyysityökalu on yhteydessä analyysivaiheessa, myös tälle työkalulle on määritelty vaatimukset. Analyysityökalun laatuvaatimuksiin ei ole laitettu suurta painoarvoa, sillä tarkoituksena ei ole ollut tehdä kaupallista sovellusta, vaan ensisijaisesti RAP-menetelmää tukeva tutkimustyökalu luotettavuuden evaluointiin

arkkitehtuuritasolta. Tällöin työkalun toiminnallisten ja teknisten vaatimusten saavuttaminen riittää, eikä esimerkiksi suorituskyvyn optimointi ole oleellista.

3.2.1 Toiminnalliset vaatimukset

Järjestelmän luotettavuutta arvioivan analyysityökalun on toteutettava seuraavat toiminnalliset vaatimukset:

1. Analyysityökalun täytyy päästä selaamaan ja muokkaamaan UML-mallia, jotta siitä voitaisiin analysoida mallinnetun järjestelmän luotettavuutta ja päivittää malliin työkalun laskemia luotettavuusarvoja. Seuraavat seikat liittyvät UML-mallin käsittelyyn:
   - Työkalun tulee osata hakea arkkitehtuurimalleista komponenttien tilakaaviot ja laskea komponenttien luotettavuudet niiden perusteella.
   - Komponenttien luotettavuusarvot täytyy päivittää komponenttikaaviossa oleville elementeille.
   - Järjestelmän simuloinnin mahdollistamiseksi analyysityökalun täytyy tunnistaa ja tallentaa arkkitehdin määrittelemät komponenttikohtaiset heräteviestit sekvenssikaaviosta.
   - Simulointia varten analyysityökalun täytyy tunnistaa arkkitehdin tekemä simulointimalli aktiviteettikaaviosta.

2. Arkkitehtuurimallin läpikäymisen jälkeen analyysityökalun tulee suorittaa simulointi mallista kerätyn tiedon perusteella tai ilmoittaa käyttäjälle, mikäli kerätty tieto on virheellistä tai puutteellista.

3. Simuloinnin suorittamisen jälkeen analyysityökalun on pystyttävä esittämään malleista kerättyä tietoa ja analyysin tulokset, joita ovat
   - komponenttikohtaiset Markovin ketjut
   - analysoitavan järjestelmän toimintaa kuvaava simulointimalli
   - heräteviestit, viestin sisältö ja sen vastaanottava komponentti
   - järjestelmän komponenttien virhetodennäköisyydet ja suorituskerrat kohdejärjestelmässä
   - simuloinnin aikana kuljetut suorituspolut ja polkujen virhetodennäköisyydet
   - koko järjestelmän virhetodennäköisyys.

3.2.2 Tekniset vaatimukset

Analyysityökalun tekniset vaatimukset liittyvät työkalun käytettävyyteen ja ohjelman sisäiseen rakenteeseen. Näitä vaatimuksia ovat seuraavat:

1. Graafinen käyttöliittymä, joka tarjoaa seuraavat toiminnot:
   - kaikkien toimintojen käyttö menuvalikosta
   - mallinnustyökalun tiedostojen haku tiedostoselaimen kautta
   - simuloinnin etenemisen näyttö
   - analyysin tuloksien selkeä esittäminen.

2. Työkalun rakenne on toteutettava mahdollisimman modulaarisesti ja sen komponentit vaihdettaviksi, jotta esimerkiksi mallinnustyökalun vaihtaminen ei aiheuttaisi muutoksia muualle kuin siihen liitoksissa olevaan komponenttiin.

3.2.3 Mallinnustyökalun vaatimukset

Jotta mallinnustyökalua voitaisiin käyttää analyysityökalun yhteydessä RAP-menetelmää sovellettaessa, sen tulee sisältää seuraavat ominaisuudet [33]:

1. Mallinnustyökalun täytyy tarjota mekanismi, jonka avulla ulkopuolinen ohjelma pääsee käsiksi malliin. Mallinnustyökalulla tulee tätä varten olla avoin ohjelmointirajapinta, joka sisältää metodit mallin lukemiseen ja muokkaamiseen.
2. Mallinnustyökalulla täytyy voida mallintaa QADA®:n mukaisia näkökulmia, joita tarvitaan luotettavuusvaatimusten esittämiseen mallissa. QADA®:n mukaisten näkökulmien mallintamiseksi mallinnustyökalun täytyy tukea UML 2.0 -standardia, joka tarjoaa joukon parannuksia ja uusia kaaviotyyppejä aikaisempaan UML-versioon nähden. Esimerkiksi QADA®:n rakennenäkökulman mallintamiseen tarvitaan koostekaaviota, jota ei ole aikaisemmissa UML:n versioissa.
3. Mallinnustyökalun täytyy tukea UML-profiilien luontia, joita tarvitaan mallissa olevien elementtien (esim. komponentti) laajentamiseen luotettavuusattribuutteja varten.

3.3 Toteutettavat analysointitekniikat

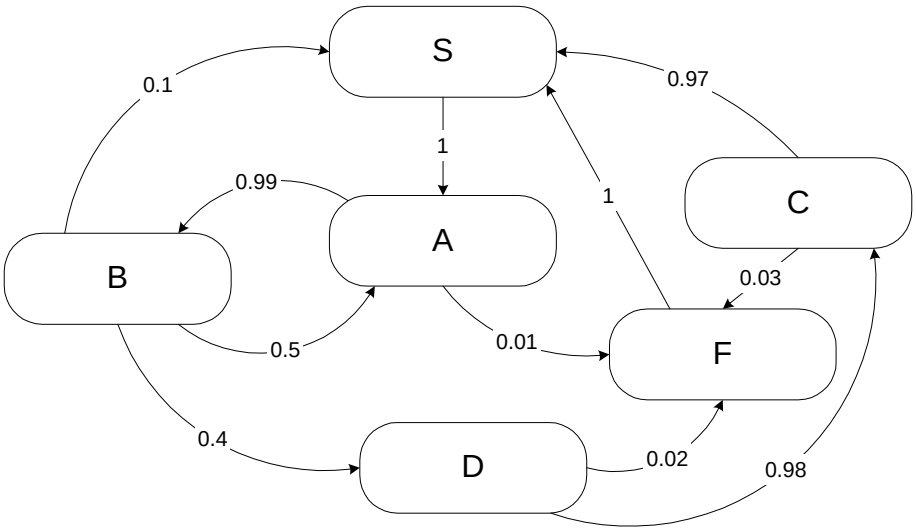
Analyysityökalun käyttämät luotettavuuden arviointitekniikat pohjautuvat *tilapohjaiseen* [32] ja *polkupohjaiseen* [2, 34] malliin. Näistä malleista voidaan johtaa kohdejärjestelmän tai sen komponenttien käyttäytymiskuvaukset graafisessa muodossa arkkitehtuurimalliin, mikä mahdollistaa luotettavuusanalyysin suorittamisen.

3.3.1 Tilapohjainen analysointimalli

Yleisesti ottaen tilapohjaista mallia voidaan käyttää sekä yksittäisten komponenttien että koko järjestelmän luotettavuuden laskemiseen niiden käyttäytymisten perusteella. Käyttäytyminen kuvataan komponentin tilojen tai järjestelmän komponenttien todennäköisyyksinä siirtyä tilasta tai komponentista toiseen. Tilapohjaiset mallit kuvataan yleensä Markovin ketjuilla. Markovin ketju on diskreettiaikainen stokastinen prosessi, jolla on Markov-ominaisuus [35]. Markov-ominaisuudella tarkoitetaan sellaista stokastista prosessia, jonka tulevaisuutta voidaan ennustaa sen nykytilan perusteella ja jonka ennustetta ei voida parantaa, vaikka prosessin kulku tunnettaisiin ennen nykytilaa [35].

Tässä työssä tilapohjaisella mallilla mallinnetaan yksittäisten komponenttien vikaantumista tilapohjaista analyysiä varten. Komponenttien virhetodennäköisyydet lasketaan niiden käyttäytymistä mallintavien tilakoneiden pohjalta rakennetuista Markovin ketjuista.

Analyysityökalua varten Markovin ketjut muodostetaan komponenttien tilakoneista korvaamalla tilakoneen tilasiirtymäehdot tilasiirtymätodennäköisyyksillä. Tämän jälkeen komponentin Markovin ketjuun lisätään komponentin vikaantumista kuvaava vikatila, johon määritellään tilasiirtymätodennäköisyydet muista tiloista [32]. Nämä todennäköisyydet perustuvat arkkitehdin arvioihin ja tietämykseen komponentista. Kuva 6 havainnollistaa Markovin ketjua, missä S on alkutila ja F on vikatila. Näin muodostetusta *vikatilamallista* voidaan laskea staattisen tilan todennäköisyydet jokaiselle tilalle, kun alkutilanne tunnetaan. Tässä tapauksessa, kun halutaan arvioida komponentin luotettavuutta, kiinnostuksen kohteena on ainoastaan vikatilan F esiintymistodennäköisyys



Kuva 6. Komponentin tilapohjainen vikatilamalli.

Havainnollistetaan komponentin virhetodennäköisyyden laskua kuvan 6 esimerkin avulla. Tilojen todennäköisyyksien laskemiseksi kuvan 6 mukaisesta Markovin ketjusta määritellään aluksi todennäköisyysvektori **p**(*n*):

$$\mathbf{p}(n) = \begin{pmatrix} p(n)_S & p(n)_A & p(n)_B & p(n)_C & p(n)_D & p(n)_F \end{pmatrix}, \qquad (2)$$

missä

$p(n)_F$ on vikatilan F esiintymistodennäköisyys.

Todennäköisyydet $p(n)_S$, $p(n)_A$, $p(n)_B$, $p(n)_C$ ja $p(n)_D$ ovat vastaavasti tilojen S, A, B, C ja D esiintymistodennäköisyydet.

Tämän jälkeen määritellään tilasiirtymämatriisi *P* seuraavasti:

$$P = \begin{pmatrix} p_{SS} & p_{SA} & \cdots & p_{SF} \\ p_{AS} & p_{AA} & \cdots & p_{AF} \\ \vdots & \vdots & \ddots & \vdots \\ p_{FS} & p_{FA} & \cdots & p_{FF} \end{pmatrix}, \qquad (3)$$

missä

p_{SA} on todennäköisyys tilasiirtymälle tilasta S tilaan A.

Muut todennäköisyydet ovat vastaavalla tavalla tilojen nimien mukaan nimettyjä tilasiirtymätodennäköisyyksiä.

Kun todennäköisyysvektori tunnetaan ajan hetkellä *n*, voidaan todennäköisyysvektori laskea ajan hetkellä *n* + 1 seuraavasti:

$$\mathbf{p}(n+1) = \mathbf{p}(n)P, \qquad (4)$$

missä

$\mathbf{p}(n)$ on edellä määritelty todennäköisyysvektori ajan hetkellä *n*

P on edellä määritelty tilasiirtymämatriisi.

Yhtälö tunnetaan myös nimellä *eteen suuntautuva Chapman-Kolmogorov-yhtälö* [36].

Todennäköisyysvektori voidaan ratkaista iteratiivisesti millä tahansa diskreetillä ajan hetkellä *n*, kun se tunnetaan ajan hetkellä 0. Koska kuvan 6 Markovin ketjun alkutilan tiedetään olevan S, sen todennäköisyysvektori hetkellä $n = 0$ on

$$\mathbf{p}(0) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Tilasiirtymämatriisi *P* on puolestaan

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0{,}99 & 0 & 0 & 0{,}01 \\ 0{,}1 & 0{,}5 & 0 & 0 & 0{,}4 & 0 \\ 0{,}97 & 0 & 0 & 0 & 0 & 0{,}03 \\ 0 & 0 & 0 & 0{,}98 & 0 & 0{,}02 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Taulukossa 3 esitetään todennäköisyysvektorin $\mathbf{p}(n)$ arvot eri ajan hetkillä *n*. Taulukosta havaitaan todennäköisyysvektorin konvergoituminen kohti staattisen tilan todennäköisyysvektoria ajan kasvaessa. Ajan hetkillä $n = 40$ ja $n = 41$ todennäköisyysvektorissa viimeisenä olevan vikatilan todennäköisyyden havaitaan pysyvän muuttumattomana. Komponentin virhetodennäköisyydeksi saadaan täten 0,009. Komponentille näin saatu virhetodennäköisyys on ns. *riippumaton virhetodennäköisyys*. Riippumattomaan virhetodennäköisyyteen ei huomioida komponentin käyttöympäristön eli järjestelmän vaikutusta komponentin vikaantumiseen, vaan se on komponentin itsenäinen ominaisuus.

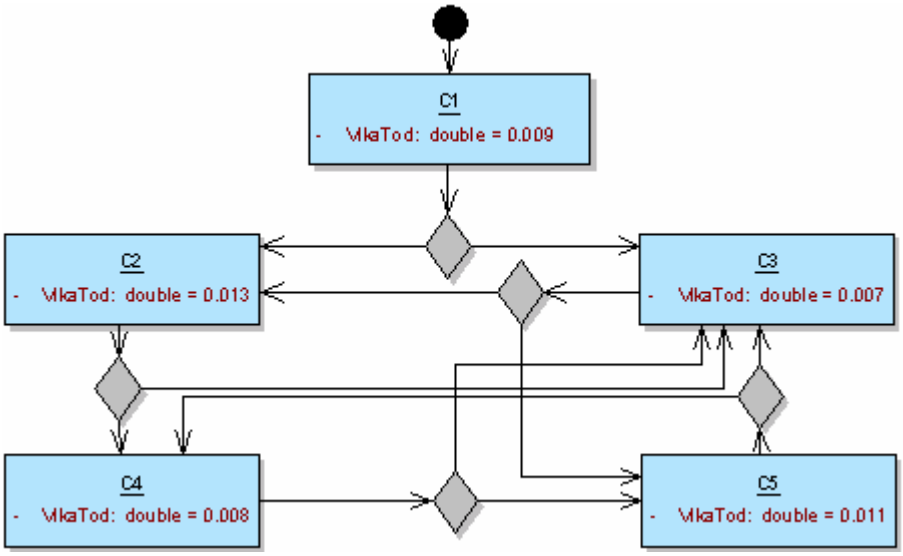
Taulukko 3. Todennäköisyysvektori p(n) ajan n funktiona.

