3 + 1 UML-näkymää arkkitehtuuriin

Juha-Markus Aalto,
Nokia Telecommunications,
Network Management Systems

Ohjelmistoarkkitehtuurin merkitys kasvaa järjestelmien koon ja monimutkaisuuden kasvaessa. Arkkitehtuurin suunnittelun ja jatkokehityksen kannalta on tärkeää, että arkkitehtuuri pystytään selkeästi esittämään. Nokiassa laajasti käytettyyn OMT++ -menetelmään on sisällytetty UML-mallinnuskieleen pohjautuvia käytäntöjä, joiden avulla laajankin järjestelmäarkkitehtuurin keskeisimmät ominaisuudet voidaan esittää ilmaisuvoimaisten kaavioiden avulla.

Arkkitehtuuri ja UML

Järjestelmäarkkitehtuuria on lähes mahdotonta toteuttaa ja ylläpitää, mikäli keskeisiä arkkitehtuuriratkaisuita ei voida kommunikoida järjestelmän toteuttajille. Ohjelmistotuotannon menetelmäkehittäjät ovatkin laatineet erilaisia kuvaustapoja ja mallintamiskieliä, joilla abstrakteja rakenteita voidaan esittää tiiviisti ja selkeästi. Viime vuosien ehkä merkittävin edistymisaskel tällä alueella on ollut yhteisen mallintamiskielen UML:n (Unified Modeling Language) määrittely. UML tarjoaa kuvauskielen ohjelmistotyön vaihetuotteiden määrittelyyn, visualisointiin ja dokumentointiin [UML].

UML soveltuu hyvin ohjelmistoarkkitehtuurien esittämiseen. UML:n yksi vahvuus on sen monipuolisuus, mutta toisaalta mallinnuskielen rikkaus tuo haasteita UML:n oppimiseen ja tarkoituksenmukaiseen soveltamiseen. Projektityössä on syytä ensin miettiä huolella, mitä tärkeää suunnitteluinformaatiota halutaan määritellä ja jakaa, ja vasta sitten valita sopivat mallintamistyökalut UML:n kaaviotarjonnasta.

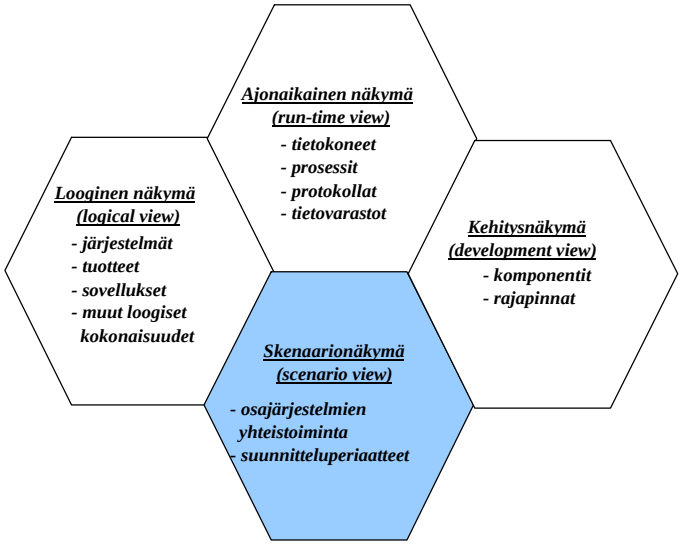
Ohjelmistoarkkitehtuurien eri näkökulmien mallintamistarpeet riippuvat muun muassa sovelluksen luonteesta, valitusta toteutusteknologiasta ja järjestelmän laajuudesta. Nokian OMT++ -menetelmää kehittäessä on todettu, että useimpien keskisuurten ja laajojen järjestelmien arkkitehtuurikuvauksen tulee vastata ainakin seuraaviin kysymyksiin [Jaaksi-Aalto-Aalto-Vättö 99]:

- *Mistä loogisista osista järjestelmä koostuu?*
- *Millainen on järjestelmän ajonaikainen konfiguraatio?*
- *Miten järjestelmä jaetaan kehitettäviin ohjelmistokomponentteihin?*
- *Mitä riippuvuuksia ja rajapintoja on komponenttien välillä?*
- *Miten arkkitehtuuri toteuttaa tärkeimmät käyttötapaukset?*

