Reaaliaikasovellukset ja UML-pohjainen mallinnusprosessi

ITK130 Johdatus ohjelmistotekniikkaan

22.10.2003

Sami Äyrämö

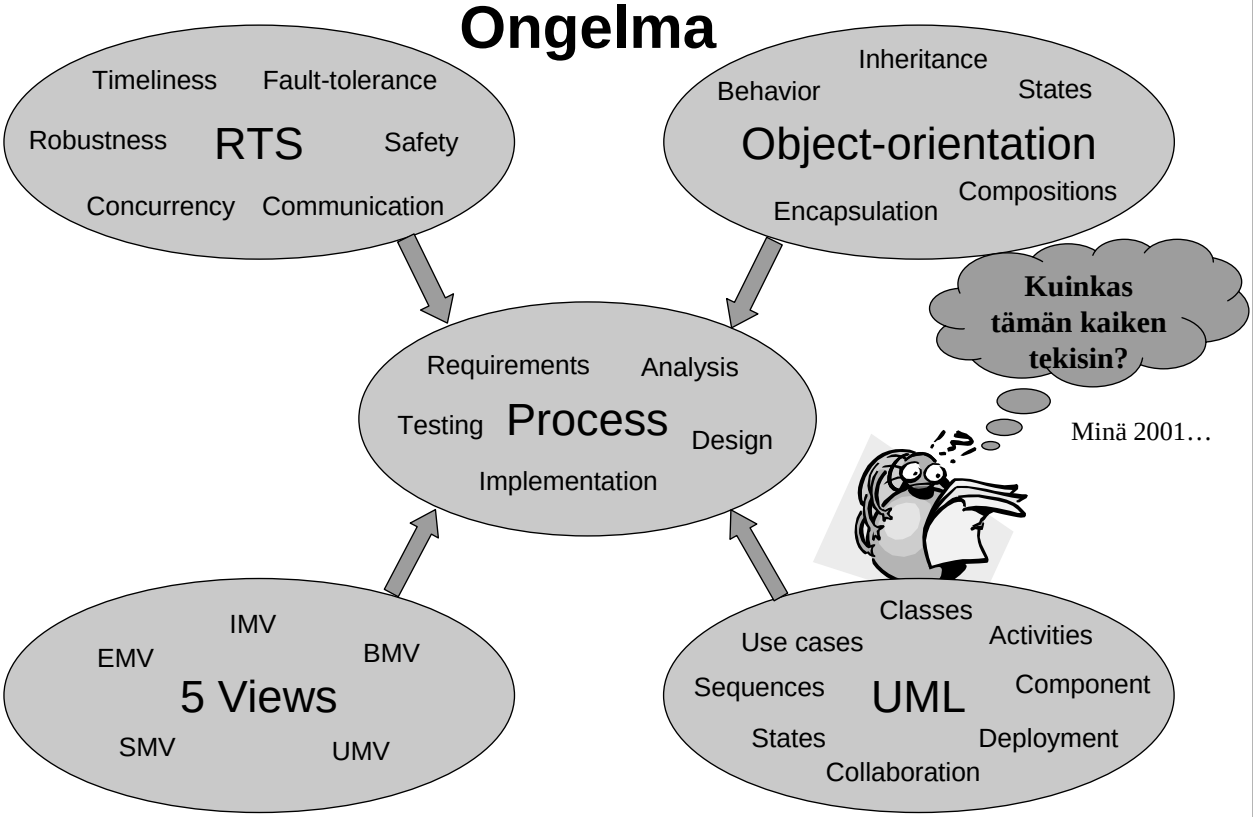
sami.ayramo@mit.jyu.fi

Sisältö

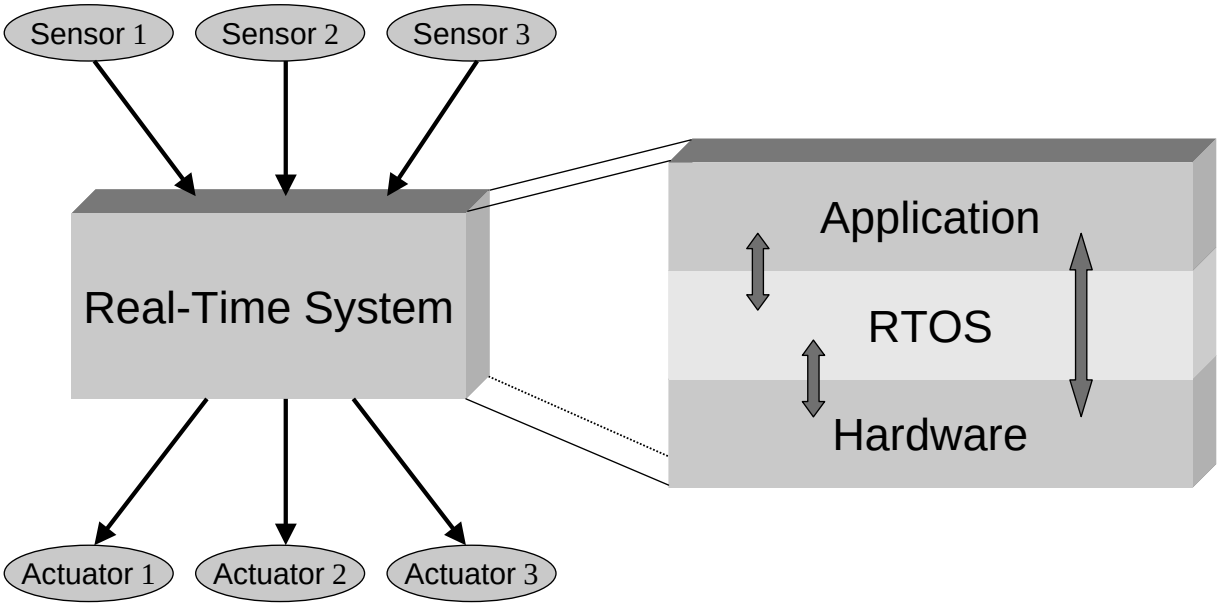
- **Reaaliaikajärjestelmät (*RTS, Real-Time Systems*)**
- **Oliot**
- **UML**
- **5 näkymää (5 views)**
- **Prosessimalli (vaiheet + näkymät + UML kaaviot)**

Lähde: S. Äyrämö: *Reaaliaikajärjestelmien mallintaminen UML-kuvauskielen avulla*, pro gradu, Tietotekniikan laitos, Jyväskylän yliopisto, 2002.