	Todennäköisyysvektori p(n)					
n	**S**	**A**	**B**	**C**	**D**	**F**
0	1	0	0	0	0	0
1	0	1	0	0	0	0
2	0	0	0,9900	0	0	0,0100
3	0,1090	0,4950	0	0	0,3960	0
4	0	0,1090	0,4900	0,3881	0	0,0129
5	0,4383	0,2450	0,1079	0	0,1960	0,0127
6	0,0235	0,4923	0,2426	0,1921	0,0432	0,0064
7	0,2170	0,1448	0,4873	0,0423	0,0970	0,0115
8	0,1013	0,4606	0,1434	0,0951	0,1949	0,0047
9	0,1112	0,1730	0,4560	0,1910	0,0573	0,0114
10	0,2423	0,3392	0,1713	0,0562	0,1824	0,0086
⋮				⋮		
40	0,1518	0,3017	0,3006	0,1181	0,1189	0,0089
41	0,1536	0,3020	0,2987	0,1165	0,1202	0,0089

3.3.2 Polkupohjainen analysointimalli

Polkupohjaisen mallin tarkoituksena on esittää graafinen kuvaus järjestelmän käyttäytymisestä polkupohjaista analyysiä varten. Kun järjestelmän käyttäytyminen tunnetaan, voidaan komponenttien käyttöä ja niiden välisiä suorituspolkuja jäljittää komponenttipohjaisissa järjestelmissä. Järjestelmän luotettavuutta voidaan analysoida, kun polkujen suoritustodennäköisyydet ja polkuihin kuuluvat komponentit tunnetaan. RAP-menetelmässä polkupohjaista mallia käytetään koko järjestelmän luotettavuuden evaluoimiseen, kun yksittäisten komponenttien luotettavuusarviot tiedetään.

Luotettavuuden evaluoimiseksi analyysityökalu kerää tarvittavan tiedon simuloimalla järjestelmän toimintaa, jota varten järjestelmän arkkitehtuurimalliin on lisätty varta vasten analyysityökalua varten kehitetty nk. *simulointimalli*. Kuva 7 havainnollistaa yksinkertaistettua simulointimallia, jossa suorakulmiot esittävät komponentteja, vinoneliöt päätöselementtejä ja nuolet näiden välisiä linkkejä. Varsinainen simulointimalli sisältää lisäksi haarautumissäännöt, jotka analyysityökalu lukee simulointivaiheessa ja tekee niiden mukaan haarautumispäätöksen. Simulointimallin yhteyteen kuuluu vielä joukko arkkitehdin määrittelemiä *heräteviestejä*, joilla kuvataan polun ensimmäiselle komponentille välittyvää tieto-objektia. Kuvan 7 tapauksessa heräteviestit vastaanottaisi komponentti C1.



Kuva 7. Yksinkertaistettu simulointimalli.

Seuraavaksi havainnollistetaan esimerkin avulla järjestelmän virhetodennäköisyyden laskemista. Virhetodennäköisyyden laskeminen suoritetaan kaikkiaan neljässä eri vaiheessa. Aluksi lasketaan komponenttien polkukohtaiset virhetodennäköisyydet, sitten komponenttien virhetodennäköisyydet kohdejärjestelmässä, minkä jälkeen lasketaan polkujen virhetodennäköisyys. Lopuksi lasketaan koko järjestelmän virhetodennäköisyys. Oletetaan, että kuvan 7 mukaisen esimerkkimallin simuloinnin tuloksena on saatu kolme suorituspolkua P1, P2 ja P3, niiden sisältämät komponentit ja polkujen suoritustodennäköisyydet. Nämä polut esitetään taulukossa 4.

Taulukko 4. Esimerkkimallin polkujen sekvenssit ja suoritustodennäköisyydet.

Polku	Komponenttisekvenssi	Polun suoritustodennäköisyys
P1	C1-C2-C4-C5-C4	0,5
P2	C1-C2-C4-C3-C2	0,3
P3	C1-C3-C5-C3-C5	0,2

Kuvassa 7 esitetyt komponenttien virhetodennäköisyydet ovat suoritusympäristöstä riippumattomia todennäköisyyksiä, jotka saatiin edellä esitetyn tilapohjaisen analyysin tuloksena. Nyt komponenteille lasketaan suorituspolkukohtaiset todennäköisyydet seuraavan kaavan mukaisesti [34]:

$$p_{ij} = 1 - (1 - p_i)^{Nij}, \qquad (5)$$

missä

p_{ij} on komponentin *i* virhetodennäköisyys polussa *j*

p_i on komponentin riippumaton virhetodennäköisyys

Nij on komponentin *i* lukumäärä polussa *j*.

Kaavalla (5) lasketut polkukohtaiset todennäköisyydet esitetään taulukossa 5.

Taulukko 5. Esimerkkimallin polkukohtaiset virhetodennäköisyydet.

Komponentti / **Polku**	**C1**	**C2**	**C3**	**C4**	**C5**
P1	0,009	0,013	–	0,016	0,011
P2	0,009	0,026	0,007	0,008	–
P3	0,009	–	0,014	–	0,022

Polkukohtaisten todennäköisyyksien laskemisen jälkeen komponenteille lasketaan virhetodennäköisyydet koko järjestelmässä seuraavan kaavan mukaan:

$$p_{Si} = \sum_{j=1}^{n} p_{ij} p_{Pj} \,, \tag{6}$$

missä

p_{Si} on komponentin *i* virhetodennäköisyys kohdejärjestelmässä

p_{ij} on komponentin *i* virhetodennäköisyys polussa *j*

p_{Pj} on polun *j* suoritustodennäköisyys

n on polkujen lukumäärä.

Kaavan (6) mukaan lasketut komponenttien järjestelmäkohtaiset virhetodennäköisyydet esitetään taulukossa 6.

Taulukko 6. Komponenttien virhetodennäköisyydet kohdejärjestelmässä.

C1	C2	C3	C4	C5
0,009	0,014	0,005	0,010	0,010

Seuraavaksi lasketaan suorituspolkujen virhetodennäköisyydet kaikkien polkuun kuuluvien komponenttien virhetodennäköisyyksien perusteella seuraavan kaavan mukaan [2]:

$$p_{Rj} = 1 - \prod_{\forall i \in j} (1 - p_{Si}), \tag{7}$$

missä

p_{Rj} on polun j virhetodennäköisyys

p_{Si} on komponentin virhetodennäköisyys kohdejärjestelmässä.

Polkujen P1, P2 ja P3 virhetodennäköisyyksiksi saadaan 0,052, 0,052 ja 0,038 vastaavasti. Nyt koko järjestelmän virhetodennäköisyys voidaan laskea polkujen virhetodennäköisyyksien summana painottaen niitä polkujen suoritustodennäköisyyksillä seuraavasti:

$$p_J = \sum_{j=1}^{n} p_{Rj} p_{Pj}, \tag{8}$$

missä

p_J on koko järjestelmän virhetodennäköisyys

p_{Rj} on polun j virhetodennäköisyys

p_{Pj} on polun j suoritustodennäköisyys

n on järjestelmän kaikkien suorituspolkujen lukumäärä.

Tämän esimerkin järjestelmän virhetodennäköisyydeksi saadaan 0,049.

4. Analyysityökalu

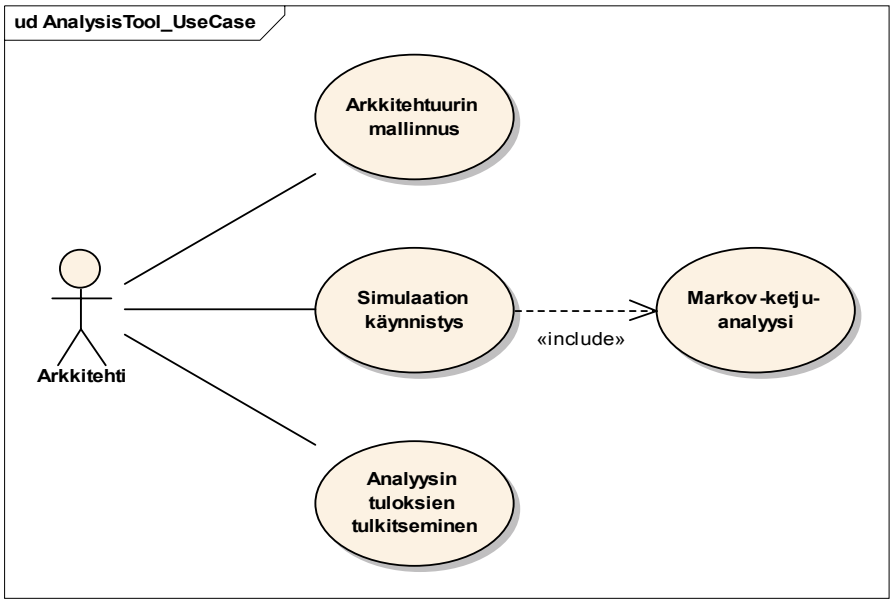
Analyysityökalun päätarkoitus on automatisoida luotettavuuden analysointi ohjelmiston arkkitehtuuritasolta. Analyysityökalu toimii arkkitehdin apuvälineenä, jotta arkkitehtuurin kyky toteuttaa sille asetetut luotettavuusvaatimukset voitaisiin ennustaa tai vaihtoehtoisista arkkitehtuurikuvauksista voitaisiin nopeasti tehdä johtopäätöksiä niiden keskinäisestä paremmuudesta luotettavuuden kannalta.

Tässä luvussa käydään ensin läpi analyysityökalun arkkitehtuurin rakenne ja komponenttien rajapinnat, sitten tutustutaan työkalun toteutukseen ja lopuksi testaukseen.

4.1 Arkkitehtuuri

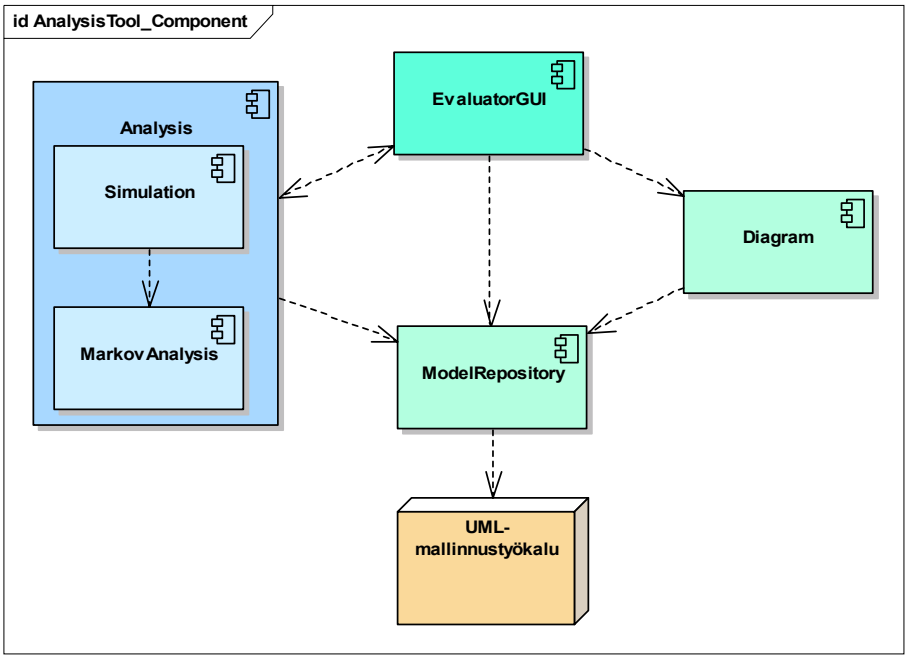
Analyysityökalun arkkitehtuurin suunnittelun tavoitteena oli suunnitella selkeä komponenttipohjainen järjestelmä, jonka komponentit ovat vaihdettavissa, jotta esimerkiksi ulkopuolisen mallinnustyökalun vaihtaminen aiheuttaisi muutoksia vain siihen yhteydessä olevaan komponenttiin.

Arkkitehtuurin suunnittelu alkoi ohjelman käyttäjälähtöisten vaatimusten kuvaamisella UML:n käyttötapauskaavion avulla (kuva 8). Analyysityökalun käytettävyyden haluttiin olevan mahdollisimman suoraviivaista ja helppoa. Kuvassa 8 esitetyssä käyttötapauksessa arkkitehti suunnittelee analysoitavan järjestelmän arkkitehtuurin, lisää sinne analyysityökalun tarvitsemat simulointi- ja vikatilamallit ja käynnistää simulaation, minkä jälkeen analyysityökalu esittää analyysin tulokset.



Kuva 8. Analyysityökalun käyttötapauskaavio.