Jako loogisiin osajärjestelmiin antaa laajastakin järjestelmästä yleiskuvan ja auttaa hahmottamaan mm. toiminnallisuuden allokointia eri järjestelmän osille. Ajonaikaisen konfiguraation määrittely kuvaa käytettävissä olevat tietokoneresurssit ja prosessien allokoinnin niille sekä ajonaikaiset riippuvuudet, mikä auttaa esimerkiksi vikatilanteiden selvittämisessä. Jako ohjelmistokomponentteihin ja komponenttien riippuvuuksien tunteminen luovat pohjan konfiguraationhallinnalle (software configuration management) sekä kehitys- ja integrointijärjestyksen suunnittelulle. Arkkitehtuurin tarkastelu käyttötapausten (use case) kannalta auttaa mm. kehitettävän järjestelmän suorituskyvyn arviointia käyttäjän näkökulmasta.

3 + 1 näkymää

Tässä esiteltävä "3 + 1" –näkymän malli on kehitetty Philippe Kruchtenin tunnetuksi tekemän "4 + 1" –näkymän [Kruchten 95] mallin pohjalta. Pääerot Kruchtenin alkuperäiseen malliin verrattuna ovat UML:n käyttöönotto kuvauskielenä sekä näkymien sovittaminen OMT++ -menetelmän arkkitehtuurisuunnittelukäytäntöihin. Kuvassa 1 on esitetty yhteenveto 3 + 1 UML-pohjaisesta arkkitehtuurinäkymästä, joita ovat looginen näkymä (logical view), ajonaikainen näkymä (run-time view), kehitysnäkymä (development view) sekä skenaarionäkymä (scenario view).

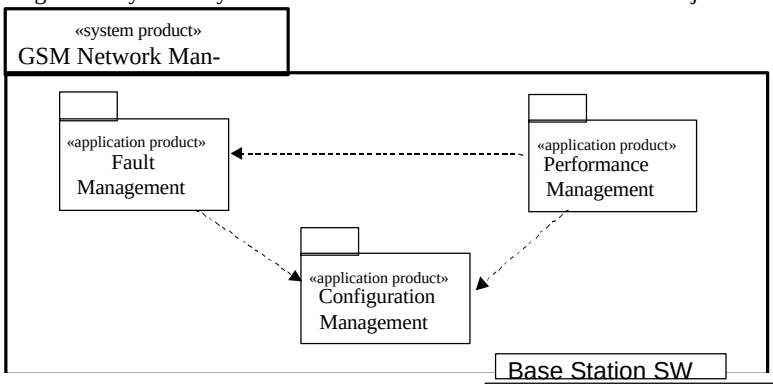


Kuva 1. 3 + 1 näkymää arkkitehtuuriin.

Looginen näkymä

Looginen näkymä arkkitehtuuriin kuvaa järjestelmän jaon loogisiksi osakokonaisuuksiksi, kuten erikseen paketoitaviksi tuotteiksi, sovelluksiksi tai vaikkapa sovelluskehyksiksi. Loogisen näkymän avulla voidaan jäsentää laajojen järjestelmien ja tuoteperheiden tuoterakennetta, ja sillä voidaan esittää keskeisten osajärjestelmien riippuvuuksia. Loogisen näkymän esittämiseen voidaan käyttää UML:n *pakkauksia* sekä riippuvuuksien esittämiseen riippuvuusnuolia. Järjestelmän osien luonnetta voidaan havainnollistaa *ste-*

reotyyppimäärityksillä, kuten «application» tai «application product». Kuvassa 2 loogista näkymää käytetään kuvitteellisen verkonhallintajärjestelmän tuotejaottelun kuvaamiseen[1].