Ongelma



RTS:n yleinen malli ja kerrosrakenne



Reaaliaikajärjestelmän määritelmä

*"A real-time computer system may be defined as one which controls an environment by receiving data, processing them, and taking action or **returning results sufficiently quickly** to affect the functioning of the environment at that time."*

James Martin, *Design of Real-Time Computer Systems.* Englewood Cliffs, N.J.: Prentice-Hall, 1967.

*"Any information processing activity or system which has to respond to externally-generated input stimuli **within a finite and specified period**."*

S.J. Young, *Real-Time Languages:design and development*. Ellis Horwood, Chichester, England 1982 .

Reaaliaikajärjestelmän piirteitä

- Tehtävien suorittamiselle aikarajoitteita
- Oikea looginen vaste ei riitä, vaan se pitää saada myös oikeaan aikaan
- Vasteaika voi vaihdella (esim. määrätty ajanjakso)
- Joskus vasteajan täytyy olla sekunnin murto-osia, joskus taas riittää muutama päiväkin, kunhan toiminta tapahtuu oikeaan aikaan
- Yleensä suoritetaan useita tehtäviä rinnakkain, mikä edellyttää tehtävien (*taskien*) skedulointia (Jane W. S. Liu, *Real-Time Systems*, Prentice Hall, 2000 tai kurssi TIE342 Reaaliaikajärjestelmät)
- Hyvin usein ns. sulautettuja järjestelmiä
    - Käyttäjät voivat olla myös ihmisiä, mutta yleensä ne ovat laitteita, jotka lähettävät palvelupyyntöjä RTS:lle.

Reaaliaikajärjestelmien erityisvaatimukset

1. Oikea-aikaisuus
   - aikarajoitteet
2. (pseudo-) rinnakkaisuus
   - kommunikointi, synkronointi, priorisointi,...
3. Virheettömyys, kestävyys ja vikasietoisuus
   - virheen korjaus, toiminta vikatilanteissa, deadlockit,...
4. Ennustettavuus
   - satunnaisuuden hallinta, keskeytykset
5. Muut rajoitteet
   - esim. fyysiset rajoitteet

✓ Tarvitsevatko erityishuomiota?

✓ Voidaanko näihin vaikuttaa mallinnuskielen valinnalla?

✓ Missä vaiheessa prosessia niihin pitäisi reagoida?

UML (Unified Modeling Language)

• Graafinen kuvausnotaatio ohjelmistojen visualisointiin, määrittelyyn, rakentamiseen ja dokumentoimiseen

• ”*UML on kuvaustapa, joka sijoittuu ohjelmointikielten ja suunnittelumenetelmien välimaastoon…*”

- M. Rintala ja J. Jokinen: *Olioiden ohjelmointi C++:lla*, Suomen ATK-kustannus 2000.

• Soveltuu ensisijaisesti oliopohjaisten järjestelmien ja ohjelmistojen mallintamiseen

• UML ei ole: prosessi, ohjelmointikieli tai määrittelykieli

• Ei määrittele, missä järjestyksessä kaavioita pitäisi käyttää

UML (Unified Modeling Language)

- Booch method + OMT + OOSE => UML
- Työkaluja
  - Rational Rose, Rhapsody (I-Logix), MS Vision, …
  - ArgoUML (www.argoUML.org), ilmainen!
- Kirjallisuutta:
  - *OMG UML specification*, version 1.5, March 2003,
    - (www.omg.org)
  - *UML Notation Guide 1.1* (netissä useita linkkejä)
  - *UML Semantics 1.1* (netissä useita linkkejä)
  - B.P. Douglass, *Doing Hard Time : Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns*, Addison-Wesley, 1999.
  - B.P. Douglass*, Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*, Addison-Wesley, 2002.
  - H. Gomaa: *Designing Concurrent, Distributed, and Real-Time Applications with UML*, Addison-Wesley, 2000.

UML kaaviot ja elementit

- Käyttötapauskaavio (*Use Case diagram*)
- Luokkakaavio (*Class diagram*)
- Käyttäymiskaaviot (*behavioral diagrams*)
  - Tilakaavio (*State diagram*)
  - Aktiviteettikaavio (*Activity diagram*)
  - Vuorovaikutuskaaviot (*interaction diagrams*)
    - Sekvenssikaavio (*Sequence diagram*)
    - Yhteistoimintakaavio (*Collaboration diagram*)
- Toteutuskaaviot (*implementation diagrams*)
  - Komponenttikaavio (*Component diagram*)
  - Sijoittelukaavio (*Deployment diagram*)
- Kaavioille yhteisiä kuvauselementtejä
  - Paketit (package)
  - Stereotyypit (stereotype)
  - Muistilaput (note)
  - Rajoitteet (contraints) (Kuvataan yleensä tekstikenttien text strings avulla)

Oliokaavio (Object diagram)

• ”*A static object diagram is an instance of a class diagram; it shows a snapshot of the detailed state of a system at a point in time.*”

  • OMG UML specification, version 1.5, pp. 3-35, March 2003

• Kuvaa olioita ja niiden data-arvoja

• Luokkakaavio voi sisältää olioita, joten luokkakaavio, jossa ei ole luokkia vaan olioita, on oliokaavio!

