TAMPERE UNIVERSITY OF TECHNOLOGY

PEKKA ALHO

APPLICATION OF NEW AUTOMATION SOFTWARE DESIGN AND INTEGRATION TECHNOLOGIES IN TEACHING

Master of Science Thesis

Examiner: Professor Seppo Kuikka
Examiner and topic approved in the Automation, Mechanical and Materials Engineering Faculty Council on 19.08.2009

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Automation Engineering
ALHO, PEKKA: Application of New Automation Software Design and Integration Technologies in Teaching
Master of Science Thesis, 95 pages, 33 appendix pages
September 2009
Major: Automation Technology
Examiner: Professor Seppo Kuikka
Keywords: Automation software engineering, study material design, learning, OSGi, UML automation profile, .NET Micro Framework

Automation software is produced by Finnish companies from various sectors of industry, including processing, electronics and forest machinery industries. This thesis is part of the Research and development environment for distributed automation systems (HAJA, or Hajautettujen automaatiojärjestelmien tutkimus- ja kehitysympäristö in Finnish) research project, the goal of which is to produce a research and development environment for distributed automation systems in order to test new software technologies. The new technologies are important for ensuring the competitiveness of the companies in the future.

The work done in this thesis is part of a university prototype for this environment which aims to raise the learning capabilities of students up to the best international standards in the area of automation software engineering, in addition to studying new technologies. In order to meet these goals this thesis presents three cutting edge automation software design and integration technologies: UML Automation Profile, OSGi Framework and .NET Micro Framework. Connecting themes for these technologies are that they raise the level of abstraction in software development and support reusability.

The Automation Profile is an UML extension that includes modelling concepts and three new diagram types that can be used to model software for automation systems. OSGi adds a dynamic modularity layer to Java platform and supports component based software engineering. Communication between components in OSGi is possible to implement with service-oriented architecture. .NET Micro Framework enables software development for resource constrained and embedded devices with a modern object-oriented language.

The university prototype, i.e. material produced for courses ACI-32020 and ACI-32040, includes weekly exercises, assignments and written educational material. In order to improve the pedagogic quality of the material, this work also includes research on basic learning theories and their applications. Important learning aspects for the assignment development included induced realisation or insight, learner-centric approach, repetition, deep processing and hands-on experience. The successfulness of the material is hard to analyze before it has been put to use on the courses but the quality of learning process should generally be better if a pedagogic approach and educational psychological principles are applied already in the design phase. Nevertheless, the material should not be seen as a finished product but rather as a target for development, based on the feedback from students and the experiences of course staff.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Automaatiotekniikan koulutusohjelma
ALHO, PEKKA: Uusien automaatiosovellussuunnittelu ja -integrointiteknologioiden käyttö opetuksessa
Diplomityö, 95 sivua, 33 liitesivua
Syyskuu 2009
Pääaine: Automaatioteknologia
Tarkastaja: professori Seppo Kuikka
Avainsanat: automaation ohjelmistotuotanto, opintomateriaalin suunnittelu, oppiminen, OSGi Framework, UML-automaatioprofiili, .NET Micro Framework

Automaatiosovelluksia tuotetaan useissa suomalaisissa eri alojen teollisuusyrityksissä. Sovelluksia löytyy mm. prosessi-, elektroniikka ja metsäkoneteollisuuden tuotteista. Teollisuuden kilpailukyvyn takaamiseksi ”Hajautettujen automaatiojärjestelmien tutkimus- ja kehitysympäristö” (HAJA) -tutkimusprojektin tarkoituksena on tuottaa tutkimus- ja kehitysympäristö hajautetuille automaatiojärjestelmille, jossa testataan uusia ohjelmistoteknisiä innovaatioita ja teknologioita. Uusien teknologioiden tuomat edut ovat tärkeitä etenkin nykyisessä taloudellisessa tilanteessa, kun yritysten täytyy onnistua viemään projektit läpi entistä nopeammin rajoitetuilla resursseilla. Ympäristöstä aiotaan tuottaa sekä yliopisto- että teollisuusprototyypit, joilla voidaan arvioida erilaisten uusien automaation ohjelmistotekniikan tuottavuutta parantavien teknologioiden etuja ja puutteita.

Tämä diplomityö on tehty osana kyseisen projektin yliopistoprototyyppiä, jonka tehtävänä on teknologioiden arvioinnin lisäksi TTY:n opiskelijoiden oppimisvalmiuksien kohottaminen alan kansainväliselle huipputasolle. Tavoitteen saavuttamiseksi työssä esitellään kolme kehityksen kärkeä edustavaa automaatiosovellussuunnittelu ja -integrointiteknologiaa, sekä niiden käyttöönotto automaation tietotekniikan opetukseen. Teknologiat ovat TTY:llä kehitetty UML-automaatioprofiili ja sen käytön mahdollistava suunnittelutyökalu AP Tool, OSGi Framework ja .NET Micro Framework. Oppimateriaaleihin integroidut teknologiat eivät muodosta yhtenäistä kokonaisuutta, mutta tästä huolimatta niillä on yhtenäisiä piirteitä, kuten abstraktiotason nosto automaation ohjelmistotuotannossa ja uudelleenkäytettävyyden tukeminen.

Automaatioprofiili on ohjelmistotekniikassa yleisesti suunnittelun tukena käytetyn UML-mallinnuskielen laajennos. Profiili täydentää kieltä automaatiosuunnitteluun sopivalla käsitteistöllä ja kaavioilla, joilla voidaan kuvata vaatimuksia, säätörakenteita ja automaatiotoimintoja. UML:n käyttö automaatio-ohjelmistojen mallintamisessa on toistaiseksi harvinaista, mutta tämä johtuu osittain mm. UML:n epämuodollisuudesta, mikä voi johtaa väärinkäsityksiin suunnittelussa. Automaatioprofiili korjaa tämän puutteen automaatioalakohtaisella käsitteistöllään, joka voidaan nähdä myös automaatiosuunnittelun eri osa-alueita yhdistävänä tekijänä. Profiili on suunniteltu alustariippumattomaksi ja helposti laajennettavaksi erilaisten standardien mukaisilla käsitteillä. Profiilin käyttäminen suunnittelussa on mahdollista Eclipse-ohjelmointiympäristöön pohjautuvalla AP Tool -työkalulla.

OSGi Framework on OSGi Alliancen hallinnoima standardi, joka lisää suosittuun Java-ohjelmistoalustaan siitä puuttuvan dynaamisen moduulikerroksen, joka mahdollistaa ohjelmistojen komponenttipohjaisen kehityksen. Komponenttien välinen

kommunikointi voidaan OSGissa toteuttaa palvelupohjaisesti, jolloin komponentit voivat kertoa mitä palveluita ne tarjoavat ja odottavat olevan saatavilla muilta komponenteilta. Palvelupohjaisuus yhdistettynä dynaamisuuteen, eli komponenttien pysäyttämiseen ja uudelleenlataamiseen ajonaikaisesti, tekevät OSGista tehokkaan ja robustin työkalun, joka mm. mahdollistaa komponenttien ajonaikaisen päivittämisen ja edistää komponenttien uudelleenkäytettävyyttä. Näiden etujen saavuttaminen edellyttää kuitenkin komponenttien huolellista suunnittelua. Java-pohjaisuuden ansiosta OSGi-sovellukset ovat alustariippumattomia, mutta mm. roskienkeruun ja ajonaikaisen kääntämisen takia suoritus ei ole determinististä. Vaikka OSGi onkin alun perin kehitetty tietoverkkoihin liitettyihin sulautettuihin laitteisiin, jotka edellyttävät ohjelmistoalustaltaan hyvää suorituskykyä, OSGia ei Java-pohjaisuutensa takia voi helposti suositella kovia reaaliaikavaatimuksia sisältäviin kohteisiin.

.NET Micro Framework on Microsoftin sulautettuihin laitteisiin tarkoitettu .NET-ohjelmistoalustan kevyin versio. Alusta tukee ohjelmistokehitystä nykyaikaisella C#-ohjelmointikielellä Visual Studio -ohjelmointiympäristössä. Kehittynyt oliopohjainen ohjelmointikieli mahdollistaa abstraktiotason nostamisen aiempaa korkeammalle perinteiseen C- ja Assembly-pohjaiseen sulautettujen laitteiden ohjelmistokehitykseen verrattuna. Abstraktiotason nosto parantaa edellytyksiä käyttää kehittyneitä suunnittelumalleja apuna toteutuksessa ja nopeuttaa ohjelmistokehitysprosessia.

Teknologioiden oppimateriaaleiksi toteutettiin viikkoharjoitukset, harjoitustyöt ja kirjallista materiaalia opetusmonisteisiin kursseille ACI-32020 Automaation reaaliaikajärjestelmät ja ACI-32040 Automaation ohjelmistokomponentit ja sovelluspalvelut. Toteutettujen oppimateriaalien didaktisen laadun varmistamiseksi työssä on perehdytty oppimisteorioihin ja niiden sovelluksiin. Diplomityön teknisen luonteen ja pituuden takia aiheen syvällinen esittely ei ole mahdollista – eikä välttämättä edes hyödyllistä – mutta didaktisten näkökohtien huomioiminen materiaalin tuottamisessa pitäisi parantaa sen laatua. Oppimisteorioita ja niiden sovelluksia käytetään tässä työssä materiaalin suunnittelun tukemisen lisäksi mallinnuksen merkityksen pohdiskeluun oppimisen kannalta ja oppimistasojen arviointiin.

Tärkeiksi periaatteiksi oppimateriaalin suunnittelussa nousivat oivalluttaminen, oppijakeskeisyys, materiaalin ymmärtäminen ulkoa opettelemisen sijaan ja käytännön kokemuksen saaminen. Opiskelijoiden lähtötiedot on otettava huomioon, jotta materiaalin vaikeustaso olisi sopiva ja opiskelijoiden oppimismotivaatio säilyisi. Motivaatio ja kiinnostus aiheeseen ovat tärkeitä myös aiheen ymmärtämisen kannalta. Motivaatiota voidaan tukea esim. käyttämällä kiinnostavia ja käytännönläheisiä esimerkkejä materiaalissa tai tarjoamalla valinnanvaraa harjoitustyöaiheissa. Käytännön kokemuksen saamista nykyaikaiset oppimisteoriat pitävät tärkeänä osana oppimisprosessia. Tätä kokemusta on mahdollista hankkia viikkoharjoituksista ja harjoitustöistä em. kursseilla.

Tässä diplomityössä oppimisteorioita ja niiden soveltamista on käytetty työtä yhdistävänä tekijänä. Lopullisia johtopäätöksiä opetusmateriaalin toimivuudesta on hankala tehdä ennen kuin ne on otettu käyttöön kurssien opetuksessa, mutta materiaalin tulisikin muuttua dynaamisesti palautteen ja opetuksen tarpeiden mukaan. Mahdollisia kohteita jatkotutkimukselle voisivat olla esimerkiksi mallintamisen käyttö oppimisen tukena ja teknologioiden jatkotestaaminen teollisuusprototyypeillä

PREFACE

This work has been carried out in the Department of Automation Science and Engineering at Tampere University of Technology as a part of the HAJA research project.

First, I would like to express my gratitude to the supervisors of this work, Prof. Seppo Kuikka and M.Sc. Timo Vepsäläinen. Things got a little busy during the last weeks of writing this thesis, thank you for guidance and always having time to comment and explain everything to me.

I would also like to thank my colleagues, especially researchers M.Sc. Outi Laitinen and M.Sc. David Hästbacka, for valuable comments on various things.

My parents and friends deserve special thanks for their support throughout the working process.

Finally, I would like to thank the English lecturers at TUT for interesting lectures and sparking an interest in the English language.

The writing of this thesis has been an interesting journey – one you are so engaged to take that you look around at the end of it and cannot really believe how far you have come. Hopefully I will be able to take many more like it in the future.

Tampere 9.9.2009

_____

Pekka Alho

CONTENTS

ABBREVIATIONS AND NOTATION

AP	UML Automation Profile.
API	Application Programming Interface.
ASE	The Automation Software Engineering research group or the Department of Automation Science and Engineering.
AUKOTON	Research project, full name in English is Uninterrupted development path for automation applications - From P&I diagram to control software.
bundle	Unit of modularisation in OSGi framework.
CBSE	Component-Based Software Engineering.
DCS	Distributed Control System, an automation system where the computing units are deployed in a decentralized manner.
Eclipse	A popular software development platform comprising of an IDE and a plug-in system to extend it.
EDK	Embedded Development Kit.
Equinox	An OSGi framework implementation.
HAJA	Research project that this thesis is part of, full name in English is Research and development environment for distributed automation systems.
IDE	Integrated Development Environment.
IEC-61131-3	Standard that defines four languages for PLC programming.
OMG	Object Management Group, the consortium developing UML.
OPC	Object Linking and Embedding for Process Control, automation industry standard used for communication of real-time plant data between devices.
OSGi	Open component framework for Java platform, developed by the OSGi Alliance.
PBL	Problem-Based Learning, an instructional strategy.
PLC	Programmable Logic Controller, a computer type used in automation and control systems.
QoSFT	UML profile for modelling Quality of Services and Fault Tolerance.
RT	Real-Time.
SOA	Service Oriented Architecture.
SysML	Systems Modeling Language, a UML extension.
TUT	Tampere University of Technology.
UML	Unified Modeling Language.
USVA	Research project, full name in English is The impact of new technologies and standards on automation design.

1. INTRODUCTION

This certainly is not the easiest of times for automation software engineers. Today's industrial environment is plagued by growing project sizes, increased pressure for doing more profit with less resources, intensive schedules, current global economical situation, inconsistent practices, communications with several project parties, unrefined software engineering methods, version control… The list of challenges is not a short one. Although companies may be reluctant to channel funds to research when at the same time they must consider firing their valuable human resources, this is a time when the decisions to invest or not to invest in technological innovations can really make or break the companies – or even whole industries.

The writing of this thesis started as an integral part of a research project called Research and development environment for distributed automation systems (HAJA, or Hajautettujen automaatiojärjestelmien tutkimus- ja kehitysympäristö in Finnish) which is a joint research project between Automation Software Engineering (ASE) research group of Tampere University of Technology (TUT) and laboratory of Information and Computer Systems in Automation of Helsinki University of Technology (TKK). This project has a goal to develop an R&D environment for distributed automation systems, as per the project name. The environment is intended to be used to research new automation software related technologies, and to teach the use of new design techniques for people connected to automation, including students, researchers and industrial experts. Although the word *environment* in name of the research project and in this thesis is used in singular form, it might be more suitable to think about the implementation as a set of environments, in plural, as the environment is implemented by several subprojects and assignments that do not necessarily form a cohesive entity as suggested by the definition of the term.

The goal for this thesis was to contribute to the creation of a university prototype for the R&D environment, starting by choosing appropriate technologies and supporting software, investigating them and then adapting them to university teaching in the form of assignments and studying material. The prototype has a key role in fulfilling one of the goals set for the research project: making sure that the level of students is up to the best international standards in the area of automation software engineering.

The choice of technologies for the research environment was partly dictated by the needs of teaching in the current courses managed by the ASE group and partly by the focal points of the group's research efforts, so the final decision about technologies to be researched was easy. The real research problems formed out to be the evaluation of

the technologies and how to implement the environment, i.e. the chosen automation software technologies, in the university teaching.

An important and integral part of this environment is the AP Tool software (Vepsäläinen 2008), which is a modelling tool that uses an Automation Profile UML extension developed for automation software and automation engineering related modelling. Both the Automation Profile (Ritala & Kuikka 2007) and AP Tool have been developed by the Automation Software Engineering research group of TUT to improve the efficiency of automation development process. Former was developed in a research project known as USVA (The impact of new technologies and standards on automation design), which studied new technologies and their advantages and disadvantages in automation development. Latter has been developed in an ongoing research project called AUKOTON (Uninterrupted development path for automation applications - From P&I diagram to control software) which aims to unify and improve automation design process with uniform concepts and technologies like UML and model driven-driven development. AP Tool has written instructions and software-based guidance implemented in the application (Rauhamäki 2009b), but these instructions are intended primarily for automation engineers with industrial backgrounds rather than university students who usually have only limited knowledge of the actual development process of automation systems. Integration of the tool to university teaching requires further work, part of which is done in this thesis.

Another technology that will be examined in this study is the OSGi Service Platform, an open specification for a Java-based dynamic component model, maintained by OSGi Alliance[1]. A core part of the OSGi Platform specification is the OSGi Framework. The Framework defines a unit of modularisation known as a bundle, life-cycle management for bundles and a service-oriented architecture (SOA) in a single virtual machine. The Platform provides tools to dynamically change the composition of the OSGi-based software without restarts.

Finally I will cover .NET Micro Framework; a small .NET runtime that is targeted at resource constrained embedded devices. Micro Framework allows the development of embedded software with modern development tools and C# language, allowing the engineers to focus on more abstract design concepts instead of writing, for example, low level Assembly or C code. So far the embedded devices have not benefited much from the advances in software engineering. Use of advanced programming languages and integrated development environments (IDEs) can cut both development time and costs by simplifying or even automating routine tasks, raising abstraction level and reducing number of errors in software code.

The chosen technologies target a fairly wide range of different kinds of automation software engineering, from small embedded devices to programmable logic controller (PLC) and distributed control system (DCS) based automation systems used in industrial plants. As a connecting theme, however, all raise abstraction level in the

[1] Formerly known as Open Services Gateway initiative but this name is now obsolete.

automation development and at least AP Tool and OSGi promote component reusability – the .NET Micro Framework represents the somewhat more traditional object-orientation paradigm as it enables embedded software to be written with an object oriented programming language. They all are also cutting-edge software technologies that have potential to bring significant gains to the productivity of automation software engineering.

Focus of this study is in the automation software technologies, not in the learning process. However, since the results are intended to be used in university teaching, the thesis will also discuss briefly some basic background for learning and how the environment should be implemented in order to support the learning process.

The research process for this work began by first studying the technologies. This included reading source material and doing hands-on work and empirical experiments with the technologies. After becoming sufficiently familiar with them, assignments and exercise instructions were written for courses ACI-32020 Real Time Systems in Automation and ACI-32040 Software Components and Services in Automation. Model solutions for the assignments were also developed. Finally the supporting lecture material for AP Tool – to be used in the course about real time systems – and OSGi – for the component course – were written. Since the automation software courses are held in Finnish, the resulting material of this work is also mostly in Finnish.

The contents of this thesis are arranged as follows:

Chapter 2 presents the technical background information that is required to understand the rest of the work. This information includes an understanding of automation software design and its present state, component-based software engineering and UML, a standardized modelling language.

Chapter 3 covers learning theories and how they should be applied to assignment and exercise development to improve the pedagogical quality of the work.

Chapter 4 covers the technologies studied for the research and development environment. The technologies include the Automation Profile, AP Tool, OSGi Framework and .NET Micro Framework.

Chapter 5 presents the results i.e. produced study materials and assignments. Discussion about the results, including the assessment of their significance, can be found in the *chapter 6*.

The thesis is concluded by the *chapter 7* which also gives some recommendations.

2. TECHNOLOGICAL STARTING POINTS

This chapter presents the technical starting points for this thesis. These concepts are the basic building blocks that can be considered to be necessary knowledge to understand the rest of the chapters. The matters discussed in this chapter represent a varied collection of ideas, ranging from automation design to software engineering. One of the challenges encountered in writing of this thesis has definitely been creation of a cohesive entity out of the individual concepts and technologies.

The chapter starts by introducing automation software engineering and its general practices. Next is a description of components and their usage in software engineering which is an integral aspect of automation software education at TUT. After the component section is the last subchapter which explains basics of UML, a modelling language that is not only an important design tool for software engineering but also a central element for the Automation Profile and AP Tool.

2.1. Automation Software Engineering

Automation is an interdisciplinary branch of technology, covering a wide range of topics from information technology to hydraulics. As a result, automation software can be found in almost all kinds of places and devices like rock crushers or elevators. Automation software engineering refers to designing software-based automation or control systems for processes or industrial machines; target platforms can be e.g. PLCs, microcontrollers or PCs.

Automation software is often, but not always, part of an automation system. Automation system could be a single PLC or complex network of devices used to control an entity like a plant or an industrial process. In process automation the systems often form complex, hierarchical structures made of computers, controllers, I/O devices and other hardware that are connected via fieldbuses or other networks. Each hierarchy level has its own functionality and responsibilities. The lowest level is made of field devices like sensors, actuators, whereas the highest level is usually the control room applications which may be connected to other information networks. An example of a modern DCS can be seen in figure 2.1.

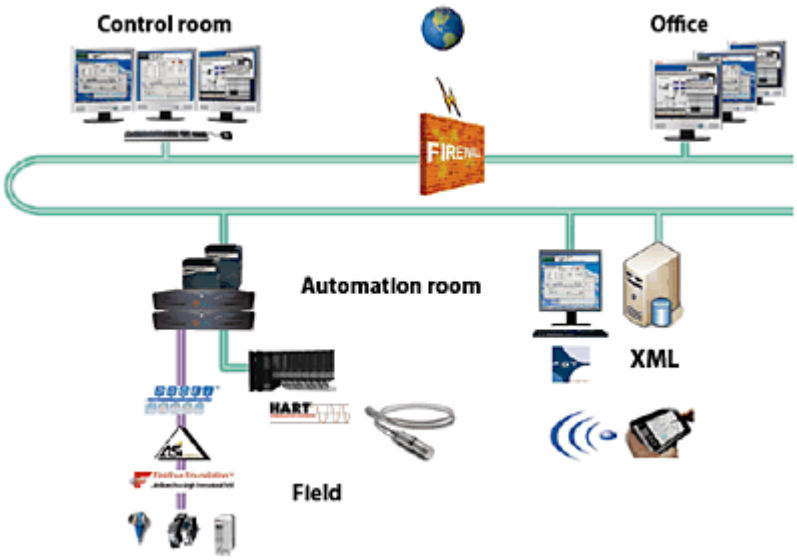


Figure 2.1. *An example of a modern DCS network (Metso 2009).*

As a result of being used in automation systems that manage operations of industrial processes and machines, automation software often has certain characteristics. Firstly, automation systems typically need to react to stimuli coming from sensors and control actuators based on these inputs. The reactions need to happen inside definite time limits, which means that they are *real-time systems*. In hard real-time systems the performance requirements are absolute. A failure to meet the deadlines could lead to a malfunction or a break down in the system. In soft real-time systems the deadlines can be occasionally missed. (Douglass 2004, p. 3.)

Secondly the execution platform for automation software is frequently an *embedded system*. Embedded systems are computers designed to perform certain tasks dedicatedly, contrary to general-purpose PCs. They are stripped of all excess peripheral devices and run on minimal hardware, for example small microcontroller boards. The minimal hardware sets challenges for programming as the available resources and computing capacity are limited and the program code is often written in low-level languages. (Douglass 2004, p. 2.)

And thirdly, automation systems are often *distributed systems*, meaning that the devices and computers forming the system are decentralized and communicate through networks. The structure of the system may change over time and it has to tolerate failures in individual devices.

Software development for modern PLC and DCS based automation systems differs from traditional software engineering also in the way that they are programmed. Most of automation systems are not programmed with general-purpose languages like C++; instead the programming is done with graphical function block diagram (FBD) and sequential function chart (SFC) languages usually influenced by IEC-61131-3, IEC-61499 or other automation standards. OSGi and .NET Micro Framework technologies described in this study are more similar to the general-purpose software engineering technologies and cannot be used in PLC programming – whereas AP Tool can be used

to support a wider range of automation software development applications, including PLC and DCS programming.

According to Rodriguez (2003), there is "a vast agreement about the characteristics of good control software". These features are largely similar to those of general software engineering – good control software is structured, modular, maintainable, well documented and written with uniform style, etc. Automation software, however is often used with safety-critical systems and thus needs to be especially robust and safe. Achieving all these goals is not effortless, and it could be easy to compromise on critical qualities in the pressure of tight schedules. Writing of high-quality software presumes proficient practices and principles and a systematic approach – this is indeed a part of the IEEE's definition for software engineering (IEEE 1990, p. 67).

Automation software is not a separate entity; it is usually one part of a larger automation system which furthermore is a part of some production process or machine. The production of the software needs to take into consideration the life-cycle and production process of the whole system. A generic example of a process plant automation system development life-cycle is presented in (SAS 2001): After preliminary analysis the development process starts with *specification phase* that produces customer requirements, functional specification and contracts. *System design phase* aims to produce software and hardware design specifications, including software module design specification. *Implementation phase* produces the necessary hardware and software needed to run the process, and includes factory acceptance testing and delivery release. *Installation phase* includes installation and testing of hardware. *Operative testing* makes sure that the system is ready for start-up with cold commissioning, followed by hot commissioning with actual process chemicals and materials. If everything is in order, the system can be released and the responsibility of the system is transferred to the customer. The customer *validates* the system by validation process and performance testing, after which it is ready for *production phase*.

The aforementioned life-cycle process is an ideal case, similar to waterfall process in general software engineering, but should nevertheless give a good overall picture of what is included in the development of an automation system. Notable characteristic is the amount of testing that must be done before the system is ready for production, compared to many software life-cycle models.

A Finnish guidebook for writing automation software (SAS 2005), targeted mainly for PLC-based systems, presents a more advanced life-cycle model for automation software called spiral. Spiral model is also commonly used in traditional software engineering. Other well-known software life-cycle models include the aforementioned waterfall model, rest of the incremental models like EVO (evolutionary delivery) that build the software by going through several iterations that keep adding features, prototyping, and agile software development which shortens the iterations even more, up to a level where the iteration becomes an ongoing process. (Haikala & Märijärvi 2006, pp. 35-47.)

Good automation software should furthermore be based on known standards (Rodriguez 2003). PLC programming for example has the essential IEC 61131-3 standard that defines four programming languages and a newer standard IEC 61499 for distributed computing. These standards, however, do not comment *how* the software should be written or developed. For quality control there are standards that can be used to ensure that the software is produced with appropriate processes and meets quality requirements, for example ISO 9126. Other fairly common standards define information and communication related concepts, e.g. ANSI/ISA-95 for integrating control and enterprise level systems and ISA-88 for batch process control.

The new automation software development and integration technologies presented in this study can offer solutions to some of the problems of automation software engineering. The Automation Profile and AUKOTON tool chain can unify concepts and engineering process and help maintain traceability of requirements. .NET Micro Framework can speed up the development process and raise the abstraction level for embedded device programming, enabling use of more sophisticated design patters and methods. Module system provided by the OSGi Framework can improve software maintainability, portability, modularity and reusability. But still, these are only some of the new technologies and methods developed by modern-day software engineering, and probably even more software innovations could be used to improve productivity of automation software engineering.

2.2. Component-Based Software Engineering

The central idea of the Component-Based Software Engineering (CBSE) is to enable the building of software from reusable components. This has two main benefits, namely extensibility and maintainability; functionality of software can be easily extended by adding new components or it can be fixed by changing outdated components to new ones. Statement from the whitepaper of OSGi Framework, an example of a component technology, formulates the solution in the following way: "Software component architectures address an increasing problem in software development: The large number of configurations that need to be developed and maintained. The standardized OSGi component architecture simplifies this configuration process significantly." (OSGi Alliance 2007b.)

CBSE has been one of the prime factors for development of Service Oriented Architecture (SOA) which extends the component concept to loosely coupled[2] and distributed services. These approaches share many key ideas but also differ in their use purposes (Kuikka 2009, p. 3). OSGi is closer to CBSE technologies but it has features of SOA – this viewpoint is discussed in chapter 4.3.2.

[2] Coupling is the degree of dependency that a software module has to other modules. Loose coupling can be achieved e.g. with the use of interfaces so that the dependence is to the service interface instead of the module that implements the service. This adds a degree of independency as the implementation can be more easily changed.

A component is a unit of modularisation and outside of CBSE the term is used with this generic meaning. In CBSE components are defined more specifically: they are reusable and independent software packages that encapsulate executable code. Components are used through interfaces that act as contracts that define how the component should be used and what services it expects to be available. Component-based software development happens by choosing and building suitable components and assembling them together. (Kuikka 2009.)

In CBSE components conform to a component model that defines the structure of the components and how they are used. Common component models include JavaServer Faces (JSF) and Enterprise Java Beans (EJB) from Sun Microsystems, COM and .NET from Microsoft and CORBA standard of Object Management Group (OMG).

A unit of modularisation in OSGi is a module but what is the difference between a software component in CBSE and an OSGi module? In the context of this thesis, component is a finished product that can be adapted to specific needs by parameterization; it also suggests a larger entity than module. By some definitions a component contains executable code whereas a module might contain only pictures or some other type of data. Otherwise they are generally the same; both are reusable, self-contained and employed through interfaces. Although the terms are often used interchangeably, a component could be considered to be more specific definition than a module, especially when used with CBSE. Finally it should be noted that component is also used as a term for other purposes, like configuration management where it can be used to imply a configuration item, i.e. a program file or a document in addition to a standalone software component (Haikala & Märijärvi 2006, pp. 257; 329-330).

2.3. UML 2

Much of the software engineering work consists of making specifications (Haikala & Märijärvi 2006, pp. 62-63). The requirement specification phase of the software development process aims to produce unambiguous requirement specifications that are used as inputs for the next steps in the development life-cycle to describe *what* needs to be done. The design phase attempts to solve the problem *how* the requirement specifications could be implemented by making design decisions, which includes considering and choosing appropriate software architecture and design patterns. These design decision are documented in the technical specifications.

The specifications include descriptions of behavioural and non-behavioural requirements, constraints and limitations that can be depicted with written text that is usually supplemented with tables, charts and such. In addition to these documentation techniques, the specifications often contain abstract models with graphical notations. There are a number of different kinds of notations that can be used for this task but nowadays the Unified Modeling Language (UML) is perhaps the most well-known and widespread. UML is a standardized graphical modelling language that can be used to create a variety of software-related models. The specification of UML is developed by a

consortium known as OMG; the first version of UML was accepted by OMG in 1997. Version 2.0, which extended the language notably, was adopted in 2005 (OMG 2009). The new version also features improved support for component development and CBSE (Kuikka 2009, p. 5).

UML has 13 types of diagrams that can be divided into two groups. Structural diagrams model static structures using objects, attributes and relationships. A common example is the class diagram. Behavioural diagrams describe system dynamics and interaction. Examples of behavioural diagrams include use case diagrams, state machines and sequence diagrams.

OMG has specified the UML 2 using a metamodelling approach, i.e. they have developed a metamodelling architecture called Meta-Object Facility (MOF) and used it to define the UML metamodel. This approach is shown in figure 2.2. Meta-metamodelling language is on the highest level (M3) and can be used to define new metamodels of modelling languages on level M2. Software designers can use the concepts of the level M2 language to create new level M1 models of the actual software that is being developed and finally the run-time instances of the objects are on level M0. UML is designed to be easily expandable by profiles that extend UML metamodel with domain-specific modelling elements. Profiles can take advantage of and use other existing profiles and their elements. More detailed explanations on UML extension mechanisms can be found e.g. in (Vepsäläinen 2008).

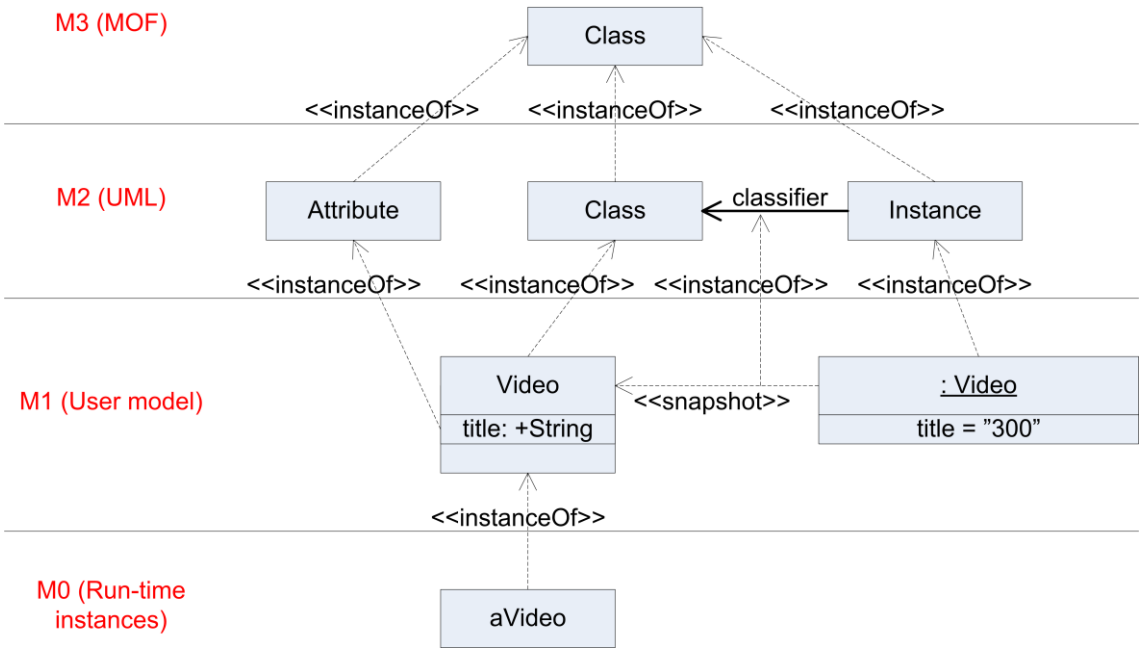


Figure 2.2. *The four-layer metamodel hierarchy (OMG 2009, p. 19).*

UML is often criticized of language bloat, and the resulting difficulty to learn UML properly can cause misuse of symbols and syntax. However, basics of UML are fairly easy to learn and there is no need to learn every aspect and diagram type of the

language. Diagrams can be used for fast sketching or explaining ideas in which case the use of proper symbols is not a top priority as it is more important to convey the main points of an idea. If the diagrams are part of a specification document, they should be accompanied with a descriptive text which should clear any possible misunderstandings that may rise from the diagrams.