Käyttötapauskuvauksen ja analyysityökalun vaatimusmäärittelyn (esitetty luvussa 3) perusteella lähdettiin suunnittelemaan ohjelman tarvitsemia komponentteja ja arkkitehtuurin rakennetta. Kuvassa 9 esitetään analyysityökalun arkkitehtuuriin liittyvät komponentit ja niiden riippuvuussuhteet.



Kuva 9. Analyysityökalun komponenttien riippuvuudet.

Huolimatta siitä, että arkkitehtuurin suunnitteluun ei tietoisesti haettu mitään olemassa olevaa *arkkitehtuurimallia* (engl. *architectural pattern*), sen rakenne muistuttaa arkkitehtuurimallia, joka tunnetaan myös nimellä *liitutauluarkkitehtuuri* (engl. *blackboard pattern*). Tällaisessa mallissa on tyypillisesti yhteinen tietovarastokomponentti eli liitutaulu, sitä käsittelevät komponentit ja keskitetty ohjauskomponentti. Liitutaulua vastaava komponentti analyysityökalun arkkitehtuurissa on ModelRepository, joka sisältää muiden komponenttien käyttämän yhteisen arkkitehtuurimallin. Ohjauskomponenttina puolestaan toimii EvaluatorGUI. Taulukossa 7 esitetään yhteenveto analyysityökalun komponenttien tehtävistä ja niiden rajapintojen näkyvyyksistä muille komponenteille.

Seuraavaksi määritellään analyysityökaluun kuuluvien komponenttien vastuut ja niiden rajapinnat osana arkkitehtuurikuvausta. Komponentit esitellään seuraavassa järjestyksessä: EvaluatorGUI, Diagram, Analysis, Simulation, MarkovAnalysis ja ModelRepository.

EvaluatorGUI

Käyttöliittymäkomponenttina EvaluatorGUI pääasiassa ohjaa muita komponentteja joko suoraan tai välillisesti ohjelman käyttäjän tuottamien tapahtumien mukaisesti. EvaluatorGUI ei tarjoa palveluita muille komponenteille, lukuun ottamatta suorituksen edistymispalkin päivittämistä tarjoavaa *performProgresBarStep*-metodia. Tätä metodia käyttää ainoastaan Simulation-komponentti informoidakseen ohjelman käyttäjää analyysin edistymisestä.

Taulukko 7. Yhteenveto analyysityökalun komponenteista.

Komponentti	Tehtävä	Komponentin näkyvyys
EvaluatorGUI	Tarjoaa graafisen käyttöliittymän	Vain Analysis-komponentti näkee tämän
Diagram	Mallin kaavioiden esittäminen	Näkyy vain EvaluatorGUI:lle
Analysis	Arkkitehtuurimallin luotettavuuden analysointi	Näkyy vain EvaluatorGUI:lle
Simulation	Arkkitehtuurimallin simuloiminen	Näkyy vain Analysis-komponentille
MarkovAnalysis	Komponenttien tilapohjainen analysointi	Simulation- ja Analysis-komponentit näkevät tämän
ModelRepository	Mallin säilyttäminen ja sen käsittelyn kapseloiminen	Näkyy kaikille komponenteille

Diagram

Diagram-komponentin tehtävänä on UML-kaavioiden esittäminen analysoitavasta arkkitehtuurimallista. Tällä tavoin käyttäjä voi tarkkailla esim. simulaatiomallia työkalusta käsin tarvitsematta avata sitä mallinnustyökalussa. Kaavioiden esittämiseen Diagram-komponentti tarjoaa kaksi metodia, jotka esitetään taulukossa 8.

Taulukko 8. Diagram-komponentin rajapinta.

Metodi	Parametrit	Paluu-arvo	Poikkeukset	Tehtävä
showSimulationDiagram	–	bool	Kaaviota ei löydy	Simulaatiomallin esittäminen
showStateDiagram	–	bool	Kaaviota ei löydy	Markov-ketjun esittäminen

Analysis

Analysis-komponentin tehtävä on arkkitehtuurimallin luotettavuuden analysoiminen simulaation ja Markov-ketju-analyysin tuloksista. Simulation- ja MarkovAnalysis-komponentit ovatkin Analysis-komponentin alikomponentteja. Analysis-komponentin tarjoamat metodit esitetään taulukossa 9. Analyysi käynnistetään *analyseSystem*-metodilla ja analyysin tulokset saadaan *getAnalysisResults*-metodilla. Järjestelmä analysoidaan kohdassa 3.3.2 esitetyn esimerkin mukaisesti.

Taulukko 9. Analysis-komponentin rajapinta.

Metodi	Parametrit	Paluuarvo	Poikkeukset	Tehtävä
analyseSystem	–	–	Simulaatio ei onnistu	Järjestelmän simulointi
getAnalysisResults	–	ResultTable	ResultTablea ei ole luotu	Analyysin tulosten hakeminen

Simulation

Simulation-komponentin tehtävä on suorittaa järjestelmän simulointi luotettavuuden analysoimista varten arkkitehdin tekemästä arkkitehtuurimallista. Komponentti tarjoaa kolme metodia, jotka esitetään taulukossa 10. *SimulateSystem*-metodilla, joka toteuttaa suurimman osan koko työkalun toiminnallisuudesta, käynnistetään mallin simulointi. *SimulateSystem*-metodin kutsumisen jälkeen simulaation tuloksista saadaan yhteenveto kutsumalla *getSimulationResult*-metodia. Polkukohtaiset virhetodennäköisyydet saadaan *getPathReliability*-metodilla, jolle annetaan parametrina lista polkuun kuuluvista komponenteista.

Taulukko 10. Simulation-komponentin rajapinta.

Metodi	Parametrit	Paluuarvo	Poikkeukset	Tehtävä
simulateSystem	–	bool	Heräteviestit virheellisiä, simulointimalli virheellinen	Järjestelmän simuloiminen
getSimulationResult	–	Hashtable	Hashtablea ei ole luotu	Simulaation tulosten hakeminen
getPathReliability	ArrayList	double	–	Polun luotettavuuden laskeminen

MarkovAnalysis

MarkovAnalysis-komponentti vastaa analysoitavan arkkitehtuurin komponenttien virhetodennäköisyyksien laskemisesta. Tätä varten komponentin rajapinta sisältää *doMarkovAnalysis*-metodin. Muita metodeja ei komponentilla ole. *DoMarkovAnalysis*-metodin kutsumisesta seuraa analysoitavan komponentin vikatilamallia esittävän Markovin ketjun hakeminen ModelRepository-komponentilta, josta lasketaan komponentin virhetodennäköisyys kohdassa 3.3.1 esitetyn esimerkin mukaisesti. Virhetodennäköisyyden laskemisen jälkeen päivitetään ModelRepositoryssa olevaa mallia, jolloin laskutulos on muiden komponenttien hyödynnettävissä. Metodi tuottaa poikkeuksen, jos analysoitavan komponentin tilakaaviota ei löydy tai tilakaavio on mallinnettu virheellisesti.

ModelRepository

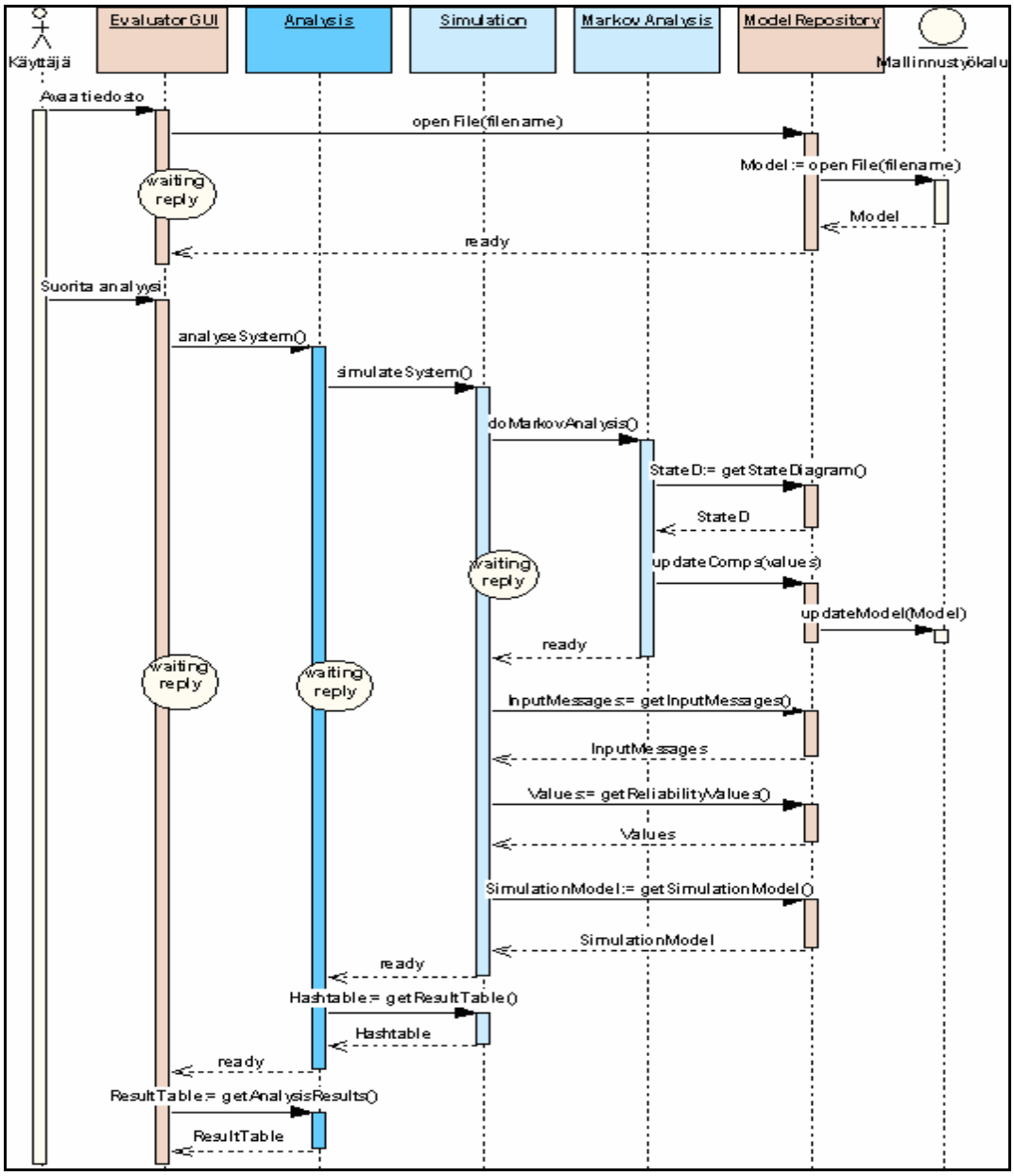
ModelRepository-komponentin tehtävä on kapseloida arkkitehtuurimallin käsittely ja tarjota vakiosäilytyspaikka mallille, josta muut komponentit voivat sitä vuorollaan käyttää. ModelRepository on ainoa komponentti, joka on yhteydessä ulkopuoliseen mallinnustyökaluun. Komponentin rajapinnan sisältämät metodit esitetään taulukossa 11. *OpenFile*-metodi avaa arkkitehtuurimallin sisältävän tiedoston parametrina annettavan tiedoston nimen perusteella. Arkkitehtuurimallista haetaan tietoa komponentin tarjoamilla *getStateDiagram*-, *getInputMessages*-, *getSimulationModel*- ja *getReliabilityValues*-metodeilla ja malliin voidaan päivittää komponenttien virhetodennäköisyysarvoja *updateComps*-metodilla, jolle arvot syötetään parametrina.

Taulukko 11. ModelRepository-komponentin rajapinta.

Metodi	**Parametrit**	**Paluu-arvo**	**Poikkeukset**	**Tehtävä**
openFile	String	Model	Mallia ei löydy	Mallin sisältävän tiedoston avaus
getStateDiagram	–	StateD	Tilakaaviota ei löydy	Tilakaavion hakeminen
getInputMessages	–	Messages	Viestejä ei löydy	Heräteviestien hakeminen
getSimulationModel	–	SModel	Mallia ei löydy	Simulaatiomallin hakeminen
getReliabilityValues	–	double[]	Arvoja ei löydy	Todennäköisyys-arvojen hakeminen
updateComps	double[]	–	–	Todennäköisyys-arvojen päivitys malliin

Analyysityökalun tyypillinen käyttöskenaario on kuvassa 10. Tässä skenaariossa ohjelman käyttäjä suorittaa ainoastaan kaksi komentoa: arkkitehtuurimallin tiedoston avauksen ja analyysin käynnistyksen. Kun käyttäjä avaa tiedoston, EvaluatorGUI-komponentti kutsuu ModelRepositoryn *openFile*-metodia, joka edelleen välittää kutsun mallinnustyökalulle. Onnistuneen tiedostonavauksen jälkeen ModelRepository tallentaa mallin, ja EvaluatorGUI saa tiedon tästä. Tämän jälkeen analyysityökalun käyttäjä käynnistää järjestelmän analyysin.