5-VIEWS

• UMV (User Model View): käyttäjän näkökulma, joka mallinnetaan yleensä käyttötapausten avulla

• EMV (Enviromental Model View): kuvaus järjestelmän toteutusympäristöstä

• SMV (Structural Model View): kuvaus datan ja erilaisten rakenteellisten elementtien välisistä riippuvuuksista järjestelmän sisällä

• BMV (Behavioral Model View): kuvaus ohjelmiston dynaamisesta toiminnasta ja käyttäytymisestä. Voi johtaa SMV-näkymän muutoksiin, jos rakenne ei mahdollista toiminnallisuuksien toteuttamista tai se ei täytä laadullisia vaatimuksia

• IMV (Implementation Model View): kuvaus rakenteellisten ja toiminnallisten mallien toteutuksesta

(R.S. Pressman, *Software Engineering: A Practitioner 's Approach*. European Adaption, 5th Edition, McGraw-Hill International, 2000)

3-vaiheinen mallinnusprosessi

- Prosessimalli reaaliaikajärjestelmien kehittämiseen UML-notaatiota ja Pressmanin viittä näkymää käyttäen
- Vain määrittely- ja mallinnusvaiheet (ei toteutus- / testausvaiheita)
- Tavoite löytää ne UML-kaaviot, joilla mallinnusprosessi kantaa prosessivaiheiden yli niin, että sitä voidaan mahdollisimman helposti seurata molempiin suuntiin
- Edellisen vaiheen on annettava syötteenä lähtötiedot seuraavalle
- Yksinkertaisuus (= kuvataan vain tarvittavat asiat, tavoite ei ole käyttää kaikkia UML-kaavioita)
- Unohdetaan tässä yhteydessä muut ohjelmistoprosessiin liittyvät asiat kuten: roolit, työnjako, metriikat, aikataulut, resurssit jne.

3-vaiheinen mallinnusprosessi

- Vaatimusmäärittely
  - Toiminnalliset vaatimukset
  - Toimintaympäristö
  - Ei-toiminalliset vaatimukset
- Yleissuunnittelu
  - Ohjelmiston yleisrakenteen ja arkkitehtuurin kuvaaminen
  - Toimivuuden arviointi
- Yksityiskohtainen suunnittelu
  - Suunnitelma siitä, kuinka hyväksi todettu rakenne toteutetaan

Näkymät vs. vaiheet/ Vaiheet vs. kaaviot

	Vaatimusmäärittely	Yleissuunnittelu	Yksityiskohtainen suunnittelu
UMV:	X		
EMV:	X		
SMV:		X	X
BMV:	X	X	X
IMV:			X

	UMV	EMV	SMV	BMV	IMV
Käyttötapauskaavio	**X**	**X**			
Luokkakaavio			**X**		
Tilakaavio				x	x
Aktiviteettikaavio				**X**	**X**
Sekvenssikaavio	**X**			**X**	
Yhteistoimintakaavio	x			x	
Komponenttikaavio					x
Sijoittelukaavio		**X**			

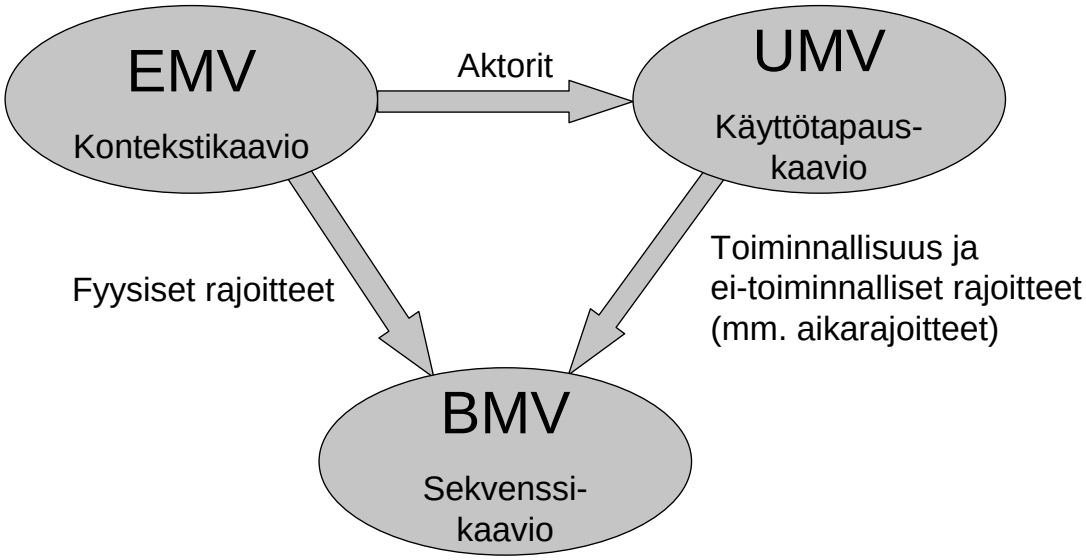
X: prosessimalliin valitut kaaviotyypit

x: kaaviotyyppi jolla myös voidaan kuvata näkymää

Vaatimusmäärittelyvaihe

- Kuvataan toimintaympäristö:
  - ympäristön laitteet ja liitynnät
  - esim. RTS ei yleensä toimi välittömästi ihmisten ohjaamana, vaan se kommunikoi suoraan muiden laitteiden kanssa
  - auttaa ymmärtämään toimintaympäristön ja sen tuomat rajoitteet
- Kuvataan toiminta: käyttäjän näkökulma järjestelmän toiminnasta
  - ei mietitä kuitenkaan sisäistä toteutusta vielä = BLACK BOX!
- Päätetään ei-toiminnalliset vaatimukset (esim. aikarajoitteet)
- EMV, UMV ja BMV näkymät!

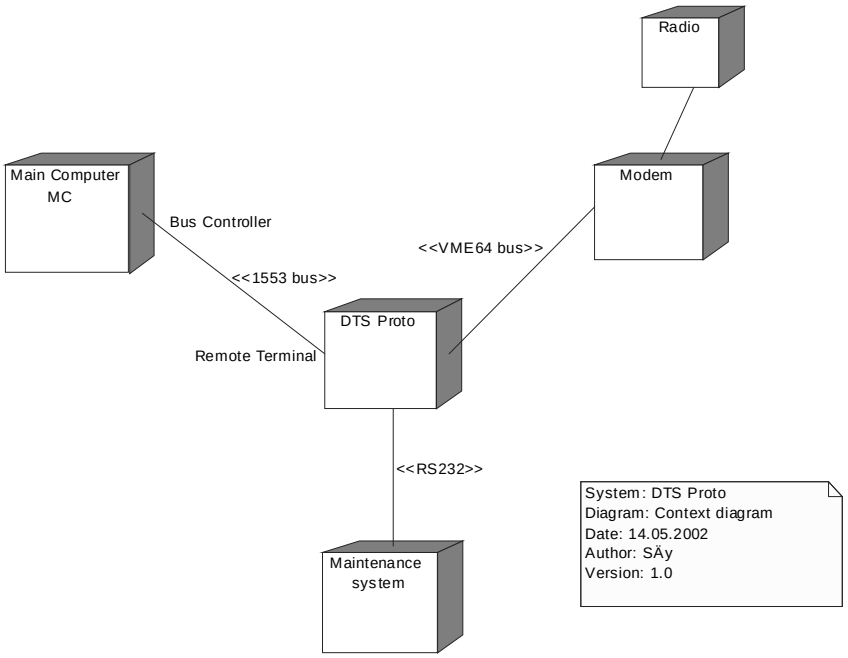
Vaatimusmäärittelyvaihe: näkymät - kaaviot