UML 2 and its improvements, including better extension mechanisms, have opened new possibilities to the use of UML in automation and control domain. Antila (2008) presents possible ways to use standard UML 2 techniques in batch processes and machine control and concludes that UML could be used to improve intelligibility of complex systems and improve communication between project parties. However, this approach is not without impediments. Automation engineers' lack of knowledge in standard software engineering methods and techniques like UML can be seen as a hindrance for the use of UML in automation domain and this problem is further highlighted by the large size of UML 2. But considering how UML has fairly fast after its initial publication become de facto standard modelling language of software industry (Bissell 2003), these problems should not be insurmountable, especially if UML is extended with profiles that have concepts that automation engineers are already familiar with. Most of software engineers probably do not have complete grasp of the language and its nuances but this has not stopped UML from gaining popularity in the traditional software domain.

3. LEARNING

Rationale for including basics of learning theories in a technical text is that only describing different technologies and how they work does not guarantee learning and is even less likely to give the knowledge of how the technologies should be applied in practice. The technical side of education – use of multimedia and interactivity – is usually well under control in technological universities, as the lecturers are familiar with innovative technologies, but educational skills have more variance as there are no obligatory requirements for pedagogical formal training. The quality of learning process could be better if a pedagogic approach and educational psychological principles were applied already in the design phase. (Manninen et al. 2000, pp. 49-50.)

3.1. Learning Theories

As for terminology, in English language *pedagogy* is used for "study of teaching methods, including the aims of education and the ways in which such goals may be achieved" (Encyclopædia Britannica 2009). In Finnish and German languages the word *didactics* is used to describe research on teaching and learning (Uusikylä & Atjonen 2002), but it is rarely used in English.

Learning can happen in a number of different ways. Ways to classify types of learning include: formal learning, i.e. education vs. informal or learning by play; understanding vs. rote learning that focuses on memorization; e-learning with the help of computers and Internet; learning-by-doing, also known as hands-on learning and so on. In psychology and education, learning is defined as a *process* that brings together various influences and experiences to acquire new knowledge, skills, values, understanding, etc. (Maples & Webster 1980). Learning theories can be categorized in different ways but most often used frameworks are behaviourism, cognitivism and constructivism (Manninen et al. 2000). (Smith 1999.)

Behaviourism focuses on objectively observable aspects of learning and assumes that learning is manifested by a change of behaviour. The behaviour is shaped by the environment and conditioning, of which the most famous example is the Pavlov's Dogs. (Smith 1999.)

Cognitivist orientation offers a bit more modern viewpoint, transforming the focus from environment to the individual. Cognitive theories emphasize the internal mental process; how memory works and the importance of prior knowledge. In practice, this translates to developing learning skills and capacity, i.e. learning how to learn. (Smith 1999.)

Next orientation repeatedly mentioned in the literature is constructivism, the so-called fashion theory of the 90s. Constructivism is actually an umbrella for different smaller trends but the common theme is the view of learning as a process in which the learner constructs new ideas from prior experiences. Constructivism has seen its share of criticism but its influence on modern pedagogy is quite indisputable. In practice this has appeared as stressing the importance of understanding as a goal of science instruction and by fostering student engagement, for example by choosing interesting topics for assignments. (Matthews 2000, pp. 161-192.)

Other popular learning theories place emphasis on a number of areas; most common ones include humanist, social and individual orientations. Brief introductions can be found e.g. in (Manninen et al. 2000, pp. 24-28) or (Smith 1999). The key viewpoints that these theories bring out are the importance of motivation for learning, and how a positive learning culture can enhance learning.

3.2. Educational Applications

Aforementioned theories are the underlying foundations that many of the more practical education strategies are based on either directly or by implication. An example of one such strategy is Problem-Based Learning (PBL). PBL is an approach to learning where students work in small groups and attempt to solve real-life based problems under guidance of an instructor. It can be described as "a systematic attempt to apply findings of cognitive psychology to educational practice" (Dolmans & Schmidt 1996) and was pioneered by medical schools in US and Europe in 1970s (Spencer & Jordan 1999).

The term problem based learning is used with varied meanings, for example, it can be used to describe both an educational method and a curricular philosophy. Generally it is understood to mean an instructional strategy in which students identify issues raised by specific problems to help develop understanding about the underlying concepts and principles (Spencer & Jordan 1999), as described above. According to Spencer & Jordan, PBL is associated with several benefits like promoting deep learning and retention of knowledge, more stimulating learning environment, motivation improvements and promoting of interaction between students and staff.

Educational applications of learning theories can also be used to analyze learning, and one way to do so is by defining levels of learning. The levels can be interpreted as a classification of learning objectives that educators can set for students. One such arrangement commonly used is Bloom's taxonomy (Bloom & Krathwohl 1956), first presented in 1956. The taxonomy is divided into three domains – affective, psychomotor and cognitive – to create a holistic approach to learning, but the cognitive domain is the one that is most commonly referred to. Nowadays the cognitive domain is almost synonymous with Bloom's taxonomy. The levels originally presented in the cognitive domain are knowledge, comprehension, application, analysis, synthesis and evaluation. During the 1990s the taxonomy was updated to reflect the relevant contemporary work by a group of cognitive psychologists, lead by Lorin Anderson, a

former student of Bloom's (Anderson & Krathwohl 2001). The new taxonomy changes nouns to verbs and reflects a more active form of thinking, as seen in figure 3.1. The new taxonomy has been designed to be useful for a broader audience than the original one.

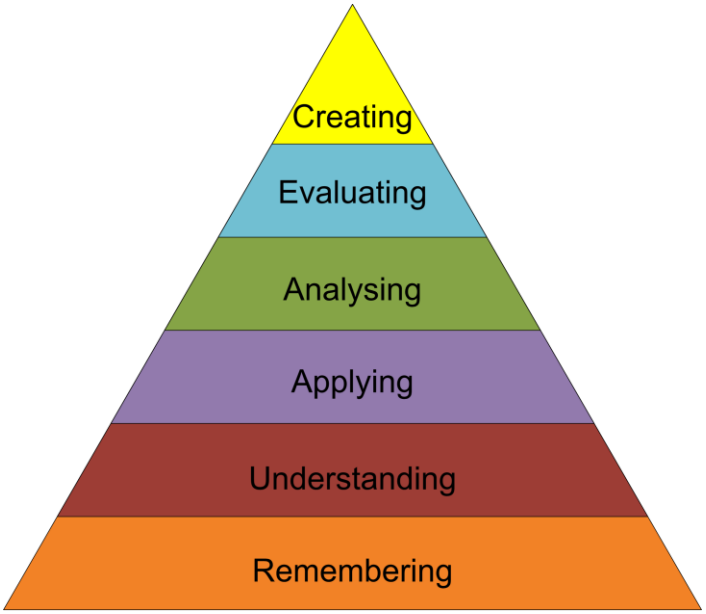


Figure 3.1. *New version of the cognitive domain of Bloom's taxonomy (Anderson & Krathwohl 2001).*

The new terms are defined as following:

- "**Remembering**: Retrieving, recognizing, and recalling relevant knowledge from long-term memory.
- **Understanding**: Constructing meaning from oral, written, and graphic messages through interpreting, exemplifying, classifying, summarizing, inferring, comparing, and explaining.
- **Applying**: Carrying out or using a procedure through executing, or implementing.
- **Analyzing**: Breaking material into constituent parts, determining how the parts relate to one another and to an overall structure or purpose through differentiating, organizing, and attributing.
- **Evaluating**: Making judgments based on criteria and standards through checking and critiquing.
- **Creating**: Putting elements together to form a coherent or -improvement of teaching methods and material based on feedback." (Anderson & Krathwohl 2001, pp. 67-68)

3.3. Use of Modelling in Support for Learning

Use of interactivity and multimedia to enhance learning has been researched to some extent, for example Manninen et al. (2000) present in their report the results of using pedagogical and technological innovations in Finnish vocational education. In this context interactivity can be considered as the use of WWW and multimedia as augmentation of teaching with a variety of presentation technologies, like PowerPoint slides combining pictures and text.

Research on using software models to enhance learning, a topic that is more interesting from the viewpoint of this thesis, seems to be scarce. However, UML models, and especially class diagrams, are essentially formalized concept maps, abstract representations of some idea or system. Use of concept maps is a more broadly researched topic, so we can gain some insight to the usefulness of modelling by examining this subject.

Concept maps are diagrams that depict relationships between concepts, consisting of concept nodes that are often enclosed with boxes or circles and connecting lines that may be supplemented with text that clarifies the relationship between nodes. Concept mapping has been used in a variety of fields, including science teaching and learning (Chang et al. 2001), health planning (Trochim et al, 2004), and as a research and evaluation tool (Markham et al. 1994). Furthermore, the use of concept maps is supported by constructivism. Constructivists hold that prior knowledge is used as a framework for acquiring new knowledge and concept maps can help us to identify and organize these knowledge structures. Once a concept map has been established, it can be expanded to accommodate new concepts, linking new ideas and knowledge to the old. Overall the usefulness of concept maps in enhancing learning seems to be an established fact, supported by several researches. E.g. Nesbit and Adesope (2006) did a large-scale meta-analysis on 55 concept maps studies, and concluded that the use of concept maps was associated with increased knowledge retention.

These results seem to indicate that the use of model diagram representations of software or automation system could, indeed help to give better understanding of the architecture and aid the planning process. Alternatively they could be used in education for similar purposes as conventional concept maps, e.g., to help students to conceptualize assignments, when done by the students themselves. Actually the usefulness of model diagrams in software engineering is not shocking news, as class diagrams and suchlike have a fairly long history of use in software industry.

Finally it should be noted that although concept maps and model diagrams are a useful abstraction tool, they may not suit to all learning styles equally well. Use of concept mapping generally produces an increase of knowledge, but usually there is also a group of people that register no change in their level of understanding (Irvine et al. 2005, p. 11). Diagrams suit especially well to visually oriented persons, whereas for non-visually-oriented people text-based descriptions may be easier to grasp. Model

diagrams should, indeed, always be accompanied by textual explanations, especially if they are part of software documentation.

Although model diagrams have a good power of expression, they have their own limitations. One aspect that should be taken into consideration when designing diagrams, and more specifically static structure diagrams, is the maintenance of readability. Human mind is able to receive considerable amounts of information as long as it is presented in a suitable format. One of the strengths of modelling is the ability to transfer large amounts of data in reasonably compact diagrams. Care should be taken to keep the diagrams understandable. This can be achieved by information hiding, e.g. by using sub-diagrams or composite design pattern[3].

Attention should also be paid when choosing the graphical notation for diagrams. For diagrams to be properly used for purposes of transferring knowledge, appropriate graphical elements (shapes) must be chosen to represent different features or abstract model elements. These elements can be complemented with text to augment limitations of graphical expression. One such occasion is when the expressive powers of diagrams is not sufficient to fully describe the situation, and the resulting diagram would be loose, i.e. it would allow implementations that do not match the desired system. These written restrictions can be either added straight to the diagram if the language permits it, or added to the related documentation. In UML this can be done with *constraints.* For example, if two relations are mutually exclusive, this can be expressed with the constraint {xor}. (Haikala & Märijärvi 2006, p. 129)

3.4. Exercise and Assignment Development

This subchapter tries to tie the new technologies together with the pedagogical viewpoints and apply them to produce suitable exercises and assignments. By focusing on the pedagogic aspects, the writer hopes that the resulting tasks will be well designed and facilitate improved learning. The included aspects for software assignment and study material development are: induced realisation or insight, learner-centric approach, repetition, deep processing and hands-on experience.

The design of the assignments and exercises should aim to **induce realisation in the learner**; by guiding the thoughts to the direction of the right answer the learner can get the positive experience of insight and is also more likely to remember the gained knowledge when compared to rote learning.

Designing the difficulty level to be sufficiently challenging while still being neither too easy nor too difficult requires careful **learner-centric approach**. Automation software engineering students, for example, typically do not have many years of programming experience. There might also be fairly experienced software engineering students among them doing the exercises but it is better to design the exercises with the less experienced programmers in mind. This is true especially in the case of the weekly

[3] Composite design pattern allows objects to be composed into tree structures to represent part-whole hierarchies (Gamma et al. 1995).

exercises that are intended to give brief hands-on experience of the subject and have limited time available for completion, and thus should have reasonably detailed instructions. Assignments, on the other hand, have much less constrained time limits and students will have more time to familiarize themselves with the subject if they have limited programming experience. Study material should not make too many presumptions about the knowledge level of students. If the material is written in a way that expects that the reader is an expert in software technology, the message of the text could be clouded by a multitude of difficult and new terms.

Repetition can be an effective tool for memorization, and is usually required for knowledge to be transferred to the long-term memory. Students are introduced to subjects when the information is first presented on lectures. Lectures themselves usually have some repetition – the subjects are first introduced, then explained and later recapped. The next step is doing the weekly exercises that repeat some of the things that have been discussed on lectures and let the students get hands-on experience on the subjects. After doing the relevant exercise, students can start working on the assignment which gives them a chance to try their hand with both general ideas learned on the course and with a specific technology that is used to implement the assignment. Finally the students will go through the material for the course exam, thus committing the facts to the long-term memory.

The goal of the education should be to encourage **deep processing** and understanding instead of rote learning and memorizing. According to Hartley (1998, p. 51), some conditions that can drive students to surface processing include heavy workload, an excessive amount of course material, a lack of opportunity to pursue subjects in depth or a lack of choice over subjects and methods of study. Ways to encourage deep processing include project work, learning by doing, using problem-based learning, group assignments, encouraging student reflection, allowing for independent learning, providing authentic tasks, rewarding understanding and penalising reproduction, and involving students in the choice of assessment methods. The assignments on the relevant courses have a fairly good selection of topics to choose from and they should cover most items on the list of ways to encourage deep processing. A possible way to enhance assignments could be to allow the students to affect the selection of evaluation method. Some other points are really out of the educator's reach, like the work-load of the students. Lecturers can influence the amount of work that is needed to pass a course but the workload should be reflected by the amount of credits the course is worth of, and thus students should be able to choose a number of courses they can handle based on the total credits of all the courses they have selected.

Giving a chance for the students to put theory into practice and gaining **hands-on experience** is valuable for learning, as noted by Hartley. Especially since the software topics tend to be on a very abstract level, hands-on training can really help to get the big picture. The weekly exercises aim to give basic mechanical experience and assignments let the students to apply their gathered knowledge in practice.

4. RESEARCHED TECHNOLOGIES

The technologies that are presented in this chapter were studied for the purposes of producing learning material and exercises for university students. Thus they are given a general overview rather than a thorough and detailed analysis. The reviewed technologies are the Automation Profile and the accompanying AP Tool, OSGi Framework and .NET Micro Framework.

4.1. Automation Profile

Development of the Automation Profile (AP) was started as a part of the USVA-project, which aimed to evaluate new automation technologies and their usefulness. The Profile tries to unify the terms and concepts in the automation industry and introduces them as a UML extension that allows automation designers to design and develop the automation systems and their requirements by creating UML model diagrams. Use of a standardized modelling language in the design phase of the automation software lets the engineers to focus on the more abstract design and architectural decisions, and makes it easier to reuse existing solutions to design problems, thus avoiding the need to solve same problems repeatedly in new projects.

Automation industry, however, does not have any prevailing standards for automation-related concepts or modelling, as discussed in chapter 2.1. UML itself is too software-oriented to be suitable for general automation design, but it does have an excellent extension mechanism and several existing profiles that fit automation-related sub-tasks. (Ritala 2006.)

4.1.1. Base Profiles

Creating a UML profile that would cover the whole automation field from the scratch would not only be an intimidating task but also redundant work. The Automation Profile is thus based on several existing domain-specific UML profiles that have been standardized by OMG. The AP builds on the key concepts of these profiles and extends them to suit the needs of the automation industry. The base profiles include the Systems Modeling Language (SysML), Real-Time Profile (RT Profile) and UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms (QoSFT).

SysML is a UML-based modelling language that has been developed for systems engineering. As UML is quite software-centric, SysML extends it to be more flexible and expressive. It supports the specification, analysis, design, verification and validation

of systems and systems-of-systems, including hardware, processes, personnel and information (OMG 2008b). It defines among other things several diagram types, most of which are based on existing UML diagram types. These include e.g. a requirement diagram that can be used to manage requirements on a more specific level than use case diagrams of UML. Other features of SysML that have been used in the AP, in addition to requirement definitions, include *Blocks* that can be considered to be a more general-purpose version of classes. Blocks are basic building elements that can be specialized with UML stereotypes. They can be flexibly connected by Ports and Flows, another part of SysML that has been used in the Profile.

UML Profile for Schedulability, Performance and Time, more commonly known as the *Real-Time Profile*, extends UML's somewhat lacking timing model to better suit the needs of the time- and resource-critical systems modelling (OMG 2005). Time-, schedule- and performance-related aspects of the real-time systems can be modelled with basic UML, but the RT profile provides extended ways to specify the timeliness of the systems and subsequently to perform quantitative analysis on them (Douglass 2004). The profile is quite large and has been divided into packages that suit different real-time modelling related tasks, namely General Resource Modelling Framework, Analysis Models and Infrastructure Models. Most relevant of these from the perspective of automation industry is the General Resource Modelling Framework that contains all the basic concepts of real-time systems.

Specification for the Quality of Service and Fault Tolerance Characteristics and Mechanisms UML profile extends UML's capabilities in the aforementioned applications. The profile itself is divided into two basic frameworks, QoS Modelling Framework and Fault Tolerance Framework. The QoS Framework makes it possible to associate requirements to the model elements in UML models and FT Framework includes notations to model risk assessments. (OMG 2008a.)

4.1.2. Design Principles and Architecture

The Automation Profile extends UML by defining the automation-related concepts that have been arranged into four subprofiles according to their subject. In addition to offering notions for the modelling, the AP also defines three diagram types that can be used in the modelling process. Use of the developed modelling concepts is not limited only to the new diagrams, as they can also be used with the standard UML diagrams. (Vepsäläinen 2008.) Development of the Profile continues in parallel with respective tool development in the on-going AUKOTON-project.

Although the Profile is intended to be usable in a wide range of different automation applications, its focus is in the software-based control and measurement applications, which could be described as the backbone of the automation industry. Included concepts range from abstract requirement-related terms to fairly low-level design elements. There is some support for physical devices that are needed e.g. when designing a distributed system. All model elements are non-platform specific, as one of the design principles

for the AP was to make it a viable tool for all automation-related modelling needs, regardless of the runtime environment.

According to Ritala, the following design principles were utilized in the development process:

- "Familiarity of concepts: the modelling elements of the profile present various commonly known concepts required in the development of software-oriented applications within the automation industry.
- Reuse and extensions: the primary mechanisms for implementing the modelling elements of the automation profile are reusing and extending modelling elements from the UML V2 and existing UML profiles.
- Modularity: the modelling elements of the profile have been divided into small distinct packages with a detailed responsibility. Thus, the user of the profile will be able to easily pick the elements he/she needs.
- Extensibility: the profile allows user-level extension of some concepts covered by the profile (such as control algorithms) in order to better suit application-specific needs." (Ritala 2006, p. 10)

As mentioned earlier in this chapter, the profile has been divided into four separate subprofiles, each of which has its clearly defined areas of responsibility. The Requirements subprofile can be used to specify functional and non-functional requirements. The second subprofile, Automation Concepts, includes notions that are needed when modelling automation-specific systems, such as control structures, process simulators and alarms. The Distribution and Concurrency subprofile can be used to model how the components of the system are distributed and how they communicate. It also includes some terms specific to concurrent systems like synchronization and mutexes[4]. The fourth subprofile, Devices and Resources, can be used to depict simplified physical devices, device interfaces and other resources in the models. The subprofiles are represented with more details in the next chapter.

4.1.3. AP Subprofiles

The Automation Profile consists of four subprofiles that cover the most common aspects of automation applications. The subprofiles have some dependencies between themselves and to the base profiles, so they are not entirely independent (Vepsäläinen 2008, p. 60), but they have been designed to be modular so that the developers can pick and use only those packages that they need.

[4] Synchronization is used to coordinate actions between processes. Mutexes or mutual exclusion algorithms are an example of synchronization used to prevent the simultaneous use of a resource from more than one thread at a time.

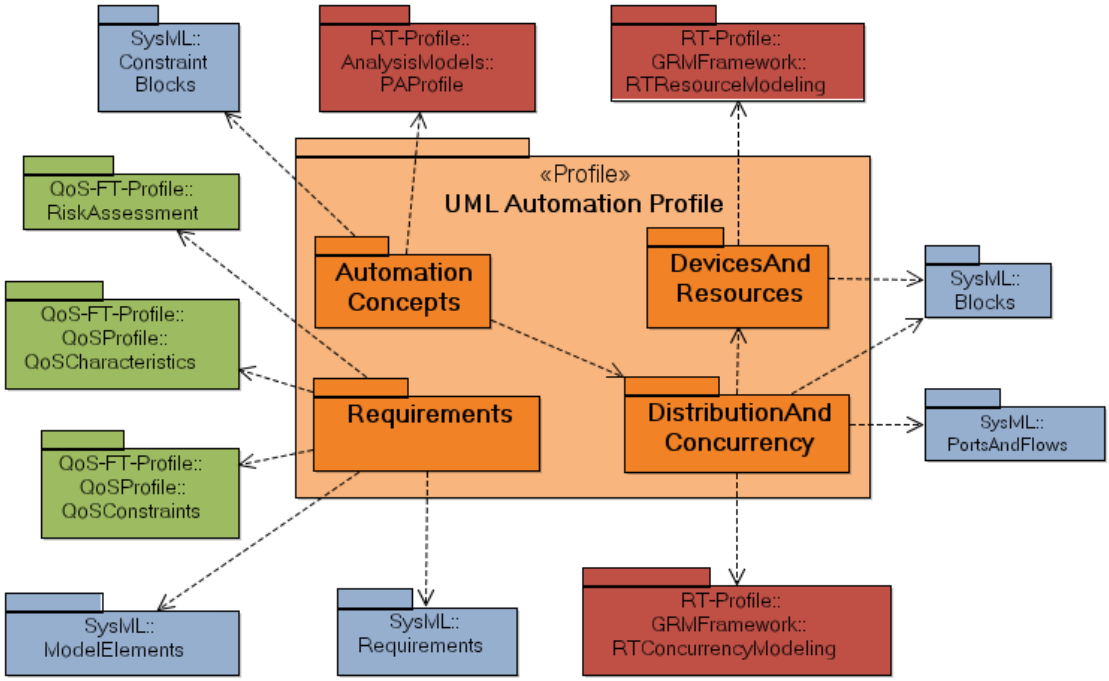


Figure 4.1. *The Automation Profile shown with subprofiles and dependencies to base profiles.*

Figure 4.1 shows the subprofiles and the interdependencies the Profile has in addition to the dependencies to the base profiles. The subprofiles will be discussed next on a package-level. A good source for more detailed information about the Automation Profile is (Rauhamäki 2009a). The descriptions of the packages in this chapter are based on the Profile specification (Ritala 2006) unless noted otherwise.

Requirements

Functional and non-functional requirements together with constraints and limitations are used in the specification process to produce specification documents that define the system and are used in the design and implementation phases. An accurate specification for the software forms the basis for building a well-functioning system. Flaws or inaccuracies in the specification on the other hand can mean that the software does not fulfil its intended purpose sufficiently well. Errors of this kind are usually not found until the system testing, which is the last testing phase. Errors found this late are substantially more expensive to fix when compared to finding the errors earlier in the specification phases.

Specifications created in one development phase are used as inputs for the next phase, therefore describing its requirements. Every phase gets its requirements from the previous one, describing what needs to be done in this new phase. The new phase will produce a specification document describing how the matters defined in its input documents should be done on this level, thus forming a chain from the abstract customer requirements to detailed software code. (Haikala & Märijärvi 2006, pp. 62-

63.) One point of this subprofile and especially the *AutomationRequirement* concept is to import and merge all requirements from phases prior to automation software specification to a single requirement model.

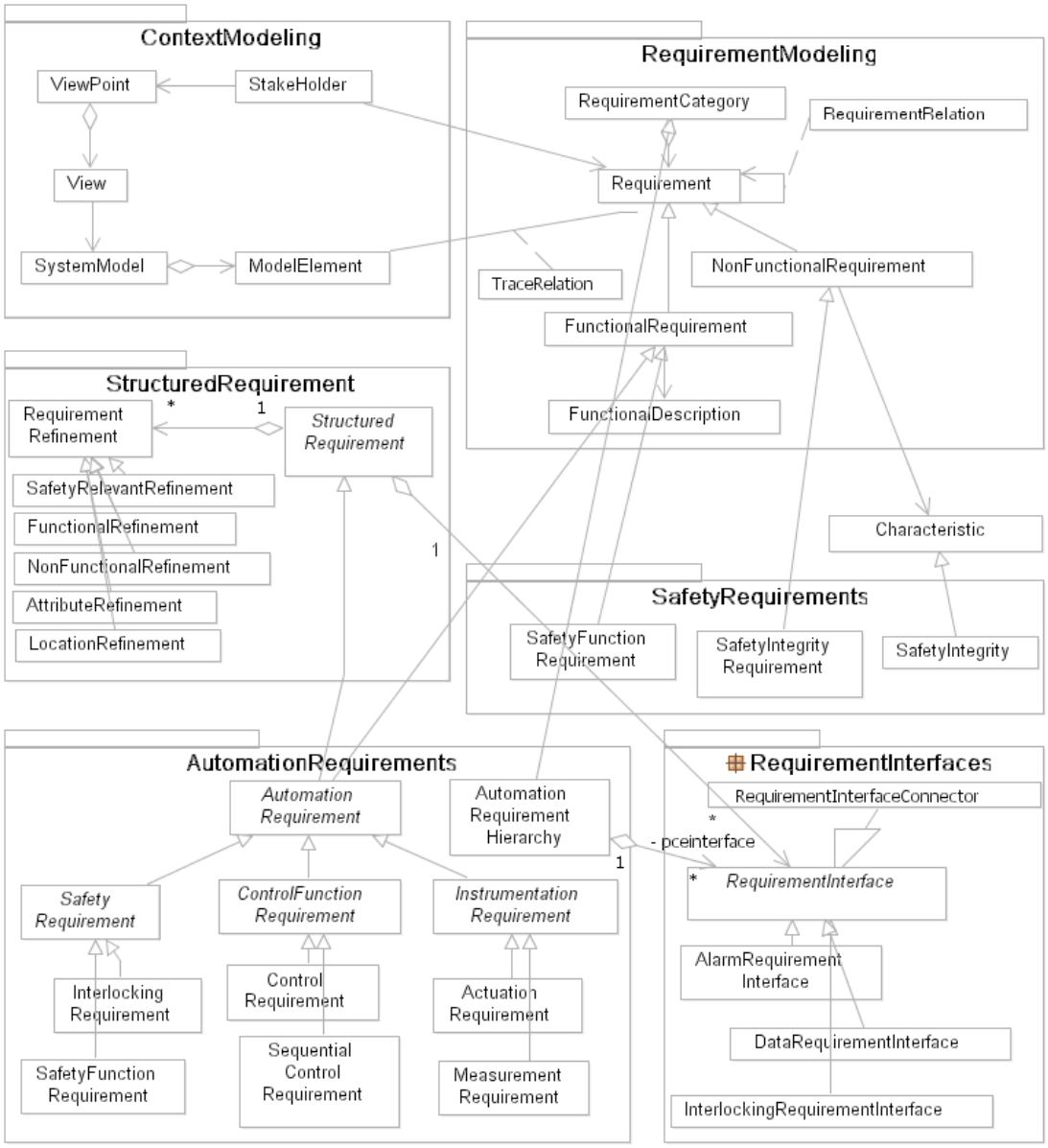


Figure 4.2. *Requirements subprofile.*

The traceability of the requirements is important for determining that the end product meets all the requirements. Traceability means that we can follow the path from the customer requirements to the specifications documents to check which features implement the requirements, and further still, which code modules realise any given requirement (forward traceability). Or vice versa, what requirements any given code module realises (backwards traceability). Unfortunately the maintenance of traceability

is hard work and often requires use of external programs. (Haikala & Märijärvi 2006, pp. 97-98.)

Customer requirement specification is usually done in the preliminary development phase. The Requirements subprofile offers support for the modelling of different types of requirements that are related to the development of automation software, in addition to standard UML features like use case diagrams. The profile does not remove the need for careful requirements specification but provides tools to present the requirements in a graphical form. Perhaps most importantly, at least from the requirements management perspective, it can be used to connect the requirements to the blocks and diagrams implementing them, ensuring that features are traceable backward and forward in the development phases.

Figure 4.2 on page 21 shows the packages and concepts of the Requirements subprofile. The *RequirementModeling* package can be used to model functional and non-functional requirements. A notable feature is the ability to create trace relations between requirements and modelling elements.

Safety of the industrial processes and automation equipment is a fundamental concern in automation design and negligence on this area could possibly cost human lives. *SafetyRequirements* package modifies the concepts of functional and non-functional requirements with safety-point-of-view, its concepts originating from the functional security standard IEC61508.

ContextModeling package enables the creation of customized views of models for associated persons according to their area of expertise, hiding irrelevant information.

AutomationRequirements provides a package of automation and process industry related requirement concepts that originate e.g. from measurements, actuators and controllers. The package is intended to be used in automatic processing of requirements as part of the uninterrupted development path that is being developed in the AUKOTON-project.

StructuredRequirements package provides concepts that can be used to attach additional information to requirements. Main candidates for use are the automation requirements, which are more strictly defined than most of the Profile's concepts. The additional information can include features and properties, and interfaces that are defined by the *RequirementInterfaces* package. (Hästbacka & Vepsäläinen 2009.)

Automation Concepts

The Automation Concepts subprofile contains more concrete modelling concepts than the Requirements subprofile. They are needed for the specification of the functionality and behaviour of the system, which is usually done in the basic design phase. These concepts include control sequences and loops that can be used to define control algorithms. (Ritala 2006.)

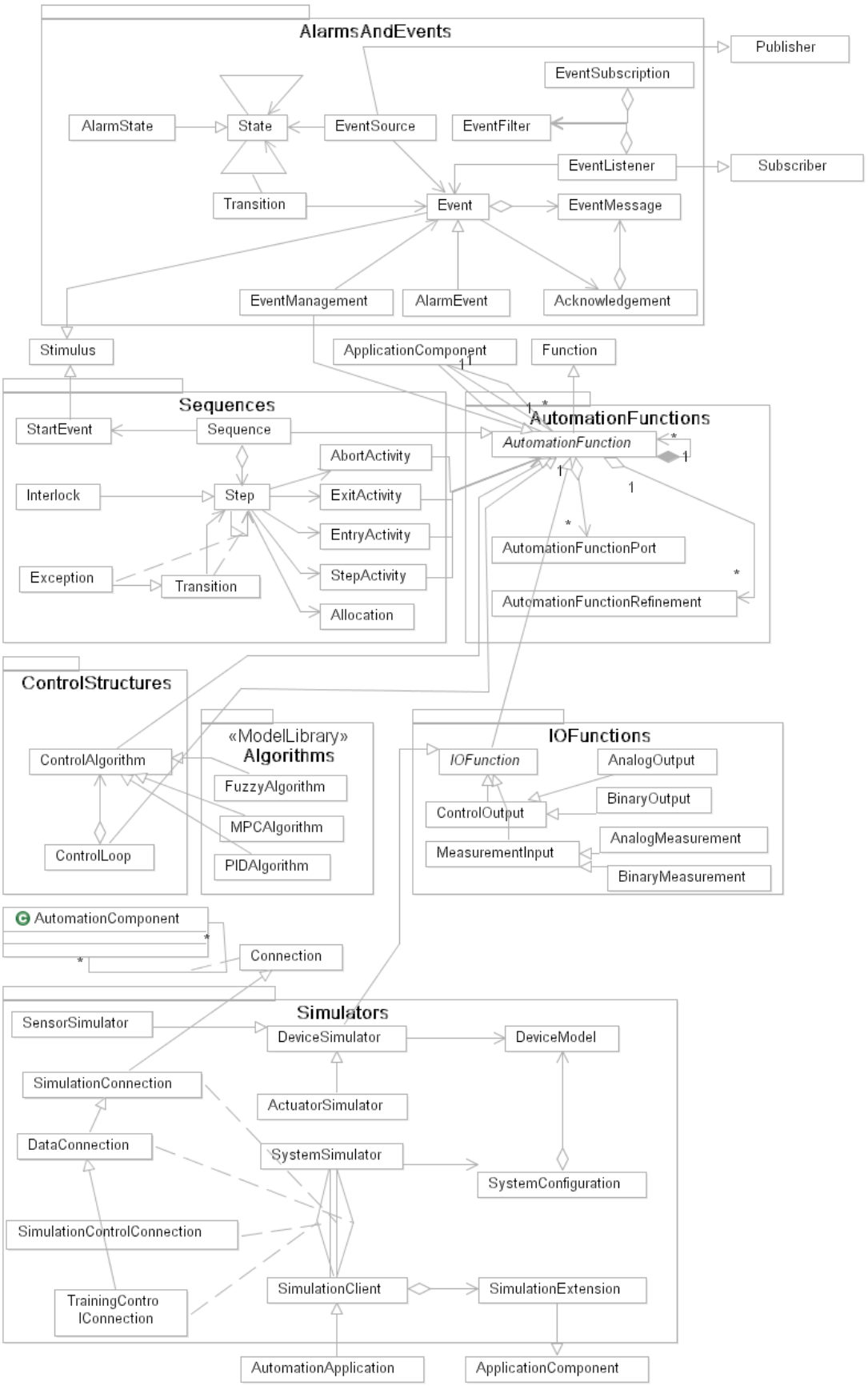


Figure 4.3. *AutomationConcepts subprofile.*

Figure 4.3 on page 23 shows the structure and connections of the AutomationConcepts subprofile. *AutomationFunctions* package acts as the basis for automation concepts, defining a general-use automation function, on top of which many concepts of other packages can build up by extending the base concept.