Käyttäjän antama komento välittyy EvaluatorGUI:n kautta Analysis-komponentille, joka kutsuu Simulation-komponentin *simulateSystem*-metodia. Ennen varsinaista simulaation suorittamista Simulation-komponentti kutsuu MarkovAnalysis-komponentin *doMarkovAnalysis*-metodia. Tämän metodin kutsumisen jälkeen MarkovAnalysis hakee järjestelmän komponenttien vikatilamallit tietovarastosta, suorittaa niille Markov-analyysin ja päivittää analyysin tuloksena saatavat luotettavuusarvot tietovarastoon. Kun Simulation-komponentti saa tiedon Markov-analyysin valmistumisesta, haetaan tietovarastosta tarvittava informaatio, kuten heräteviestit, komponenttien luotettavuusarvot ja simulointimalli, ja suoritetaan simulointi. Simuloinnin suorituksesta ilmoitetaan Analysis-komponentille, joka pyytää *getResultTable*-metodia ja kutsuu simuloinnin tuloksia niiden analysoimista varten. Analyysin valmistumisesta puolestaan ilmoitetaan EvaluatorGUI:lle, joka kutsuu Analysis-komponentin *getAnalysisResults*-metodia. *GetAnalysisResults*-metodi palauttaa analyysin tulokset käyttöliittymäkomponentille, joka näyttää tulokset analyysityökalun käyttäjälle.



Kuva 10. Järjestelmän analyysin suoritussekvenssi.

4.2 Toteutus

Seuraavissa kohdissa tarkastellaan analyysityökalun toteutusta ja toteutukseen liittyviä rajoitteita ja reunaehtoja. Tarkoituksena ei ole tuoda esille analyysityökalun yksityiskohtaista toteutusta vaan kertoa toteutuksesta yleisellä tasolla.

4.2.1 Tekninen toteutus ja rajoitteet

Enterprise Architectin tarjoamat liityntävaihtoehdot

Mallinnustyökaluksi valittu Sparx Systemsin Entrerprise Architect tarjoaa kolme eri mekanismia, joilla ulkopuolinen työkalu, esim. analyysityökalu, voi vaihtoehtoisesti liittyä siihen:

1. XML-pohjaisen rajapinnan kautta
2. kiinteä toteutus COM-rajapinnan kautta
3. erillinen toteutus COM-rajapinnan kautta.

Sopivimman vaihtoehdon valitseminen näistä kolmesta liityntämekanismista pohjautui kahteen analyysityökalulle asetettuun vaatimukseen. Näistä ensimmäinen vaatimus oli, että ulkopuolisen mallinnustyökalun tulee olla mahdollisimman helposti vaihdettavissa. Toisena valintaan vaikuttanut vaatimus oli, että arkkitehtuurimallia on pystyttävä lukemisen lisäksi myös muokkaamaan.

Ensimmäinen liityntämekanismi XML (eXtensible Markup Language) -pohjaisen rajapinnan kautta tarjoaa parhaimmat edellytykset ensimmäisen vaatimuksen toteuttamiseen. XML [37] on alustariippumaton tekstimuotoinen laajennettavissa oleva tiedon esitystapa, jota nykyisin useat mallinnustyökalut käyttävät mallien siirtämiseksi työkaluista toiseen XMI (XML Metadata Interchange) -formaatin [38] mukaisesti. Enterprise Architectin XML-rajapinta tarjoaa kuitenkin vain operaatiota mallin lukemiseen eikä muokkaamiseen, mikä oli toisena vaatimuksena. Tätä vaihtoehtoa ei voitu siksi käyttää.

Toisessa vaihtoehdossa mallinnustyökalun yhteydessä toimiva työkalu sisällytetään kiinteästi mallinnustyökaluun, jolloin työkalu käynnistetään mallinnustyökalun työkaluvalikon kautta. Tässä vaihtoehdossa Enterprise Architect tarjoaa monipuolisimman rajapinnan näistä kolmesta vaihtoehdosta, mikä mahdollistaa mallinnustyökalun ja siihen liitetyn työkalun välille parhaan vuorovaikutuksen. Esimerkiksi arkkitehtuurimallin elementtien valintoja ja mallinnustyökalun työkaluvalikon käyttöä voidaan seurata lisätyökalun kautta tässä vaihtoehdossa. Tämän liityntämekanismin runsaasta toiminnallisuudesta huolimatta sen katsottiin olevan liian riippuvainen Entrerprise Architect -mallinnustyökalusta.

Kolmannessa vaihtoehdossa lisätyökalu toteutetaan irrallaan mallinnustyökalusta. Lisätyökalu käynnistetään omasta exe-tiedostosta itsenäisen sovelluksen tapaan. Tässä liityntätavassa arkkitehtuurimallin avaaminen lisätyökalun kautta käynnistää automaattisesti prosessin Enterprise Architectista tausta-ajoon, jolloin lisätyökalun tarjoama mal-

linkäsittelyrajapinta on käytettävissä. Tämä liityntätapa valittiin analyysityökalun toteutukseen, sillä sen katsottiin vastaavan parhaiten analyysityökalun teknisiä vaatimuksia.

Kaksi jälkimmäistä liityntätapaa käyttävät COM (Component Object Model) -rajapintaa mallinnustyökalun ja lisätyökalun välillä. COM on Microsoftin spesifikaatio ohjelmistokomponenttien väliselle kommunikoinnille [39]. Ohjelmistokomponentit voivat olla hajautettuna monissa prosesseissa tai tietokoneissa.

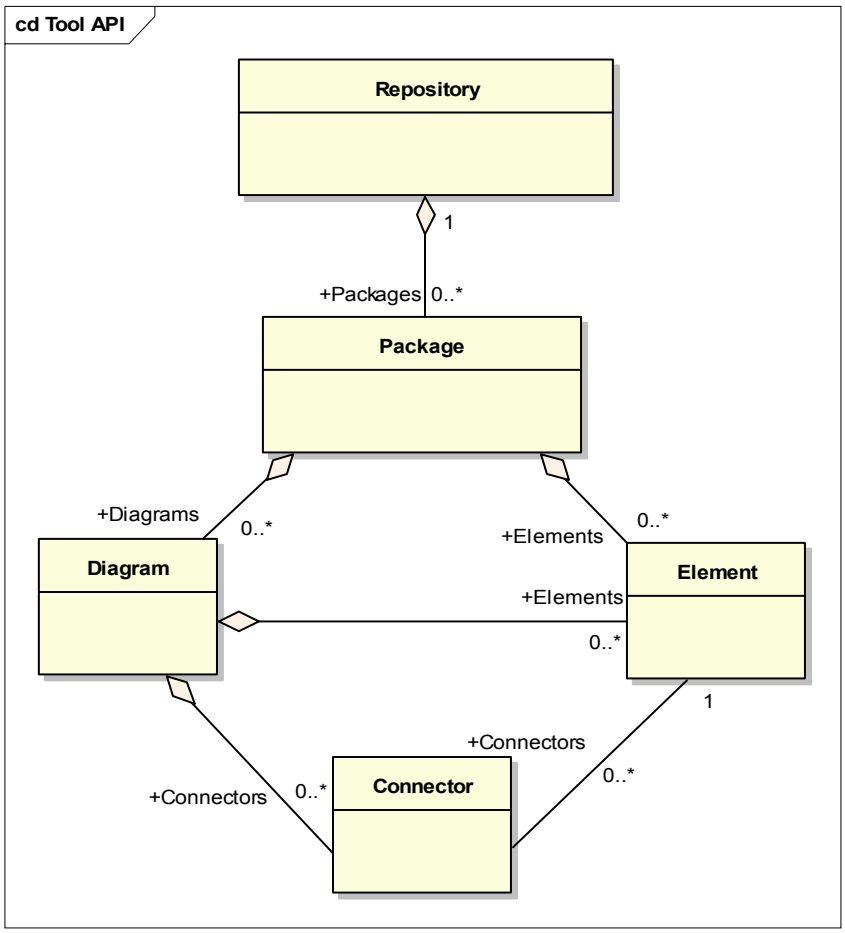
Kehitysympäristö

Koska analyysityökalun ja Entrerprise Architectin välinen kommunikointi päätettiin toteuttaa COM-rajapinnan kautta, kehitysympäristöksi olisi voitu valita mikä tahansa COM-objektien muodostamiseen pystyvä ympäristö. Luonnolliselta vaihtoehdolta tuntui Microsoftin Visual Studio .NET 2003 [40], johon oli myös valmis esimerkkisovellus. Myös Borlandin Delphiä harkittiin, mutta aikaisemman kokemuksen puuttuessa tästä kehitysympäristöstä luovuttiin.

Visual Studion tarjoamista useista ohjelmointikielistä valittiin C#, koska mallinnustyökalun mukana tullut esimerkkisovellus oli ohjelmoitu sillä. Vaikka C#:sta ei ollutkaan aikaisempaa kokemusta, sen syntaksi oli kuitenkin nopeasti omaksuttavissa ja siinä oli piirteitä hyvin paljon Javasta ja C++:sta, joista tämän opinnäytetyön kirjoittajalla oli aikaisempaa kokemusta.

Enterprisen Architectin rajapinta

Enterprise Architect tarjoaa monipuolisen kokoelman arkkitehtuurimallin käsittelytoimintoja COM-pohjaisen rajapinnan kautta. Rajapinta on jaettu hierarkkisesti osiin kuvan 11 osoittamalla tavalla. Ylimpänä lohkona olevan Repository-luokan kautta päästään käsiksi arkkitehtuurimallin muihin osiin. Analyysityökalun ModelRepository-komponentti luo ilmentymän tästä mallinnustyökalun Repository-luokasta, jonka kautta mallia käsitellään. Repository-luokka sisältää hakemistorakenteen, joka on organisoitu Package-luokalla muodostettujen pakettien avulla. Paketit puolestaan sisältävät esim. arkkitehdin mallintamia UML-kaavioita, jotka ovat Diagram-luokan ilmentymiä. Viime kädessä kaaviot muodostuvat Element-luokan elementeistä, joiden ilmentyminä voivat olla esim. luokka, komponentti tai objekti, ja elementtejä yhdistävistä Connector-luokan linkeistä. Element-luokan jäsenet eivät välttämättä kuulu mihinkään kaavioon, vaan ne voivat sisältyä suoraan pakettiin kuvan 11 mukaisesti.



Kuva 11. Enterprise Architectin rajapinnan hierarkkinen rakenne.

Enterprise Architectin rajapinnan laajuuden takia rajapinnan sisältämiä metodeja ja attribuutteja ei ole mielekästä esittää tässä kokonaisuudessaan. Esimerkkinä rajapinnan käytöstä tarkastellaan kuvassa 12 esitettävän ohjelmakoodin avulla arkkitehtuurimallin selausta Enterprise Architectin rajapinnan tarjoamien palveluiden kautta. Tässä esimerkissä arkkitehtuurimallista etsitään *Dynamic View* -nimisestä paketista aktiviteettikaavio, jonka löytymisen jälkeen kaikkien kaavion sisältämien elementtien tyyppi tulostetaan. Tämän paketin aktiviteettikaaviota käytetään varsinaisessa toteutuksessa esim. simulointimallin mallintamiseen. Esimerkissä oletetaan, että Repository-luokasta on luotu mallin sisältävä ilmentymä nimeltä *repository*.

```
(1)  EA.Package root = (EA.Package)repository.Models.GetAt(0);
(2)  for (int i = 0; i < root.Packages.Count; i++)
     {
(3)   EA.Package package = (EA.Package)root.Packages.GetAt(i);
(4)   if (package.Name.Equals("Dynamic View"))
       for (int j = 0; j < package.Diagrams.Count; j++)
       {
(5)     EA.Diagram diagram = (EA.Diagram)package.Diagrams.GetAt(j);
(6)     if (diagram.GetType().Equals("Activity"))
         for (int k = 0; k < diagram.Elements.count; k++)
         {
(7)       EA.Element element = (EA.Element)diagram.Elements.GetAt(k);
(8)       Console.WriteLine("Elementin tyyppi on: " + element.GetType());
         }
       }
     }
```

Kuva 12. Esimerkkiohjelma mallin käsittelyyn.

Kuvan 12 ohjelmalistaus toteuttaa seuraavat vaiheet:

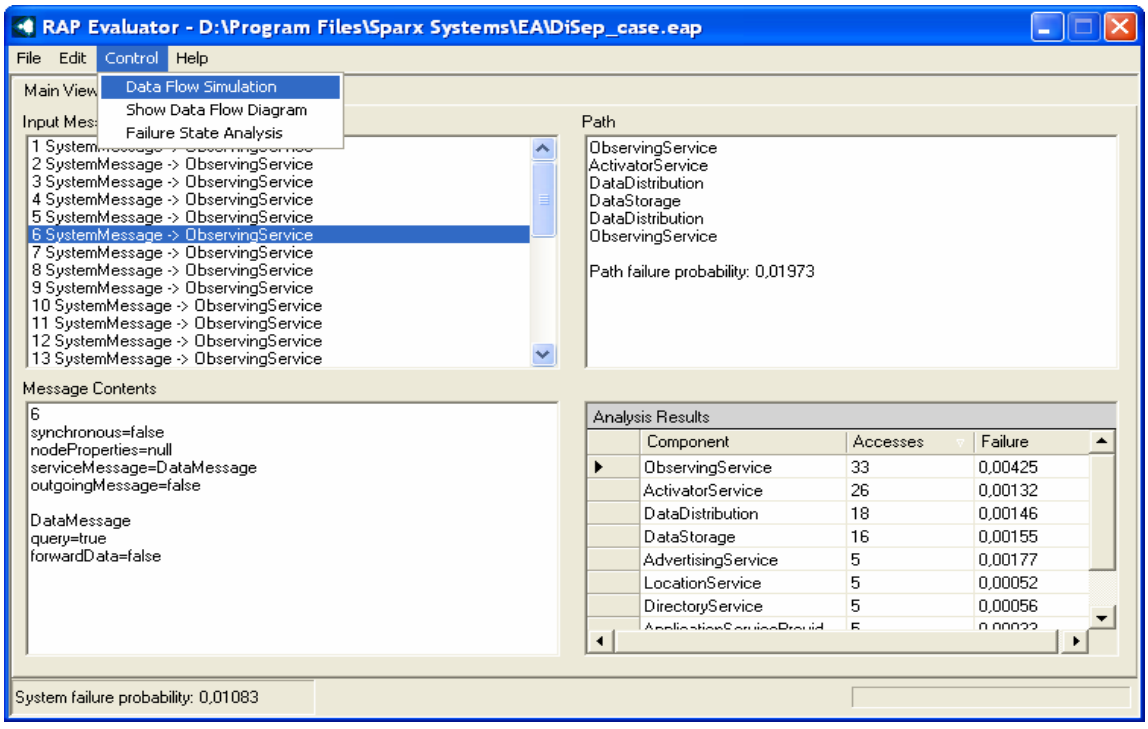
1. Haetaan arkkitehtuurimallin juurihakemisto muuttujaan *root*.
2. Luodaan silmukka, jossa käydään kaikki juurihakemiston paketit läpi.
3. Paketti tallennetaan muuttujaan *package*.
4. Jos paketin *package* nimi on *Dynamic View*, käydään paketin sisältämät kaaviot läpi silmukassa.
5. Kaavio tallennetaan muuttujaan *diagram*.
6. Jos kaavio *diagram* on tyyppiä *Activity* eli aktiviteettikaavio, käydään läpi kaikki kaavioon kuuluvat elementit silmukassa.
7. Tallennetaan elementti muuttujaan *element*.
8. Haetaan elementin tyyppi *GetType*-metodilla ja tulostetaan se.