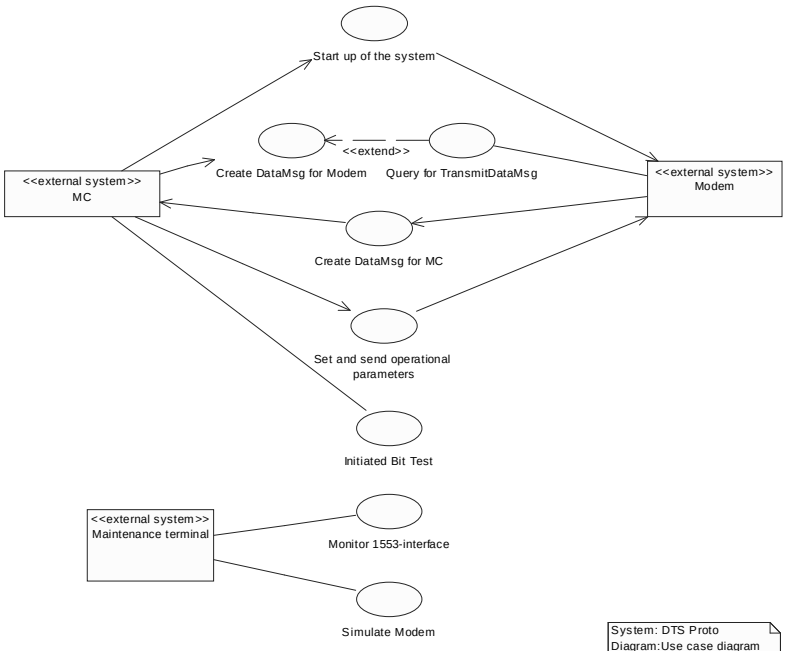
Kontekstikaavio (Context diagram)

- Kuvaa mallinnettavan järjestelmän ympäristön rakennetta sekä vuorovaikutuksessa toimivia laitteita (voivat olla aktoreita -> hyödyllinen RTS:n mallintamisessa)
  - Ei varsinainen UML-kaavio
  - Voidaan toteuttaa sijoittelukaavion avulla

Vaatimusmäärittelyvaihe: EMV/kontekstikaavio



Vaatimusmäärittelyvaihe: UMV/käyttötapauskaavio



Vaatimusmäärittelyvaihe: UMV/käyttötapauksen tarkentaminen

Transfer DataMsg to Modem

Yhteenveto: MC lähettää DTS Protolle sanoman, josta DTS Proto muodostaa datasanoman modeemille.

Aktorit: MC

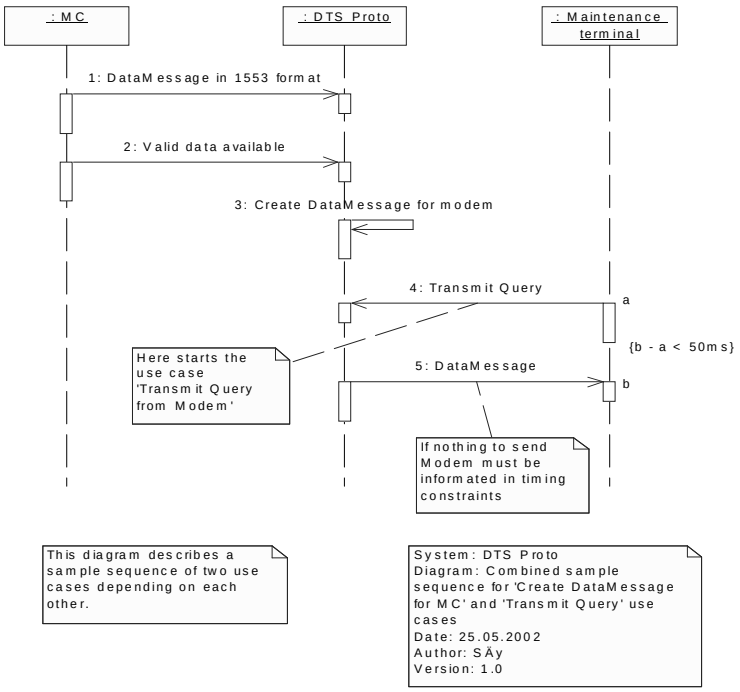
Esiehdot: DTS Proto sekä sen rajapinnat MC:lle ja modeemille ovat toiminnassa.

Kuvaus: MC ilmoittaa lähettäneensä uuden sanoman [1], jolloin DTS Proto lukee sanoman ja muodostaa siitä uuden modeemisanoman. MC lähettää myös sanoman pituuden [2]. Muodostettuaan uuden Modeemisanoman DTS Proto tallettaa sen seuraavaa lähetystiedonkyselyä varten.

Poikkeukset: [1] Jos väylä DTS Protolle ei ole toiminnassa, MC käynnistää DTS Proton uudelleen. [2] Jos pituus on nolla, ei uutta sanomaa muodosteta.

Jälkiehdot: DTS Proto on valmis vastaamaan uudella modeemisanomalla Modeemin lähetystiedonkyselyyn.