Control structures are often an integral part of almost any automation application. *ControlStructures* package together with *Algorithms*, containing model library implementations for most commonly used control algorithms, and *IOFunctions* package, i.e. inputs and outputs for control, form the basic building blocks for creating a myriad of different control solutions.

Sequences package features concepts for creating control sequences, also known as sequential function charts, as defined by the fairly well known IEC 61131-3 PLC programming language standard (IEC 2003). Control sequences are made of action steps and conditioned transitions between them.

Alarms and events are information produced by the automation system describing changes and occurrences in the state of the system, defined with the concepts of the *AlarmsAndEvents* package. They can be stored to various history logs or used to trigger safety functions or interlocks, as defined by the *SafetyFunctions* package, to prevent possible accidents or damage (Hästbacka & Vepsäläinen 2009).

The last package of the AutomationConcepts subprofile is the *Simulators*. This package can be used to create and connect to various kinds of simulators and models that are often used in automation industry to estimate how the system would function in certain situations or to test control parameters.

Distribution and Concurrency

Large-scale automation systems often have distributed devices that are connected by fieldbuses or other networks. These devices can include PCs used as high-level monitoring and control stations, workstations and servers; distributed control CPUs and PLCs operating near the actual process; and low-level smart sensors and operating devices. Thus the components of the automation software used to run the system tend to be distributed. This fact, combined to the needs that automation software have for timely and reliable communication, creates a call for well-defined specification for the distribution of the components. (Ritala & Kuikka 2007.)

Systems featuring all these elements obviously can include quite a lot of computing executed simultaneously, not to mention that even fairly simple programs can have multiple threads. Basic UML offers some means to handle situations where concurrency is an issue – generally this means activity and sequence diagrams. Given the importance of deterministic computing in the automation field and combining that with the increase of concurrent computing in general, the need for accurate modelling methods for concurrency is evident.

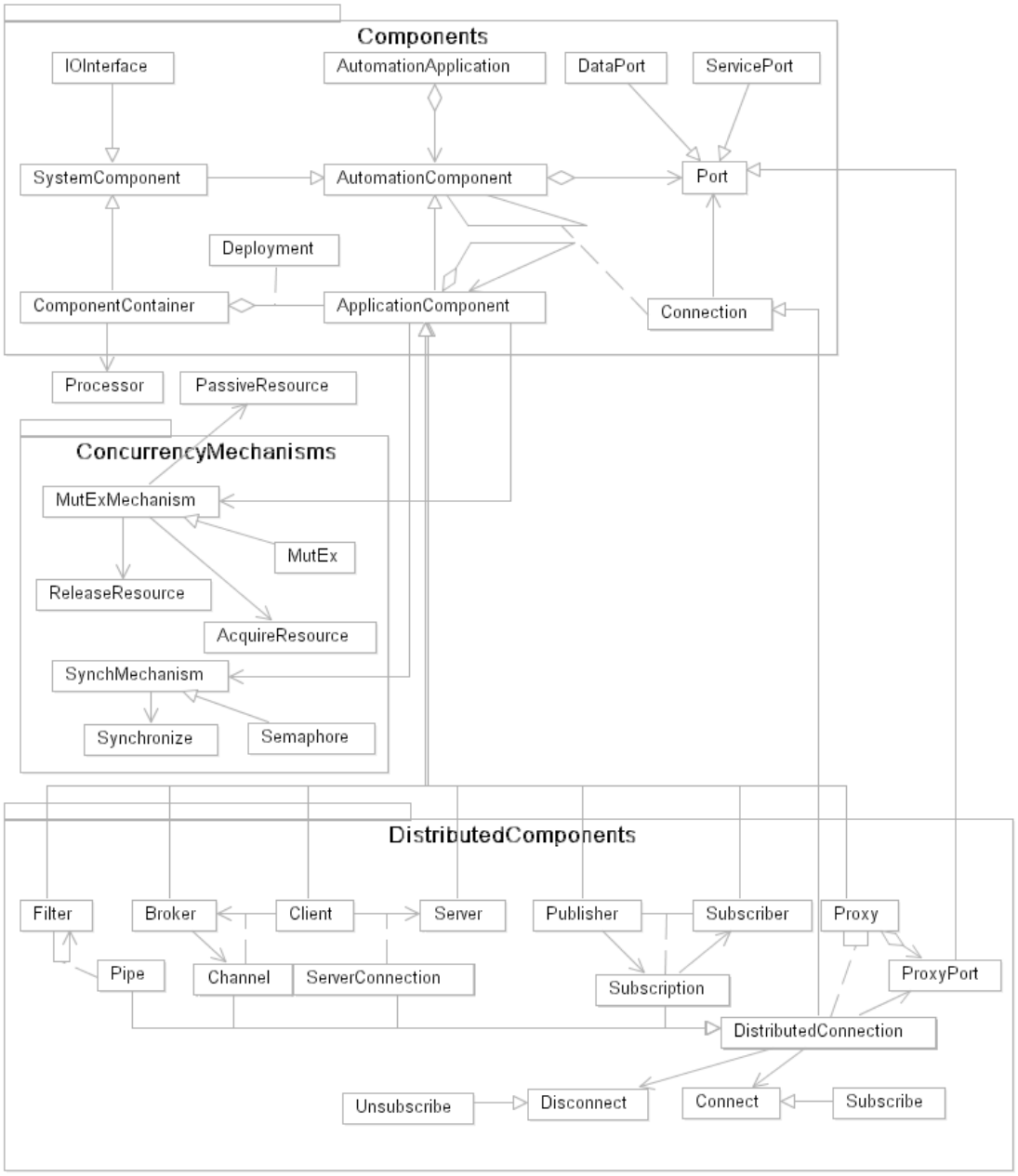


Figure 4.4. *DistributionAndConcurrency subprofile.*

Figure 4.4 presents the packages and concepts of the DistributionAndConcurrency subprofile. The *Components* package can be used to describe the structure of the component based automation applications and the way in which those components are distributed to the processors. Concepts of the *DistributedComponents* package correspond to the most commonly used communication policies and mechanisms of distributed computing, making it possible to hide the details of the distribution from the components. The *ConcurrencyMechanisms* package provides concepts that can be used to protect resources from simultaneous access or to synchronize concurrent units with each others.

Devices and Resources

Although the Automation Profile is designed for development and modelling of automation software, there is often a need to include some hardware concepts on one level or another to the design process, since the automation software tend to be device-oriented. The software can interact and communicate with different kinds of devices through a variety of interfaces that may range from diverse and simple device drivers to bulky interface standards.

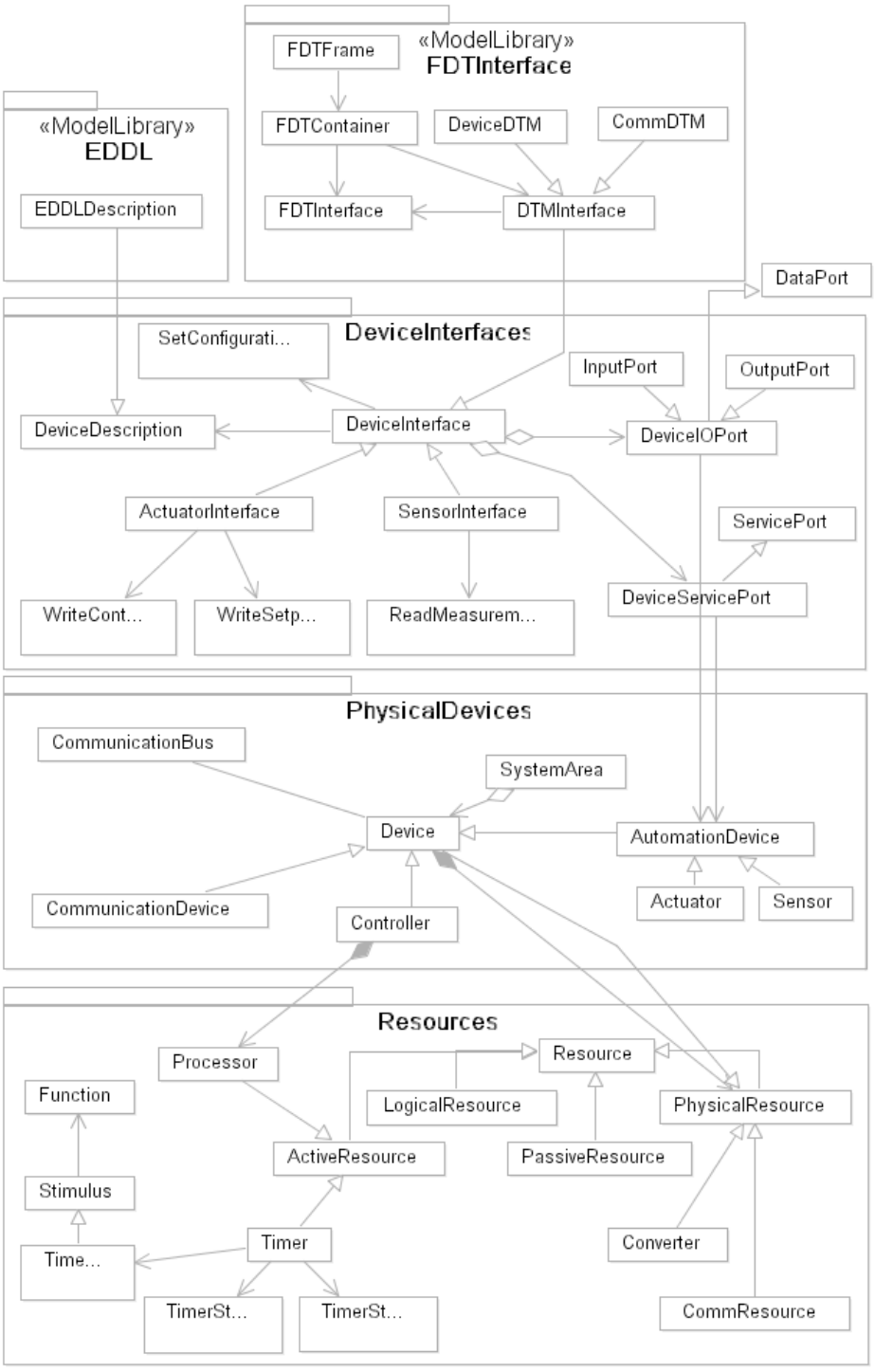


Figure 4.5. *DevicesAndResources subprofile.*

The Devices category of the subprofile also covers the automation related physical equipment that is used in the automation systems. This includes not only sensors and actuators, but also communication devices and controllers. In the case that the user does not find an appropriate device class from the subprofile, it has extension capabilities that allow users to extend the profile and add their own devices (Ritala 2006, p. 50). The devices may need to be taken into consideration in the architecture design when deciding how the software components are deployed to the processing units.

Resources are logical or physical assets that are used to fulfil specific needs. Resources may often have a limited availability, which means that their proper modelling is especially important in concurrent and distributed environments. A typical example of a resource in an automation system is a timer used to trigger events.

Figure 4.5 on page 26 shows the contents of the DevicesAndResources subprofile. In addition to the aforementioned logical/physical resources, the *Resources* package contains the concepts for active/passive resources.

The *PhysicalDevices* package enables the modeller to link the automation software to the devices of the system. The included terms are basic as it is not the Profile's intended use to create detailed hardware models.

Interfaces specified by the *DeviceInterfaces* package are parts of software or third-party applications that can be used to realize the connections between the automation software and the actual devices.

The Automation Profile can be extended with support for new standardized interfaces easily. The package *FDTInterface,* adding support for Field Device Tool (FDT) interface integration standard, is included as an example of one such extension so it is not very likely to be used in the university courses. Another package included as an example is the *EDDL* which provides support for the Electronic Device Description Language (EDDL), demonstrating the Profile's extensibility potential for standardized device description languages that can be used to specify the capabilities of automation devices.

4.1.4. Diagrams

In addition to being usable with any standard UML V2 or SysML diagram, the Automation Profile also defines three automation-specific diagram types that can be seen in figure 4.6. These diagram types extend the existing UML and SysML diagram types for automation domain purposes and accept only certain modelling elements. The Profile specification itself does not define these correct elements for the diagrams but the implementation of the profile in AP Tool imposes restrictions to the available elements in each diagram type.

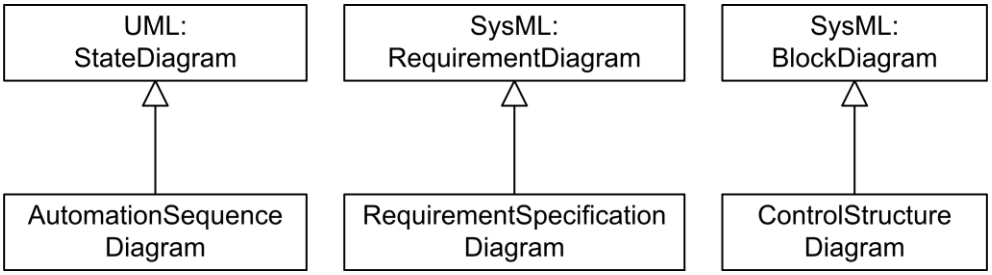


Figure 4.6. *Diagram types defined by the Automation Profile are based on SysML and UML diagrams (Ritala 2006).*

RequirementSpecificationDiagrams (RSD) are used to model the requirements in the customer requirements specification phase, one of the first things that the automation designer should do. RSDs are used with the concepts of the Requirements subprofile.

ControlStructureDiagrams (CSD) are used to mo del the structure and architecture of the automation software. It is likely to become the most familiar of the diagrams to designers using the Automation Profile as it can be used during the rest of the development process after requirements specification for different design tasks, like modelling of control loops. CSDs are very flexible as they support a large portion of the Profile's concepts, excluding only the Requirements subprofile. CSDs also support SysML's <<block>> stereotype as a basic building block that can be extended with new stereotypes to fit the modelling needs. (Rauhamäki 2009a, pp. 20-21.)

Although CSDs can be used to model functionality of the system, they are somewhat limited in this aspect. For more detailed presentation of the functionality, the Profile offers *AutomationSequenceDiagrams* (ASD) from the AutomationConcepts subprofile. In addition to modelling the execution order of CSD diagram elements, it can also be used for more general modelling purposes, e.g. process sequence modelling. (Rauhamäki 2009a, pp. 21-22.)

4.1.5. Utilization of the Automation Profile

This chapter gives basic guidelines about how the Automation Profile can be used to support the automation software development and considers possible problems that may be encountered in the process. The first version of the Profile was developed in 2006 but – as mentioned earlier – the improvement of the Profile is still underway. Recently the Requirements and AutomationConcepts subprofiles have seen improvements, leaving especially the DistributionAndConcurrency subprofile waiting for an update.

One of the key design principles of the Profile was to make it platform independent. This design decision has inherently some pros and cons attached to it. Obviously, it allows flexibility when choosing the hardware platform while maintaining the

reusability of old design solutions and models. Because the Profile has been developed to be easily extendable, support for new standards or features can be added easily. Another benefit is that the Profile can be used in a wide range of different automation-related design tasks as it is not restricted by narrow support for platform specific concepts.

As for the downsides, platform independency can complicate the design process. For example, situations might arise when there is no exact match for a required modelling element, a situation which requires a workaround solution. Although the use of high-level modelling languages aims at reducing development costs, the cost of the modelling work itself may and probably will rise due to increased complexity and amount of work, even if the total cost goes down, as discussed in chapter 4.1. With platform independent solution there exists the danger that the Profile might be either too limited (not possible to model a specific feature) or too generic (presentation of the modelled feature too simple) to be useful. In the worst case it is possible to be both if some parts are too loosely-defined and others too constrained. Overall the platform independency can be considered an essential requirement for the possible success of the Profile.

The Profile is moderately large as an entity, including many design concepts from different areas of expertise, so it can be hard to master all of its aspects, although there is no need for a normal automation software engineer to do so. User groups can choose to learn and use only the relevant subprofiles, which would be the recommended approach for the most of the AP users.

According to Ritala, use of UML to model automation applications is still rare, partly due to the informality of UML which can cause misunderstandings between developers (Ritala & Kuikka 2007). Sometimes it can be difficult to understand how a certain concept has been intended to use, especially with vaguely defined concepts or if the profile specifications seem to contradict with common sense, e.g. a Processor from DevicesAndResources subprofile is an ActiveResource, not a PhysicalResource as one might expect. To prevent confusion about the concepts and their use, good guidance and examples of use are crucial in addition to good design of the Profile. The implementation of the guidance for the Profile has been part of the research the ASE research group has done and includes a written guide with examples and user support features integrated to AP Tool. These are covered more extensively in (Rauhamäki 2009b). The Profile also has its own weekly exercise on the course ACI-32020 which aims to introduce students to the use of AP Tool and the Profile.

Alternatives for the Automation Profile present altogether different approaches as there are not any other implementations for comprehensive automation-specific modelling languages. Possible UML-based options include using the basic UML, an approach that might best suit traditional software engineers hired to design automation applications because in all probability they already are familiar with UML and know how to use it. UML extensions like SysML offer another solution but since they have not been designed for automation software design, their suitability for this task is

somewhat questionable. Finally there is the possibility of not using UML at all in design process, which means that the modelling must either be done with some less well-known or non-standard modelling language, risking misunderstandings, or be neglected completely, missing out on all the benefits of modelling altogether.

4.2. AP Tool

AP Tool is a modelling application based on the Eclipse-based open-source modelling tool Topcased. AP Tool enables automation software architecture modelling using the concepts and diagram types defined by the Automation Profile. The Tool has been implemented in an Eclipse environment because there are high quality open source tools based on the Eclipse Modeling Framework (EMF) and UML 2 plug-ins that support extensions to the UML metamodel, thus enabling the development of it with relatively small resources. AP Tool and its development process are covered more extensively in (Vepsäläinen 2008).

AP Tool has been developed as a part of the AUKOTON research project and is an integral part of the AUKOTON development process, the uninterrupted development path from P&I diagrams to control software, including code generation[5] based on models. However, this thesis will focus on the modelling capabilities of the Tool, ignoring code generation aspects, as they are still in the early prototyping phases.

To help the software design process and modelling, there are a number of support features implemented in the Tool; in addition to the integrated help documentation they include cheat sheets and wizards. Cheat sheets are a form of Eclipse aids introduced in Eclipse version 3.0 that are meant to guide a developer through a series of tasks to achieve some overall goal. They integrate to the main Eclipse windows so that there is no need to switch between Eclipse and the help window. (Tiedt 2005.) Wizards are user interfaces consisting of sequential dialogue boxes, used to complete some specific task, like creating a new project. Additionally the Automation Profile exercise described in chapter 5.2.2 acts as a tutorial for AP Tool.

The user might encounter some challenges in the use of the Tool as it is still under development. This includes software bugs and missing minor features. Additionally the software might be slow if used on underpowered PCs or virtual machines, as the extended Eclipse platform is not as fast as programs built from scratch, but generally it is satisfactorily usable and fast to be used for educational and demonstrational purposes.

4.3. OSGi

OSGi Service Platform is an open Java-based module system specification maintained by the OSGi Alliance. The Alliance – originally known as the Open Service Gateway

[5] Actually the process of generating executable applications with AP Tool could be more appropriately described as composing applications by instantiating and linking existing type circuit types into applications, but code generation is sufficiently good generalization.

initiative – was founded by Ericsson, IBM, Oracle, Sun Microsystems, and others in March 1999. Its members today include a number of well-known companies from different business areas, for example Deutsche Telekom, Motorola, Nokia, Red Hat, Samsung Electronics, SAP AG, Siemens, and SpringSource. (OSGi Alliance 2009b.)

The main selling point of OSGi technology is that it brings a modularity layer to Java. Modularity can be used to divide programs to logical segments that can be developed in their own right, a procedure which can be used to simplify development by increasing the level of abstraction and enabling better division of labour, increase reusability, and improve maintainability. Support for modularity in the standard Java language is quite limited. As Java packages are only a way to organize classes into groups, packaging does not affect code visibility. Additionally Java class path does not know anything about code versions. In a worst-case scenario one might have several versions of a class loaded simultaneously and class path will return the one that it finds first which can create problems that are very difficult to solve. (Hall et al. 2009, pp. 5-7.)

OSGi Service Platform specification implements an environment that enables development, deployment and management of services dynamically. Dynamic software development may sound like yet-another software buzzword but here it is used to signify that individual services can be started, stopped and updated without restarting the whole framework. The OSGi Framework was originally targeted at networked embedded devices like residential Internet gateways that have co-operating pieces of software from several parties, but has since proved its applicability in much broader use. Present uses of OSGi include smart phones, home automation, telematic systems in cars, enterprise server solutions when combined with Spring Framework, and Eclipse's plug-in system which has been OSGi-based starting from Eclipse version 3.0. (OSGi Alliance 2007b.)

4.3.1. OSGi Architecture and Technology

A core part of the OSGi specification is the OSGi Framework. The framework is responsible for component lifecycle management, dependency resolution, class loading, service registrations, and event management (McAffer et al. 2009). Although the terms OSGi framework, OSGi runtime, and OSGi Service Platform are often used similarly, they have different meanings. The framework is the runtime that implements and provides OSGi's core functionality. Service Platform refers to the OSGi specification and includes OSGi standard services in addition to the core framework. Standard services are a collection of Alliance-defined reusable application programming interfaces (APIs) for commonly used tasks like administration and logging.

The current version of OSGi specification is 4.1 from April, 2007 (OSGi Alliance 2007a). As the Alliance only defines the specification, there are currently several runtime implementations, many of which are open source. For a listing of some of the

available implementations see Table 4.1. Although the amount of available frameworks can be somewhat confusing, this gives flexibility to choose the framework that best suits the needs of the project in hand.

Table 4.1. *OSGi framework implementations.*

Implementation	Licence	Developed by	Description
Concierge	BSD style	open source	Compact and highly optimized, implements OSGi release 3.
Equinox	EPL	Eclipse Foundation	Framework reference implementation.
Felix	Apache 2.0	Apache Software Foundation	Compact, designed for easy embedding.
Knoplerfish	BSD style	Makewave	Mature, also available commercially.
mBedded Server	commercial	ProSyst	Optimized for embedded devices.
Newton	GNU AGPL	open source	-

The OSGi framework is divided into several layers that can be seen in the figure 4.7 which depicts the framework's architecture. Conceptually most important of these layers are the *Module Layer*, *Life Cycle Layer*, and *Service Layer*. Security model in OSGi is based on the Java's security model which provides a safe sandbox for applications to run (Venners 1997). *Security Layer* itself is an optional layer based on the Java 2 security architecture which adds permissions and security-related services (OSGi Alliance 2007a) to the Java model.

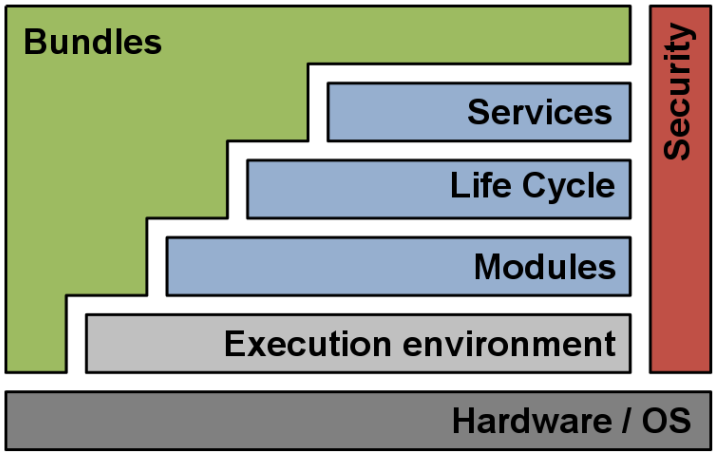


Figure 4.7. *OSGi architecture (OSGi Alliance 2007a).*

Software components that make up the OSGi applications are called *bundles*. They are OSGi's unit of modularization, as defined by the Module Layer. A bundle is a self-describing package that can both provide and consume services. The physical implementation of the bundle is a JAR file containing code, other resources, e.g. images or libraries, and supplemented with a manifest file that gives the description of the

bundle's contents and requirements. Java code in the bundles is organized into standard Java packages with the related folder structure. The metadata that the manifest file contains includes information about the bundle's identity, i.e. name and version, list of imports and exports, and optional information like the minimum Java version that the bundle requires (Bartlett 2009, p. 16). As the bundles are just JAR-files with some metadata, they have the advantage of being able to be used even in non-OSGi contexts. This reusability aspect is further supported by the convention of placing OSGi-related code into different package from the business logic code in the bundles.

In order to share their Java types, bundles must export appropriate packages. This is done by listing them in the manifest file's *Export-Package* line. Exported packages can be tagged with a version number. The published API for the bundle is made of the Java packages that the bundle has exported in this manner. Other packages in the bundle are considered to be part of the internal implementation and are not revealed to outside. As the package is the unit of visibility in the OSGi, extra care should be taken with their naming and placing of classes into them. In addition to exporting, bundles can state their dependencies by importing packages in the same way with *Import-Package* or *Require-Bundle* lines. Import-Package specifies only the name of the required package whereas Require-Bundle also expects a certain bundle, thus being more specific. Good OSGi practices recommend the use of Import-Package line as it decreases coupling between bundles. OSGi also has the ability to control version numbers of imported and exported packages. Bundles can impose requirements for version numbers or indicate that a package is not a mandatory requirement.

The bundles have dependencies between themselves but they are not organized hierarchically, instead forming a community of bundles (McAffer et al. 2009). OSGi applications do not have a main class or a main method that would be responsible for creating new bundles. Instead bundles can define a *bundle-activator* class that is used to start or stop the bundle. Bundle-activator must implement the corresponding interface with start and stop methods. These methods have a single parameter, called *BundleContext*. *BundleContext* object is the gateway that is used to gain access to the framework and its functionality.

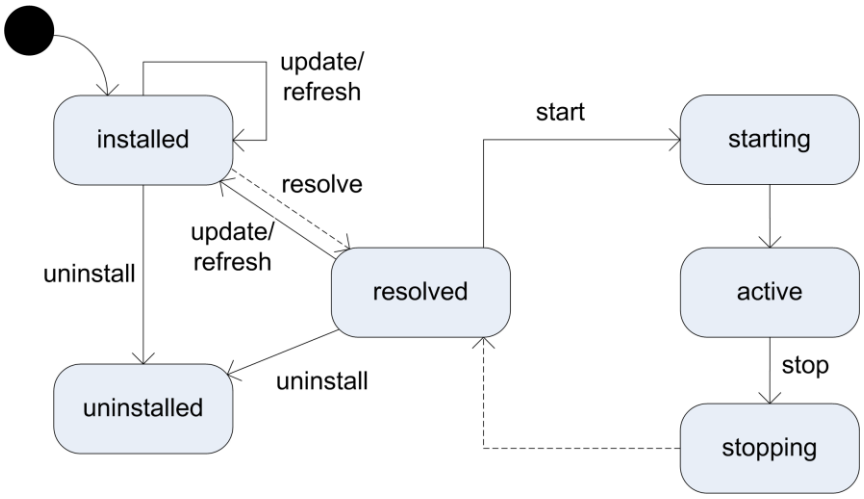


Figure 4.8. *Bundle lifecycle states (OSGi Alliance 2007a). Dash line indicates automatic or implicit transformation.*

Life cycle of the bundles is controlled by the Life Cycle Layer in the framework, which offers the possibility to dynamically change the states of the bundles in a running system. Bundles begin their existence in the installed state as shown in figure 4.8. If all their dependencies are met, they will be moved to the resolved state. If the bundle has an activator, it can be used to acquire resources when the bundle is started and transformed to the active state. Usually the framework handles bundle start-up automatically but it can also be done manually, as can be installation, refreshing etc. Manual management of bundle states of course is not a practical alternative when the number of bundles starts to grow but it can be used to troubleshoot applications.

OSGi applications can be based on bundles alone – most basic applications could have e.g. only two or three bundles with one bundle activator, but creating this kind of software is not using OSGi's strengths and could be achieved even without OSGi. By taking advantage of OSGi's ability to create services we can increase software complexity and modularity by increasing the decoupling between bundles. Services are contracts typically based on a Java interface. It is a good practice to separate the interface specification and implementation by placing the classes that implement the service to different packages from the ones used to export the interface. Services are defined by the Services Layer which consists of the actual services offered by the bundles and the *Service Registry* that holds the service registrations.

Service Registry is the entity through which the coordination of the services offered and required by the bundles happens. When a publisher, i.e. a bundle, wants to publish a service, it calls the *registerService* method in its *BundleContext*, whereas a service requester can discover a service using the *getServiceReference* method from its BundleContext (Chappell & Khanderao 2009). Since the programmer normally cannot

affect the order in which the bundles are started, race conditions[6] can sometimes come up when all of the services required by the bundle are not yet available at the start-up. Correspondingly when writing bundles, one cannot make the assumption that services would be always available. Naturally OSGi and its implementations offer tools to help manage service dynamics, including simple methods like start levels and more complex ones, like Service Trackers and Declarative Services that are suitable even for larger applications.

Release 1 of OSGi had the *service listener* APIs as the only way to work with OSGi service model but they required complex code to handle service starting order and concurrency issues. They are still supported but not recommended to be used. (McAffer et al. 2009.)

Service Activator Toolkit (SAT) is a 3rd party solution for dynamic services that was developed as an alternative to low-level service listeners during early OSGi versions. It acts as an abstract bundle activator that can also take care of some instantiation tasks that would otherwise need to be written to standard bundle activator code. SAT adds some overhead for each bundle in the system and thus may not be suitable for applications with bundle numbers ranging in thousands. SAT was originally developed for Equinox but should be framework-independent, just like most other Equinox bundles. (McAffer et al. 2009.)

Service Trackers are customisable objects added in OSGi release 2 that can be used to track services, as the name implies. Service Trackers abstract much of the complex actions that are needed to handle the dynamic starting and stopping of services but writing the required code while maintaining scalability is complicated. (McAffer et al. 2009)

Declarative Services (DS) is the latest solution to control dynamic service issues, added in OSGi release 4. Bundles declare in an XML file what services they register and acquire while DS takes care of binding and unbinding the services lazily at an appropriate time, which completely eliminates the need for service management code and the bundles actually do not need bundle activators at all. DS significantly simplifies the service-oriented model by declaratively handling the dynamics of services. (McAffer et al. 2009.)

Spring Dynamic Modules (Spring DM) is intended for building Spring Framework based applications that can be deployed in an OSGi execution environment. Spring DM is tightly interconnected to Spring Framework and needs a large amount of additional Spring and logging bundles to function. Spring Framework itself is an open-source application framework that is used as an alternative to Enterprise JavaBeans (EJB) model. (Walls 2009) Like DS, Spring DM also enables publishing of services in a declarative fashion, but considering the increased complexity there may not be a strong

[6] Race condition in computing refers to a situation where correct operation of software presumes that certain process (or bundle in the case of OSGi) must perform a required task before other processes can be executed normally. Failure to do so could lead to erratic behaviour or system crash.

enough rationale to warrant the use Spring DM, unless the use of the Spring Framework can bring additional benefits that outweigh the complexity.

The simple way that OSGi adds modularity features to Java might be one reason behind the rising popularity of OSGi. Limiting the overall complexity of the application framework can be a benefit to software development as it makes easier to keep things under control. Even though the basic methods of the OSGi Framework are uncomplicated, this does not mean that OSGi is not capable of handling more complex needs and situations. However, with dynamic services things quickly start to get complex, especially when considering all the above mentioned alternatives for the dynamic service management. Declarative Services does a good job at simplifying things but it lacks a rich programming model for component declaration and consumption, and can complicate debugging by hiding implementation details. It is good then that OSGi Alliance also seems to recognize these problems and according to draft-versions, the next OSGi Service Platform Core Specification should contain a new component model inspired by Spring DM known as the Blueprint Service. (Walls 2009.)