4.2.2 Analyysityökalun käyttöliittymä

EvaluatorGUI-komponentilla toteutettavan analyysityökalun graafisen käyttöliittymän tehtävä on mahdollistaa ohjelman hallinta ja tarjota käyttäjälle tietoa analyysin tuloksista. Kuvassa 13 esitetään analyysityökalun pääikkuna. Työkalun valikkorivi on jaettu neljään valikkoon. ”File”-valikko sisältää tiedoston avaus- ja sulkemistoiminnot. ”Edit”-valikko pitää sisällään tekstin kopiointi- ja liittämistoiminnot. ”Control”-valikossa on toiminnot analyysin käynnistämiseen, simulointimallin esittämiseen ja komponenttien

tilapohjaisen analyysin suorittamiseen. ”Help”-valikosta saadaan tietoja työkalusta ja työkalun tekijästä. Käsiteltävä arkkitehtuurimallin sisältävä tiedosto näkyy työkalun otsikkorivillä.

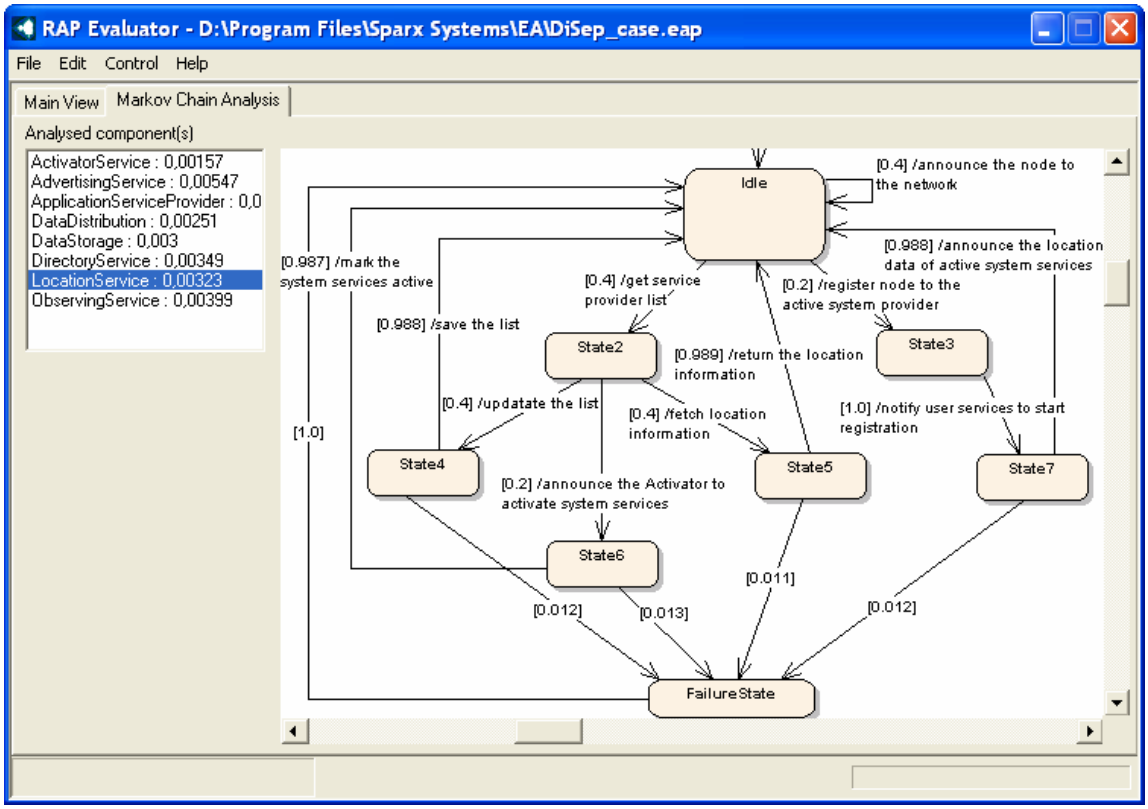
Analysoitavasta arkkitehtuurimallista kerätty ja analysoinnin tuloksena saatu informaatio esitetään työkalun pääikkunassa neljässä eri kentässä. Kuvassa 13 vasemmalla ylhäällä oleva lista näyttää arkkitehtuurimalliin sisältämät heräteviestit. Kenttä näyttää vain viestin nimen ja sen vastaanottavan komponentin. Listasta voidaan hiirellä valita jokin viesti, jolloin vasemmalla alhaalla oleva tekstikenttä näyttää valitun viestin sisällön. Viestin valitseminen päivittää myös oikealla ylhäällä olevaa tekstikenttää, joka näyttää simuloinnin tuloksena kyseisellä heräteviestillä saatuun suorituspolkuun kuuluvat komponentit suoritusjärjestyksessä ja tämän polun virhetodennäköisyyden. Oikealla alhaalla oleva yhteenvetotaulukko sisältää simulaation aikana läpikäydyt komponentit ja niiden suorituskerrat sekä komponenttien virhetodennäköisyydet kyseisessä järjestelmässä. Pääikkunan alalaidassa oleva statuspalkki näyttää koko järjestelmän virhetodennäköisyyden.



Kuva 13. Analyysityökalun pääikkuna.

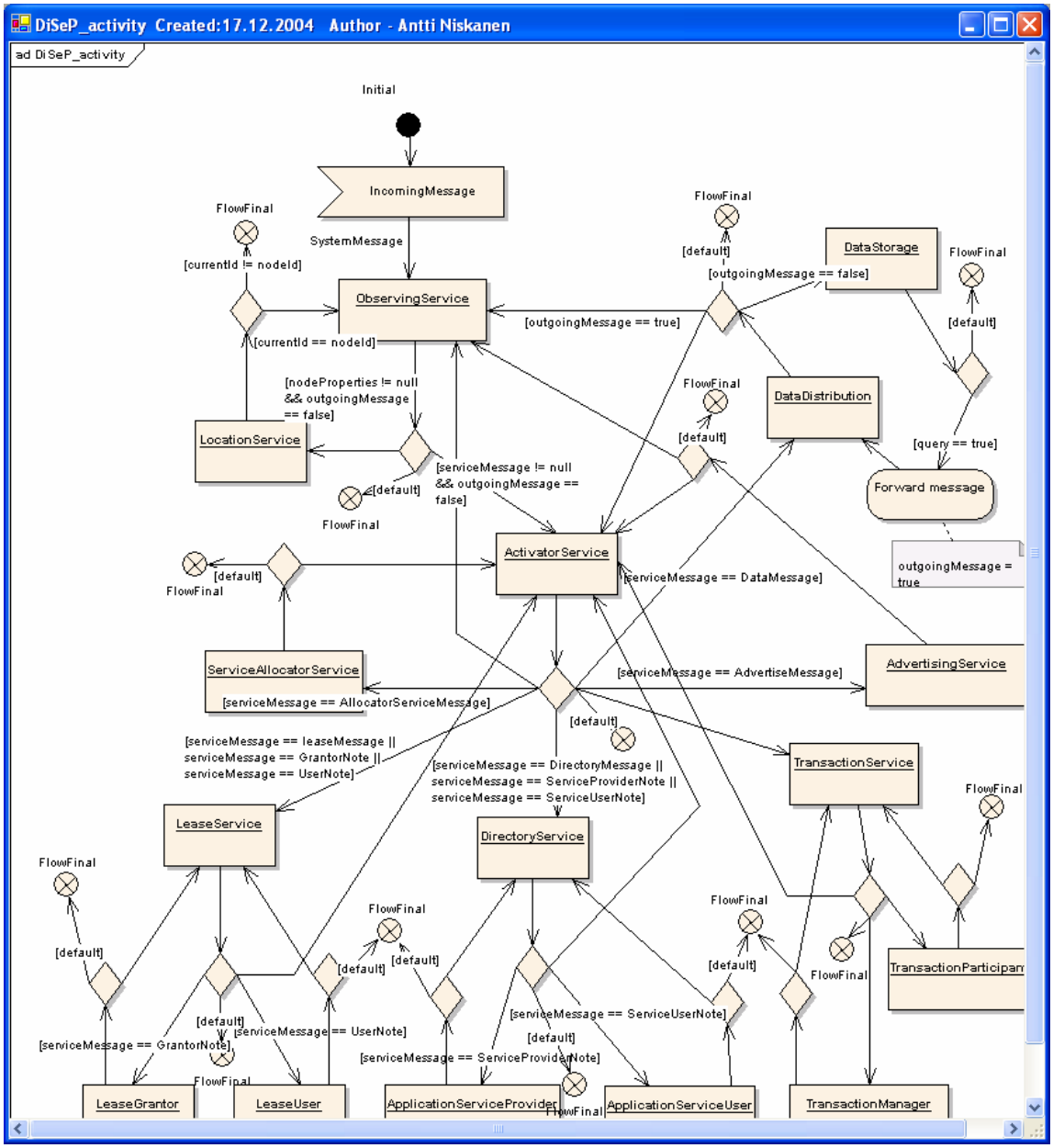
Analyysityökalussa on tilapohjaista analyysia varten oma pääikkunasta välilehdellä eroteltu ikkuna, joka on kuvassa 14. Ikkunan vasemmassa laidassa luetellaan komponentit, jotka on analysoitu niiden arkkitehtuurimallissa olevien vikatilamallien (esitelty kohdassa 3.3.1) mukaan. Komponenttien nimien vieressä on niiden järjestelmästä riippumattomat

virhetodennäköisyydet, jotka on saatu tilapohjaisen analyysin tuloksena. Ikkunan oikeassa laidassa esitetään listasta valitun komponentin vikatilamallia esittävä tilakaavio.



Kuva 14. Analyysityökalun komponenttitason analyysiä esittävä ikkuna.

Arkkitehtuurimallin sisältämää simulointimallia voidaan tarkkailla analyysityökalusta käsin tarvitsematta avata simulointimallia mallinnustyökalussa. Simulointimalli avataan kuvassa 15 olevaan erilliseen ikkunaan. Ikkunan otsikkorivillä on mallin nimi, luontipäivämäärä ja tekijä.



Kuva 15. Simulointimallin esitys analyysityökalussa.

4.3 Testaus

Analyysityökalun testauksen tavoitteena oli lähinnä varmentaa toiminnallisten vaatimusten toteutuminen. Työkalun mahdollisten suorituskykyongelmien etsiminen ei kuulunut testauksen piiriin, koska suorituskyvylle ei ollut asetettu erityisiä vaatimuksia. Testaus jaettiin seuraaviin päävaiheisiin:

1. komponenttikohtainen testaus
2. integrointitestaus
3. tulosten oikeellisuuden tarkistus.

Analyysityökalun kehitysvaiheessa komponentit testattiin aluksi itsenäisesti muista komponenteista riippumattomasti. Komponenttien testaus toteutettiin *black-box*-periaatteella, jossa komponenteille syötettiin joukko todellista tilannetta jäljitteleviä herätteitä ja tarkkailtiin komponenttien vastauksia niihin. Komponenttien sisäistä rakennetta korjattiin sitä mukaa, kun virheitä tuli ilmi niiden toiminnassa. Tätä vaihetta toistettiin kaikkien komponenttien kohdalla, kunnes niiden antamissa vastauksissa ei havaittu enää epäloogisuutta käytetyllä testijoukolla.

Seuraavaksi testattiin kokonaisuudessaan analyysityökalua, joka rakennettiin itsenäisesti testatuista komponenteista työkalun arkkitehtuurimallin mukaiseksi. Tässä vaiheessa testattiin analyysityökalun käyttöliittymän tarjoamien toimintojen oikeellisuus, kuten myös analyysityökalun mallinkäsittelytoiminnot. Mallinkäsittelytoimintojen testaukseen rakennettiin muutamia erilaisia Enterprise Archtitectilla luotuja arkkitehtuurimalleja. Osaan testijoukkona käytettävistä malleista jätettiin tarkoituksella virheitä, jotta analyysityökalun virheenkäsittelymekanismien oikea toiminta voitiin varmistaa. Analyysityökalu ilmoittaa käyttäjälle virhetapauksista esimerkiksi seuraavilla viesteillä, jos käyttäjä on yrittänyt analysoida virheellisesti mallinnettua arkkitehtuurimallia:

1. Invalid message format.
2. Illegal operator '>', Simulation aborted.
3. Invalid simulation diagram, Simulation aborted.