Vaatimusmäärittelyvaihe: (BMV/sekvenssikaavio)



Vaatimusmäärittelyvaihe: yhteenveto

- RTS:n toimintaympäristön kuvaaminen kontekstikaavioilla
- Käyttötapausten kuvaaminen sekä sanallinen ja graafinen tarkentaminen
  - sanalliset kuvaukset
  - sekvenssikaaviot
- Aktoreiden kerääminen kontekstikaavion perusteella
- Huom. RTS:ssä myös ajastimet voivat olla aktoreita (eli tapahtumaketjuja käynnistäviä toimijoita)
- BLACK BOX!!
- Vaihtoehtoja RTS:n vaatimusmäärittelylle:
  - EERL-listat (Ellis, John R., *Objectifying Real-Time Systems*, SIGS Books, April 1994)

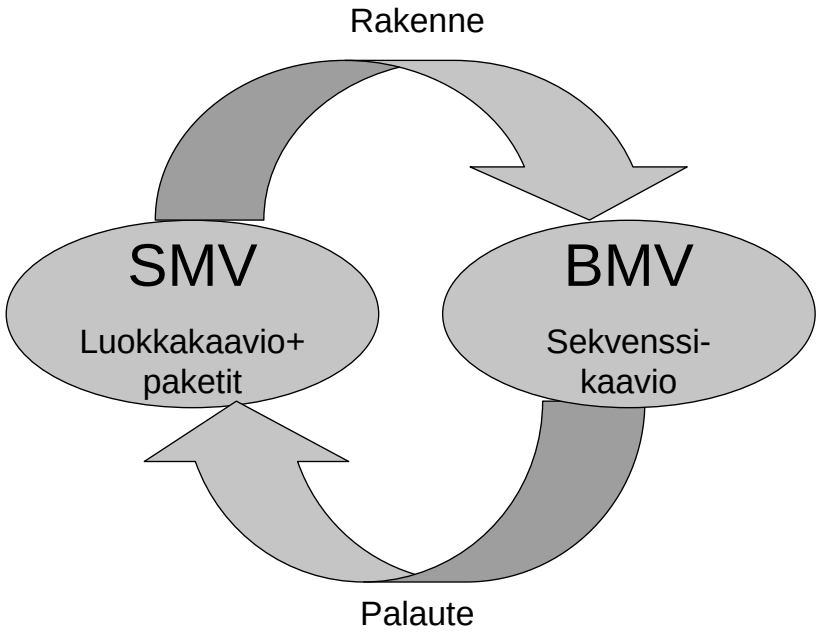
Yleissuunnitteluvaihe 1

- Arkkitehtuurin suunnittelu (Kurssi: *TIE355 Ohjelmistoarkkitehtuurit* !!)
- Alkuehtoina vaatimusmäärittelyn tuotokset
- Lähtökohtana olioarkkitehtuuri
- Alijärjestelmien kuvaaminen (package-komponentit)
- Ns. analyysitasoisten luokkien valitseminen ja kuvaaminen (ei toteuttaminen!)
- Kummat ensin…luokat vai alijärjestelmät?
- Tavoite: kuvataan ohjelman rakenne ja ilman yhtään koodiriviä analysoidaan, toteuttaako se sille asetetut laatuvaatimukset … aika kova juttu …
- Kuvattavat näkymät:
  - Rakenne: SMV
  - Toiminta: BMV

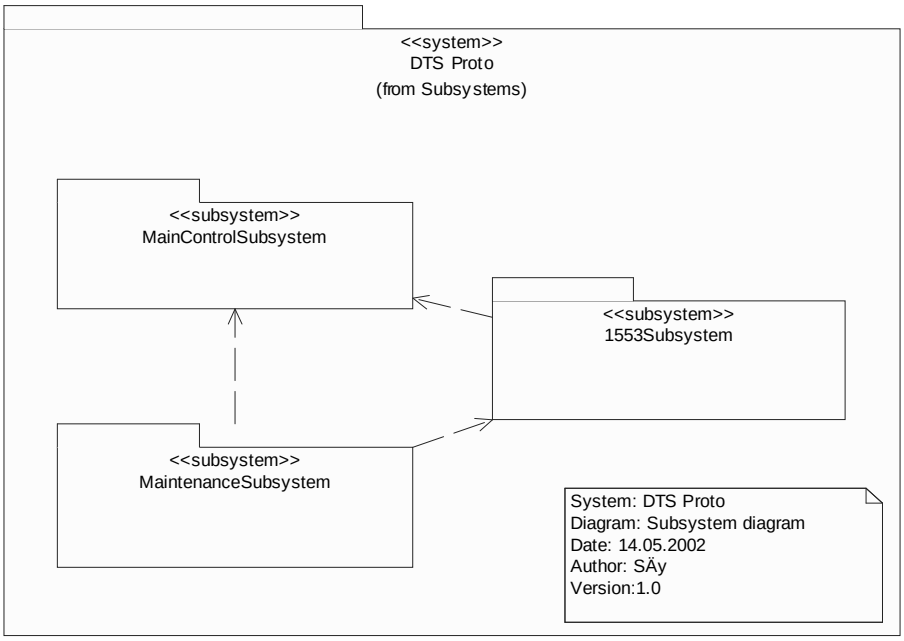
Yleissuunnitteluvaihe 2

- Alijärjestelmäjako (SMV)
  - pakettinotaatio
  - luokat
- Alijärjestelmien välisten liityntöjen kuvaaminen vaikeaa UML:n pakettinotaatiolla
  - kuinka alijärjestelmät kommunikoivat keskenään
  - käytettävä matemaattisia ja sanallisia kuvauksia apuna
  - luokkien väliset assosiaatiot mahdollistavat hieman enemmän
- Alijärjestelmäjaon jälkeen etsitään sopivat luokkakandidaatit
  - *boundary-, entity-*, *control*-stereotyypit
  - RTS:ssä myös *active*-stereotyyppi (taskit)
- Vaihtoehtoisesti voidaan etsiä luokkakandidaatit ja muodostaa niistä sopivia alijärjestelmiä
  - keskenään tiukasti kytketyt luokat: sama alijärjestelmä
  - keskenään löyhästi kytketyt luokat: eri alijärjestelmiä