As the amount of alternative solutions seems to suggest, dynamic software development is not a free lunch. Although it increases software reusability and flexibility, complexity is also increased in one way or another. Even though the implementation technology would simplify implementation, modularity still increases the difficulty of debugging, especially if the technology hides the implementation like in the case of DS. To decrease the number of possible problems, best practices for dynamic programming should be followed. Some of these are brought up in the educational material to help students with the OSGi assignments and also to demonstrate possible hardships of dynamic software development.

4.3.2. OSGi and Service Oriented Architecture

One method designed to tackle the ever-increasing complexity in software development is Service Oriented Architecture (SOA), or more extensively Service Oriented Computing (SOC). SOA has been created to speed the development and distribution of new systems and on the other hand to ease the integration of different systems. Although it is usually associated with Web Services, OSGi also fits under the metaphorical SOA umbrella but uses a different kind of approach. Unlike Web Services, OSGi services are running in the local virtual machine (VM). SOA is based around the concept of developing loosely coupled services that communicate with each other and then composing these services together.

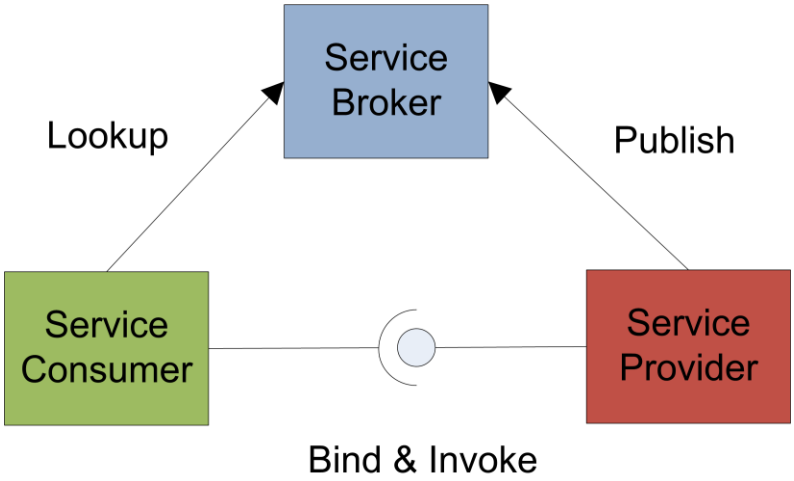


Figure 4.9. *Basic model for Service Oriented Architecture. Adapted from (Papazoglou 2003).*

Service providers offer services and service consumers use them. Coordination between services happens through a service broker (Service Registry in the case of OSGi). This basic model for SOA is presented in figure 4.9. This allows software developers to reuse services in several applications or allow services developed by different organizations to co-operate. In an ideal situation the software engineers could pick necessary services from a variety of libraries containing third-party and in-house developed services, and then combine them together as the final program. Resulting strategic benefits for successfully applying service orientation design principles can include increased ROI (Return On Investment), reduced IT burden and increased operational agility (Erl 2008, pp. 55-65).

Although the OSGi framework specification was not developed self-purposefully as a SOA technology – the development started in 1998 – the resulting programming model by and large complies with principles of service orientation. OSGi Service Registry acts as the service broker between bundles, basic building blocks of OSGi applications that can both publish and request services. (Chappell & Khanderao 2009.) For a more detailed analysis we can compare how OSGi enables the implementation of service orientation principles, presented in (Erl 2008, pp. 71-73).

Standardized Service Contract – services express their purpose and capabilities via a service contract. In OSGi this is done through bundle API, or in an XML file in the case of Declarative Services.

Service Loose Coupling – OSGi promotes use of designs that increase loose coupling of services as services in OSGi are dynamic, which means they can come and go without warning and the developer must be prepared for this. E.g. the Whiteboard Pattern is an alternative to the Listener Pattern where, instead of keeping track of listener bundles in the event source bundle, the listeners register themselves to the OSGi's Service Registry. Event source bundle tracks the listener services through the registry and calls them when there is an event to deliver. This way the event source

bundle does not have to worry about listeners that forget to call removeListener method when stopping. (OSGi Alliance 2004)

Service Abstraction principle promotes hiding of implementation details which is encouraged in OSGi by placing the implementation classes to different packages from the API packages. The implementation packages are invisible to the outside world.

Service Reusability – one of the core concepts of SOA, bundles need to be developed with reusability and dynamic best practices in order to actually create reusable components or services. This principle is supported by the decoupling of bundles. Additionally OSGi supports the dynamic starting and stopping of bundles so that the bundle composition can be easily changed.

Service Autonomy – bundles have a contract (manifest in the case of OSGi) that expresses their functional boundaries, and exercise control over their execution environment.

Service Statelessness – DS provides a rich component lifecycle model, including lazy loading of components. Without the use of dynamic services, statelessness is difficult to achieve in OSGi.

Service Discoverability – DS simplifies service discoverability notably. Offered services are simply declared in an XML contract file.

Service Composability – OSGi application is a community of bundles that enables adding and removing of bundles without stopping the framework. In the optimal case unit of composability would be a service but in OSGi it is a bundle which may contain a number of services.

Considering these principles OSGi could indeed be considered as a SOA technology, but in a smaller scale than enterprise level. However, it should be noted that even though OSGi *enables* the development of service-oriented solution logic, attention must be paid to principles of service design for the benefits to realize – but this naturally applies to all service-based technologies. There has been some discussion about whether OSGi is a SOA or not, e.g. in (Normington 2007), but even the OSGi Alliance mentions SOA in their OSGi whitepaper (OSGi Alliance 2007b, p. 3).

Code reusability and use of off-the-shelf packages have been promoted for 20-30 years but they have not proved to be the technological breakthrough or the silver bullet that would solve all problems of software development (Brooks 1987). Designing of reusable software takes extra effort when compared to conventional software design and the resulting components might still become too general-purpose to fit any specific needs or run too slowly (Haikala & Märijärvi 2006). Nevertheless, the trend for reusability has been slowly but constantly rising and it has successfully been used to improve quality and productivity of software engineering (Schmidt 1999). OSGi's take on service oriented computing in a dynamic environment can be considered as another step to the direction of easier development of complex software, as in any large systems it is unlikely that the whole application would be developed by only one software house.

4.3.3. Equinox

Equinox is a modular and dynamic Java runtime environment based on OSGi framework R4 specification. It was chosen as the implementation platform for the assignments because of its prevalence, as this makes it one of the most probable frameworks that students might encounter and work with in the future. It also means that there are many Equinox based tutorials and guides available in the Internet and literature. Equinox's tight integration with Eclipse IDE is an additional benefit since the research group is well familiar with Eclipse as it is being used on several courses and as the implementation platform for AP Tool.

Eclipse had its own modularity system but moved to OSGi in version 3.0: Eclipse plug-ins are now OSGi bundles and Equinox is the basic framework under all Eclipse applications. Equinox is used in all Eclipse-based systems from airline check-in kiosks to application servers like WebSphere (McAffer 2008), but overall Equinox is more tuned for desktop and server applications whereas some of the other OSGi implementations are more suited to the embedded space (Sugrue 2009).

Equinox offers two mechanisms to support collaboration between bundles, namely services and extensions (McAffer 2009). The typical OSGi mechanism for this is the service registry but Equinox also adds the extension registry as a complementary approach. Former approach, i.e. services in OSGi, has been discussed in previous chapters. Extensions are the traditional tightly coupled Eclipse collaboration mechanism where bundles can declare extension points to which other bundles can contribute extensions.

Until recently Eclipse plug-ins have been almost entirely based on extensions, but situation could change now that tool support for DS has been added in the latest Eclipse version 3.5 Galileo. Unlike extensions that are scanned and registered at start-up and then loaded lazily, services are not automatically registered. Compared to extensions, services had the problem that Eclipse platform did not know at start-up what bundles were going to be necessary, and thus had to load all bundles, leading to a slow start-up. DS solves this problem by activating bundles when they are needed. (Bartlett 2007.)

4.4. .NET Micro Framework

Microsoft .NET Micro Framework (.NET MF) is a .NET-based software platform for resource constrained or embedded devices that cannot run the heavier Windows Embedded CE or .NET Compact Framework platforms. It includes a small version of the .NET Common Language Runtime (CLR), known as TinyCLR, and supports software development with Visual Studio IDE and C# programming language. C# is a modern and usable high-level language that supports object-oriented and component-oriented programming paradigms. Debugging can be done either on hardware or emulator through Visual Studio. The runtime has a fairly small footprint of few hundred kilobytes, depending on how much of the provided functionality is used, and runs on

low-speed 32-bit processors (Microsoft 2009). The power consumption is in reasonable limits, making batteries a viable power-source option for .NET MF -based systems. MF also includes support for the Device Profile for Web Services developed by Microsoft, a minimal implementation for Web Services specification. (Kühner 2008.)

Compared to more traditional embedded software development, which is often based on C or Assembly programming languages, .NET Micro Framework can support faster software development for embedded platforms: C# and object orientation raise abstraction level, cutting the need for tinkering with small details and enabling use of software design patterns and other advanced software design methods. .NET MF removes the need to write code for hardware interfaces by abstracting the hardware access when using standard hardware platforms like in the Traxster robot (see chapter 5.2.3). Hardware interfaces are implemented by objects that are remarkably easier to use than switching bits in registers. Use of standardized hardware platforms that are produced inexpensively in large numbers can also cut hardware costs. Additionally .NET MF makes it easy for a software engineer experienced in .NET programming to transfer into embedded software developing, since real differences between .NET and .NET MF are fairly small from the developers' point of view.

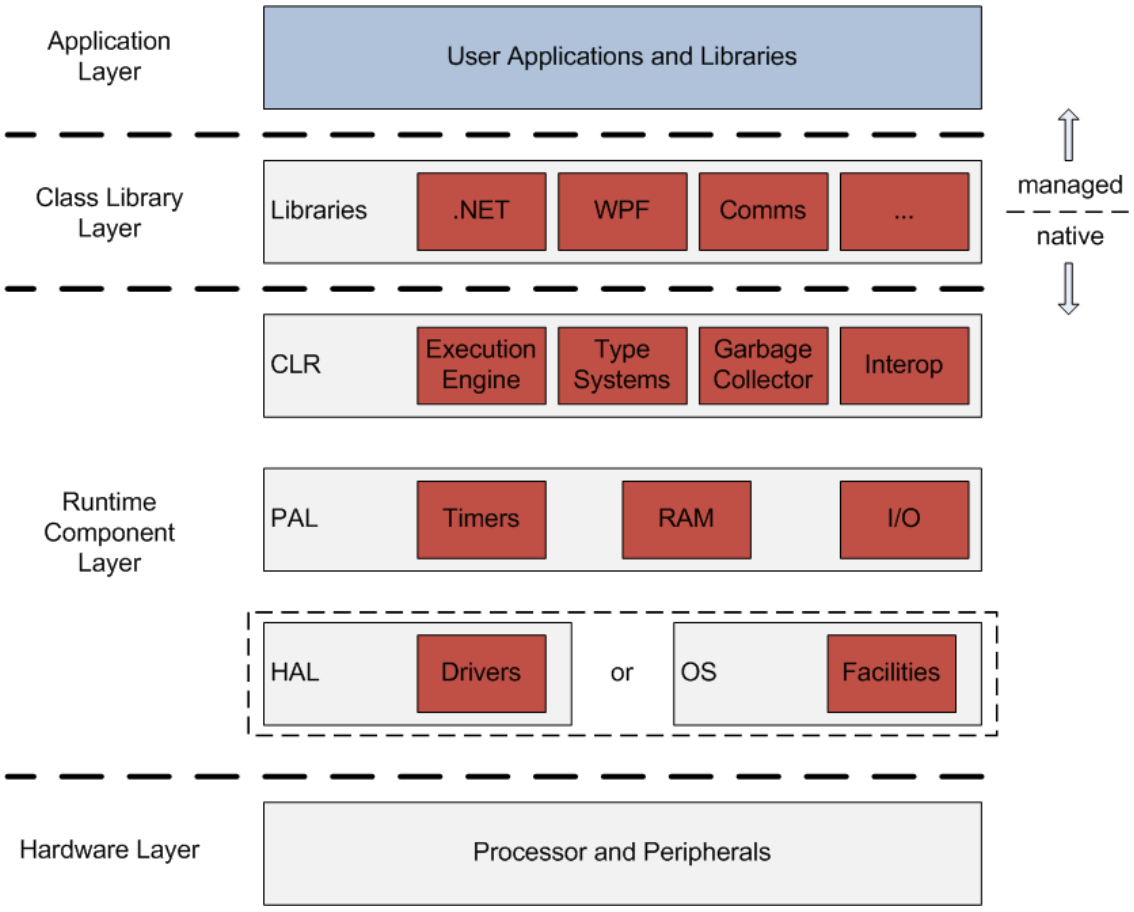


Figure 4.10. *The layers of the .NET Micro Framework architecture (Microsoft 2009).*

Architecture of .NET MF can be seen in figure 4.10. Top layer is the user application and external libraries, followed by the class libraries of .NET MF. Next layer is made up by the TinyCLR, i.e. Runtime Component Layer that consists of CLR, Hardware Abstraction Layer (HAL) and Operating System (OS) or Platform Abstraction Layer (PAL) components. The hardware makes up the bottom layer. The C# compiler and .NET MF Visual Studio plug-in produce size-optimised managed code in an intermediary-language that the TinyCLR executes. The TinyCLR takes care of memory management, thread and code execution, exception handling and other system services.

Raise of abstraction level often happens at the expense of performance, and .NET Micro Framework is no exception to this rule. Execution is not as effective as precompiled C or Assembly code as .NET MF does not have a native code compiler. Nevertheless, the execution is fast enough for most applications but it is not deterministic – MF is not a pure real-time system. Features like garbage collection that make the development process easier, can slow down the execution for the few milliseconds that could be crucially important in a hard real-time system. Additionally, lack of support for other programming languages other than C# can be seen as a downside and options for execution of unmanaged or native code are limited. MF also has restrictions on the available C# libraries. Most notably the choice of data structures is limited to non-multidimensional arrays.

A useful feature for debugging and rapid prototyping is the Extensible Emulator that is provided with the framework. The emulator supports emulation of several types of communication components and peripherals, including GPIO ports, SPI devices, I^2C devices and serial ports (Kühner 2008, p. 317). Support for other devices or hardware simulation can be added by manually writing a suitable component. The emulator application has a graphical interface by default, but it can also be turned into a console application. The emulator makes it possible to start debugging the software before the hardware is available, thus providing another tool to cut development time. It is also useful for educational purposes as students can develop and test their software at home.

Suitable applications for .NET Micro Framework can be found e.g. in home automation, industry control, robotics, microcontrollers and other small embedded devices. Possible alternatives for .NET MF in small-scale systems could be writing the software and start-up routines in Assembly or C languages; larger systems could use embedded Linux as an operating system. The .NET MF is a viable option in other than hard real-time applications; the choice of implementation technology should be based on needs and existing expertise.

5. RESULTS

The results of this study are presented here in brief. First a short description is given of the courses, then of the related work done for them.

5.1. University Prototype

Goal of this thesis was to create a university prototype of the research and development environment for distributed automation systems. The environment is not singular software or hardware system, but a collection of possible beneficial technologies. The educational material and assignments developed for this thesis and the research project form the university prototype for this environment.

The components of the prototype are the selected technologies, namely the Automation Profile, OSGi Framework, and .NET Micro Framework. The prototype will be put to practical testing when the material is adopted and put to use in the university courses. Future plans for the environment include possible industrial prototypes at least for the Automation Profile and AP Tool.

The attached material is mostly in Finnish since it is the language the courses are given in. An exception to this is made by the lecture notes of ACI-32020 which are in English. In this thesis the following terms are used to describe the tasks generated for students: exercise is a guided learning event with hands-on training, lasting no more than two hours. Assignment is a large task that the student does either independently or with a student colleague, aiming to create complete software, including the related documentation, thus forming a usable entity.

5.2. ACI-32020 Real Time Systems in Automation

ACI-32020 Real Time Systems in Automation is a course arranged by TUT's Department of Automation Science and Engineering. The course focuses in the basics and the design of real time software systems. Students will become familiar with basics of real time systems, reliability, performance and safety aspects, use of UML language and architecture design (Tampere University of Technology 2008). Large portion of the work required to complete the course is taken by the practical work in which the students design and develop a soft real-time application using UML-based methods and a suitable integrated development environment (IDE). The students can choose the subject of this assignment from a set of given automation-related topics. The subjects available in 2008 are presented in Table 5.1.

Table 5.1. *Assignment subjects for course ACI-32020 in academic year 2008-09.*

Name of the assignment	Programming language	Bonus points for the use of AP Tool
Batch process control	Java	yes
Tank process pressure and level control	Visual C++	yes
Tank process monitoring with a PDA device	C#	no
Microsystem actuator control	C++	yes
Robot control (new for 2008)	C#	yes

Assignment instructions given to students on this course are moderately specific about technical details so that the students can concentrate more on the design process instead of small details. This course is a mandatory requirement for ACI-32040 which covers software components and services in automation.

5.2.1. Educational Material

Educational material for the course ACI-32020 includes Real Time UML book by B.P. Douglass (2004), Power Point slides and a short lecture notes booklet written by the lecturing professor, Seppo Kuikka (2008). The booklet covers general topics related to automation software development, such as automation systems and reliability aspects. The new educational material created for this course is intended to augment the lecture notes with an introduction to the Automation Profile and AP Tool. Since the lecture notes are in English, the material is directly based on chapters 4.1 and 4.2 of this thesis, with most references to projects and some of the discussion removed. Practical examples of the Profile's use are provided in the AP exercise that is covered in the next subchapter.

The educational material presumes that the reader has basic understanding of UML which is taught on a recommended requirement course, Introduction to Software Engineering. All students will not have completed the preliminary course but as UML is an integral part of the course's content, they get to familiarize themselves with it in weekly exercises and assignment.

5.2.2. Automation Profile Exercise

The Automation Profile was used for the first time in 2008 for the bonus sections of assignments. To give students basic understanding of the Profile and AP Tool, an exercise was created by ASE researcher Vepsäläinen. The exercise consists of creating a model for elevator control software, including a Requirement Specification Diagram, two Control Structure Diagrams and an Automation Sequence Diagram. The exercise has clear step-by-step instructions for completion, as its purpose is to serve as an introductionary example of how the Profile should be used.

For the academic term 2009-2010 I updated the exercise to reflect the development and changes made to the Profile and AP Tool, including Automation Concepts subprofile improvements and slight UI modifications. I also clarified the structure of the

exercise slightly to improve comprehension of different elements. The exercise instructions are in Appendix 1: UML Automation Profile Exercise.

5.2.3. .NET Micro Framework Assignment

Planning of a new assignment for the course ACI-32020 started in the beginning of summer 2008. Starting point was an embedded development kit (EDK) for Microsoft .NET Micro Framework which the research group already had in their possession. The EDK consisted of EMAC's iPac-9302 single board computer with an ARM9 processor and the necessary software to run Micro Framework applications on it. (SJJ Embedded Micro Solutions 2008). The .NET MF is described in chapter 4.4.

The goals were firstly to create an interesting and easily approachable assignment for the students, and secondly that it would be straightforward to apply the Automation Profile in the software planning phase of the assignment. A robotic application was considered appropriate for fulfilling these goals.

Figure 5.1. *Assembled Traxster robot.*

After comparing different robot kits, it was decided that the best choice was RoboticsConnection's Traxster .NET Micro Framework Robot Development Kit which was using Plus-version of the aforementioned EDK. (Summerour Robotics Corporation 2008). Use of this kit made developing the assignment considerably easier compared to some of the other kits, since it had a second board to control the hardware and included a .NET MF library to control all the sensors and actuators of the robot. Traxster kit included only one infrared sensor so it was ordered with additional sensors, including an ultrasonic range finder, some additional infrared sensors and an electronic compass. The built-up Traxster robot can be seen in figure 5.1.

Programming of the Traxster robot is carried out with C# language in Microsoft Visual Studio IDE. Since the .NET Micro Framework is more limited than full-sized .NET framework, all features of the language are not available. One of the most limiting restrictions is that Micro Framework only supports ArrayList and Dictionary data structures. Nevertheless, support for many other useful features, e.g. threads and exception handling, is retained.

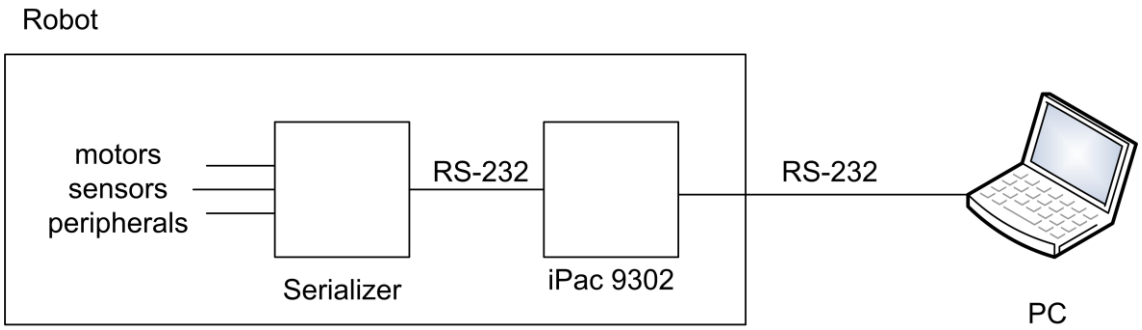


Figure 5.2. *Simple diagram of the Traxster robot.*

Figure 5.2 shows a simplified diagram of the Traxster robot. PC can be connected to the iPac-9302 board via serial cable (RS-232) in order to transfer the program to the robot or to debug it. iPac-9320 runs the .NET Micro Framework CLR, the user's program and Serializer library component. iPac communicates with the Serializer board via another RS-232 port using the Serializer library. Serializer board handles all low-level communication with sensors and other peripherals like servos, and commands the electric motors.

In order to help the software development for the robot and especially to allow the students to write and test their own programs at home, a software-based robot emulator[7] was developed. This emulator, concisely named TraxEmu, was built using Micro Framework's Extensible Emulator feature. The basic emulator actually emulates only the iPac 9302 board and rest of the hardware emulations are custom software added to this. TraxEmu needs to be used with a modified version of the Serializer library since

[7] In computer science terms emulator and simulator are sometimes used interchangeably. Simulator might be more appropriate term here but emulator is used instead since the engineered software is based on the Micro Framework's Extensible Emulator.

the emulator is not fast enough to directly replace the hardware. TraxEmu uses a simulated model of the Traxster to calculate speed, direction and sensor measurements that are shown to the user and communicated through the emulated serial connection to the user's program. Unfortunately the first completed version of the emulator was lost in a hard-disk failure and had to be re-written. As there are some differences between the emulator and the actual hardware, it would be recommendable that students occasionally check that their software behaves as expected when used with the Traxster robot.

Assignment instructions for the robot are in two parts. First part describes the problem that needs to be solved and second part gives more detailed instructions for execution. Former gives a brief description of the robot and Micro Framework, general information about the robot, what needs to be done in order to complete the assignment, and detailed requirements. Latter part covers the topics of how to create a new Micro Framework project in Visual Studio, how to use the supplied Serializer library, how to use the emulator and the actual robot, and finally gives some basic guidelines for the structure of the program.

Use of the Automation Profile in the assignment is implemented via a bonus point system. Students can gain an increase of one grade (on a scale of 0-5) if they complete the pre-assigned bonus tasks. All assignment topics have a bonus section in the work description but the actual bonus tasks differ somewhat depending on work specifications. All assignments except for the PDA (see Table 5.1) will grant half of the bonus if the students use Automation Profile to model software requirements and structure. In the robot assignment another half of the bonus comes from using proportional plus integral (PI) control to improve the movements of the robot. Assignment instructions are in Appendix 2: .NET Micro Framework Assignment.

5.3. ACI-32040 Software Components and Services in Automation

ACI-32040 Software Components and Services in Automation is a follow-up course for the ACI-32020. The course aims to introduce students to building automation software out of components and Web Services (Tampere University of Technology 2008). Like the preceding course, ACI-32040 also has a number of exercises and one larger assignment that is carried out in pairs. This time the assignment instructions are looser and give greater freedom and responsibility for the students to create a working software architecture.

5.3.1. Educational Material

Educational material for course ACI-32040 consists of self-study booklet written by professor Kuikka (2009) and PowerPoint slides. The booklet starts by covering software component models and their use in software development. It then proceeds to Web-services and SOAs and finishes with the topic of application of components and

services in automation industry. The chapter about component models presents and describes JavaBeans, EJB, COM, .NET and CORBA, but does not contain anything about the OSGi Service Platform. One goal for this thesis work was to produce a subchapter for this material describing the OSGi technology.

The educational material written about OSGi for this thesis is included in the Appendix 3: OSGi Educational Material. The material is based on the chapter 4.3 of this thesis but has been rewritten in Finnish with university students new to Java programming in mind. However, it can be difficult to understand if the reader has completed only the minimum requirements for the course – namely the ACI-32020 and basic programming courses Programming I&II – or has no prior experience of Java, but this applies to most of the topics covered on this course. All English terms except bundle have been translated to Finnish, since bundle does not have an established Finnish translation.

The new material starts by giving motivation why OSGi is an important technology by presenting current applications of OSGi and rationalizing the need for modularization. It then explains why the existing features of Java are insufficient for the needs of the CBSE and why an additional modularity layer is needed. The study material tries to retain an implementation-free point of view, only introducing different implementations for OSGi Framework. Architecture of the Framework is presented on the basic level that is required to understand how the OSGi applications are formed and function. Dynamic services are discussed only briefly, as they are a fairly complex subject with several alternative implementations. The final part of the material introduces best practices for designing OSGi applications that can also help to give better understanding of the OSGi framework and especially its dynamic sides. The best practices can additionally be useful for the students that choose to do the OSGi assignment.

5.3.2. OSGi Exercise

The automation software component and service course ACI-32040 had an existing weekly exercise about OSGi Framework and Eclipse plug-ins for 2008 but after writing first draft for the larger OSGi assignment it was concluded to benefit from a rewrite. The old exercise instructions used a ready-made Eclipse view to implement the graphical user interface (GUI) whereas the new assignment requires a bit more flexible interface. Although Equinox implementation for OSGi framework has several good tutorials available on the Internet, especially up-to-date information about using dynamic services was hard to find on the moment of writing the exercise. Thus the weekly exercise is a good place to give a brief introduction to creating a graphical UI, introduce information about advanced OSGi features and some other good programming conventions for writing OSGi-based programs.

The topic of the exercise is to implement a simple OSGi-based calculator with a basic GUI. The calculator part of the exercise is kept as simple as possible so that the

students can concentrate on how the OSGi-related code is implemented. The exercise has been divided into three parts. First part covers the creation of the calculator logic and developing a GUI with Visual Editor plug-in for the Eclipse. In the next phase the calculator is changed to a basic OSGi application using only a minimal amount of OSGi's features. Finally on the third part the application is converted to be service-based without going too deeply into the details, while keeping the needs of the OSGi assignment in mind. This three-part structure aims to introduce OSGi in a piecemeal fashion to give a better understanding about different features of the Framework. Exercise instructions are included in Appendix 4: OSGi Exercise.

5.3.3. OSGi Assignment

The OSGi assignment is intended to give hands-on experience for students about component-based software engineering and OSGi technology. Objective of the assignment is to create a remote batch process control and monitoring utility for a mini batch process via OLE for Process Control (OPC) connection. The application will be done with the Eclipse IDE and the Equinox implementation of the OSGi specification. The assignment instruction does not provide specific directions; instead it reads more like a list of customer requirements. Choice of the implementation method for services will be left for the students, although Declarative Services are recommended in the instructions. Similarly the Visual Editor plug-in is recommended for development of the GUI. The assignment score will be based on implemented features, how the application has been divided into components, service design, and how well it follows software engineering and OSGi best practices, e.g. separating the implementation from interface.

It is not always possible for software engineers or even companies of today to do everything by themselves, so this assignment is a good chance to practice utilisation of components provided by 3rd parties, i.e. the course staff. The staff will provide a component that will act as an interface to the hardware. The interface can either be connected to the simulated model that is also used on the batch process control assignment of the ACI-32020 course, or to the real batch process hardware through an OPC connection.

Choice of assignment topic and possibility to test application with real equipment aims to motivate students and it would probably be most interesting to students studying process automation. The open-ended instructions and use of external components are similar to PBL approach discussed in chapter 3.2. The assignment instructions are included in Appendix 5: OSGi Assignment.

6. DISCUSSION OF THE RESULTS

Discussion of the results includes analysis of whether the results correspond with the set objectives and does this thesis answer the research questions set in the introduction chapter. The scientific and practical significance of the results will be assessed and compared to the pedagogical theories and goals set in chapter 3.

6.1. Technologies

The research problems that I set in the introduction chapter were the evaluation of the technologies and how to implement the environment in the university teaching. Chapter 4 is a general overview to the selected technologies and remains on the factual level, whereas here is a more speculative analysis of the technologies.

The Automation Profile is perhaps the most important of the chosen automation software technologies, as it could potentially be used thorough the automation field to aid development process and unify development concepts. However, to students the Profile represents a new area of expertise, as they probably have limited experience of automation software development and the related concepts. For them the Profile introduces new notions and practices and thus needs clear guidance. The AP exercise and educational material serve this purpose to some extent but there might be a need for additional material. The Automation Profile user guide (Rauhamäki 2009a) explains concepts used in the Profile but is probably too detailed for this purpose and written mostly with automation engineers in mind, so the possible additional reference material should be more simplified. This could be implemented as a part of the textual guidance integrated to AP Tool (Rauhamäki 2009b), from where it could be easily referred to and printed if necessary. In any case, more accurate estimation for the need of such material needs feedback from the students based on the currently available material.

One of the main benefits the Profile can offer is the possibility to support the traceability of requirements from the specifications to the implementation (Vepsäläinen 2008), as noted in the Requirements subprofile section of the Automation Profile chapter. The ability of AP Tool to use trace relations is demonstrated in the Automation Profile exercise and brought up in the educational material but it should also be emphasized in the personal guidance for the assignments. The industrial prototype of the HAJA environment might benefit from separate guidance for the use of trace relations.

Although the significance of the AP is being emphasized here, the other two technologies are also entitled for their places in this thesis. Anyone who has written software for embedded devices can attest that it is rarely a simple and easy process, but

.NET Micro Framework manages to make it just that, by enabling the use of a modern object-oriented language and Visual Studio IDE in the development process. MF does not need a separate operating system for the hardware and supports commonly used embedded peripherals and interconnections like I2C, SPI and USB. Although the framework does not include full C# libraries, it has support for commonly needed features like threads and exception handling. These features reduce the amount of basic routine code needed e.g. for schedulers and communication interfaces, thus decreasing software development times. The downside is that the choice of hardware is restricted to standardized platforms that are supported by .NET MF. To add support for a new platform, the Framework first needs to be ported to it which is a complex process and requires complete understanding of the hardware (Kühner 2008, p. 12).

The benefits of OSGi technology and modularity have been highlighted to some extent in the related chapter, but could be briefly summarised with easier runtime maintenance and upgrading, increased component abstraction, and software decomposition. OSGi is an extension of Java technology so it has additional benefit of being platform independent. Reusability and transportability make OSGi a powerful tool as the rapid development speeds of modern industry also imply shorter product lifetimes (Tommila et al. 2005, p. 77). Advantages of Java over the application-oriented languages used in automation domain include object-orientation, exception handling and garbage collection, in addition to aforementioned transportability. Although it is fast, standard Java is not a deterministic language which somewhat limits its applicability in the automation domain. For real-time applications Sun Microsystems has a commercial implementation of Real-Time Java specification, the aptly named Java Real-Time System (Sun 2009). Apparently Java RTS has not yet been used in combination with OSGi technology, but it could be an option if somebody wants the benefits of OSGi and Java in a hard real-time environment.

Even though SOA is usually associated with distributed computing and enterprise systems, OSGi was found to fulfil the characteristics of service-orientation in a smaller scale, i.e. in one virtual machine. However, draft of the next OSGi specification, 4.2, includes specification for Distributed OSGi and Distribution Software (DSW), a software platform that supports the binding and injection of services in other address spaces or across machine boundaries, using various existing software systems like Web Services or CORBA (OSGi Alliance 2009a).

6.2. Learning

This subchapter includes discussion of the results based on earlier learning and assignment development chapters. The basic learning theories from chapter 3.1 are examined from the viewpoint of general technological university education. The applications of learning theories from chapters 3.2 and 3.4 are reviewed with consideration to the created material and the courses relevant to this thesis.

Behavioural theories held the view that behaviour and learning is shaped by the environment. The learning process in the courses organized at TUT could be generally described by the following process: Students listen to lectures and do exercises to learn basic theory, and then get to apply this knowledge by doing assignments, for which they receive guidance and feedback. Learning or the change in behaviour is evaluated by forming the final grade from a combination of the scores of the exam and the assignment. Educator's role is to arrange the environment to support the learning process.