Ensimmäinen viesti näytetään, jos heräteviestien formaatti on virheellinen. Toinen viesti näyttää operaattorin, jota on käytetty väärin simulointimallissa olevissa päätössäännöissä. Viimeinen viesti ilmaantuu, jos simulointimallin muodostamiseen on käytetty vääriä elementtejä tai simulointimalli on muuten virheellisesti muodostettu.

Viimeiseksi tarkistettiin analyysityökalun käyttämien laskentamenetelmien virheettömyys. Markov-analyysin tulosten todentamiseksi luotiin viitteessä [32] esitetty Markovin ketju, josta laskettiin vikatilan esiintymistodennäköisyys. Huolimatta siitä, että viitteessä [32] vikatilan laskemiseen käytettiin analyysityökalun laskentatavasta poikkeavaa Poisson-approksimaatiota, molemmissa päädyttiin samaan lopputulokseen. Polkujen ja koko järjestelmän virhetodennäköisyyden laskennan oikeellisuus tarkistettiin manuaalisesti, koska laskemiseen käytetyt kaavat olivat suhteellisen yksinkertaisia.

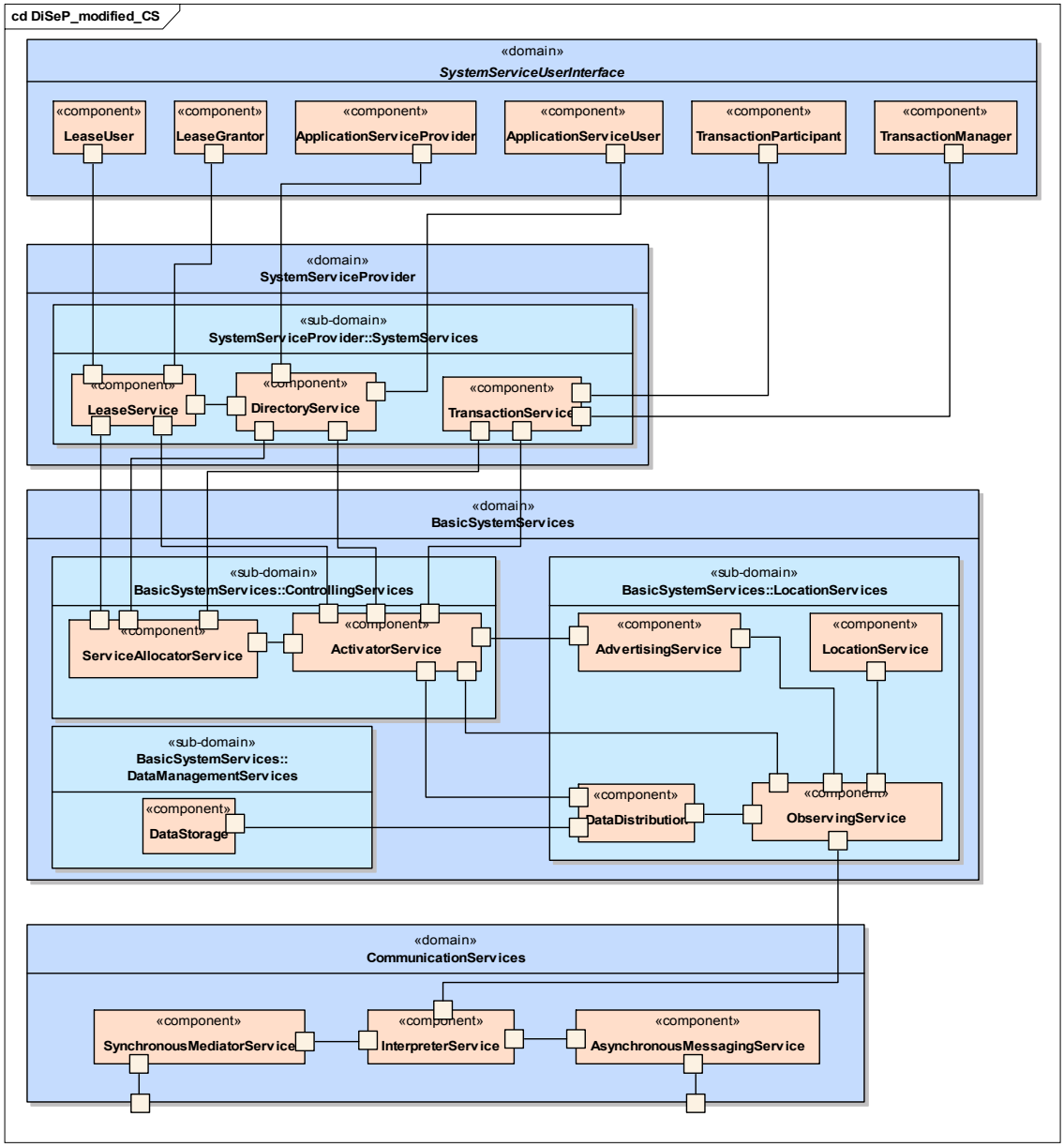
5. Tapaustutkimus

Tässä luvussa esitetään tapaustutkimus, jonka avulla demonstroidaan järjestelmän virhetodennäköisyyden analysointia sen arkkitehtuurista analyysityökalua käyttäen. Tämän tapaustutkimuksen kohdejärjestelmänä käytetään palveluiden hajautusalustaa, jota kutsutaan nimellä Distribution Service Platform (DiSeP). DiSePin tarkoituksena on tarjota joustava verkottunut ympäristö siinä olevien laskentayksikköjen väliseen vuorovaikutukseen. Tällaisia laskentayksiköitä, jotka toimivat samalla verkon solmuina, voivat olla esimerkiksi matkapuhelimet, PDA-laitteet tai tietokoneet. Tässä joustavalla verkottuneella ympäristöllä tarkoitetaan verkkoympäristöä, johon laskentayksikkö voi spontaanisti liittyä ja tarjota samalla muille yksiköille omia palveluitaan sekä saada käyttöönsä muiden yksiköiden tarjoamia palveluita. DiSePissä ei ole lainkaan keskitettyä verkkopalvelinta; sen sijaan mikä tahansa verkkoon kuuluvista laskentayksiköistä voi hoitaa palvelimen virkaa. Jos palvelimen tehtävää hoitava yksikkö kytkeytyy irti verkosta, sen tehtävä siirtyy automaattisesti jollekin toiselle yksikölle.

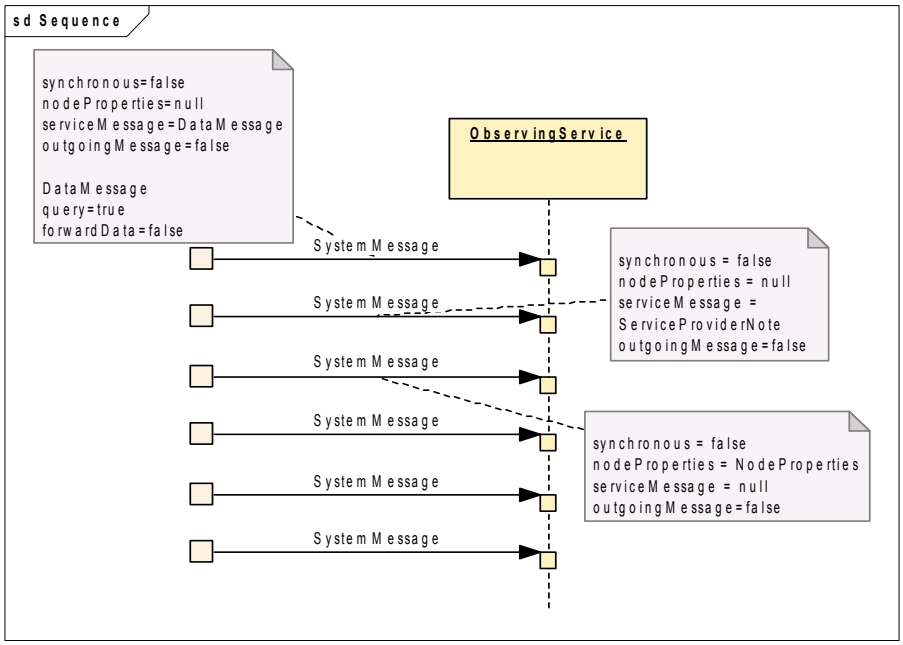
Kuva 16 esittää DiSePin konseptuaalista arkkitehtuuria, joka on jaettu neljään eri kerrokseen. Ylimpänä kerroksena oleva SystemServiceUserInterface mahdollistaa solmussa olevien palveluiden välittämisen käyttäjälle. Seuraavassa kerroksessa oleva SystemServices huolehtii järjestelmän palveluiden rekisteröinnistä ja jakamisesta. Jokaisen verkossa olevan solmun tulee toteuttaa nämä SystemService-lohkon palvelut, mutta ne ovat aktiivisia vain yhdessä solmussa kerrallaan. Muut solmut käyttävät aina sen solmun SystemService-palveluja, jossa ne ovat aktiivisena. BasicSystemServices-kerros tarjoaa autonomisesti toimivia palveluita, jotka on jaettu LocationServices-, ControllingServices- ja DataManagementServices-lohkoihin. Näistä LocationServicesin tarjoamat palvelut päivittävät listaa verkossa olevista solmuista ja niiden tunnistetiedoista, ja lisäksi ne lähettävät tietyin väliajoin ryhmäviestin muille solmuille, jotta ne tunnistaisivat kyseessä olevan solmun. LocationServices toisin sanoen mahdollistaa dynaamisesti muuttuvan verkon ylläpidon. ControllingServices-lohkon tehtävä on ohjata solmun toimintaa ja tarkkailla SystemServicesin palveluiden käyttöastetta. DataManagementService sisältää komponenttien yhteisen tietovaraston. Alin kerros CommunicationServices tarjoaa palveluja solmujen väliseen kommunikointiin.

Virhetodennäköisyyden arvioimiseksi arkkitehtuurista DiSePin käyttäytymisen tulee olla hyvin tiedossa. Sen perusteella laaditaan analyysiä varten järjestelmän käyttäytymistä kuvaava simulointimalli, joka voidaan liittää osaksi arkkitehtuurikuvausta. Mitä tarkemmin käyttäytyminen tunnetaan, sitä tarkemmin simulointimalli voidaan tehdä. Kuvassa 15 esitettiin DiSePin luotettavuuden evaluoimiseen käytetty simulointimalli. Simulointia varten täytyy määritellä joukko simulointimallille syötettäviä heräteviestejä. Simulointivaiheessa jokainen mallille syötetty heräteviesti vastaa yhtä käyttöskenaariota järjestelmästä, jonka käyttäytymistä simulointimallilla kuvataan. Näiden käyttöskenaa-

rioiden voidaan ajatella koostuvan kohdejärjestelmän komponenttien suoritussekvensseistä, joita tarvitaan analyysin myöhemmässä vaiheessa. Tällainen viestipohjainen lähestymistapa simulointiin on DiSePin tapauksessa hyvin luontevaa, koska sen verkkoympäristössä olevat solmut kommunikoivat joka tapauksessa viestien välityksellä. Simuloinnissa käytetäänkin DiSePin omia viestejä vastaavia viestejä simulointimallin heräteviesteinä. Kuva 17 esittää heräteviestien määrittelyyn käytettyä sekvenssikaaviota. Tässä tapauksessa kaaviossa oleva ObservingService-komponentti vastaanottaa muutamia heräteviestejä. Kuvan sisältämät kommenttilaput eivät varsinaisesti kuulu viestien määrittelyyn, mutta ne havainnollistavat viestien sisältöjä, jotka eivät muuten näkyisi kaaviossa. Lisäksi todellisuudessa heräteviestejä määritellään huomattavasti enemmän.



Kuva 16. DiSePin konseptuaalinen arkkitehtuuri.



Kuva 17. Heräteviestit simulaatiota varten.

Ennen kuin kohdejärjestelmälle voidaan suorittaa luotettavuuden analysointi, täytyy siinä olevien komponenttien riippumattomat virhetodennäköisyydet selvittää. Mikäli järjestelmä sisältää uusia komponentteja, joiden vikaantumiskäyttäytymisestä ei ole aikaisempaa kokemusta, joudutaan niitä varten arkkitehtuurikuvaukseen lisäämään kohdassa 3.3.1 esitetyn kaltaiset vikatilamallit virhetodennäköisyyksien arvioimiseksi. Kuvassa 14 esitetyssä analyysityökalun komponenttitason analysointia esittävässä ikkunassa nähdään DiSePin LocationService-komponentin vikatilamallia esittävä Markovin ketju. DiSePin tapauksessa arkkitehtuurikuvaukseen lisättiin vikatilamallit vain käytetyimpien komponenttien kohdalla, koska kriittisimmille komponenteille haluttiin mahdollisimman tarkat arvot, toisaalta harvoin käytetyt komponentit eivät juuri vaikuta analyysin lopputulokseen.