Yleissuunnitteluvaihe: näkymät ja kaaviot



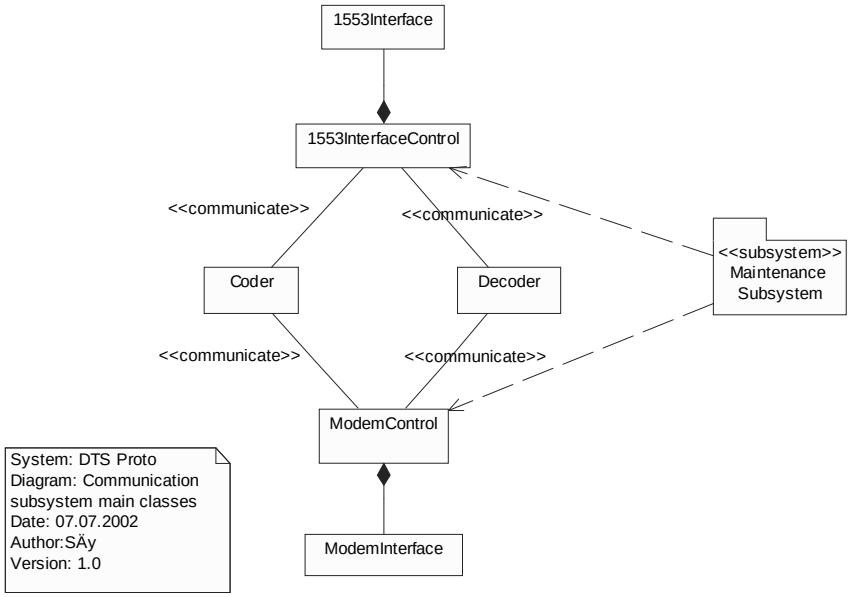
Yleissuunnitteluvaihe: SMV/alijärjestelmät



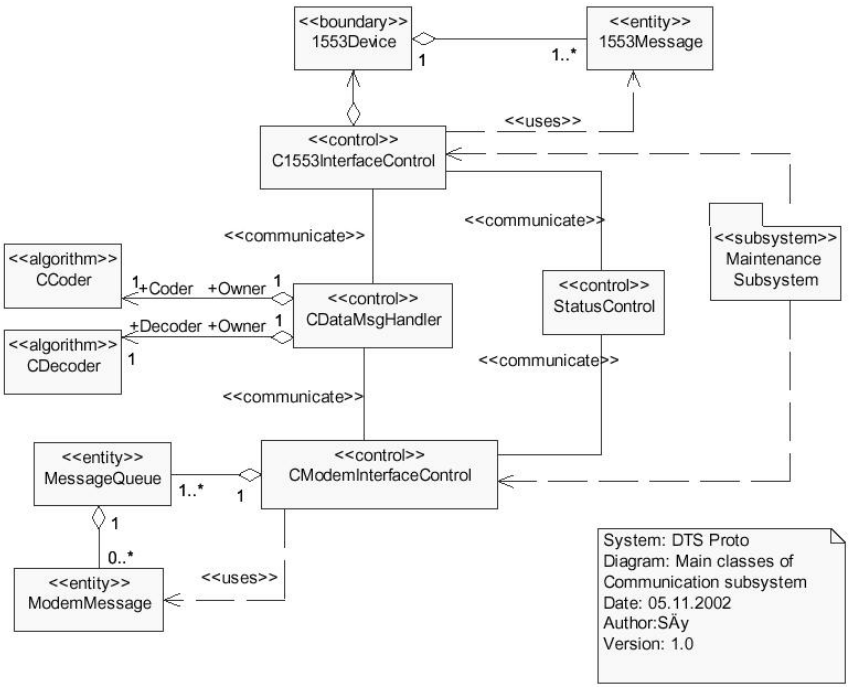
Yleissuunnitteluvaihe: alijärjestelmät

- Kolme alijärjestelmää kuvattuna pakettinotaatiota käyttäen
- Kaavio esittää vain riippuvuudet alijärjestelmien välillä
- Ohjaa alijärjestelmien suunnittelu- ja toteutusjärjestystä
- Tarvitaan lisäksi kuvaukset alijärjestelmien vastuista - sanallinen kuvaus
- Jokainen luokkakandidaatti sijoitetaan yhteen alijärjestelmään
- Yksi alijärjestelmä voi sisältää useita luokkia
- Tarkastellaan luokkakandidaattien toimivuutta testaamalla käyttötapauksia luokkarakenteelle sekvenssikaavioiden avulla (BMV)
  - Pyritään arvioimaan täyttyvätkö koko järjestelmälle asetetut vaatimukset
- Jos BMV osoittaa luokkakaavion riittämättömäksi palataan tarkastelemaan luokkarakennetta SMV-näkymän avulla

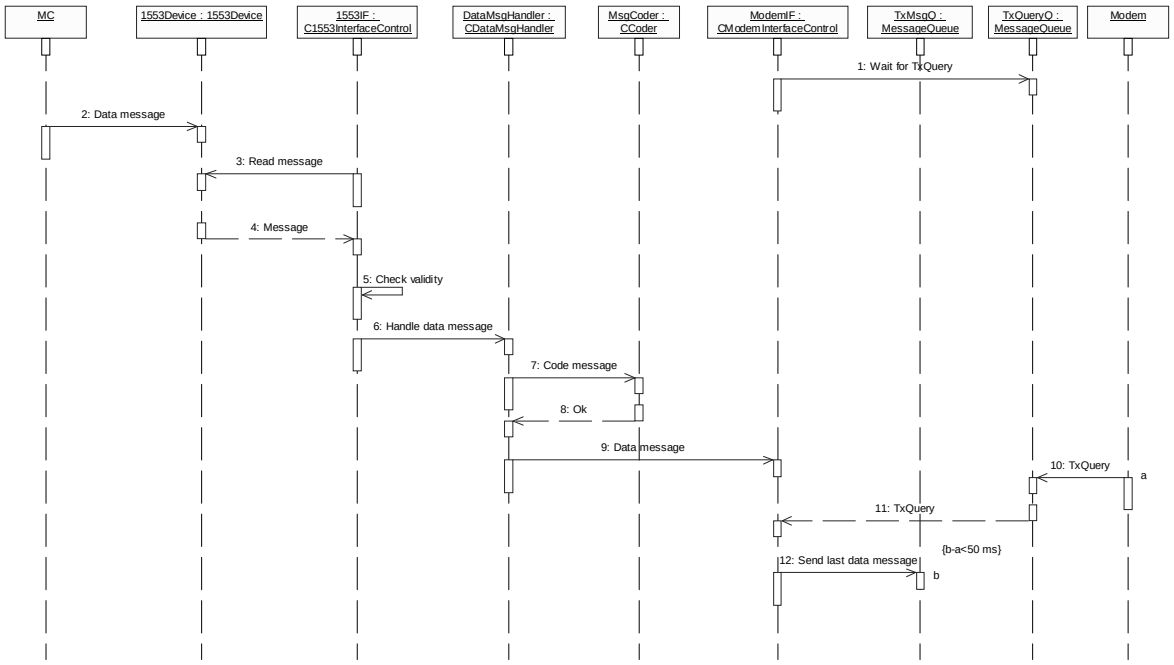
Yleissuunnitteluvaihe: SMV/luokkakaavio (analyysikaavio)