Cognitivist viewpoint emphasized the internal mental process. This viewpoint is rarely brought up on individual courses but the importance of finding a suitable studying style is emphasized in the freshman year. Cognitivism uses computers as a metaphor for human mind; instructions come in, are processed, and lead to certain outcomes. Instruction should be well-organized and clearly structured to aid memorization and building of logical relationships between ideas. With this in mind, I have tried to organize the structure of the educational material, exercises and assignment instructions as clearly as possible and tried to avoid overly extensive and complex explanations that would likely only confuse the students reading them. It is all too easy to forget all the effort it has taken to learning everything that one knows about automation and software engineering.

Constructivist theories considered learning as an active, constructive process that builds on previous knowledge. Graphical presentations of knowledge can be used to aid conceptualization, and thus help to add new information in relation to old. The role of the teacher is to act as a facilitator that helps the learners to discover facts and principles themselves with intuitive thinking and learning by doing. This is also a basis for criticism of constructivist teaching model, as especially with novice learners the mental models necessary to facilitate learning by doing do not yet exist. Instead, with novices the learning should start with active teaching and learners being cognitively active during learning (Mayer 2004).

Bloom's taxonomy is a useful tool to estimate the learning levels that students have achieved. The first level – remembering – already requires that the relevant knowledge can be recalled from long-term memory. Lectures are a part of the process to commit matters to the long-term memory, aided with exercises and reading the material. If students miss or skip most of the lectures, they are unlikely to learn much from the exercises either, as they do not have necessary knowledgebase that the exercises are intended to expand. Weekly exercises also aim to give understanding of the subject which is necessary to be able to apply the knowledge in the assignment and thus it is not sensible to make difficult or complex exercises as students likely have not yet learned to apply their knowledge. Finally the exam should consist of a range of questions, some of which test remembering, others understanding and applying of knowledge. Generally the higher levels of learning, i.e. ability to analyze, evaluate and create, as defined in the taxonomy, require much more thorough knowledge of the subject.

Principles for exercise and assignment development listed earlier included induced realisation, learner-centric approach, repetition, deep processing and hands-on experience. Assignments aim to encourage deep processing and understanding with loose task descriptions, especially on the course ACI-32040 where the loose problem description of the assignment is close to the principles of PBL. In real-life the problems are rarely well-structured, as the case often is with many educational exercises. Interesting assignment topics can improve motivation and thus encourage learning and deep processing. Lecture notes also try to provide interesting and concrete examples, e.g. the Boeing example in OSGi material (Appendix 3: OSGi Educational Material).

The goal for this thesis was to contribute to the creation of the university prototype of the environment, and more specifically, that the prototype, i.e. learning material, should raise the learning readiness of students in the area of automation software development and integration technologies. The success of the results cannot be fully analyzed before the material has been put to use in the educational courses, but I am convinced that the work done in this thesis has benefited from the study of the learning and pedagogical theories and that the implementation of the technologies to education was accomplished satisfactorily. Because of the technical nature of this thesis, this background information had to be presented in a fairly short format, but it is nevertheless questionable if a longer and more profound discussion of learning theories would have added anything to the produced educational material.

The covered automation software development and integration technologies may not form a cohesive environment or entity, but with so diverse technologies it may not have been even desirable, even though the Automation Profile is being used in the bonus section of the .NET Micro Framework assignment. Instead the educational applications for the technologies can be considered as smaller prototype subprojects for the main HAJA research project that also fulfil educational needs.

7. CONCLUSIONS

This chapter collects the conclusions and gives some recommendations. People that might be interested in this work include the ASE research group, personnel related to AUKOTON and HAJA projects, persons working in automation software industry who are interested in the new technologies presented in this thesis, and those interested in the pedagogical aspects of technical university education.

Automation software is being developed by various companies that create automation products and systems, ranging from electronics to processing industry. This thesis has been carried out as a part of the HAJA research project which aims to develop a research and development environment for automation software, based on new software technologies that utilize paradigms like object orientation and component-based development. Technologies that were chosen for implementation to teaching were the Automation Profile UML extension and AP Tool, OSGi framework, and .NET Micro Framework.

The subjects of the study material represent cutting edge technologies in the fields of software and automation. To ensure that the assignments and other educational material produced would meet sufficient pedagogical standards, learning theories and their applications were studied and used as guidelines for the development process. However, the material and assignments presented in this thesis should not be viewed as finished end products but rather as targets for development based on student feedback and course staff experiences. The material also needs to be updated based on updates and changes in technologies.

The key points of learning theories that were used in material design were mostly based on cognitivist and constructivist theories and their applications. The material aims to give the students a chance to get hands-on experience on interesting tasks to increase learning motivation. The exercises and assignments were designed to develop gradually more challenging as the level of understanding of the students increases. Other important points for material development that were brought up were induced realisation or insight, learner-centric approach, repetition, and deep processing. However, if the learning theories are mentioned only as a fashionable trend and not used in the implementation with thought, the pedagogic quality of the end result likely will not be different from just ignoring the theories.

The technologies together with learning theories form a diverse combination and one concern with this work was to find the best way to combine them to a coherent entity. The common themes I have chosen to emphasize as uniting factors are the applications of learning theories and component-orientation.

The UML Automation Profile could be used to support reusability of commonly used solutions to design problems stored in model libraries. The current implementation of AP Tool does not have this feature but it should be possible to be implemented in the future. Use of UML to model automation applications is still rare, partly due to the informality of UML, but the Profile could be used to unify development concepts. It has also been designed to be platform independent and extendable to better suit application-specific needs and standards. Finally, the traceability of requirements that the Profile supports is especially important for the safety and quality of software. However, AP Tool is not a finished commercial product and although it fulfils its purpose as a modelling tool, there are still features that need to be implemented and the user interface needs some refinements. Additionally, AP Tool can be hard to use if the user has now prior experience of UML. Some of these problems could be taken care of with extra material about UML. One way to implement the material could be the integrated help material in AP Tool.

Controlling modularity of software is a problem that grows with size of the applications. Simple class and package-based modularisation is easy to maintain as long as application size remains fairly small but maintenance can quickly get out of hand. OSGi offers a solution to this by offering a dynamic component model to Java platform, something that is missing from standard Java. Especially the feature to dynamically start and stop OSGi bundles seems like an attractive feature in automation domain, as this would enable software updates and fixes without stopping the whole framework. Naturally this kind of robustness does not come automatically with any technology and requires careful design. Downside for OSGi use in automation industry is the non-deterministic nature of Java caused by the garbage collection mechanism and JIT-compilation among other things, even though on average the performance of Java and other JIT compilers comes close to code precompiled with e.g. C or C++ compilers.

Fundamentally OSGi is a simple technology but obtaining full benefits from the framework requires utilization of dynamic services. There are various methods to do this but the simplest solution would be to use Declarative Services. DS runtime uses XML based declarations to track provided and required services, and reduces the amount of required OSGi-specific boilerplate code. Compared to other approaches to implement services dynamically, DS manages better to reduce coupling between bundles and does it in a simple way.

Although .NET Micro Framework may not be a CBSE technology per se, C# gives possibility to implement reusability with principalities associated with object oriented languages like abstraction, encapsulation, inheritance and modularity in the embedded platforms. Ability to use C# and Visual Studio IDE with .NET Micro Framework features like hardware emulators enables rapid application development and prototyping.

Constructivist theories and research done on the use of concept maps seem to support the use of modelling to enhance learning, but this could be a subject for further research. The adaptation of the technologies, especially OSGi and the Profile, in

automation software production could be possible as an industrial prototype. Although the next version of OSGi has not been yet published, its draft version contains interesting new features like support for distribution. AP Tool could benefit from the implementation of additional features and UI refinements in the future, and its potential for further development could be tested by extending support for new platforms.

BIBLIOGRAPHY

Anderson, L. W., & Krathwohl, D. R. 2001. A taxonomy for learning, teaching and assessing: A revision of Bloom's Taxonomy of educational objectives: Complete edition. New York: Longman. 336 p.

Antila, M. 2008. Applying UML 2 techniques to batch processes and machine control. Espoo: Helsinki University of Technology. 94 p.

Bartlett, N. 2007. A Comparison of Eclipse Extensions and OSGi Services. Retrieved August 26, 2009, from EclipseZone: http://www.eclipsezone.com/articles/extensions-vs-services/.

Bartlett, N. 2009. OSGi in Practice. Draft. Retrieved August 31, 2009, from Neil's point-free blog: http://neilbartlett.name/blog/osgibook/.

Bissell, A. 2003. UML 2.0 -- a major revision of the industries de facto software modelling language. Software World, January 1, 2003. Retrieved August 31, 2009, from The Free Library: http://www.thefreelibrary.com/UML+2.0+--+a+major+revision+of+the+industries+de+facto+software...-a097232543.

Bloom, B. S., & Krathwohl, D. R. 1956. Taxonomy of Educational Objectives: The Classification of Educational Goals, by a committee of college and university examiners. Handbook I: Cognitive Domain. New York: Longman. 207 p.

Brooks, F. P. 1987. No Silver Bullet: Essence and Accidents of Software Engineering. IEEE Computer , 20 (4), pp. 10-19.

Chang, K. E., Sung, Y. T., & Chen, S. F. 2001. Learning through computer-based concept mapping with scaffolding aid. Journal of Computer Assisted Learning , 17, pp. 21-23.

Chappell, D., & Khanderao, K. 2009, April 17. Universal Middleware: What's Happening With OSGi and Why You Should Care. Retrieved May 18, 2009, from SOA World Magazine: http://soa.sys-con.com/node/492519.

Dolmans, D., & Schmidt, H. 1996. The advantages of problem-based curricula. Postgraduate Medical Journal (72), pp. 535-538.

Douglass, B. P. 2004. Real Time UML. Boston: Addison Wesley. 694 p.

Encyclopædia Britannica. 2009. pedagogy. Retrieved July 27, 2009, from Encyclopædia Britannica Online: http://www.britannica.com/EBchecked/topic/448410/pedagogy.

Erl, T. 2008. SOA: Principles of Service Design. Crawfordsville: Prentice Hall. 573 p.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. M. 1995. Design Patterns: Elements of Reusable Object-Oriented Software. Reading (MA): Addison-Wesley. 416 p.

Haikala, I.;& Märijärvi, J. 2006. Ohjelmistotuotanto. In Finnish. Jyväskylä: Talentum. 440 p.

Hall, R. S., Pauls, K., & McCulloh, S. 2009. OSGi in Action. Draft. Manning. 375 p.

Hartley, J. 1998. Learning and Studying. A research perspective. London: Routledge. 179 p.

Hästbacka, D., & Vepsäläinen, T. 2009. Improvements to the UML automation profile. Draft 18.5.2009, Tampere. 37 p.

IEC. 2003, January 21. IEC 61131-3 Ed. 2.0 English. Retrieved July 14, 2009, from IEC Webstore: http://webstore.iec.ch/webstore/webstore.nsf/artnum/029664.

IEEE. 1990, September 28. IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology. Retrieved August 7, 2009, from IEEE Xplore: http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=4148.

Irvine, H. J., Cooper, K., & Jones, G. 2005. Concept mapping to enhance student learning in a financial accounting subject. Proceedings of the Accounting Educators Forum. Sydney: CPA Australia and Charles Sturt University. pp. 1-19.

Kuikka, S. 2008. Notes on Automation Software. Lecture notes. Tampere: Tampere University of Technology. 30 p.

Kuikka, S. 2009. Automaation ohjelmistokomponentit ja sovelluspalvelut. Lecture notes. In Finnish. Tampere: Tampere University of Technology. 178 p.

Kühner, J. 2008. Expert .NET Micro Framework. Berkeley: Apress. 424 p.

Manninen, J., Nevgi, A., Matikainen, J., Luukannel, S., & Porevuo, M. 2000. Osaajien koulutus 2000-luvulla. In Finnish. Retrieved July 27, 2009, from Leonardo da Vinci - programme: http://www.leonardodavinci.fi/publications/valo99/teema1/osaajat.pdf.

Maples, M. F., & Webster, J. M. 1980. "Thorndike's connectionism". Theories of Learning.

Markham, K. M., Mintzes, J. J., & Jones, M. G. 1994. The concept map as a research and evaluation tool: Further evidence of validity. Journal of Research in Science Teaching , 31 (1), pp. 91-101.

Matthews, M. R. 2000. Constructivism in Science and Mathematics Education. National Society for the Study of Education, 99th Yearbook. Chicago: University of Chicago Press. pp. 161-192.

Mayer, R. 2004. Should there be a three-strikes rule against pure discovery learning? American Psychologist, 59 (1), pp. 14–19.

McAffer, J. 2008. Getting Started with OSGi & Equinox. Retrieved August 26, 2009, from DZone Refcardz: http://refcardz.dzone.com/refcardz/equinox?oid=ban00027-0

McAffer, J., Vanderlei, P., & Archer, S. 2009. Equinox and OSGi: The Power Behind Eclipse. Draft. Addison-Wesley Professional.

Metso. 2009. metsoDNA CR - a complete automation platform for better process results. Retrieved September 3, 2009, from Metso: http://www.metso.com/Automation/ip_prod.nsf/WebWID/WTB-070110-2256F-21D48.

Microsoft. 2009. The .NET Micro Framework Platform SDK. Retrieved August 12, 2009, from Microsoft Developer Network: http://msdn.microsoft.com/en-us/library/cc533015.aspx.

Nesbit, J. C., & Adesope, O. O. 2006. Learning With Concept and Knowledge Maps: A Meta-Analysis. Review of Educational Research , 76 (3), pp. 413-448.

Normington, G. 2007, February 12. Is OSGi a SOA? Retrieved May 25, 2009, from Mind the Gap: http://underlap.blogspot.com/2007/02/is-osgi-soa.html.

OMG. 2005, January 2. UML Profile for Schedulability, Performance, and Time, version 1.1. Retrieved June 24, 2009, from OMG: http://www.omg.org/technology/documents/formal/schedulability.htm.

OMG. 2008a, April. Documents associated with UML for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms, v1.1 . Retrieved June 25, 2009, from OMG: http://www.omg.org/spec/QFTP/1.1/.

OMG. 2008b, November. SysML Specifications. Retrieved June 24, 2009, from SysML: http://www.sysml.org/specs.htm.

OMG. 2009, February 4. UML Infrastructure specification Version 2.2. Retrieved August 14, 2009, from OMG: http://www.omg.org/spec/UML/2.2/Infrastructure/PDF/.

OSGi Alliance. 2004, August 17. Listeners Considered Harmful: The "Whiteboard" Pattern Technical Whitepaper. Retrieved August 25, 2009, from OSGi Alliance: http://www.osgi.org/wiki/uploads/Links/whiteboard.pdf.

OSGi Alliance. 2007a, April. OSGi Service Platform Core Specification Release 4, version 4.1. Retrieved April 27, 2009, from OSGi Alliance: http://www.osgi.org/download/r4v41/r4.core.pdf.

OSGi Alliance. 2007b, June 7. OSGi Technical Whitepaper. Retrieved April 27, 2009, from OSGi Alliance: http://www.osgi.org/wiki/uploads/Links/OSGiTechnicalWhitePaper.pdf.

OSGi Alliance. 2009a, March 7. OSGi Service Platform Release 4 Version 4.2 - Early Draft 3. Retrieved September 9, 2009, from OSGi Alliance: http://www.osgi.org/download/osgi-4.2-early-draft3.pdf.

OSGi Alliance. 2009b. About / Members. Retrieved April 27, 2009, from OSGi Alliance: http://www.osgi.org/About/Members.

Papazoglou, M. P. 2003. Service-Oriented Computing: Concepts, Characteristics and Directions. Fourth International Conference on Web Information Systems Engineering. WISE 2003. pp. 3-12.

Rauhamäki, J. 2009a. User guide for the UML Automation Profile. Tampere: Tampere University of Technology. 169 p.

Rauhamäki, J. 2009b. Draft. In Finnish. Guidance for model-driven automation application development. Tampere: Tampere University of Technology.

Ritala, T. 2006. USVA-WP2-Profile_specification. Tampere: USVA project. 73 p.

Ritala, T., & Kuikka, S. 2007. UML Automation Profile: Enhancing the Efficiency of Software Development in the Automation Industry. 5th IEEE International Conference on Industrial Informatics (INDIN 2007). Vienna. pp. 885-890.

Rodriguez, M. A. 2003, November 1. Guidelines for control software organization and development. Retrieved August 7, 2009, from ISA | Living Documents: http://www.isa.org/InTechTemplate.cfm?Section=InTech&template=/ContentManagement/ContentDisplay.cfm&ContentID=31247.

SAS. 2001. Laatu automaatiossa - parhaat käytännöt. In Finnish. Saarijärvi: Suomen Automaatioseura ry. 245 p.

SAS. 2005. Automaatiosovellusten ohjelmistokehitys. In Finnish. Helsinki: Suomen Automaatioseura ry. 152 p.

Schmidt, D. C. 1999, January. Why Software Reuse has Failed and How to Make It Work for You. C++ Report. Retrieved August 31, 2009, from Douglas C. Schmidt: http://www.cse.wustl.edu/~schmidt/reuse-lessons.html.

SJJ Embedded Micro Solutions. 2008. .NET Micro Framework EDK. Retrieved February 26, 2009, from SJJ Embedded Micro Solutions: http://www.sjjmicro.com/EDK.html.

Smith, M. K. 1999. Learning theory - models, product and process. Retrieved July 28, 2009, from the encyclopedia of informal education: http://www.infed.org/biblio/b-learn.htm.

Spencer, J. A., & Jordan, R. K. 1999. Learner centred approaches in medical education. BMJ (318), pp. 1280-1283.

Sugrue, J. 2009, January 12. Equinox Refcard Now Available - An Interview With Jeff McAffer. Retrieved May 28, 2009, from Javalobby: http://java.dzone.com/articles/meet-refcard-author-equinox-wi.

Summerour Robotics Corporation. 2008. Traxster .NET Micro Framework Robot Kit. Retrieved February 26, 2009, from RoboticsConnection.com: http://www.roboticsconnection.com/p-85-traxster-net-micro-framework-robot-kit.aspx

Sun Microsystems. 2009. Sun Java Real-Time System. Retrieved September 2, 2009, from Sun Developer Network (SDN): http://java.sun.com/javase/technologies/realtime/index.jsp.

Tampere University of Technology. 2008. TTY:n Opintojaksokuvaukset 2008-2009. In Finnish. Tampere.

Tiedt, P. 2005, December 13. Building cheat sheets in Eclipse. Retrieved July 22, 2009, from IBM developerWorks: https://www6.software.ibm.com/developerworks/education/os-cheatsheets/os-cheatsheets-a4.pdf.

Tommila, T., Hirvonen, J., Jaakkola, L., Peltoniemi, J., Peltola, J., Sierla, S., & Koskinen, K. 2005. Next generation of industrial automation. VTT Research Notes. Espoo: VTT. 108 p.

Trochim, W. M., Milstein, B., Wood, B. J., Jackson, S., & Pressler, V. 2004. Setting Objectives for Community and Systems Change: An Application of Concept Mapping for Planning a Statewide Health Improvement Initiative. Health Promotion Practice , 5 (1), pp. 8-19.

Uusikylä, K., & Atjonen, P. 2002. Didaktiikan perusteet. In Finnish. Juva: WS Bookwell Oy. 268 p.

Walls, C. 2009. Modular Java: Creating Flexible Applications with OSGi and Spring. Excerpt. The Pragmatic Programmers. 260 p.

Venners, B. 1997, January 8. Java's security architecture. Retrieved August 25, 2009, from JavaWorld.com: http://www.javaworld.com/javaworld/jw-08-1997/jw-08-hood.html.

Vepsäläinen, T. 2008. UML profile tool for automation and control system design. In Finnish. Tampere: Tampere University of Technology. 177 p.

APPENDIX 1: UML AUTOMATION PROFILE EXERCISE

ACI-32020 Automaation reaaliaikajärjestelmät
UML AP -harjoitus, syksy 2009

UML-automaatioprofiiliharjoitus

Aihe:
Tutustuminen UML-automaatioprofiiliin, profiilin kaaviotyyppeihin ja profiilia tukevaan mallinnustyökaluun. UML-automaatioprofiili tuo UML-mallinnukseen automaatiotoimialan käsitteitä ja sen tavoitteena on edistää automaation ohjelmistokehitystä mahdollistamalla toimialakohtaisten käsitteiden hyödyntämisen osana mallinnusta.

Johdanto:
Harjoituksen aiheena on tutustua pintapuolisesti Systeemitekniikan laitoksella kehitettyyn UML-automaatioprofiilin, sen määrittelemiin kaaviotyyppeihin sekä kehitettyyn profiilia tukevaan työkaluun. Koska harjoituksessa ei ehditä käsitellä tarkasti koko profiilia ja sen käsitteiden merkitystä, keskitytään harjoituksessa työkalun käyttöön liittyviin käytännön asioihin siten, että profiilia ja työkalua voidaan hyödyntää harjoitustyön määrittely- ja suunnitteluosuudessa. Harjoitustöiden automaatioprofiiliin liittyviä bonus-osioita varten opiskelijoille jaetaan myös dokumentti, joka opastaa profiilin käsitteiden käyttöä. Pääasia tässä harjoituksessa on, että kunkin kaaviotyypin tekeminen, kaavioiden luominen ja mallin hallinta työkalulla tulee selväksi.

Harjoituksen 1. osassa luodaan uusi projekti ja sille vaatimusmäärittelykaavio.

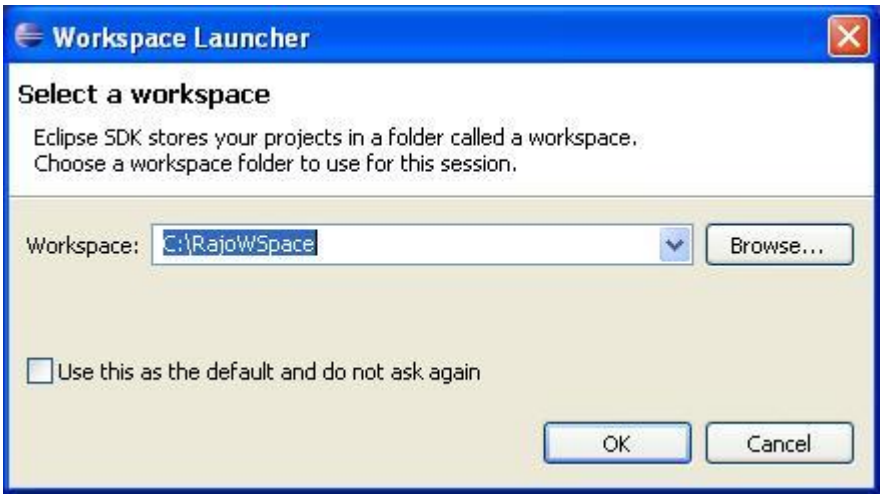
Harjoituksen 2. osassa luodaan säätörakennekaaviot. Säätörakennekaavioita käytetään automaatio-ohjelmiston arkkitehtuurin mallintamiseen.

Harjoituksen 3. osassa luodaan automaatiosekvenssikaavio, joita käytetään sovelluksen toiminnallisuuden mallintamiseen, ja täydennetään vaatimusmäärittely jäljitysrelaatioilla.

Työkalun raskaudesta – ja mm. virtualisoinnista – johtuen työkalu ei välttämättä toimi aina erityisen sulavasti. Työkalun uudelleenkäynnistäminen aika-ajoin saattaa helpottaa ongelmaa.

1. Aloitus ja vaatimusmäärittelykaavio

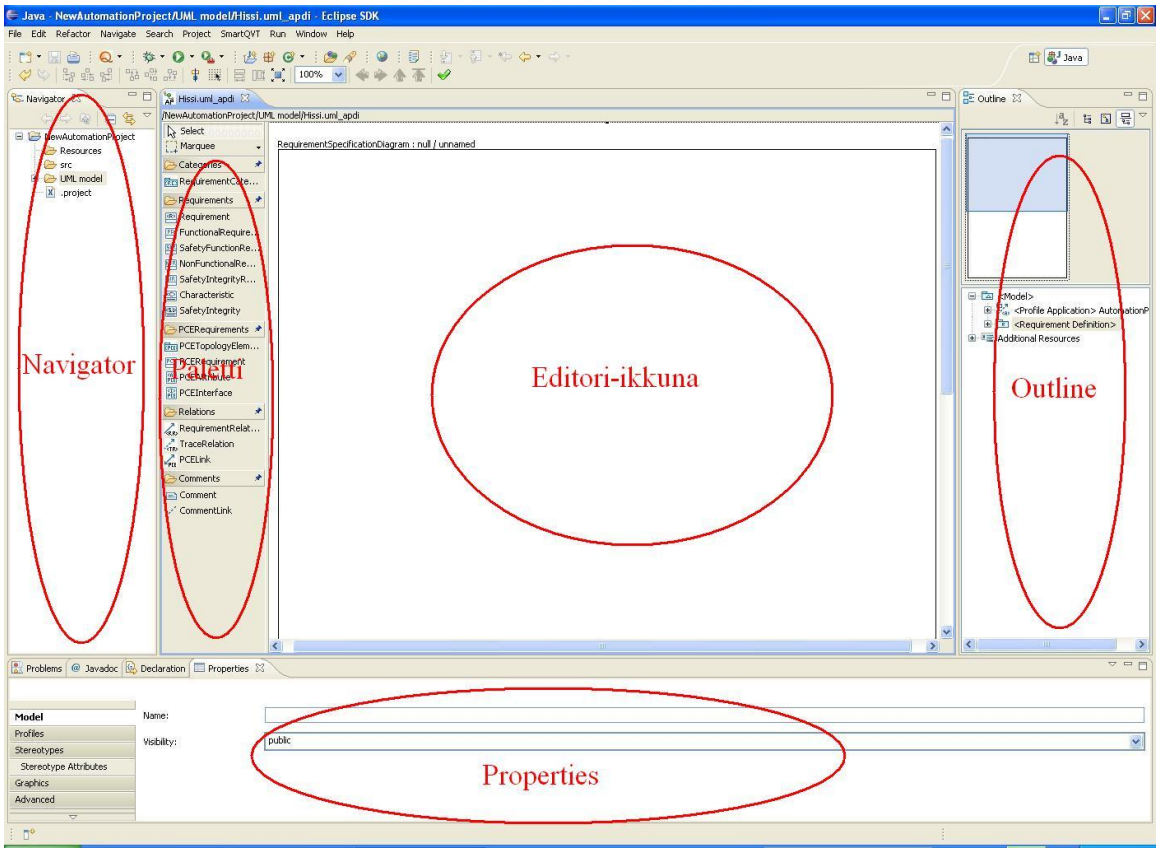
*K*äynnistä *UML AP Tool* työpöydältä. Tällöin aukeaa *Workspace Launcher* -ikkuna jossa valitaan käytetty työtila. Luo ***Browse…***-napilla avatussa ikkunassa My documents:n alle sopiva hakemisto työtilaa varten ja paina ***OK***. Työkalun pääikkuna avautuu.



Kuva 1. *Workspace Launcher -ikkuna.*

Luodaan aluksi mallinnusprojekti: ***File → New → Project…*** Valitse avautuneessa projektin luonti -ikkunassa ***UML AP Projects*** ja edelleen **Automation Modeling Project** ja paina ***Next***. Nimeä projektisi ja paina ***Finish***. Edellinen loi työtilaan uuden projektin, joka sisältää kansiot Resources, src ja UML model. Valitse UML model -kansio ja edelleen: ***File → New → Other… → Topcased → Topcased Diagrams → UML AP model With UML AP Tool*** ja paina ***Next***. (Ole tarkkana, että valitset nimenomaan UML AP model With UML AP Tool!) Anna luotavalle mallille nimi **Hissi** ja valitse templateksi **RequirementSpecificationDiagram**. Klikkaa lopuksi **Finish**. Edellä luotiin uusi malli nimeltä ”Hissi” ja samalla vaatimusmäärittelykaavio, jonka käsittelyllä jatkamme seuraavaksi.

Seuraava kuva 2. esittää työkalu pääikkunaa projektin ja mallin luomisen jälkeen. Erilaiset näkymät ja niiden sijainti on kuitenkin vapaasti käyttäjän valittavissa, muokkaa tällä kertaa pääikkuna vastaamaan kuvan 2 ryhmittelyä. Ei käytössä (ei näkyvissä) olevia näkymiä saa tarvittaessa käyttöön valitsemalla ***Window → Show View → …*** jne.

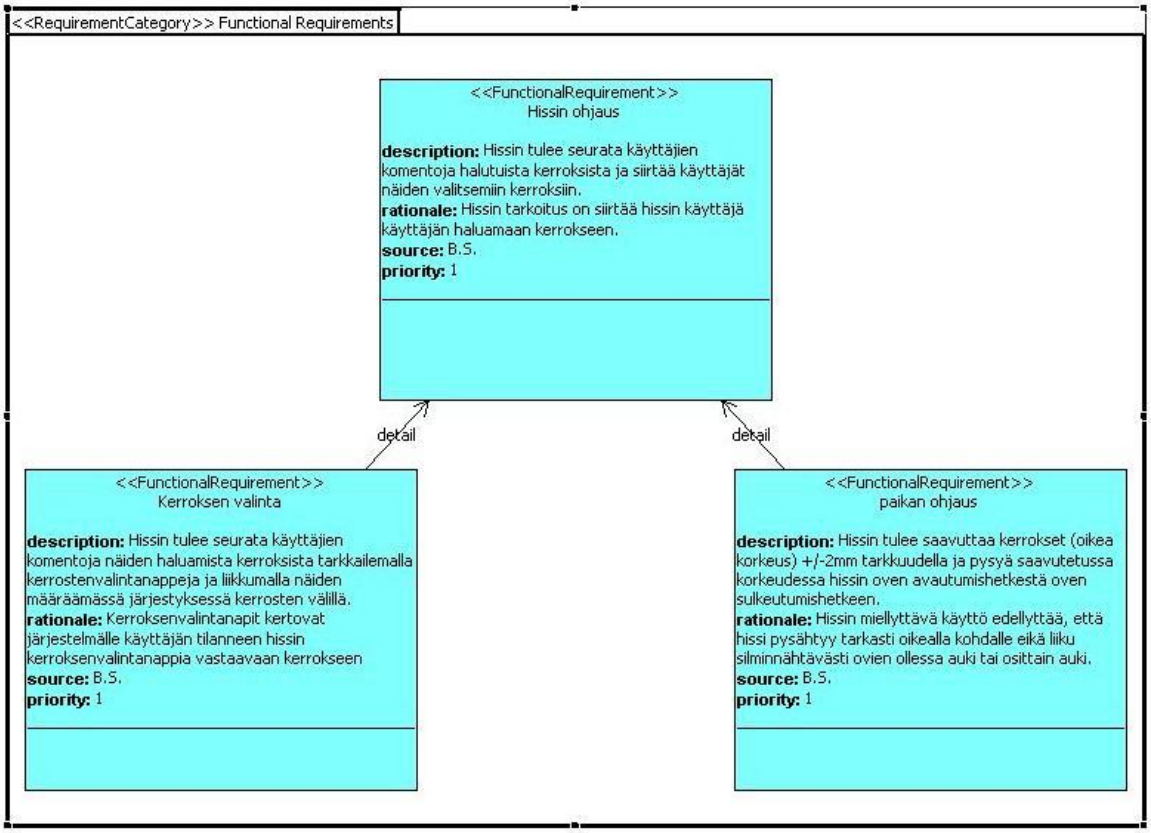


Kuva 2. *UML AP työkalu uuden Automation Modeling Projektin ja mallin luomisen jälkeen. Pääikkuna koostuu näkymistä joissa mallia tarkastellaan eri näkökulmista sekä editoreista joissa mallia muokataan. Kuvan esittämä sijoittelu on ainakin harjoituksen laatijalle luontevin.*

Valitse Outline-näkymästä (esittää mallihierarkian puukaaviona) juurielementti (<Model>) ja anna sille Properties-näkymän kautta nimi ”**Hissi**” – jos näkymä ei ole näkyvissä, saat sen esille edellisessä kappaleessa mainitulla tavalla.

Valitse tämän jälkeen Outline-näkymästä luotu kaavio (Kaaviot esitetään lihavoidulla fontilla ja niihin liittyy -ikoni) ja anna sinne nimi ”**Requirements**”. Mallin elementtien ominaisuuksien muuttaminen on mahdollista Properties-näkymän kautta, kun haluttu elementti on valittu joko Outline-näkymästä tai editori-ikkunasta. Erona on lähinnä se, että editori-ikkunasta valittuna myös elementtien (jotkin) graafiset ominaisuudet ovat muokattavissa. Ominaisuudet on jaettu Properties-näkymässä erilaisiin kategorioihin, kuten Model, Graphics, Advanced ja Automation.