Luotettavuuden evaluointi kohdejärjestelmästä analyysityökalun avulla on hyvin suoraviivaista, kun järjestelmän arkkitehtuurikuvaukseen on lisätty simulaatiomalli heräteviesteineen ja siinä oleville komponenteille vikatilamallit tarpeen mukaan. Kuva 13 esittää tilannetta analyysityökalun pääikkunasta DiSeP-järjestelmän analyysin jälkeen. Taulukkoon 12 on koottu yhteenveto DiSePin komponenttien virhetodennäköisyyksistä. Suluissa olevien komponenttien riippumattomat virhetodennäköisyydet ovat puhtaasti arkkitehdin arvioita, koska niille ei ollut käytettävissä vikatilamalleja.

Taulukko 12. DiSePin komponenttien virhetodennäiköisyydet.

Komponentti	**Riippumaton virhetod.**	**Virhetod. DiSePissä**	**Suorituskerrat**
ActivatorService	0,0016	0,0013	26
AdvertisingService	0,0055	0,0018	5
ApplicationServiceProvider	0,0019	0,0002	5
ApplicationServiceUser	(0,006)	0,0002	2
DataDistribution	0,0025	0,0015	18
DataStorage	0,0030	0,0016	16
DirectoryService	0,0035	0,0006	5
LeaseGrantor	(0,006)	0,0001	1
LeaseService	(0,006)	0,0002	2
LeaseUser	(0,006)	0,0001	1
LocationService	0,0032	0,0005	5
ObservingService	0,0040	0,0043	33
ServiceAllocatorService	(0,006)	0,0002	3
TransactionManager	(0,006)	0,0001	1
TransactionParticipant	(0,006)	0,0001	1
TransactionService	(0,006)	0,0001	1

Analysoidun DiSeP-järjestelmän kokonaisvirhetodennäköisyydeksi saatiin 0,011. On huomattava, että tämä koko järjestelmän virhetodennäköisyys riippuu paljolti siitä, minkälaisia käyttöskenaarioita heräteviestien muodossa arkkitehti ottaa mukaan simulointiin. Analyysin tuloksena saatu järjestelmän virhetodennäköisyys ei välttämättä vastaa täysin konkreettisen toteutuksen virhetodennäköisyyttä. Analyysistä saatava hyöty piilee siinä, että arkkitehti voi nopeasti evaluoida useita erilaisia arkkitehtuurikandidaatteja ja valita niistä parhaan tai muuttaa järjestelmää ja suorittaa analyysin uudelleen.

6. Pohdinta

Luotettavuuden mallipohjainen analysointi on askel kohti entistä laadukkaampia ohjelmistoja. Luotettavuuden tärkeys verrattuna muihin laatuattribuutteihin korostuu etenkin kriittisissä reaaliaikajärjestelmissä. Toisaalta ohjelmistoarkkitehtuurilla on huomattava merkitys valmiin ohjelman luotettavuuteen, joten menetelmien kehittäminen arkkitehtuurin luotettavuuden analysointiin on erittäin perusteltua. Koska mikään luotettavuuden arkkitehtuuritason analysointimenetelmä ei olisi tehokas ilman oikeanlaista työkalutukea, tavoitteena on helpottaa ohjelmistoarkkitehdin työtä automatisoimalla analysoinnin raskain eli laskennallinen osuus.

Tässä opinnäytetyössä pystyttiin osoittamaan, että arkkitehtuuritason luotettavuuden analysoinnin automatisointi on mahdollista arkkitehtuurimallia hyödyntävän analyysityökalun avulla. Parhaassa tapauksessa esiteltyä ratkaisua voitaisiin tulevaisuudessa käyttää vähentämään arkkitehtuurin suunnittelusta johtuvia luotettavuusongelmia lopullisessa ohjelmistotuotteessa. Näin olisi mahdollista pienentää ohjelmiston testauksesta aiheutuvia kuluja, mutta kokonaan ohjelmiston testauksen tarvetta ei kuitenkaan pystyttäisi poistamaan. Luotettavuusanalyysin toimivuus riippuu tietenkin olennaisesti myös itse analysointimenetelmästä eikä pelkästään analyysityökalusta.

Teknisesti työkalulla tehty esimerkkiarkkitehtuurin luotettavuusanalyysi onnistui erittäin hyvin. Erityisesti tila- ja polkupohjaiset analysointitekniikat toteutettiin onnistuneesti. Myöskään työkalun virheenkäsittelymekanismissa ei havaittu puutteita, mikä toisaalta voi olla merkki liian suppeasta testitapausten määrästä. Työkalun testitapauksista oli vaikea saada hyvin kattavaa, koska erilaisia virheellisesti muodostettuja vikatila- tai simulointimalleja voi olla lähes lukematon määrä.

Vaikka työkalun käyttäminen oli helppoa ja analyysin tulokset lupauksia antavia esimerkkiarkkitehtuurin kohdalla, työkalu ei vielä vastaa teollisuuden tarpeita. Luottavuuden analysoinnissa käytetyllä simulointimallilla ei tällä hetkellä pystytä kuvaamaan kaikenlaisten järjestelmien käyttäytymistä riittävän tarkasti. Esimerkkinä käytetyn DiSePin arkkitehtuuri on hieman vaikuttanut simulointimallin kehitykseen ja sen nykyiseen muotoon, mikä puolestaan vähentää mallin yleiskäyttöisyyttä. Lisäksi käytetyt heräteviestit eivät välttämättä kata kaikkia järjestelmän suorituspolkuja, mikä heikentää analyysin tarkkuutta.

Työkalun käyttämät arkkitehtuurimallit mallinnettiin kaupallisella Sparx Systemsin Entrerprise Architect UML -mallinnustyökalulla. Mallinnustyökalu osoittautui hyväksi valinnaksi, eikä sen käyttämisessä ollut suurempia ongelmia. Ainoana miinuksena mainittakoon mallinnustyökalun mukana tulleiden ohjelmointiesimerkkien vähäisyys, mikä vaikeutti analyysityökalun ohjelmoinnin aloittamista. Toisaalta siihen vaikutti myös

ohjelmointiympäristönä käytetty Visual Studio .NET 2003, josta työn tekijällä oli vain hyvin vähän aikaisempaa kokemusta.

Työkalun jatkokehityksen kannalta olennaisinta olisi parantaa kohdejärjestelmän käyttäytymisen kuvaustapaa, koska analyysin luotettavuuden kannalta on hyvin tärkeää tietää järjestelmässä olevien komponenttien käyttötilanteet mahdollisimman tarkasti. Lisäksi työkalun suorittaman analyysin voisi laajentaa kattamaan luotettavuuden lisäksi myös saatavuuteen, joka on toinen RAP-menetelmän huomioimista laatuattribuuteista. Työkalutukea saatavuuden analysointiin mietittiin tutkimusongelman määrittämisen yhteydessä mutta se rajattiin pois, koska haluttiin keskittyä tärkeämmäksi katsottuun luotettavuuden analysointiin.

Analyysityökalu suunniteltiin tukemaan ja helpottamaan luvussa 3 esitetyn RAP-menetelmän käyttöä, joka on puolestaan osa laatuohjattua ohjelmistokehitystä tukevaa QADA®-metodologiaa. Vaikka tässä työssä työkalu toteutettiin erillisenä ohjelmana, työkalun tarkoitus on tulevaisuudessa toimia osana suurempaa arkkitehtuuritason laadun analyysiohjelmistoa, jossa arkkitehtuurista analysoidaan luottavuuden lisäksi myös muita laatuattribuutteja.

7. Yhteenveto

Ohjelmistojen luotettavuuden parantaminen on tärkeää, sillä ihmiset käyttävät päivittäisten asioiden hoitamiseen yhä useammin tietotekniikkaa. Ohjelmistojen virhetilanteet voivat aiheuttaa taloudellisia tappioita, tapaturmia tai jopa kuolemantapauksia. Ohjelmiston arkkitehtuuritason luotettavuusanalyysin tarkoituksena on löytää mahdollisia ongelmakohtia kohdejärjestelmän arkkitehtuurista mahdollisimman varhaisessa kehitysvaiheessa, jolloin muutoksia on helpompi tehdä kuin myöhemmässä vaiheessa. Olemassa olevat luotettavuuden analysointimenetelmät ovat tehottomia ilman työkalutukea, koska luotettavuuden analysointiin liittyy paljon laskentaa.

Tässä työssä toteutettiin työkalu mallipohjaiseen analysointiin RAP-menetelmän tueksi. Työkalun tehtävänä oli suorittaa luotettavuuden evaluoinnin laskennallinen osuus, jotta arkkitehti pystyisi nopeasti analysoimaan, toteuttaako arkkitehtuuri sille asetetut luotettavuusvaatimukset, ja vertailemaan eri arkkitehtuurikandidaatteja ja valitsemaan niistä luotettavuuden kannalta parhaan. Työkalun suorittamaa analysointia varten kehitettiin kohdejärjestelmän käyttäytymistä kuvaava simulointimalli ja sen komponenttitason virhetilanteita kuvaava vikatilamalli. Molemmat mallit kuvattiin Enterprise Architect -työkalulla varsinaisen analysoitavan järjestelmän arkkitehtuurikuvauksen yhteyteen. Mallien suunnittelun jälkeen toteutettiin analyysityökalu siten, että se pystyi käsittelemään simulointi- ja vikatilamalleja Enterprise Architectin ulkoisen ohjelmointirajapinnan kautta.

Työkalun luotettavuusanalyysi todettiin teknisesti toimivaksi yksinkertaisessa tapaustutkimuksessa, jossa analysoitiin onnistuneesti esimerkkiarkkitehtuurin luotettavuutta. Kuitenkin työkalun käyttämän simulointimallin todettiin olevan liian rajoittunut, jotta sillä pystyttäisiin kuvaamaan tarkasti minkä tahansa järjestelmän käyttäytyminen.

Kaiken kaikkiaan tässä työssä esitelty työkalu arkkitehtuuritason mallipohjaiseen luotettavuuden analysointiin on askel kohti aiempaa laadukkaampien ohjelmistojen kehittämistä. Työkalu on kuitenkin vielä kaukana valmiista ja vaatii paljon parannuksia ennen kuin sillä on edellytyksiä teollisuuden käyttöön.

Lähdeluettelo

[1] Reussner, R. H., Schmidt, H. W. & Poernomo, I. H. 2003. Reliability prediction for component-based software architectures. Journal of Systems and Software, Vol. 66, No. 3, s. 241–252.

[2] Krishnamurthy, S. & Mathur, A. P. 1997. On the estimation of reliability of a software system using reliabilities of its components. 8th International Symp. Sotfware Reliability Engineering (ISSRE97).

[3] Yacoub, S., Cukic, B. & Ammar, H. 1999. Scenario-based reliability analysis of component-based software. 10th International Symposium on Software Reliability Engineering (ISSRE99).

[4] Immonen, A. 2005. A method for predicting reliability and availability at the architectural level. Hyväksytty: Research Issues in Software Product-Lines – Engineering and Management, T. Käkölä & J. C. Dueñas (toim.).

[5] Merilinna, J. & Matinlassi, M. 2004. Evaluation of UML Tools for Model-Driven Architecture. 11th Nordic Workshop on Programming and Software Development Tools and Techniques. Turku, Finland: Åbo Akademi.

[6] Bass, L., Clements, P. & Kazman, R. 1998. Software Architecture in Practice. Reading, MA: Addison-Wesley. 452 s.

[7] Shaw, M. & Garlan, D. 1996. Software Architecture: Perspectives on an Emerging Discipline. Upper Saddle River, NJ: Prentice Hall Publishing. 242 s.

[8] Bosch, J. 2000. Design and use of software architectures: adopting and evolving a product-line approach. Harlow: Addison-Wesley. 354 s.

[9] ISO/IEC, 2001. ISO/IEC 9126-1 International Standard: Software engineering – Product quality. Part 1: Quality model. 25 s.

[10] Weiss, D., Lai, C. & Tau, R. 1999. Software product-line engineering: a family-based software development process. Reading, MA: Addison-Wesley. 426 s.

[11] Purhonen, A., Niemelä, E. & Matinlassi, M. 2004. Viewpoints of DSP Software and Service Architectures. Journal of Systems and Software, Vol. 69, No. 1–2, s. 57–73.

[12] Kruchten, P. 1995. The 4+1 View Model of Architecture. IEEE Software, Vol. 12, No. 6, s. 42–50.

[13] Jaaksi, A., Aalto, J.-M., Vättö, W. & Aalto, A. 1999. Tried & True Object Development: Industry-Proven Approaches with UML. Cambridge Univ.: Cambridge University Press. 315 s.

[14] Hofmeister, C., Nord, R. & Soni, D. 1999. Applied Software Architecture. Reading, MA: Addison-Wesley. 397 s.

[15] Dobrica, L. & Niemelä, E. 2000. A strategy for Analyzing Product Line Software Architectures. VTT Publications 427. Espoo: VTT Technical Research Centre of Finland. 124 s.

[16] Matinlassi, M. & Niemelä, E. 2003. The Impact of Maintainability on Component-based Software Systems. 29th Euromicro Conference. Antalya, Turkey.

[17] Matinlassi, M., Niemelä, E. & Dobrica, L. 2002. Quality-driven architecture design and quality analysis method. A revolutionary initiation approach to a product line architecture. VTT Publications 456. Espoo: VTT Technical Research Centre of Finland. 129 s. + liitt. 10 s.

[18] Matinlassi, M. 2004. Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA and QADA. 26th International Conference on Software Engineering (ICSE'04). Edinburgh, Scotland, United Kingdom.

[19] OMG, 2003. Unified Modeling Language (UML) 2.0 Specification. Object Management Group.

[20] OMG, 2005. Object Management Group (1.4.2005), URL: http://www.omg.org.