Yleissuunnitteluvaihe: SMV/luokkakaavio (tarkennettu kaavio)



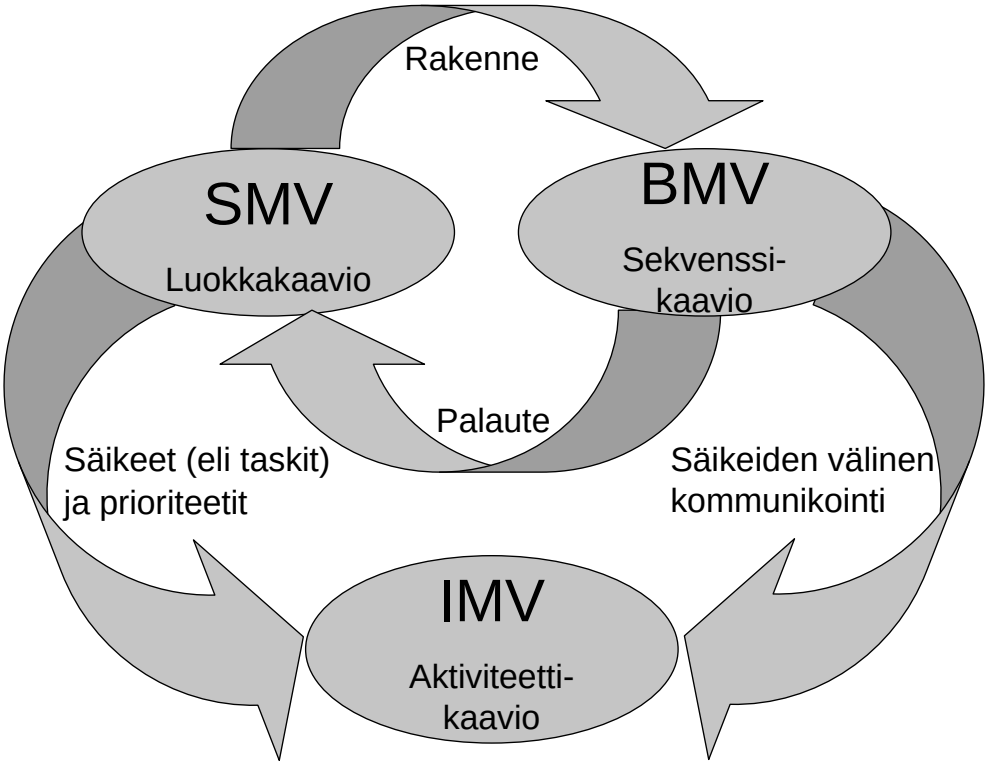
Yleissuunnitteluvaihe: BMV/Sekvenssikaavio