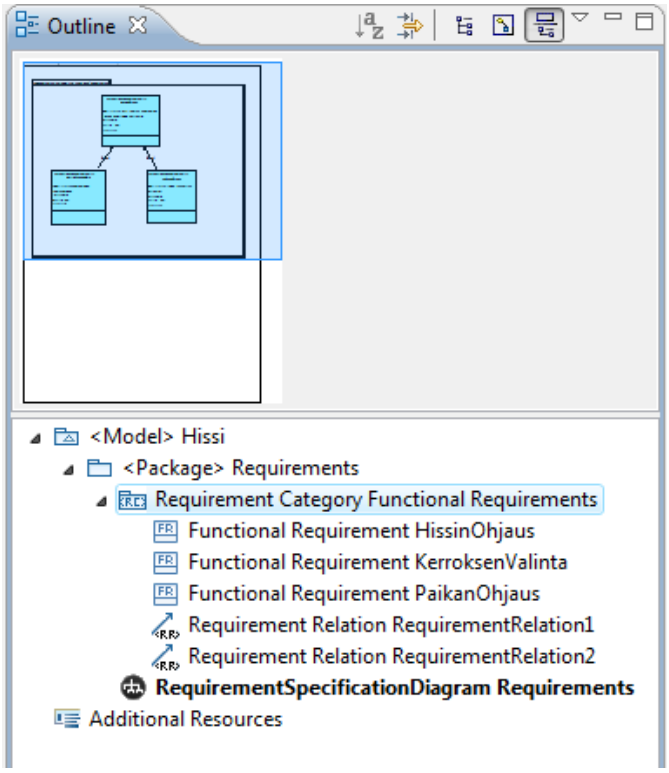
Piirrä seuraavaksi kuvan 3 esittämä vaatimusmäärittelykaavio, joka käsittelee hissin toiminnallisia vaatimuksia. Pohjalla oleva elementti kuvassa on vaatimuskategoria (RequirementCategory), vaatimukset ovat toiminnallisia vaatimuksia (FunctionalRequirement) ja relaatiot vaatimusten välillä vaatimusrelaatioita (RequirementRelation). Elementit ovat lisättävissä editori-ikkunan vasemmalla puolella sijaitsevasta paletista.



Kuva 3. *Vaatimusmäärittelykaavio hissin toiminnallisista vaatimuksista.*

Vaatimusten lisäämisen jälkeen niiden ominaisuudet ovat muokattavissa (Properties-näkymän lisäksi) myös tuplaklikkaamalla vaatimusta ja syöttämällä halutut arvot aukeavan dialogin kenttiin. Aseta vaatimuksille tekstit, jotka esiintyvät myös kuvassa 3.

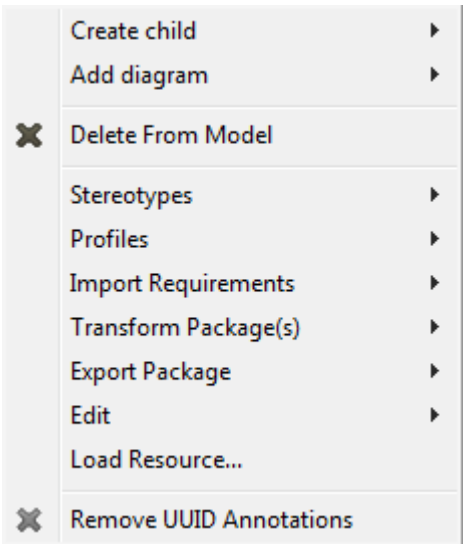
Kaavion ja ominaisuuksien asettamisen jälkeen Outline-näkymän mallin tulisi näyttää suunnilleen samanlaiselta kuin kuvassa 4.



Kuva 4. *Outline-näkymä edellisten vaiheiden suorittamisen jälkeen.*

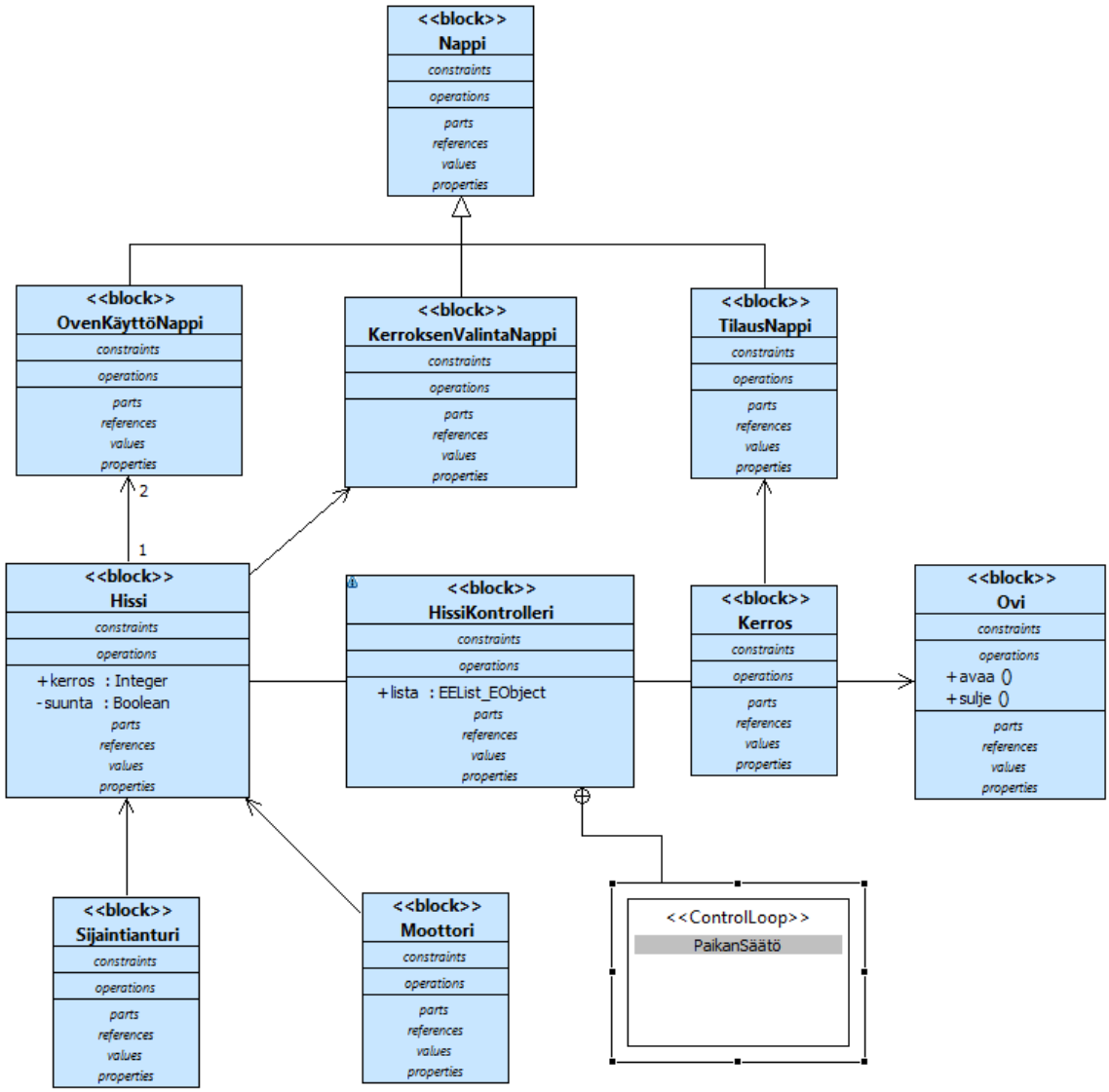
2. Säätörakennekaaviot

Outline-näkymään liittyy myös valikko, jonka avulla voidaan mm. luoda uusia kaavioita ja elementtejä sekä suorittaa erilaisia valitusta elementistä riippuvia toimintoja, joita ei läpikäydä tässä harjoituksessa erityisen tarkasti. Seuraava kuva 5 esittää <Model> Hissi -elementtiin (mallin juurielementti) liittyvän valikon, jonka saa näkyviin valitsemalla Hissin hiirellä ja painamalla tämän jälkeen hiiren oikeaa korvaa.



Kuva 5. *Outline-näkymän <Model> -elementtiin liittyvä valikko.*

Lisää edellä mainittua valikkoa käyttäen <Model> Hissi -elementille uusi pakkaus valitsemalla ***Create Child → Packaged Element | Package***. Anna luodulle pakkaukselle Properties-näkymästä nimeksi vaikkapa ”**HissiBlocks**”. Lisää tämän jälkeen malliin uusi säätörakennekaavio (Control structure diagram) valitsemalla äsken luotu pakkaus Outline-näkymästä ja sen valikosta edelleen: ***Add diagram→ ControlStructureDiagram***. Anna luodulle diagrammille nimeksi vaikkapa ”**Blocks**”. Piirrä tämän jälkeen kuvan 6. mukainen kaavio. Kuva muistuttaa aiemmassa UML-harjoituksessa piirrettyä luokkakaaviota. SysML:n lohkon (Block) voidaankin katsoa olevan eräänlainen toteutukseen (luokka, fyysinen laite tms.) kantaa ottamaton versio UML:n luokasta (Class). Lohkot voivat luokkien tapaan määritellä myös ominaisuuksia ja operaatioita.

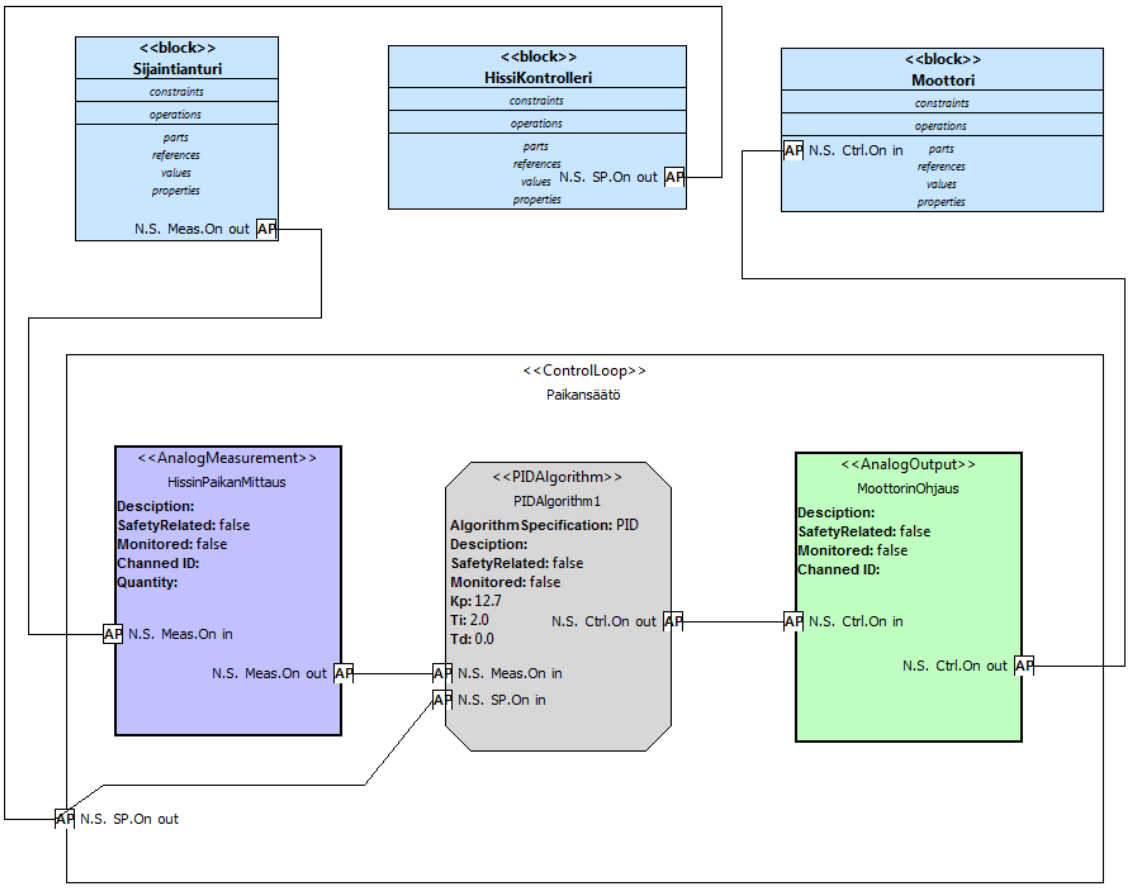


Kuva 6. *Säätörakennekaavio (ControlStructureDiagram) hissiin liittyvistä käsitteistä.*

Lohkojen väliset assosiaatiot löytyvät paletilta nimellä Association ja niiden ominaisuuksia, kuten lukumääräsuhteita ja lukusuuntaa, on mahdollista muokata Properties-näkymän First End ja Second End -kategorioiden kautta. Hissikontrollerin ja Paikansäädön välinen yhteys on Containment, HissiKontrolleri siis sisältää paikansäätöpiirin. Attribuuttien ja metodien lisääminen lohkoille onnistuu paletilta SysMLObjects-kategorian alta.

Aiemmasta harjoituksesta poiketen kaavio sisältää myös Sijaintianturin, Hissin nostamiseen ja laskemiseen käytettävän moottorin sekä Paikansäätö-säätöpiirin (ControlLoop).

Luodaan seuraavaksi varsinainen säätöpiirin (Paikansäätö) määrittely, mutta tehdään se uudessa kaaviossa. Valitse Outline-näkymästä <Package> HissiBlocks ja sen valikosta (avautuu hiiren oikealla korvalla) ***Add diagram→ ControlStructureDiagram***. Anna luodulle uudelle kaaviolle nimeksi esimerkiksi **”Säätöpiiri”** valitsemalla kaavio Outline-näkymästä ja tämän jälkeen Properties-näkymää käyttäen. Valitse tämän jälkeen Outline-näkymästä edelliseen kaavioon luodut <ControlLoop> Paikansäätö, <Block> Moottori, <Block> Sijaintianturi ja <Block> Hissikontrolleri. Useita elementtejä voi valita samaan aikaan pitämällä Ctrl-näppäintä pohjassa. Valittuasi elementit raahaa (hiiren vasenta korvaa pohjassa pitämällä) elementit uuteen kaavioon ja viimeistele säätörakennekaavio vastaamaan kuvan 7. kaaviota.



Kuva 7. *Hissin paikansäätöpiiri säätää paikkaa sijaintimittauksen perusteella hissiä liikuttavaa moottoria toimilaitteena käyttäen.*

Paikansäädön sisällä olevat Hissinpaikanmittaus ja Moottorinohjaus voidaan mieltää ohjelmallisen rajapinnan säätöpiirin antureille ja toimilaitteille tarjoaviksi elementeiksi. Ne ovat siis tavallaan säätöpiirin käyttämiä "ajureita", jotka voisivat toteuttaa mm. *lukituksiin* ja *suojauksiin* liittyviä toimintoja. Hissinpaikansäädin puolestaan on PI-säädin (Td-termi jätetty huomiotta Td: 0). Kuvassa näkyvät portit ovat AutomationFunctionPortteja ja niiden ominaisuuksia, kuten tiedon kulkusuuntaa (in/out/inout) tai tyyppiä (Measurment/SetPoint/Control/jne.) voi muuttaa Automation-välilehdeltä. Porttien väliset kytkennät löytyvät paletilta nimellä Connector.

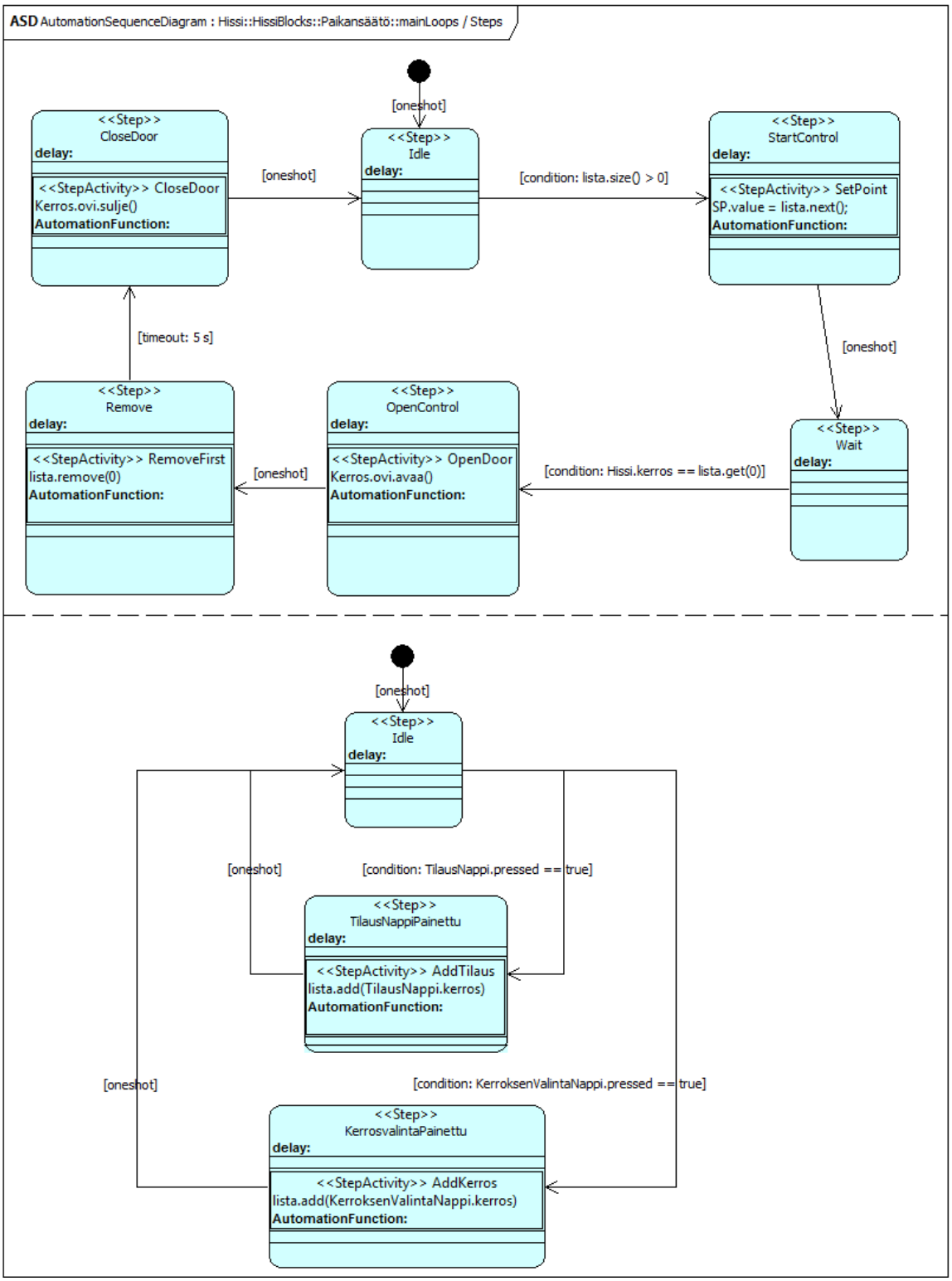
3. Automaatiosekvenssikaavio ja vaatimusmäärittelyn täydentäminen

Luodaan seuraavaksi automaationsekvenssikaavio, jolla hahmotellaan Hissikontrollerin toimintaa. Valitse siis <Block> Hissikontrolleri Outline-näkymästä ja valikosta edelleen: ***Create Child → OwnedBehavior Sequence***. Valitse luotu Sequence ja anna sille nimeksi vaikkapa **"mainLoops"**. Luo tämän jälkeen uusi automaationsekvenssikaavio Outline-näkymän kautta valitsemalla <Sequence> mainLoops ja edelleen valikosta: ***Add Diagram → AutomationSequenceDiagram***. Anna kaaviolle nimeksi vaikkapa **"Steps"**.

Luo seuraavaksi kaavion pohjaksi 2 kappaletta Region-elementtejä ja niille nimiksi "Tapahtumat" ja "Hissinohjaus". Hahmottele tämän jälkeen kuvan 8 kaltainen automaation sekvenssikaavio.

Kuvan 8 sekvenssikaavio on sisältää 2 Regionia, jotka kuvaavat toisistaan erillään eteneviä säikeitä. Pyöristetyt nelikulmiot ovat askelia (Step) ja niissä suoritettavat toiminnot StepActivityjä. Askelien väliset siirtymät ovat Transitioneja, joiden siirtymäehtoja - kuten muidenkin elementtien tärkeitä ominaisuuksia – voi muokata Tuplaklikkaamalla avautuvan dialogin avulla tai Properties-näkymän avulla.

Kuvan yläosan säie siirtyy alussa automaattisesti Idle-askeleeseen (Step), josta siirrytään Start-askeleeseen, jossa hissin paikkaa säätävän säätöpiirin asetusarvoksi kirjoitetaan (kohde)listalla seuraavana oleva kerros. Hissin saavuttua kerrokseen avataan ovi, poistetaan kerros listalta, odotetaan 5s, suljetaan ovi ja siirrytään Idle-askeleeseen, josta kierto alkaa uudestaan heti, kun listalla on uusia kerroksia. Alempi säie lisää listalle kerroksia, kun käyttäjän painavat kerroksenvalintanappeja tai tilausnappeja. *(Todellisuudessa hissin toimintalogiikka on varmasti monimutkaisempi, mutta tämä esimerkki riittänee kaaviotyyppiin tutustumiseen.)*



Kuva 8. *Automaation sekvenssikaavio hissin ohjauksesta.*

Täydennetään vielä aluksi luomaamme vaatimusmäärittelyä. Etsi Outline-näkymästä <Package> Requirements ja sen alta lihavoidulla tekstillä oleva RequirementSpecificationDiagram, jota klikkaamalla kaavion pitäisi avautua. Lisää Kerroksen valinta- ja paikan ohjaus -vaatimuksille TraceRelation-jäljitysrelaatiot. Siirry tämän jälkeen Properties-näkymässä Automation-kategoriaan ja aseta Kerroksen valinta

-vaatimuksen jäljitysrelaation ModelElement-ominaisuuden arvoksi ”**Region Tapahtumat**” ja Paikan ohjaus -vaatimuksen jäljitysrelaation ModelElement-ominaisuuden arvoksi **”ControlLoop Paikansäätö**”. Kokeile tämän jälkeen tuplaklikata TraceRelation-elementtejä kaaviossa. Tuplaklikkauksen pitäisi siirtää editori kaavioon, jossa ModelElement-viitteen viittaama elementti esiintyy, tai avata dialogi kaavion valitsemiseksi, mikäli elementti esiintyy useassa kaaviossa. (Paluu Vaatimusmäärittelykaavioon toiseen kaavioon siirtymisen jälkeen tapahtuu kätevimmin editori-ikkunan yläpuolella olevaa vihreää, vasemmalle osoittavaa Go to the previous diagram -nuolta klikkaamalla.)

Lopuksi: Työkalu mahdollistaa myös yleisimpien UML-kaavioiden, kuten käyttötapauskaavioiden, luokkakaavioiden, sekvenssikaavioiden, tilakaavioiden jne. piirtämisen. Verrattuna joihinkin muihinkin UML-työkaluihin, käyttämämme UML AP -työkalu on kuitenkin varsin tarkka mallin ”muodosta”.

Erityisesti eri kaaviotyyppejä voidaan luoda vain tietyn tyyppisten elementtien omistettavaksi. Esimerkiksi käyttötapauskaavion (Use case diagram) juurielementtinä (=> kaavio luotavissa Outline-näkymän valikon Add diagram -alavalikosta) voivat toimia tyypiltään Package (UML::Package) tai siitä periytyvän tyypin elementit, kuten Model (<Model> Outline-näkymässä) tai BlockDefinition. Erilaiset pakkaukset (tai pakkauksesta periytyvät tyypit) voivat toimia myös esimerkiksi luokkakaavioiden juurielementteinä.

Toimintaa tai käyttäytymistä kuvaavien kaavioiden juurielementit ovat suurimmalle osalle esimerkiksi luokkaa tai pakkausta tuntemattomampia tyyppejä, kuten Activity, Interaction ja Sequence. Näillä kaavioilla kuitenkin tyypillisesti kuvataan esimerkiksi luokan (Class) tai siitä periytyvän tyypin (kuten vaikkapa Block) sisäistä toimintaa tai vaikkapa pakkauksen luokkien yhteistoimintaa. Pakkauksille juurielementtien luominen on automatisoitu siten, että automaation sekvenssi, sekvenssi- ja tilakonekaavioiden luominen on mahdollista suoraan Add diagram -alavalikosta.

Kuitenkin esimerkiksi ennen luokan toimintaa kuvaavan kaavion luomista tulee luoda juurielementti kaaviota varten. Esimerkiksi haluttaessa kuvata luokan toimintaa tilakaaviolla valitaan ensin haluttu luokka, luodaan sen omistukseen StateMachine (Outline-näkymän valikosta ***Create Child → OwnedBehavior StateMachine***), valitaan luotu StateMachine ja lopulta luodaan tämän omistukseen uusi tilakaavio. Create Child -alavalikosta, joka on käytettävissä mm. luokan (Class) tai lohkon (Block) ollessa valittuna, löytyvät juurielementit esimerkiksi tilakaavioiden (juurielementti StateMachine tai Sequence), automaation sekvenssikaavioiden (juurielementti Sequence), sekvenssikaavioiden (juurielementti Interaction) tai aktiviteettikaavioiden (juurielementti Activity) luomiseen. Mikäli aikaa on vielä jäljellä, voit kokeilla joitakin näistä toiminnoista.

APPENDIX 2: .NET MICRO FRAMEWORK ASSIGNMENT

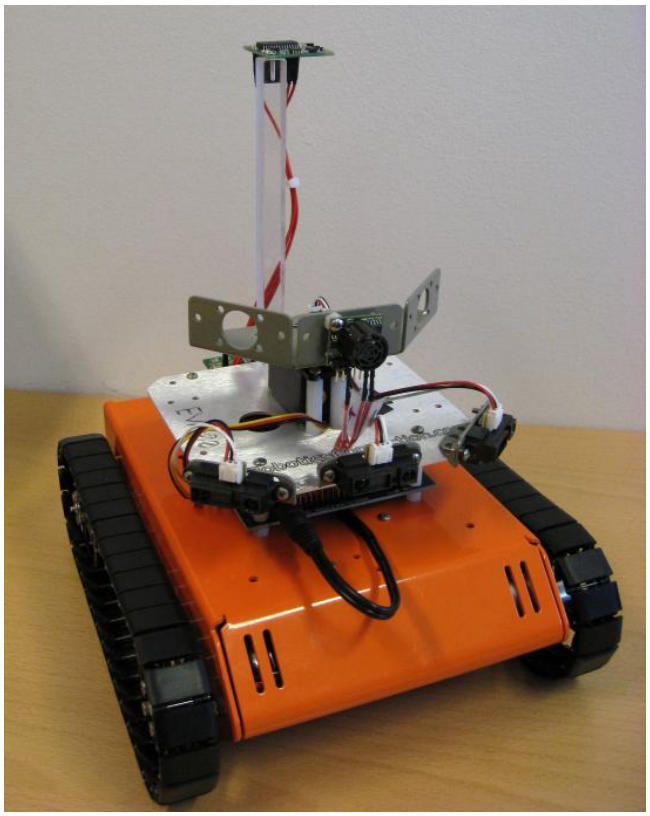
ACI-32020 Automaation reaaliaikajärjestelmät **syksy 2009**
Harjoitustyö 5

ROBOTIN OHJAUS

Taustaa

Traxster II on RoboticsConnectionin valmistama alusta robottien rakentamista varten. Runkoon on kiinnitetty erilaisia sensoreita, joilla robotti voi havainnoida ympäristöään. Robotin ohjaaminen tapahtuu EMACin valmistamalla iPac 9302 -sulautetulla tietokoneella, jossa suoritetaan .NET Micro Framework TinyCLR (Common Language Runtime) -tulkkia. Laiteohjaukset lähetetään iPac-korttiin sarjaportin kautta liitetylle Serializer Robot Controller -ohjainkortille, johon sensorit, moottorit ja muut lisälaitteet on kytketty. Serializer hoitaa matalan tason IO:t ja takaa telaketjujen moottoreille riittävän jännitesyötön.

.NET Micro Framework on Microsoftin sulautettujen ohjelmistojen kehittämisen tarkoitettu sovelluskehys. Perinteisestä sulautetusta sovelluskehityksestä poiketen ohjelmointi tehdään ilmaisuvoimaisella C#-kielellä .NET-alustan ominaisuuksia hyödyntäen. Micro Framework alusta vaatii vain muutamia satoja kilotavuja muistia ja toimii useilla ARM7 ja ARM9 suorittimilla. Ohjelmoijan käytössä on osa .NETin tarjoamista kirjastoista ja ohjelmat voidaan kehittää Visual Studio -ohjelmankehitysympäristöllä, sen tarjoamia työkaluja käyttäen. Valmiit ohjelmat ladataan iPac-kortille sarjaportin kautta.



Kuva 1. *Traxster II robotti.*

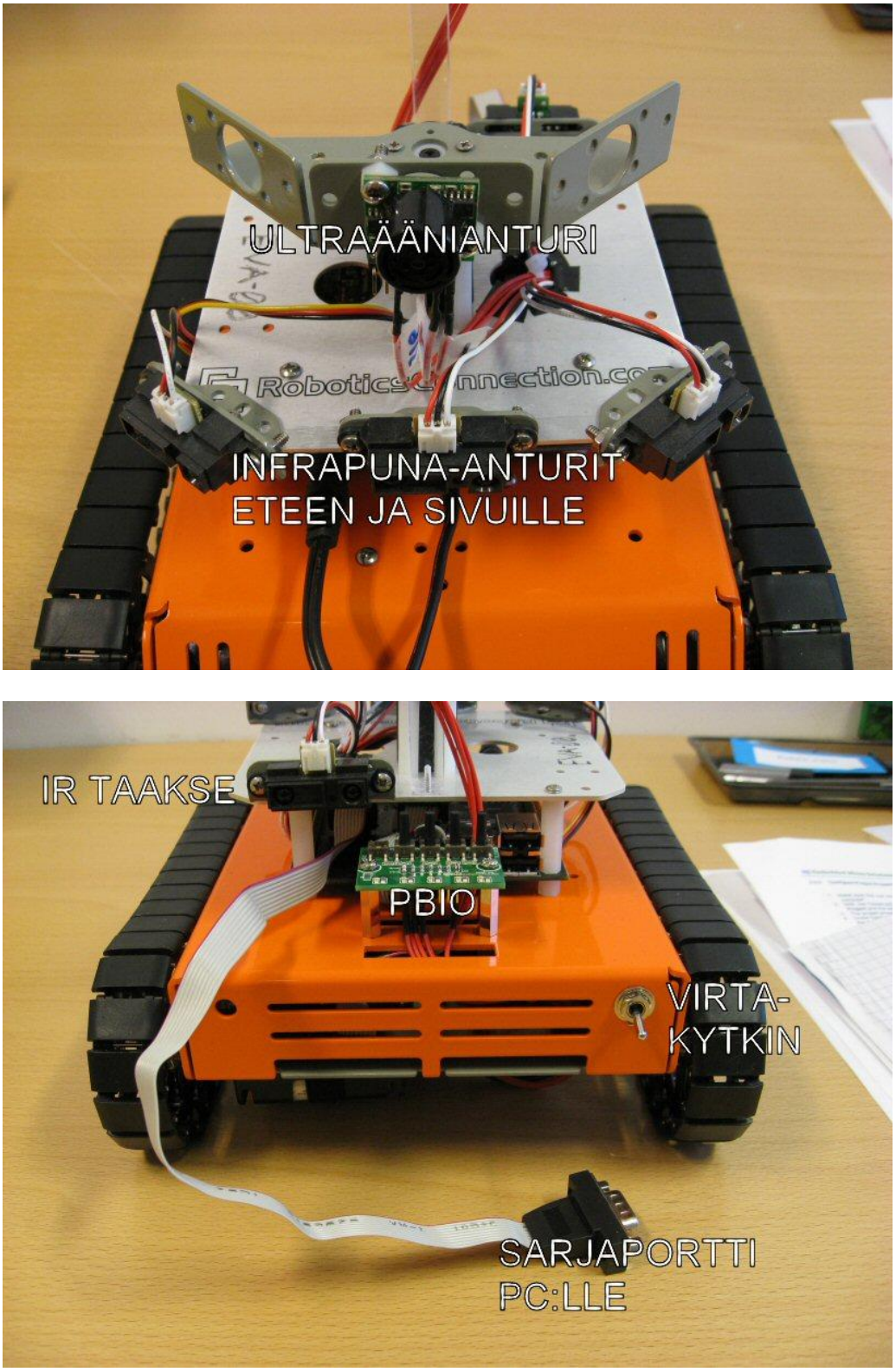
Yleistä

Harjoitustyön aiheena on toteuttaa ohjaussovellus itsenäisesti liikkuvalle robotille. Sovelluskehitys tehdään .NET Micro Framework -alustalle C#-kielellä Microsoft Visual Studio -kehitysympäristössä. Harjoitustyöhön kuuluu toteutuksen lisäksi sovelluksen määrittely, suunnittelu ja testaaminen sekä eri vaiheiden asianmukainen dokumentointi. Harjoitustyössä hyödynnetään soveltuvin osin myös UML Automaatioprofiilia sovelluksen rakenteen ja toiminnallisuuden mallintamisessa.