[21] Berkenkötter, K. 2003. Using UML 2.0 in Real-Time Development: A Critical Review. SVERTS: Specification and Validation of UML models for Real Time and Embedded Systems. San Francisco, 20.10.2003. 14 s.

[22] Koskimies, K., Koskinen, J., Maunumaa, M., Peltonen, J., Selonen, P., Siikarla, M. & Systä, T. 2004. UML työvälineenä ja tutkimuskohteena. Tietojenkäsittelytiede, No. 21, s. 19–51.

[23] Li, W. & Henry, S. 1993. Object-Oriented Metrics that Predict Maintainability. Journal of Systems and Software, Vol. 23, No. 2, s. 111–122.

[24] Bosch, J. & Molin, P. 1999. Software Architecture Design: Evaluation and Transformation. Teoksessa: IEEE Engineering of Computer Based Systems Symposium (ECBS99). S. 4–10.

[25] Kazman, R., Abowd, G. & Bass, L. 1993. Analyzing the Properties of User Interface Software Architectures. Carnegie Mellon University, School of Computer Science.

[26] Harju, H. 2002. Kustannustehokas ohjelmiston luotettavuuden suunnittelu ja arvionti. Osa 1. VTT Tiedotteita 2151. Espoo: VTT Technical Research Centre of Finland. 114 s. + liitt. 15 s.

[27] Fenton, N. E. & Neil, M. 1999. Software Metrics: successes, failures and new directions. The Journal of Systems and Software, Vol. 47, s. 149–157.

[28] McCabe, T. J. 1976. A Complexity Measure. IEEE Transactions on Software Engineering, Vol. SE 2, No. 4, s. 308–320.

[29] Halsted, M. H. 1977. Elements of Software Science. New York: Elsevier-North Holland.

[30] Smith, C. U. 1990. Performance Engineering of Software Systems. Boston, MA: Addison-Wesley.

[31] Liu, J. W. S. & Ha, R. 1995. Efficient Methods of Validating Timing Constraints. Teoksessa: Son, S. H. (toim.) Advanced in Real-Time Systems. Upper Saddle River, NJ: Prentice Hall.

[32] Thomason, M. G. & Whittaker, J. A. 1999. Rare Failure-State in a Markov Chain Model for Software Reliability. 10th International Symposium on Software Reliability Engineering. Boca Raton, Florida.

[33] Immonen, A. & Niskanen, A. 2005. A tool support for the reliability and availability prediction method. 31th Euromicro Conference. Porto, Portugal, 30.8.–3.9.2005.

[34] Cortellessa, V., Singh, H., & Cukic, B. 2002. Early reliability assessment of UML based software models. Third International Workshop on Software and Performance. Rome, Italy.

[35] Markov, A. A. 1971. Extension of the limit theorems of probability theory to a sum of variables connected in a chain. Reprinted in Appendix B of: R. Howard. Dynamic Probabilistic Systems, volume 1: Markov Chains. New York: John Wiley and Sons.

[36] Papoulis, A. 1984. Probability, Random Variables, and Stochastic Processes. 2nd ed. New York: Mc-Graw Hill.

[37] W3C. XML, a technical recommendation standard of the W3C. URL: http://www.w3.org/TR/REC-xml.

[38] OMG, 2003. XML Metadata Interchange (XMI) Specification. Object Management Group.

[39] Platt, D. S. 1998. The Essence of COM with ActiveX. A programmer's workbook. Second ed. Upper Saddle River, NJ: Prentice Hall Inc. 623 s.

[40] Microsoft Visual Studio .NET 2003. URL: http://msdn.microsoft.com/vstudio/.

Julkaisija

Julkaisun sarja, numero ja raporttikoodi

VTT Tiedotteita 2331
VTT–TIED–2331

Tekijä(t)
Niskanen, Antti

Nimeke

Työkalu luotettavuuden mallipohjaiseen analysointiin

Tiivistelmä

Mallipohjaisella analysoinnilla tarkoitetaan ohjelmiston laadun arviointia, joka perustuu ohjelmiston arkkitehtuurimalliin, joka kuvaa ohjelmiston rakenteen ja sen käyttäytymisen. Arkkitehtuuritason analysointi tehdään ohjelmistokehityksen alkuvaiheessa, jolloin mahdollisten ongelmakohtien korjaaminen on yksinkertaisempaa ja halvempaa verrattuna toteutetun ohjelmiston korjaamiseen.

Tässä työssä kehitetään ja toteutetaan työkalu ohjelmiston arkkitehtuuritason luotettavuuden analysointiin. Työkalun tarkoituksena on helpottaa ja nopeuttaa olemassa olevan analysointimenetelmän käyttöä, jotta ohjelmistoarkkitehti pystyisi näkemään, toteuttaako kohteena olevan järjestelmän arkkitehtuuri sille asetetut luotettavuusvaatimukset. Työkalun suorittamaa analysointia varten kehitettiin erilliset mallit yksittäisten ohjelmistokomponenttien ja koko järjestelmän luotettavuuden analysointiin hyödyntäen standardia graafista kuvaustapaa. Nämä mallit kuvattiin osana kohdejärjestelmän arkkitehtuurimallia.

Arkkitehtuurimallien kuvaamiseen käytettiin Sparx Systemsin Enterprise Architect -mallinnustyökalua, jonka kautta kehitettävä työkalu pystyy lukemaan ja käsittelemään arkkitehtuurimallia hyödyntämällä mallinnustyökalun tarjoamaa ulkoista ohjelmointirajapintaa. Työkalun kehitysympäristönä käytettiin Microsoftin Visual Studio .NET 2003:a ja työkalu toteutettiin C#-ohjelmointikielellä.

Yhteenvetona voidaan todeta, että teknisesti luotettavuusanalyysi onnistui esimerkkinä käytetystä arkkitehtuurimallista tässä työssä kehitetyn työkalun avulla, mutta tarvitaan huomattavasti jatkokehittelyä, ennen kuin työkalua voidaan käyttää teollisuudessa.

Avainsanat
model-based analysis, analysis tools, software quality, software systems, reliability, requirements, software architecture, Unified Modelling Language, testing

ISBN
951–38–6776–5 (nid.)
951–38–6777–3 (URL: http://www.vtt.fi/publications/index.jsp)

Avainnimeke ja ISSN
VTT Tiedotteita – Research Notes
1235–0605 (nid.)
1455–0865 (URL: http://www.vtt.fi/publications/index.jsp)

Projektinumero
E3SU00217

Julkaisuaika	Kieli	Sivuja	Hinta
Maaliskuu 2006	Suomi, engl. tiiv.	58 s.	B

Projektin nimi
Families

Toimeksiantaja(t)
ITEA-projekti / Eureka Σ! 2023, Tekes

Yhteystiedot

VTT
PL 1100, 90571 OULU
Puh. vaihde 020 722 111
Faksi 020 722 2320

Myynti

VTT
PL 1000, 02044 VTT
Puh. 020 722 4404
Faksi 020 722 4374

Published by

Series title, number and
report code of publication

VTT Research Notes 2331
VTT–TIED–2331

Author(s)
Niskanen, Antti

Title

A tool for model-based reliability analysis

Abstract

Model-based analysis is about analysing software quality from the architectural models, which describe the structure and behaviour of a system. Architectural level analysis is realised in the early development phase of the software system, when the changes caused by potential problems of a system are not as complex and expensive to make as correcting the source code.

In this thesis, a tool for analysing reliability of the software system at architectural level is developed and implemented. The purpose of the tool is to support the usage of an existing reliability analysis method and assist an architect to validate whether or not the quality requirements are met in the system architecture. For perform the analysis by the tool, separate models were developed for analysing the reliability of single software components and the whole system using a standard graphical notation. These models were described as part of the system's architectural model.

Sparx System's Enterprise Architect was used as a modelling tool for the representing the architectural models. The analysis tool was able to access the architectural models through Enterprise Architect's external application interface. For the development environment, Microsoft's Visual Studio .NET 2003 was used, and the tool was implemented using C#.

In summary, reliability analysis of the example architecture succeeded technically with the tool developed in this thesis. However, further development is considerably needed before industrial use of the tool.

Keywords
model-based analysis, analysis tools, software quality, software systems, reliability, requirements, software architecture, Unified Modelling Language, testing

ISBN
951–38–6776–5 (soft back ed.)
951–38–6777–3 (URL: http://www.vtt.fi/publications/index.jsp)

Series title and ISSN	Project number
VTT Tiedotteita – Research Notes 1235–0605 (soft back edition) 1455–0865 (URL: http://www.vtt.fi/publications/index.jsp)	E3SU00217

Date	Language	Pages	Price
March 2006	Finnish, Engl. abstr.	58 p.	B

Name of project	Commissioned by
Families	ITEA-projekti / Eureka Σ! 2023, Tekes

Contact	Sold by
VTT Technical Research Centre of Finland P.O. Box 1100, FI-90571 OULU, Finland Phone internat. +358 20 722 111 Fax +358 20 722 2320	VTT P.O.Box 1000, FI-02044 VTT, Finland Phone internat. +358 20 722 4404 Fax +358 20 722 4374

VTT Tiedotteita – Research Notes

2312 McKeough, Paterson, Solantausta, Yrjö, Kyllönen, Hilkka, Faaij, Andre, Hamelinck, Carlo, Wagener, Martijn, Beckman, David & Kjellström, Björn. Techno-economic analysis of biotrade chains. Upgraded biofuels from Russia and Canada to the Netherlands. 2005. 40 p. + app. 25 p.

2313 Sassi, Jukka, Viitasalo, Satu, Rytkönen, Jorma & Leppäkoski, Erkki. Experiments with ultraviolet light, ultrasound and ozone technologies for onboard ballast water treatment. 2005. 80 p. + app. 2 p.

2314 Häkkinen, Kai. Hankintatoimen ulkoistus metalliteollisuudessa. 2005. 77 s. + liitt. 3 s.

2315 Lahdenperä, Pertti, Nykänen, Veijo & Rintala, Kai. Elinkaarimallit. Tilapalveluhankkeiden vaihtoehtoiset toimintatavat. 2005. 56 s.

2316 Oedewald, Pia, Reiman, Teemu & Kurtti, Reetta. Organisaatiokulttuuri ja toiminnan laatu metalliteollisuudessa. 11 tapaustutkimusta suomalaisissa pk-yrityksissä. 2005. 81 s. + liitt. 4 s.

2317 Ajanko, Sirke, Moilanen, Antero & Juvonen, Juhani. Jätteiden syntypaikka-lajittelujärjestelmän ja käsittelytekniikan vaikutus kierrätyspolttoaineen laatuun. 2005. 83 s. + liitt. 21 s.

2318 Hostikka, Simo, Mikkola, Esko, Rinne, Tuomo, Tillander, Kati & Weckman, Henry. Henkilöturvallisuuden kehittäminen maanalaisissa tiloissa paloriskejä pienentämällä. 2005. 143 s. + liitt. 9 s.

2319 Weckman, Henry. Henkilöturvallisuuden kehittäminen maanalaisissa tiloissa paloriskejä pienentämällä. Tehtävä B: Poistumisturvallisuus. 2005. 93 s. + liitt. 13 s.

2320 Pöyhönen, Ilpo. Lääkintälaitteiden ohjelmistot. Suunnittelun kehityskohteita vesiputous- ja XP-mallin näkökulmasta. 2006. 61 s. + liitt. 2 s.

2321 Tsupari, Eemeli, Monni, Suvi & Pipatti, Riitta. Non-CO_2 greenhouse gas emissions from boilers and industrial processes. Evaluation and update of emission factors for the Finnish national greenhouse gas inventory. 2005. 82 p. + app. 24 p.

2323 Arnold, Mona, Kuusisto, Sari, Wellman, Kari, Kajolinna, Tuula, Räsänen, Jaakko, Sipilä, Jorma, Puumala, Maarit, Sorvala, Sanna, Pietarila, Harri & Puputti, Katja. Hajuhaitan vähentäminen maatalouden suurissa eläintuotantoyksiköissä. 2006. 74 s. + liitt. 12 s.

2324 Kivisaari, Sirkku & Saranummi, Niilo. Terveydenhuollon systeemiset innovaatiot vuorovaikutteisen kehittämisen kohteena. Case Pro Viisikko. 2006. 77 s. + liitt. 4 s.

2325 Häkkinen, Tarja, Rauhala, Kari & Huovila, Pekka. Rakennetun ympäristön kestävän kehityksen kriteerit ja indikaattorit. 2006. 89 s. + liitt. 29 s.

2327 Security-tutkimuksen roadmap. Mika Naumanen & Veikko Rouhiainen (toim.). 2006. 69 s.

2330 Apilo, Tiina & Taskinen, Tapani. Innovaatioiden johtaminen. 2006. 112 s. + liitt. 10 s.

2331 Niskanen, Antti. Työkalu luotettavuuden mallipohjaiseen analysointiin. 2006. 58 s.

Tätä julkaisua myy
VTT
PL 1000
02044 VTT
Puh. 020 722 4404
Faksi 020 722 4374

Denna publikation säljs av
VTT
PB 1000
02044 VTT
Tel. 020 722 4404
Fax 020 722 4374

This publication is available from
VTT
P.O. Box 1000
FI-02044 VTT, Finland
Phone internat. + 358 20 722 4404
Fax + 358 20 722 4374

ISBN 951-38-6776-5 (nid.)
ISSN 1235-0605 (nid.)
ISBN 951-38-6777-3 (URL: http://www.vtt.fi/inf/pdf/)
ISSN 1455-0865 (URL: http://www.vtt.fi/inf/pdf/)