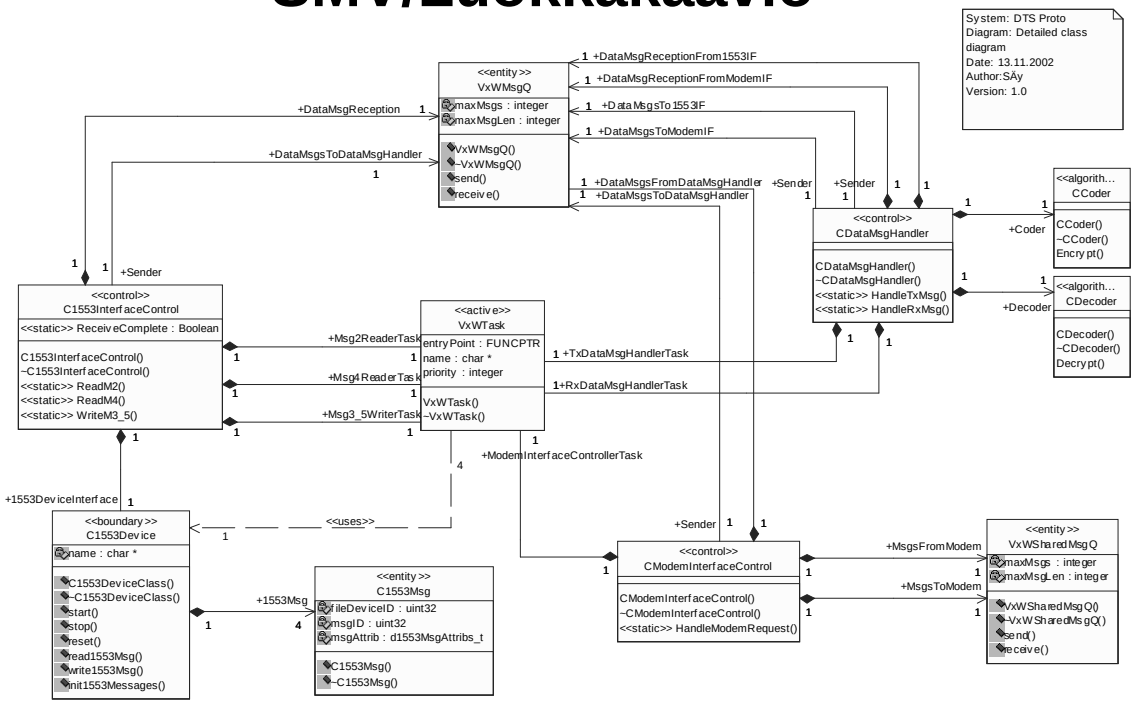
Yksityiskohtaisen suunnittelun vaihe

- Lähtökohtana yleissuunnitteluvaiheen luokkakaavio
- Tiedetään luokkien vastuut ja vuorovaikutus, joilla toiminnalliset vaatimukset täyttyvät
- Yksityiskohtaisen suunnittelun tavoitteena on tuottaa toteutuskelpoinen luokkakaavio, jossa näkyvät metodit, datarakenteet ja kommunikointiratkaisut
- Uudelleenkäytettävien luokkien valinta
- Luokkien julkisia metodeja suunniteltaessa huomioitava yleissuunnitteluvaiheen skenaariot
- Aktiivisten olioiden (taskit) ja niiden prioriteettien suunnittelu
- Taskien kommunikointi (käyttöjärjestelmä tarjoaa yleensä menetelmät)
- Taskien toiminta voidaan kuvata aktiviteettikaavioilla
- Vaiheen tuotos: toteutuskelpoiset luokat ja niiden toiminta

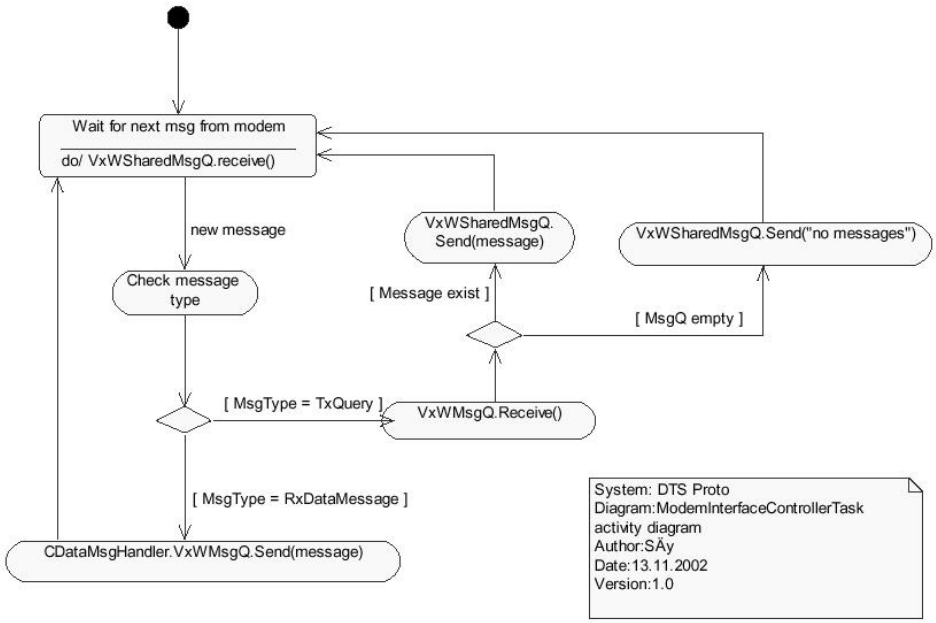
Yksityiskohtaisen suunnittelun vaihe



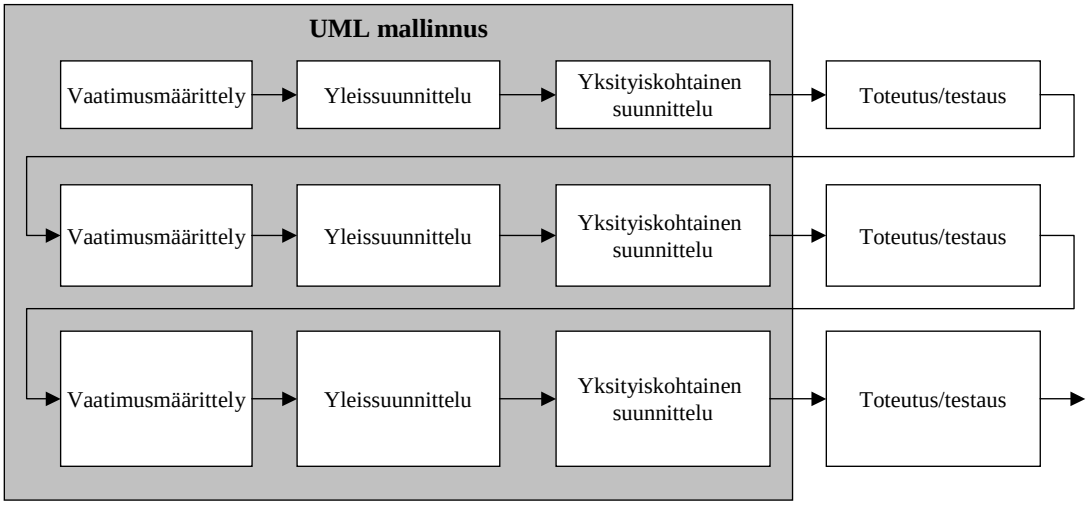
Yksityiskohtaisen suunnittelunvaihe: SMV/Luokkakaavio



Yksityiskohtaisen suunnittelunvaihe: IMV/taskien toimintosekvenssit



Yhteenveto prosessista



- Iteratiivisuus (protoilu - toteutetaan vain valitut käyttötapaukset)
- Inkrementaalisuus (include-extend -relaatiot käyttötapauksissa)

Mietittävää…?

- Saavutetaanko RTS:stä aiheutuvat vaatimukset
- Ajankuvaaminen sekvenssikaavioissa (esim. satunnainen keskeytyksen saapuminen ja siihen reagointi)?
- Muutosten lokaalisuus olioiden avulla?
- Saadaanko ohjelmistosta vikasietoisempi?
- Ongelma: Oliopohjainen rakenteen suunnittelu
  - Ylläpidettävyys vs. suorituskyky (pidetään yleensä ristiriitana)
  - Ratkaisu: kuhunkin käyttötapaukseen osallistuu mahdollisimman pieni määrä olioita? (…tämänkin opin arkkitehtuurikurssilta!)
- RTS:n käsiteköyhyys - pitäisikö keskittyä enemmän tiedonprosessoinnin kuvaamiseen (tietovuokaaviot) ja jättää oliopiirteet vähemmälle?
- Onko tarvetta uudelleenkäytettävyydelle?
- *luokat – oliot – säikeet?!*