Robotissa on seuraavat anturit ja toimilaitteet:

- 2x 7,2 V moottoria telaketjujen ohjaamiseen
- Push Button I/O board (PBIO), sisältää myös LEDejä ja summerin. Ainoastaan yksi painonappula ja summeri on kytketty kiinni.
- 180° kääntyvä servo
- Maxbotix MaxSonar EZ1 ultraäänietäisyysanturi, kantama 6-254” (alle 6” päässä olevat esteet näkyvät 6”:ssa), asennettu servon päälle. Ultraäänianturin keilakuvio on laaja, ja sopii hyvin esteiden tunnistamiseen pidemmältä matkalta.
- 4x Sharp GP2D12 infrapunaetäisyysanturia, kantama 3,9”-31,5”. Kolme anturia on suunnattu kiinteästi etusektorille ja yksi taaksepäin. Infrapuna-anturien keila on kapea, joten ne sopivat esteiden tarkkaan havainnointiin lähellä. Minimikantaman sisällä olevat esineet näyttävät olevan yli 3,9” etäisyydellä, joten liian lähelle esteitä ei robottia kannata ajaa.
- Devantech CMPS03 sähköinen kompassi - ei kerro missä päin pohjoinen on, tai edes missä päin magneettinen pohjoisnapa on. Sen sijaan se kertoo vallitsevan magneettivuon horisontaalisen komponentin suunnan, mikä voi olla Maan magneettikenttä, mutta myös mikä tahansa muu voimakas magneettikenttä.

Kompassin avulla saatavaa kulkusuunnan muutosta voidaan kuitenkin hyödyntää esimerkiksi ohjauksen takaisinkytkentä



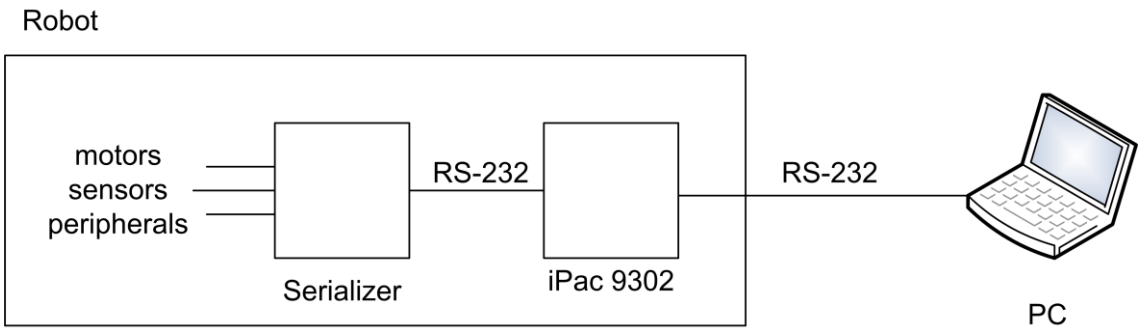
Kuvat 2 ja 3. *Lähikuvat edestä ja takaa.*

Työn suorittaminen

Robotille kirjoitetaan navigointiohjelma, jolla se pystyy liikkumaan itsenäisesti suljetussa tilassa törmäämättä esteisiin. Tila voidaan olettaa staattiseksi, eli siinä ei tapahdu äkillisiä muutoksia joihin pitäisi varautua (juuri ajetulle polulle ei ilmesty tyhjästä esteitä).

Robottia ohjataan RoboticsConnectionin kehittämän SerializerLibMF-rajapinnan kautta, jonka avulla Ipaq-kortti osaa keskustella matalan tason ohjaukset hoitavan Serializer-kortin kanssa. Rajapinnassa on mm. funktiot moottorien ja servojen ohjaamiseen ja anturien lukemiseen. Käytöstä on tarkemmat ohjeet vinkkidokumentissa.

Omaa ohjelmaa voi testata ohjelmallisesti toteutetun robottiemulaattorin kanssa, joka tarjoaa lähes saman rajapinnan kuin oikeakin robotti, mutta ei sisällä monimutkaista toiminnallisuutta. Lopullinen testaus ja palautus tapahtuvat oikealla robotilla, assistentin valvovan silmän alla.



Kuva 4. *Yksinkertainen kaavio robotista.*

Tarkennetut harjoitustyön vaatimukset

- Robotin käynnistyttyä navigointi voidaan kytkeä päälle/pois nappia painamalla. Oletuksena navigoinnin täytyy olla pois päältä, eli robotti ei lähde liikkeelle omia aikojaan.
- Navigointi voi tapahtua esim. niin että robotti pyrkii etenemään tiettyyn suuntaan, tai se voi vain pyrkiä pääsemään esteiden ohi. Navigointiratkaisun suunnittelu ja toteutus on opiskelijoiden vastuulla.
- Etenemiseen tulee olla toteutettu (ainakin) kaksi moodia, joiden välillä robotti osaa vaihtaa automaattisesti:
    - ”Täysillä eteen” – robotti päättelee antureilta saatavan palautteen perusteella, että reitti eteenpäin on selvä ja etenee reipasta vauhtia, tehden korkeintaan pieniä korjauksia etenemissuuntaan.
    - ”Esteiden väistö” – robotti huomaa lähellään esteen, jonka ohi se yrittää navigoida. Koska robotti ei erota esim. seinää tuolista, on ohjelman selviydyttävä hyvinkin erilaisista tilanteista.
- Robotti ei saa jäädä jumiin kohdatessaan esteen, sen on joko palattava takaisin tai muuten yritettävä kiertää esteet.
- Robotille täytyy suunnitella toiminnallisuus, jos se huomaa että sillä ei ole tilaa liikkua mihinkään suuntaan.

Bonustehtävän suorittaminen

Bonustehtävistä on mahdollista saada harjoitustyöhön maksimissaan yhden arvosanan korotus (0,5 arvosanan verran tehtävää kohden). Päätös bonustehtävien tekemisestä on tehtävä ennen harjoitustyön aloittamista, sillä tehtävillä on vaikutusta suunnitteluun eikä niitä voi lisätä jälkikäteen. Harjoitustyön bonustehtävät ovat:

1. **UML-automaatioprofiili osana suunnittelua**
2. **Robotin ohjaus PI(D)-säädöllä**

UML automaatioprofiilin hyödyntäminen suunnittelussa

Mallinnus sisältää erityisesti sovelluksen toiminnallisten ja ei-toiminnallisten vaatimusten mallintamisen ja johtamisen suunniteltuihin ohjelmiston osiin, joiden mallinnukseen tulee käyttää UML:n omien käsitteiden ja kaaviotyyppien lisäksi myös automaatioprofiiliin kuuluvia käsitteitä ja kaaviotyyppejä, kuten automaation sekvenssi- ja säätörakennekaavioita. Kaaviot liitetään muiden UML-kaavioiden tapaan suunnitteludokumenttiin, jossa kaavioita käsitellään sanallisesti normaaliin tapaan. (Kaaviot, myös UML AP -kaaviot, vaativat yleensä selitystä tuekseen. Toisaalta järkevällä selityksellä voi paikata kaavioissa mahdollisesti olevia notaatiovirheitä.)

Erityisesti harjoitustyön ensimmäisessä vaiheessa, jolloin mallinnetaan sovelluksen vaatimuksia ja käyttötapauksia, laaditaan käyttötapauskaavioiden lisäksi vaatimusmäärittelykaavio(ita), johon kootaan sovelluksen tärkeimmät toiminnalliset ja ei-toiminnalliset vaatimukset. Harjoitustyöstä riippuen noin ~10 vaatimusta lienee sopiva määrä. (Vaatimusten muodostamisessa kannattanee hyödyntää tätä dokumenttia.)

Harjoitustyön toisessa ja kolmannessa vaiheessa, joissa suoritetaan olioluokka-analyysi ja viimeistellään (tarkennetaan) suunnitelma ennen ohjelmointia, käytetään normaalien UML-kaavioiden lisäksi säätörakenne- ja automaation sekvenssikaavioita. Säätörakennekaavioita käytetään erityisesti säätösilmukoiden mallintamiseen. Automaation sekvenssikaaviot soveltuvat tilakaavioiden tapaan erilaisen sekventiaalisen toiminnan, kuten panosautomaation sekvenssien, kuvaamiseen. Tähän harjoitustyöhön varsinaisia panossekvenssejä ei kuulu, mutta automaation sekvenssikaavioilla voidaan korvata esimerkiksi tilakaavioita sovelluksen tilamuutoksista tai toiminnasta suoritettaessa sovellusta eri moodeissa. Kolmannessa vaiheessa myös (viimeistään) johdetaan alussa luodut vaatimukset suunniteltuihin rakenteellisiin elementteihin tai esimerkiksi sekvensseihin, jotka vastaavat vaatimusten toteutumisesta.

Vaatimusten johtaminen (TraceRelation) suunniteltuihin ohjelmiston osiin edellyttää luonnollisesti, että sekä vaatimukset että muu suunnittelu toteutetaan samaan UML AP -työkalulla kehitettävään malliin. Tästä johtuen suunnittelu kannattanee tehdä kokonaisuudessaan UML AP -työkalua käyttäen. Työkalun käyttöön on tutustuttu myös harjoituksessa, jonka kertaaminen saattaa auttaa mahdollisissa ongelmatilanteissa.

Automaatioprofiilin käsitteistä, joita tarvitaan harjoitustöiden mallintamisessa, ja niiden käytöstä on myös dokumentti, joka on saatavissa kurssin kotisivulta.

Robotin ohjaus PI(D)-säädöllä

Robotin ohjaus ja liikkuminen toteutetaan säätimillä jotta robotin liikehdintä olisi mahdollisimman sulavaa. Käytännössä tämä tarkoittaa sekä kääntymiskulman että ajonopeuden säätöä, mutta koska kyseessä ei ole säätötekniikkaan keskittyvä kurssi, ei optimaalisten viritysparametrien etsinnästä saa lisäpisteitä, eikä se emulaattorilla edes olisi mahdollista. Säädöissä on tarkoituksena hyödyntää etäisyysantureilta saatavia tietoja sekä laitteiston muita mittalaitteita (kompassi) takaisinkytkentöihin.

APPENDIX 3: OSGI EDUCATIONAL MATERIAL

OSGi

OSGi Service Platform on OSGi Alliancen kehittämä ja ylläpitämä avoin spesifikaatio Java-pohjaiselle dynaamiselle moduulijärjestelmälle. Kun OSGi Alliance muodostettiin vuonna 1999, alkuperäisenä tarkoituksena oli kehittää kotikäyttöön tarkoitettuihin reitittimiin järjestelmä, jolla useiden eri valmistajien ohjelmat ja teknologiat saataisiin toimimaan yhdessä. Sittemmin OSGin monikäyttöisyys on huomattu laajemminkin ja nykyisin se on käytössä esim. älypuhelimissa ja autojen telematiikkasovelluksissa. Lisäksi suositun kehitysympäristön Eclipsen laajennusmekanismi on nykyisin OSGi-pohjainen. OSGi ei nimenä tarkoita mitään, mutta tulee OSGi Alliancen alkuperäisestä nimestä Open Service Gateway initiative. Alliancen nykyisiä täysjäseniä ovat mm. Deutsche Telekom, Motorola, Nokia, Red Hat, Samsung Electronics, SAP AG, Siemens ja SpringSource. Tällä hetkellä spesifikaation uusin versio 4.1 on vuodelta 2007.

OSGi Alliance vastaa ainoastaan spesifikaation ylläpitämisestä. OSGi-kehyksestä (OSGi framework) onkin tarjolla useita käytännön toteutuksia, joista tunnetuin lienee Eclipsen mukana tuleva *Equinox,* joka on myös uusimman OSGi-spesifikaation referenssitoteutus. Equinox on kohtuullisen työpöytäorientoitunut ja käytössä mm. Lotus Notesissa ja IBM:n WebSphere-sovelluspalvelimessa. Vaikka mikään ei estäkään käyttämästä sitä sulautetuissa laitteissa, tähän käyttöön on kehitetty myös resursseiltaan rajoitetuille ympäristöille optimoituja pienikokoisia toteutuksia, kuten Apachen kehittämä Felix. Suurin osa toteutuksista on saatavilla erilaisten avoimen lähdekoodin lisenssien alla, mutta kaupallisiakin toteutuksia on saatavilla, esim. Concierge.

Mikä on suurin haaste missä tahansa suuressa rakennusprojektissa, kuten pilvenpiirtäjän tai suihkukoneen rakentamisessa? Vastaus on monimutkaisuus. Boeing 747-400:ssa on kuusi miljoonaa osaa, 274 km johtoja ja 8 km putkia. Konetta varten laadittiin yli 75000 piirustusta. Yksikään ihminen ei pysty ymmärtämään niin ison kokonaisuuden toimintaa täydellisesti, mutta silti uudet lentokoneet ovat vieläkin monimutkaisempia. Ainut keino suunnitella näin isoja koneita, on pilkkoa ne pienemmiksi, ymmärrettävän kokoisiksi moduuleiksi. Modulaarisuudella on useita etuja; se mahdollistaa mm. työnjaon, jolloin laskutelineen suunnittelijan ei tarvitse tietää mitään koneen viihdejärjestelmistä, abstraktiotason noston, komponenttien uudelleenkäytön muissakin koneissa ja se helpottaa huoltoa ja korjaustöitä, kun hajonneen moduulin voi korjata tai vaihtaa.

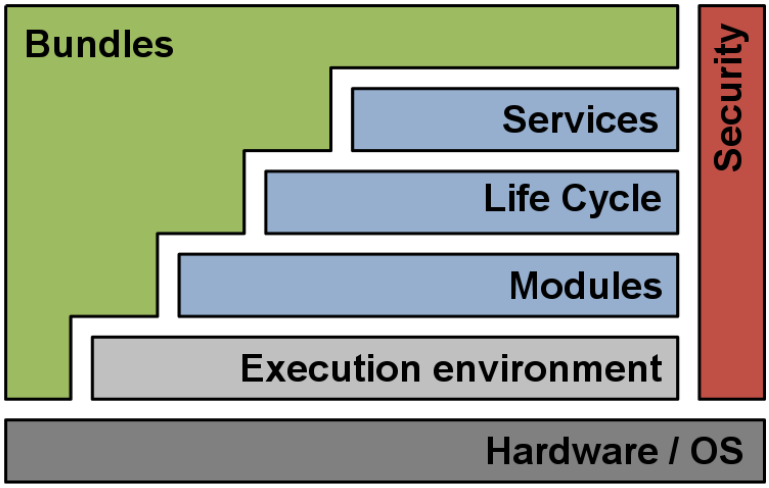
Ohjelmistotuotannossa modulaarisuudella tarkoitetaan ohjelman jakamista erillisiin osiin, eli moduuleihin. Eroa komponenttien ja moduulien välillä ei yleensä tehdä – termejä käytetään usein ristiin – mutta moduulin voidaan ajatella olevan hieman komponenttia väljemmin määritelty käsite. Kuten komponenttikin, moduuli on itsenäinen, löyhästi kytketty (muiden komponenttien sisäisestä toteutuksesta riippumaton) ja loogisesti yhtenäinen kokonaisuus.

Java on laajalle levinnyt ohjelmointikieli, mutta sen modulaarisuutta tukevat ominaisuudet ovat rajoittuneet. Javan jakeluyksikkönä toimii JAR-tiedosto, johon on pakattu käännetyt luokkatiedostot ja muut mahdolliset resurssit, kuten kuvat. Usein yhdessä JAR-tiedostossa on yksittäinen kirjasto tai jokin kokonaisuus sovelluksen toiminnallisuudesta, jolloin koko sovellus koostuu useista JAR-paketeista. Sovelluksen koon kasvaessa myös JAR-tiedostojen lukumäärä kasvaa, mutta Javasta puuttuvat kunnolliset työkalut niiden hallintaan.

Ongelmat johtuvat sopivien hallinnollisten työkalujen puuttumisen lisäksi mm. siitä että JAR-tiedostot ovat vain jakeluyksikkö, eikä niillä ole ajonaikaista vastiketta sen jälkeen kun ne on saatu ladattua Javan luokkapolkuun (Classpathiin). Tämä voi johtaa tilanteeseen, jossa samannimisestä luokasta löytyy useita versioita luokkapolusta. Luokkalataaja (Class loader) lataa aina ensimmäiseksi löytämänsä luokan, mikä voi aiheuttaa hankalasti selvitettäviä virhetilanteita, jos samasta luokasta on tarjolla useita versioita, tai luokkapolkuun on päätynyt vanhentunut toteutus luokasta jostain toisesta JARista.

Lisäksi Javassa tiedon piilottaminen onnistuu vain luokkien ja pakkauksien (*package*, eli Javan mekanismi nimiavaruuksien toteuttamiseksi) tasolla, vaikka Javan jakeluyksikkö onkin JAR-tiedosto. Koska vähänkään laajemmassa JARissa on tavanomaisesti useita pakkauksia, täytyy luokan määrittelyn olla *public*, jos sen halutaan olevan käytettävissä oman pakkauksensa ulkopuolella. Mutta tämän seurauksena se näkyy myös JARin ulkopuolelle, mikä johtaa helposti siihen että koko JAR muuttuu julkiseksi rajapinnaksi. Muita JAReihin liittyviä heikkouksia ovat mm. versioinnin ja riippuvuuksien ilmaisun puuttuminen. OSGi korjaa nämä Javan puutteet lisäämällä siihen modulaarisuuskerroksen.

OSGin perusajatus on yksinkertainen; jokaisella moduulilla – joita OSGissa kutsutaan *bundleiksi* – on oma luokkapolkunsa, joka poistaa monia Javan luokkapolun yksitasoisuudesta johtuvia ongelmista. Luokkien jakaminen bundle-komponenttien välillä tapahtuu hallitusti OSGi-kehyksen kautta. OSGi-kehys on keskeisin osa OSGin spesifikaatiota. Kehys jakautuu arkkitehtuurikuvassa sinisellä merkittyihin kolmeen osaan tai kerrokseen, jotka ovat moduulikerros, elinkaarikerros ja palvelukerros, joista jokainen tuo oman osansa OSGin toiminnallisuuteen. Näiden kanssa rinnakkaisena on punaisella merkitty turvallisuuskerros, joka laajentaa Javan omaa turvallisuusmallia. Lisäksi arkkitehtuurikuvassa on ajoympäristö, joka voi olla lähes mikä tahansa Java 2:n konfiguraatio tai profiili, esim. normaali J2SE tai mobiililaitteille tarkoitettu MIDP.



Kuva 3. *OSGin arkkitehtuuri (OSGi Service Platform Core Specification v4.1).*

OSGin moduulikerroksen määrittelemä yksikkö modulaarisuudelle on bundle. Suoritusympäristön, eli Javan, päällä oleva moduulikerros huolehtii bundle-komponenttien luokkien lataamisesta erillisiin luokkapolkuihin. Jos bundle haluaa tarjota pakkauksen muiden bundlejen käytettäväksi, se täytyy erikseen exportoida. Exportoidut pakkaukset muodostavat bundlen ohjelmointirajapinnan (application programming interface, API). Vastaavasti pakkaus täytyy importoida, että se saadaan toisessa bundlessa käyttöön.

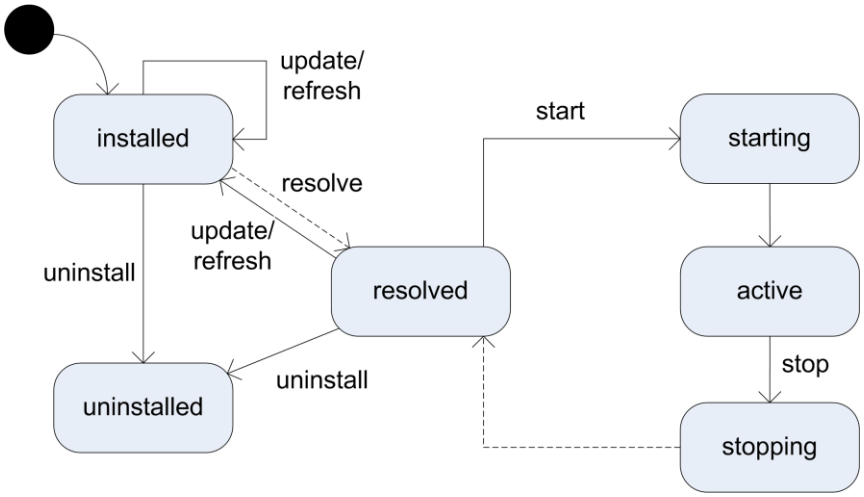
Bundle-komponentit ovat tavanomaisia JAR-tiedostoja, joihin on liitetty normaalin ohjelmakoodin lisäksi metatietoa bundlen sisällöstä ja riippuvuuksista. Tavallisesti bundlen sisällä oleva koodi on organisoitu Javan pakkauksiin, aivan kuten normaaleissakin JAReissa. Metatieto löytyy bundlen sisällä kulkevasta tiedostosta MANIFEST.MF:

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Logic
Bundle-SymbolicName: fi.tut.ase.calculator.logic
Bundle-Version: 1.0.0.qualifier
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Export-Package: fi.tut.ase.calculator.logic
Import-Package: org.osgi.framework;version="1.5.0"
Bundle-Activator: fi.tut.ase.calculator.logic.internal.bundle.Activator
```

Malli-bundlemme metadatasta löytyy nimen lisäksi komponentin versio ja vaadittu ajoympäristö. Export-Package -rivillä julistetut pakkaukset muodostavat bundlen API:n, joka tässä tapauksessa koostuu pelkästään pakkauksesta `fi.tut.ase.calculator.logic`. Import-Package -rivi kertoo että toimiakseen bundle vaatii OSGi-kehyksen perustoimintoja, jotka nekin löytyvät omasta bundlestaan `org.osgi.framework`. Lisäksi riippuvuuksia voisi esittää tarkemmin Require-Bundle -rivillä, joka määrittelee vaadittavan pakkauksen nimen sijaan bundlen nimen. Equinox-

toteutuksen ja Eclipsen kanssa työskennellessä Manifest-tiedostoa ei tavallisesti tarvitse muokata käsin, sillä Eclipse tarjoaa siihen valmiit työkalut.

Bundle-komponenttien lataamisesta huolehtii kehyksen moduulikerros, mutta niiden elinkaaresta on vastuussa elinkaarikerros, joka mahdollistaa bundlejen käynnistämisen tai sammuttamisen dynaamisesti, eli ajonaikaisesti. Bundlen elinkaaren mahdolliset tilat näkyvät alla olevassa kuvassa.



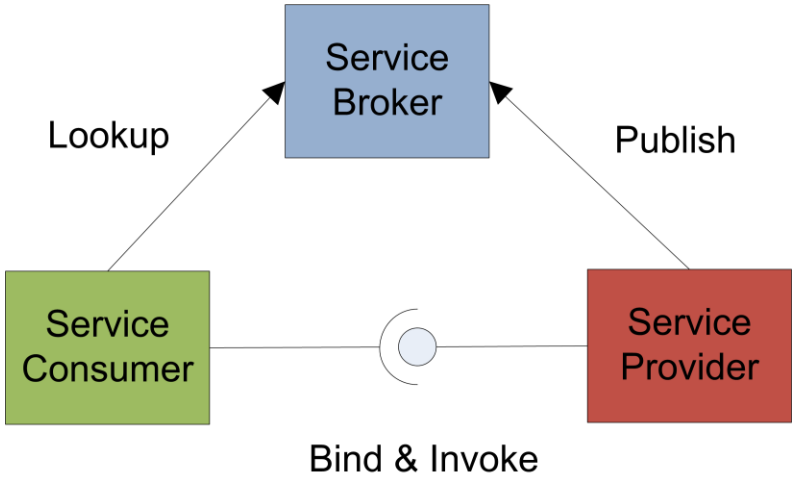
Kuva 4. *Bundlen elinkaaren tilat. Katkoviiva tarkoittaa automaattista tai implisiittistä siirtymää (OSGi Service Platform Core Specification v4.1).*

Bundle aloittaa elämänsä asennettuna (*installed*). Jos kaikki sen asettamat vaatimukset täyttyvät, eli Javan ajoympäristö vastaa määriteltyä ja tarvittavat pakkaukset ovat saatavilla, se siirtyy *resolved*-tilaan. Tästä tilasta se voidaan käynnistää käsin start-komennolla, tai määritellä käynnistymään automaattisesti heti OSGi-sovelluksen käynnistämisen yhteydessä.

OSGi-pohjaisissa ohjelmissa ei ole erillistä Main-luokkaa tai main-metodia, jota kutsuttaisiin kun ohjelmaa ajetaan – myös itse OSGi-kehys näkyy käynnissä olevassa järjestelmässä konkreettisesti system- bundlena. Jos bundle haluaa osallistua omaan elinkaareensa, sen täytyy toteuttaa erillinen aktivaattoriluokka, *BundleActivator*. Tämän luokan tulee toteuttaa *start* ja *stop* metodit, joita OSGi-kehyksen elinkaarikerros kutsuu silloin, kun bundle käynnistetään tai pysäytetään. Metodeita voidaan käyttää resurssien varaamiseen tai bundlen tarvitsemien olioiden luomiseen. Lisäksi metodien mukana tulee parametrina olio *BundleContext*, jonka kautta bundle pääsee käsiksi kehyksen tarjoamiin palveluihin. Se voi esimerkiksi hakea muita bundleja, pysäyttää tai käynnistää niitä tai rekisteröityä kuuntelemaan muutoksia niiden tiloissa.

Edellä mainitut ominaisuudet mahdollistavat sovellusten kehittämisen komponenttipohjaisesti, mutta pelkästään ohjelman jakaminen moduuleihin ei vielä tee niistä löyhästi kytkettyjä – yhden bundlen pysäyttäminen voi rikkoa koko ohjelman toiminnallisuuden, eikä uuden toiminnallisuuden lisääminenkään onnistu ilman olemassa olevien osien uudelleenkäännöksiä. OSGi tarjoaa näiden ongelmien ratkaisemiseksi palvelupohjaista arkkitehtuuria. OSGissa palvelut ovat bundleen

kuuluvia tavallisia Javan olioita, jotka toteuttavat jonkin *rajapinnan* (Javan Interface). Palvelut rekisteröidään tällä rajapinnalla palvelukerrokseen kuuluvaan Service Registryyn, josta palveluiden käyttäjät voivat niitä etsiä rajapintojen nimien perusteella. Tämä palvelupohjaisen arkkitehtuurin perusrakenne on esitetty alla olevassa kuvassa. OSGin palvelupohjainen arkkitehtuuri poikkeaa esimerkiksi Web-sovelluspalveluista siinä, että kaikki palvelut ovat käynnissä samassa virtuaalikoneessa, sen sijaan että ne olisi hajautettu tietoverkon yli.



Kuva 5. *Palvelupohjaisen arkkitehtuurin (SOA) perusmalli.*

Palvelut eivät aina ole välttämättä saatavilla, vaan niiden käytössä täytyy varautua myös tilanteeseen, jossa palvelua ei löydy. Tämä ratkaisee myös käynnistysjärjestykseen liittyvät ongelmat: jos jokin palvelu käynnistyy ennen muita, se jää odottamaan että sen tarvitsemat palvelut tulevat saataville.

OSGi tarjoaa dynaamisten palveluiden hallintaan useita työkaluja, joista periaatteeltaan yksinkertaisin on tapahtumat. Jos bundle ei löydä tarvitsemaansa palveluita, se voi rekisteröityä BundleContextin kautta kuuntelemaan päivityksiä haluamansa palvelun tilaan. Tapahtumien kuuntelun käyttö on kuitenkin jäänyt melko vähäiseksi, koska se vaatisi monimutkaista palveluiden käynnistysjärjestystä hallinnoivaa koodia. Parempi tapa on käyttää *ServiceTrackereita*, jotka huolehtivat palveluiden seurannasta ja reagoivat tapahtumiin käyttäjän puolesta. ServiceTrackeria voidaan räätälöidä osallistumaan seurattavien palveluiden elinkaareen käyttämällä toista, *ServiceTrackerCustomizer*-nimistä luokkaa.

ServiceTrackereille vaihtoehtoisena ja kehittyneempänä tapana hallita dynaamisia palveluita esiteltiin OSGin spesifikaation neljännessä versiossa uutena ominaisuutena *Declarative Services* (DS), automaattinen hallintajärjestelmä palveluiden riippuvuuksille. DS:ää varten bundlen täytyy määritellä XML-tiedostossa mitä palveluita se tarjoaa ja mitä riippuvuuksia sillä on, ja samalla tarve esim. BundleActivatorille poistuu, mikä vähentää tarvittavan OSGi-koodin määrää. Samalla DS mahdollistaa palveluiden käynnistämisen vasta silloin kun niitä tarvitaan, mikä vähentää tarvittavan muistin määrää ja käynnistymisaikaa. DS-tuki lisättiin Eclipseen

vasta uusimmassa versiossa (3.5 Galileo), minkä takia sen käyttöä ei välttämättä mainita kaikissa lähteissä. DS:ää voidaan kuitenkin pitää tällä hetkellä parhaana mekanismina palveluiden hallintaan.

Muista tavoista dynaamisten palveluiden hallintaan OSGissa on syytä mainita ainakin seuraavat:

- Service Activator Toolkit, Declarative Serviceitä ja Service Trackereita edeltänyt ja edelleen käytössä oleva laajennettu versio BundleActivatoreista.
- iPOJO, eli injected Plain Old Java Object. Apachen kehittämä, OSGin päälle tuleva komponenttimalli. Eli Javan moduulimallilaajennuksen komponenttimallilaajennus. ☺
- Spring Dynamic Modules on Spring Frameworkin (eräs suosittu Javan Dependency Injection -kehys) ja OSGin integraatio. Periaatteessa vielä joustavampi kuin DS, mutta vaatii myös Spring Frameworkin käyttämistä, mikä tekee ratkaisusta turhan monimutkaisen.

Komponenttien ja palveluiden dynaaminen hallinta tuo mukanaan joustavuutta ja robustisuutta kehitettävään ohjelmistoon ja parantaa mahdollisuuksia koodin uudelleenkäyttöön, mutta vaatii että sovelluksen suunnittelussa ja kirjoittamisessa kiinnitetään huomiota myös dynaamisuusnäkökohtiin. Huolellisen suunnittelun tuloksena ohjelmasta kuitenkin yleensä tulee selkeämpi, mikä on eduksi monimutkaisten kokonaisuuksien kanssa toimittaessa. Seuraavaksi käydään läpi OSGi-pohjaisten sovellusten suunnittelun *hyviä käytäntöjä* (best practices) eri arkkitehtuurikerrosten näkökulmista.

*Valitse importit oikein (**moduulikerros**)*

Älä oleta että koko JRE kaikkine laajennuksineen on käytettävissäsi.

- Vain java.* pakkauksia voi käyttää vapaasti.
- Muut pakkaukset kannattaa importoida (Import-Package -rivillä) Manifest-tiedostossa.

Miksi?

- Mahdollistaa vaihtoehtoisten toteutuksien tilalle vaihtamisen.
- Välttää ClassNotFoundErrorit, jos sovellustasi ajetaan testaukseen käyttämästäsi poikkeavassa OSGi-ympäristössä.

*Minimoi riippuvuudet (**moduulikerros**)*

Käytä Import-Package -riviä Require-Bundlen sijaan.

- Require-Bundle vaatii aina annetun nimisen bundlen.
- Import-Package hyväksyy vaihtoehtoisia toteutuksia vaaditulle pakkaukselle.

Käytä versiointia.

- Määrittele versionumeroalue, joka kelpaa bundlellesi.

Suunnittele bundlet huolellisesti

- Älä laita toisiinsa liittymättömiä asioita samaan bundleen.
- Suunnittele bundlet löyhästi kytketyiksi ja loogisesti yhteenkuuluviksi kokonaisuuksiksi.

Miksi?

- Esim. löydät bundlen jota haluat käyttää. Asennat sen ja huomaat että sillä on riippuvuuksia muihin bundleihin. Etsit ja asennat ne, mutta huomaat että niillä riippuvuuksia muihin bundleihin, *ad nauseam*.

*Piilota toteutuksen yksityiskohdat (**moduulikerros**)*

Sijoita sisäinen toteutus eri pakkaukseen kuin bundlen ohjelmointirajapinta (API).

- org.example.foo = API-pakkaus
- org.example.foo.internal = toteutuspakkaus

Älä exportoi toteutuspakkausta.

- Exportoi (ja importoi) vain julkinen rajapinta, ei sisäisen toteutuksen pakkauksia.
- Export-Package: org.example.foo; version=1.0

Miksi?

- Esim. kirjoitit bundlen ohjelmointirajapinnan ja sille toteutuksen. Käytit totetutuksessa public classeja, koska jouduit käyttämään ristiviittauksia pakkausten välillä. Exportoit kaikki bundlen pakkaukset. Myöhemmin päivität bundlea ja teet toteutukseen muutoksia. Kohta saat vihaisia puhelinsoittoja, vaikka olit nimenomaan *sanonut* että sisäisen toteutuksen luokkia ei saa käyttää.

*Älä oleta että bundlet käynnistyvät tietyssä järjestyksessä (**elinkaarikerros**)*

Kaikki palvelut eivät välttämättä ole käytettävissä heti alusta lähtien.

- Bundlejen käynnistysjärjestys voi vaihdella järjestelmäkohtaisesti, etkä yleensä pysty vaikuttamaan siihen.

Käytä ServiceTrackeria palveluiden seuraamiseen ja reagoi palvelun julkaisuun käyttämällä *ServiceTrackerCustomizeria*.

- ServiceTrackerCustomizer on luokka, jolla voidaan lisätä palveluiden elinkaareen hallintaan liittyviä ominaisuuksia ServiceTrackeriin.

Tai käytä deklaratiivista eli todentavaa palvelumallia, kuten OSGin Declarative Services.

Miksi?

- Jos oletat, että jokin palvelu on aina saatavilla, bundlesi voi toimia omalla koneellasi, mutta valmiiseen ohjelmaan integroituna heittää NullPointerExceptionin kun vaadittua palvelua ei löydykään.

*Muista palveluiden dynaamisuus (**palvelukerros**)*

Palvelut ovat dynaamisia ja voivat poistua palvelurekisteristä senkin jälkeen kun olet hakenut palvelun onnistuneesti.

- Bundlen täytyy reagoida käyttämänsä palvelun elinkaaressa tapahtuviin muutoksiin.

OSGi-kehys sisältää API:n näiden tapahtumien seurantaan, mutta sen metodit ovat melko vaikeakäyttöisiä. Tämän API:n pohjalta on kuitenkin kehitetty helppokäyttöisempiä työkaluja.

- Service Tracker ja Service Activator Toolkit (SAT)
- Declarative Services

Miksi?

- Kirjoittamasi bundle voi lakata toimimasta kesken ohjelman suorituksen, jos jokin sen tarvitsema palvelu on väliaikaisesti pois käytöstä, etkä hae sitä uudestaan kun se taas tulee saataville.

*Whiteboard-suunnittelumalli (**palvelukerros**)*

Jos tarvitset tapahtumankäsittelyä, Listener-suunnittelumallin käyttämisen sijaan suunnittele API:si niin, että kuuntelijan pitää rekisteröityä palveluksi.

- Yksinkertaisempaa ja robustimpaa kuin Listenerin käyttö.
- Pakottaa kiinnittämään huomiota OSGin palvelu- ja elinkaarimalleihin.

Tapahtumalähde rekisteröityy käyttämään kuuntelijan palvelua ja kutsuu sitä aina kun tapahtuma laukeaa.

- Whiteboard pohjautuu Inversion of Control -suunnittelumalliin.

Miksi?

- Esim. toteutat palvelusi käyttäen Listener-suunnittelumallin *addListener* ja *removeListener* -metodeita. Käytännössä muut bundlet kuitenkin unohtavat usein kutsua *removeListener*-metodia kun ne pysähtyvät, tai unohtavat kutsua *addListeneria* jos oma bundlesi käynnistyy uudelleen. Molemmat osapuolet tarvitsevat ylimääräistä koodia toistensa tarkkailuun, tai tapahtumat eivät toimi oikein.

*Vältä kytkentöjä OSGi-kehyksen API:in (**yleisiä käytäntöjä**)*

- Kirjoita sovelluslogiikan koodi POJOina (Plain Old Java Objects).
- Ohjelmoi rajapintojen, älä toteutusluokkien perusteella.
- Eristä OSGi-API:n käyttö minimaaliseen määrään luokkia ja sijoita ne omaan pakkaukseensa.
- Injektoi API-riippuvuudet näiden luokkien kautta POJOihin.
- Varmista että toteutusluokkasi eivät ole riippuvaisia näistä OSGiin kytketyistä luokista.
- Käytä OSGin valmiita IoC-säiliöitä, kuten Declarative Services, näiden riippuvuuksien ilmaisuun deklaratiivisesti.
    - Jätä OSGi API:n kutsuminen IoC-säiliön tehtäväksi.

Miksi?

- Esim. kirjoittamasi bundle julkaisee jonkin palvelun ja käyttää muiden bundlejen tarjoamia palveluita. Koodisi käyttää OSGi-API:ia useissa luokissa ja on sen seurauksena kytketty OSGiin. Nyt koodiasi ei voi enää helposti käyttää muualla kuin OSGi-ympäristössä.

*Nopea palaaminen kehyksen kutsuista (**yleisiä käytäntöjä**)*

Bundlejen kehittäjät käyttävät usein liikaa aikaa lataamalla etukäteen kaiken mahdollisen.

Jos käynnistyminen kestää 1 s per bundle (esim. nimipalvelukysely):

⇨ Yksi minuutti 60:llä bundlella
⇨ Viisi minuuttia 300:lla bundlella

Declarative Servicestä löytyvä laiska käynnistys on hyödyllinen ominaisuus

- Bundle käynnistyy vasta kun sen palveluita tarvitaan.

Kehyksen kutsuista täytyy palata nopeasti. Jos kutsukoodissa täytyy tehdä jotakin jonka suorittaminen kestää pitkään, niin:

- Tee se tapahtumapohjaisesti tai
- Käynnistä taustasäie suorittamaan pitkäkestoista työtä

Miksi?

- Kehitettäessä isoa OSGi-pohjaista järjestelmää komponentit saattavat toimia riittävän nopeasti moduulitestauksessa, mutta järjestelmätestauksessa kumulatiivinen käynnistymisaika voi kasvaa liian suureksi.

*Säieturvallisuus (**yleisiä käytäntöjä**)*

OSGi-ympäristössä kehyksen takaisinkutsut bundleesi voivat tapahtua useissa säikeissä yhtä aikaa. Koodisi on oltava säieturvallista!

- Älä pidä resursseja lukittuina kun kutsut metodia, jonka toteutusta et tunne. Jos kutsumasi metodi tarvitsee lukitsemaasi resurssia, voit aiheuttaa suorituksen lukkiutumisen.
- Javan monitorit on tarkoitettu matalan tason tietorakenteiden suojeluun; käytä korkean tason abstrahointeja ja aikakatkaisuja olioiden lukitsemiseen.
- Suorita jokin rinnakkaisuuteen keskittynyt kurssi, esim. OHJ-4010 Rinnakkaisuus.

Miksi?

- Rinnakkaisuudesta johtuvat virheet voivat olla todella hankalia korjata, lisäksi virheiden esiintyminen voi olla satunnaista ja tulla esiin vasta lopullisessa ohjelmassa.

APPENDIX 4: OSGI EXERCISE

ACI-32040 Automaation ohjelmistokomponentit ja sovelluspalvelut
Eclipse & OSGi -harjoitus, kevät 2010
Ohje v2.0

Aihe:

- OSGi-bundlet ja -palvelut, graafisen käyttöliittymän toteuttaminen.

Työkalut/ympäristö:

- Eclipse-platform 3.5 (www.eclipse.org)
- J2SE 1.6
- Visual Editor 1.4

Sisältö:
Harjoituksessa tehdään OSGi-sovellus kolmessa vaiheessa, joissa tuodaan OSGi-toiminnallisuus ohjelmaan vähitellen mukaan. Uutta sovellusta voi lähteä kehittämään suoraan palveluperusteisesti, mutta monesti ohjelmien kehitys tai protoilu alkaa esim. tavallisena Java-ohjelmana johon sitten tuodaan lisäominaisuuksia kun tarpeet paremmalle modulaarisuuden hallinnalle tulevat esiin. Tämä lähestymistapa pyrkii myös havainnollistamaan OSGin eri ominaisuuksien toiminnallisuutta ja jakamaan OSGia selvemmin hahmotettaviin osakokonaisuuksiin.

Harjoituksen 1. osassa luodaan laskimen toiminnallisuuden toteuttava luokka ja sille käyttöliittymä.

Harjoituksen 2. osassa muutetaan laskimen osat OSGi-komponenteiksi, eli bundleiksi.

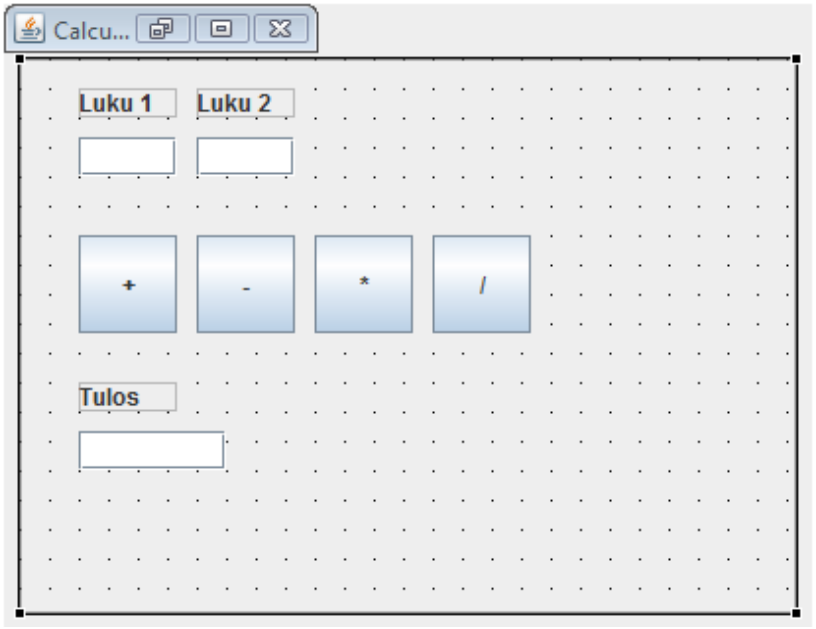
Harjoituksen 3. osassa muutetaan bundlet palvelupohjaisiksi.

Osa 1, laskin

Käynnistä Eclipse (versio 3.5, Galileo). Luodaan ensin uusi projekti, jossa laskinsovellusta kehitetään. Valitse: ***File*** **>** ***New*** **>** ***Java Project…*** ja ***Next***. Anna projektille nimeksi *Calculator* ja klikkaa ***Finish***. Luo uusi *CalculatorLogic*-luokka ***File*** **>** ***New*** **>** ***Class*** ja määrittele pakkaukseksi *fi.tut.ase.calculator* ja luokan nimeksi *CalculatorLogic.* Lisää luotuun luokkaan julkiset metodit laskimen toiminnoille (add, subtract, multiply, divide), jotka kaikki ottavat kaksi double-tyyppistä parametria ja palauttavat double-tyyppisen tuloksen.

Lisää uusi käyttöliittymäluokka Visual Editorilla muokattavaksi: ***File*** **>** ***New*** **>** ***Other…*** **>** ***Java*** **>** ***Visual Class*** ja anna luokan nimeksi *CalculatorView.* Valitse Styleksi ***Swingin*** alta ***Frame***, jonka jälkeen voit painaa ***Finishiä***.

Kasaa laskimelle yksinkertainen käyttöliittymä. Komponentit löytyvät papujen kuvalla merkityn ***Choose Bean*** nappulan alta yläreunasta. Lisää ensin paneeli ***JPanel*** Framen keskelle (***Center***) ja vaihda sille Properties-välilehdeltä tai kontekstivalikosta layoutiksi *null*. Kasaa paneelin päälle loppu laskimen käyttöliittymä: kaksi tekstikenttää syötettäville luvuille, neljä nappia matemaattisia operaattoreita varten ja yksi tekstikenttä tulokselle. Voit ottaa mallia oheisesta kuvasta.



Kuva 6. *Käyttöliittymämalli laskimelle.*

Lisää luokkaan yksityinen jäsenmuuttuja *private CalculatorLogic calculator* ja seuraava metodi, jolla laskin voidaan injektoida käyttöliittymään:

```
public void setCalculatorLogic(CalculatorLogic value){
      calculator = value;
}
```

Seuraavaksi luodaan napeille tapahtumankäsittelijät. Valitse nappi editorista ja sen kontekstivalikosta löytyy ***Events*** > ***actionPerformed***. Lisää nappuloille toiminnallisuus seuraavan mallin mukaisesti automaattisesti generoituihin tapahtumankäsittelijöihin:

```
double luku1 = Double.parseDouble(jTextField1.getText());
double luku2 = Double.parseDouble(jTextField2.getText());
jTextField3.setText(
      String.valueOf(calculator.add(luku1, luku2))
);
```

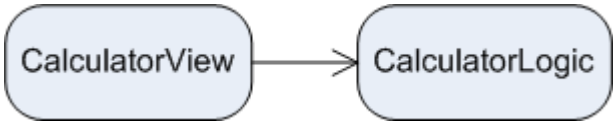
Luo uusi Java-luokka *Main* ja lisää sille *main*-metodi:

```
public static void main(String[] args) {
	System.out.println("Launching calculator");
	CalculatorView calcUI = new CalculatorView();
	CalculatorLogic calcLogic = new CalculatorLogic();
	calcUI.setCalculatorLogic(calcLogic);
	calcUI.setVisible(true);
}
```

Aja valmis laskinohjelma valitsemalla *Main*-luokka Package Explorerista ja ***Run As*** > ***Java Application*** kontekstivalikosta. Toteutuksessa on erotettu käyttöliittymä laskimen toimintalogiikasta hyvien ohjelmointiperiaatteiden mukaisesti. Näin laskimen sisäistä toiminnallisuutta voidaan muuttaa tarvittaessa helposti ilman että käyttöliittymään tarvitsee tehdä muutoksia ja vastaavasti käyttöliittymä voidaan vaihtaa toiseen koskematta laskimen toteutukseen.

Osa 2, OSGi-bundlet

Laskin on tällä hetkellä perinteinen Java-sovellus, joka tosin on suunniteltu modulaarisesti. Seuraavaksi toteutus muutetaan hyödyntämään OSGia jakamalla toteutus bundleihin. Jakaessa sovellusta komponentteihin voidaan esim. miettiä että voisiko komponenttiehdokas toimia yksin vai olisiko se järkevämpi pakata yhteen jonkin toisen osan kanssa. Omassa laskinsovelluksessamme on tällä hetkellä vain kaksi järkevää komponenttiehdokasta, käyttöliittymä ja laskinlogiikka. Etenkin jos osia on enemmän, kannattaa piirtää kaavio komponenttien välisistä riippuvuuksista, johon merkitään myös niiden välisten riippuvuuksien suunnat. Tämä helpottaa mm. kehäriippuvuuksien välttämistä.



Kuva 7. *Bundlejen väliset riippuvuudet.*

Luodaan aluksi uusi plug-in projekti laskinlogiikka-bundlelle. ***File*** > ***New*** > ***Project…*** > ***Plug-in Project*** ja paina ***Next***. Anna uudelle projektille nimeksi *fi.tut.ase.calculator.logic*. Varmista että Target Platformiksi on valittu ***an OSGi framework*** ja ***Equinox***. Paina ***Next*** ja seuraavalla sivulla poista valinta ***Generate an activator…*** Muiden asetuksien pitäisi olla ok, paina ***Finish***. Jos Eclipse kysyy haluatko avata plug-inien kehitykseen tarkoitetun näkymän, vastaa kyllä.

OSGi-projektien, eli bundlejen, nimet noudattavat yleensä pakkausten nimiä. Koska bundlet usein sijaitsevat samassa työhakemistossa, tämä nimeämiskäytäntö helpottaa niiden hallintaa etenkin bundlejen määrän kasvaessa.

Eclipse avaa projektin luotuaan Manifest-editorin, jolla voidaan helposti muokata ja hallita bundlen toiminnallisuuden määrittelevää MANIFEST.MF-tiedostoa. Bundle koostuu sisällöstä – usein Java-luokista – ja määrittelytiedostosta. XML-muotoista

määrittelytiedostoa voi muokata välilehtien asetuksien lisäksi myös käsin editorin MANIFEST.FM-välilehdeltä ja usein onkin hyödyllistä käydä sieltä katsomassa miten editorissa tehdyt muutokset vaikuttavat määrittelyyn.

Aloitetaan bundlen muokkaus siirtämällä valmiiksi toteutettu *CalculatorLogic*-luokka bundlen sisälle. Valitse luokka Package Explorerista ja avaa kontekstivalikosta ***Refactor > Move***. Valitse *fi.tut.ase.calculator.logic*-projektista src-kansio ja paina ***Create Package…*** Nimeä pakkaus samoin kuin projektikin: *fi.tut.ase.calculator.logic*. Valitse luomasi pakkaus ja paina ***OK***.

Nyt bundlella on toteutus, mutta se ei vielä näy bundlen ulkopuoliselle maailmalle. Manifest-editorissa siirry Runtime-välilehdelle (editorin saa auki tuplaklikkaamalla META-INF-kansiossa olevaa MANIFEST.FM-tiedostoa) ja Exported Packages kohdasta paina ***Add…*** ja lisää pakkaus *fi.tut.ase.calculator.logic*. Paina ***OK*** ja tallenna editori. Runtime-välilehdeltä voidaan valita mitä pakkauksia bundle exportaa. Exportatut pakkaukset muodostavat bundlen ohjelmointirajapinnan, eli API:n, jonka kautta sen tarjoamaan toiminnallisuuteen päästään käsiksi.

Luo *CalculatorViewille* uusi plug-in-projekti *fi.tut.ase.calculator.view* vastaavalla tavalla kuin edellä ja siirrä refaktoroinnilla *CalculatorView* luokka pakkaukseen *fi.tut.ase.calculator.view*. Avaa Manifest-editori ja siirry Dependencies-välilehdelle. Täältä voidaan määritellä mitä pakkauksia tai plug-inejä bundlemme vaatii toimiakseen. Lisää Imported Packages -kohtaan ***Add…***-napilla *fi.tut.ase.calculator.logic* ja *org.osgi.framework*-pakkaukset. *Org.osgi.framework*-pakkausta tarvitaan kun bundlelle kohta luodaan aktivaattori.

Luokat ovat nyt eri projekteissa ja *CalculatorLogic* on laskinnäkymän käytettävissä, mutta puutteelliset importit saattavat aiheuttaa vielä käännösvirheitä. Tallenna editori ja käy korjaamassa *CalculatorView*-luokan importit. Tämä onnistuu helpoiten avaamalla luokka Package Explorerista ja ajamalla automaattisen importtien korjauksen näppäinyhdistelmällä ***Ctrl+Shift+O***, eli ***Source > Organize Imports***. Toki korjaamisen voi tehdä myös käsin. Alkuperäisessä *Calculator*-projektissa on myös virheitä, mutta niistä ei enää tarvitse välittää, koska projektia ei enää käytetä mihinkään.

OSGi-Bundlet voivat käynnistyä joko samalla kun koko OSGi-framework käynnistyy tai sitten kun niiden luokkia tarvitaan. Bundleilla ei ole main-metodia, jossa voisi luoda uusia olioita, mutta ne voivat osallistua frameworkin toimintaan esim. toteuttamalla Bundle Activatorin-luokan.

Luo *fi.tut.ase.calculator.view*-projektiin *Activator* avaamalla Manifest-editoriin Overview-välilehti ja klikkaamalla ***Activator***-linkkiä. Täytä aukeavaan New Java Class –wizardiin pakkaukseksi *fi.tut.ase.calculator.view* ja luokan nimeksi *Activator*. Paina ***Finish***. Luodussa *Activatorissa* on valmiit stubit *start* ja *stop*-metodeille, jotka ajetaan silloin kun bundle käynnistetään tai sammutetaan. Poista *@Override*-annotaatiot ja muokkaa metodit seuraavanlaisiksi:

```
public void start(BundleContext context) throws Exception {
      System.out.println("Launching calculator");
      CalculatorView calcUI = new CalculatorView();
      CalculatorLogic calcLogic = new CalculatorLogic();
      calcUI.setCalculatorLogic(calcLogic);
       calcUI.setVisible(true);
}

public void stop(BundleContext context) throws Exception {
      System.out.println("Terminating calculator");
}
```

Kuten huomaat, start-metodi vastaa toteutukseltaan alkuperäistä main-metodia. Activatorin olisi voinut generoida myös heti alussa jättämällä projektin luonnissa valinnan ***Generate an activator…*** päälle.

Ohjelman ajamiseksi valitse ***Run > Run Configurations…*** ja valitse OSGi Framework. Luo uusi käynnistyskonfiguraatio valitsemalla kontekstivalikosta ***New***. Anna konfiguraatiolle nimeksi *Calculator*. Poista kaikki valinnat (***Deselect All***) ja valitse tarvittavat bundlet:

fi.tut.ase.calculator.logic
fi.tut.ase.calculator.view
org.eclipse.osgi

Poista valinta ***Add new workspace bundles to this launch configuration automatically*** ja käynnistä ohjelma painamalla ***Run***. Jos Eclipse avaa Save and launch –ikkunan, tallenna muutokset painamalla ***OK***. Framework pysyy käynnissä kunnes kirjoitat konsoliin komennon *close* tai pysäytät suorituksen punaisen neliön näköisellä ***Terminate***-napilla.

Osa 3, palvelut

Edellisessä osassa sovellus muutettiin bundle-pohjaiseksi, mutta se ei vieläkään ole kovin dynaaminen tai löyhästi kytketty. Seuraavaksi sovellus muutetaan palvelupohjaiseksi, mikä lisää joustavuutta ja mahdollistaa dynaamisuuden lisäämisen ohjelmaan.

Tällä hetkellä *CalculatorLogic* alustetaan vielä näkymässä. Muutetaan sen toteutusta niin, että se luo itse *CalculatorLogic*-olion ja rekisteröi sen palveluksi. Tämä tapahtuu luomalla ensin rajapintaluokka, joka rekisteröidään. Palvelua käytetään rajapintaluokan kautta.

Avaa *CalculatorLogicin* kontekstivalikko Package Explorerista ja valitse ***Refactor > Extract Interface…*** anna nimeksi *ICalculatorLogic* ja valitse ***Select All*** ja ***OK***. Koska OSGi käyttää Javan pakkauksia näkyvyyden yksikkönä, pitää bundlen sisäinen toteutus erottaa julkisesta rajapinnasta, eli API:sta, siirtämällä se toiseen pakkaukseen: valitse *CalculatorLogic*-luokalle ***Refactor > Move...***, luo pakkaus *fi.tut.ase.calculator.logic.internal* ja siirrä se sinne.

Lisätään seuraavaksi *CalculatorLogicille* aktivaattori, jossa palvelu luodaan ja rekisteröidään. Aloita avaamalla projektin Manifest-editori ja lisää Dependencies-välilehdelle Imported Packages -listaan *org.osgi.framework*. Luo aktivaattori klikkaamalla Activator-linkkiä Overview-välilehdeltä. Anna nimeksi *Activator* ja pakkaukseksi *fi.tut.ase.calculator.logic.internal.bundle*.

Palvelun rekisteröinti aktivaattorissa tapahtuu seuraavan koodinpätkän kuvaamalla tavalla:

Tällä hyvän OSGi-ohjelmointitavan mukaisella nimeämiskäytännöllä pyritään erottamaan OSGiin liittyvä koodi (eli tässä tapauksessa aktivaattori) omaan pakkaukseensa varsinaisesta sovelluksen bisneslogiikkakoodista. Tämä helpottaa koodin uudelleenkäyttöä muualla sekä palveluiden muuttamista dynaamiseksi, jota tosin ei tässä harjoituksessa opetella.

```
private ServiceRegistration registration;

public void start(BundleContext context) throws Exception {
      CalculatorLogic calcLogic = new CalculatorLogic();
      registration = context.registerService
(ICalculatorLogic.class.getName(), calcLogic, null);
}

public void stop(BundleContext context) throws Exception {
      registration.unregister();
}
```

Korjaa jälleen luokan importit (***Source*** > ***Organize Imports*** tai ***Ctrl+Shift+O***). Kun framework käynnistyy, se kutsuu *start*-metodia, joka rekisteröi laskinlogiikan toteuttavan olion *calcLogic ICalculatorLogic*-palveluna. Lisäksi aktivaattori tallentaa *registerServicen* palauttaman *ServiceRegistration*-olion rekisteröinnin purkamiseksi *stop*-metodissa.

Seuraavaksi siirrytään *CalculatorViewin* kimppuun. *CalculatorView* pitäisi muuttaa käyttämään *CalculatorLogicin* tarjoamaa palvelua *ICalculatorLogic*-rajapinnan kautta, mutta ***Refactor*** > ***Interface*** on jo tehnyt tämän muutoksen puolestamme. Bundlen *Activator* sen sijaan kaipaa hieman korjailua:

```
private ICalculatorLogic calcLogic;
private ServiceReference calcRef;
private CalculatorView calcUI;

public void start(BundleContext context) throws Exception {
   System.out.println("Launching calculator");
   calcUI = new CalculatorView();
   calcRef =
context.getServiceReference(ICalculatorLogic.class.getName());
   if (calcRef == null) {
           System.err.println("Unable to acquire Calculator Logic!");
           return;
   }
   calcLogic = (ICalculatorLogic) context.getService(calcRef);
   if (calcLogic == null) {
```

```
            System.err.println("Unable to acquire Calculator Logic!");
            return;
    }
    calcUI.setCalculatorLogic(calcLogic);
    calcUI.setVisible(true);
}

public void stop(BundleContext context) throws Exception {
    if (calcRef != null) {
            context.ungetService(calcRef);
    }
    System.out.println("Terminating calculator");
}
```

Korjaa lopuksi importit ***Organize Importsilla***. Start-metodissa laskimen käyttöliittymän luomisen jälkeen aktivaattori hakee referenssin *CalculatorLogicin* palveluihin *BundleContextin* kautta. Tämän referenssin avulla voidaan hakea varsinainen palvelu. Huomaa että emme voi enää luottaa siihen, että palvelua olisi saatavilla, joten joudumme varautumaan siihen että saamme takaisin pelkän *null*-viitteen. Lopuksi aktivaattori injektoi laskimen käyttöliittymään. Mieti mitä ongelmia tästä ratkaisusta mahdollisesti syntyy[8].

Huomaa vielä että haettu palvelu on syytä vapauttaa Stop-metodissa, eli meidän täytyy ilmoittaa Service Registrylle, että emme enää käytä palvelua kutsumalla *ungetService*-metodia.

Erotetaan vielä OSGi-koodi muusta näkymän toteuttavasta koodista refaktoroimalla *CalculatorView*-luokka pakkaukseen *fi.tut.ase.calculator.view.internal* ja *Activator* pakkaukseen *fi.tut.ase.calculator.view.internal.bundle*.

Kokeile käynnistää ohjelma: ***Run > Run Configurations… > Run***. Saatat saada konsoliin oman virheilmoituksemme ”`Unable to acquire Calculator Logic!`”, joka kertoo meille että laskimen käyttöliittymä-bundle käynnistyi ennen logiikka-bundlea. Tämän virheen voi korjata valitsemalla *fi.tut.ase.calculator.logic*-bundlen Manifest-editorin Overview-välilehdeltä vaihtoehdon ”Activate this plug-in when one of its classes is loaded”, jolloin bundle käynnistetään laiskasti vasta silloin kun sitä tarvitaan, mutta kuitenkin ennen muiden luokkien lataamista. Dynaamiset palvelut tosin ratkaisevat tämänkin ongelman ja parantavat bundlejen käynnistymisen hallintaa.

[8] Jos palvelu muuttuu tai lakkaa olemasta saatavilla sen jälkeen kun laskin on käynnistetty, tämä tapahtuma jää huomaamatta kokonaan. Toisaalta laskimen käyttöä voi jatkaa normaalisti, koska käyttöliittymällä on logiikkaolio tallennettuna, mutta esim. päivitetyt toiminnot eivät välity käyttöliittymään ennen kuin se käynnistetään uudelleen.

APPENDIX 5: OSGI ASSIGNMENT

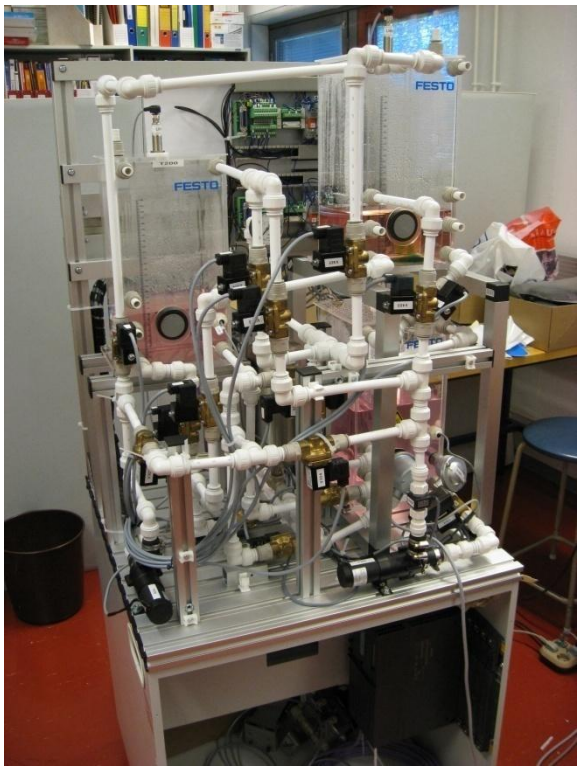
ACI-32040 Automaation ohjelmistokomponentit ja sovelluspalvelut
OSGi-harjoitustyö, kevät 2010
Ohje v1.00

Aihe:

- OSGi, Equinox, Eclipse

Työkalut/ympäristö:

- Eclipse-platform 3.5 (esim. RCP-versio, www.eclipse.org)
- Visual Editor 1.4 (Eclipse plugin)
- J2SE 1.6



Tässä dokumentissa on esitetty harjoitustyöltä vaadittava toiminnallisuus lyhyesti asiakasvaatimusten näkökulmasta. Ryhmän tehtäväksi jää suunnitella ja toteuttaa näiden vaatimusten pohjalta toimiva sovellus, jossa on sovellettu luennoilla esiteltyjä ja muuten yleisesti hyviksi katsottuja ohjelmointimalleja ja -menetelmiä.

Tehtävä:
Harjoitustyön tarkoituksena on tutustua OSGi-teknologiaan ja palvelu-pohjaisuuteen. Työssä suunnitellaan ja toteutetaan panosprosessin etähallinta- ja valvontatyökalu Bacon (Remote Batch Control).

Ohjelmalla on tarkoitus ohjata prosessin toimilaitteita ja lukea antureita. Lisäksi ohjelma voi sisältää muuta toiminnallisuutta, kuten turvallisuusominaisuuksia (varoituksia ylivuodoista, kohonneesta lämpötilasta tai paineesta). Harjoitustyön toteutus tehdään simulaattorilla, mutta simulaattori voidaan vaihtaa OPC-komponenttiin, joka ohjaa sd107-labrasta löytyvää oikeaa minipanosprosessilaitteistoa.

Ohjelman tulee olla komponenttipohjainen, jolloin sitä voitaisiin käyttää eri prosessijärjestelmien kanssa vaihtamalla laitteistorajapintakomponentti tai esim. kannettavalla päätelaitteella vaihtamalla sopivat käyttöliittymäkomponentti/-komponentit. Ohjelman toiminnallisuus tulee suunnitella siten, että se on laajennettavissa tarvittaessa bundleja lisäämällä tai vaihtamalla, esimerkiksi http-tuella,

web-käyttöliittymällä, uusilla toimilaitteilla jne. Toteutuksen tukena käytetään kolmansien osapuolien tuottamia komponentteja, jotka annetaan valmiina.

Toteutus:
Sovellus toteutetaan käyttäen OSGin Equinox-toteutusta, joka tulee Eclipse-kehitysympäristön mukana. Työn arvosteluperusteina toimivat mm. ohjelman jako bundleihin, toteutetut ominaisuudet, dynaamisuus, palvelut ja nimeämiskäytännöt.

- Bundleja voi muodostaa esim. seuraavista osista: käyttöliittymä, ydintoiminnot (core), hälytykset, toimilaitteet, anturit, tietojen tallentaminen, valmiina annettava simulaattori/OPC-komponentti (perustuu JEasyOPC:hen) jne.
- Bundleja olisi hyvä olla kohtuullisen paljon, jotta toteutus vastaa oikeaa sovellusta (normaalisti määrät vaihtelevat sadoista tuhansiin). Muista kuitenkin että määrä ei korvaa laatua.
- Palvelupohjaisuuden toteutuksen saa valita, Declarative Services[9] (DS) suositeltavin
- Dynaamisuus vaatii erityistä huomiota ohjelmoinnissa myös OSGin kanssa

Käyttöliittymän graafiseen toteutukseen suositellaan Eclipsen Visual Editor -pluginiä, mutta muutkin toteutustavat kelpaavat. Ohjeet simulaattorin käyttöön ja lyhyt esittely minipanosprosessin laitteistosta tulevat valmiina annettavien komponenttien mukana.

Yleisiä ohjeita:
OSGin yleisimpiä konsolikomentoja (voidaan kirjoittaa suoraan Eclipsen Consoleen)

- ss – short status, listaa bundlet
- refresh [id] – päivittää bundlen tilan
- restart/start/stop [id] – käynnistä/pysäytä bundle
- install/uninstall [id] – poista tai asenna bundle
- close – sulkee frameworkin

Malli pakkausten nimeämiselle:
Perusmalli domainin (www.ase.tut.fi) ja projektin (Bacon) mukaan: fi.tut.ase.bacon.
Julkaistavat rajapinnat, esim. laite *floater*, fi.tut.ase.bacon.dev.floater.
Laitteeseen liittyvä sisäinen toteutuskoodi: fi.tut.ase.bacon.internal.dev.floater.
OSGi-spesifinen koodi kuten bundle activatorit: fi.tut.ase.bacon.internal.dev.floater.fake.bundle (ei tarvita jos käytetään DS:ää).

Hyödyllisiä linkkejä:
Equinox Refcard (http://refcardz.dzone.com/refcardz/equinox)
Tutoriaali OSGin Declarative Serviceille (http://neilbartlett.name/blog/osgi-articles/)

[9] Muita vaihtoehtoja ovat pelkät bundle activatorit, Service Trackerit, SAT ja Spring DM.