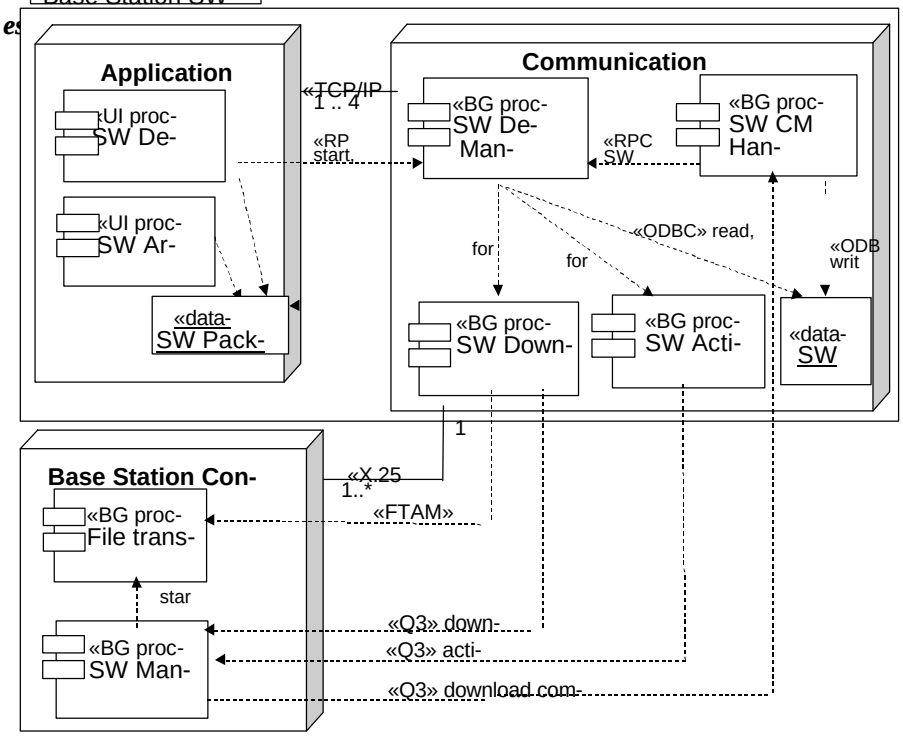
Kuva 2. Loogisella näkymällä voidaan es
ositusta osatuotteiksi.

Yleensä loogista näkymää ei sovelleta aivan pieniin sovelluksiin, vaan lähinnä laajoihin järjestelmiin. Muut seuraavaksi esiteltävät arkkitehtuurinäkymät soveltuvat hyvin kaikenkokoisten järjestelmien suunnitteluun.

Ajonaikainen näkymä

Laajat järjestelmät koostuvat tyypillisesti useista ajettavista ohjelmista, joilla saattaa olla keskinäisiä riippuvuuksia. Tällaisen järjestelmän suunnittelun, testauksen ja ylläpidon kannalta on tärkeää ymmärtää, miten prosessit sijoitellaan eri tietokoneille ja miten prosessit riippuvat toisistaan. Ajonaikainen näkymä kuvaa järjestelmän tietokoneresursseja, ajonaikaisia komponentteja ja niiden riippuvuuksia. Sen esittämiseen käytetään UML:n toteutuskaavioista *sijoittelukaaviota* (deployment diagram).

Esimerkki ajonaikaisesta näkymästä on kuvassa 3. Tässä esimerkkijärjestelmässä on 1 – 4 sovelluspalvelinta (application server) kytkettynä tietoliikennepalvelimeen (communication server), joka edelleen on kytketty vähintään yhteen tukiasemaohjaimeen (base station controller). Ajettavat ohjelmat on kategorisoitu käyttöliittymä- tai taustaprosesseiksi vastaavien stereotyyppien «UI process» ja «BG process» avulla. Nuolet prosessien välillä kuvaavat ajonaikaisia riippuvuuksia, kuten sanomarajapintaa. Kaaviossa esitetään myös tärkeimmät tietovarastot sekä prosessien riippuvuus niistä.



Kuva 3. Ajonaikainen näkymä kuvaa prosessien riippuvuuksia ja allokointia tietokoneille.

Ajonaikainen näkymä kuvaa tiiviisti monimutkaisenkin järjestelmän suorituksenaikaista arkkitehtuuria. Ajonaikaisten riippuvuuksien visualisointi mahdollistaa mm. järjestelmän suorituskyvyn, skaalautuvuuden, luotettavuuden ja modulaarisuuden arvioinnin jo suunnitteluvaiheessa. Ajonaikainen näkymä auttaa mielekkään kehitys- ja integrointijärjestyksen suunnittelua sekä myöhemmin myös virheenjäljitystä. Se ei kuitenkaan kuvaa ohjelmistokoodin staattista eli käännösaikaista rakennetta. Tätä tarkoitusta varten käytetään *kehitysnäkymää*.

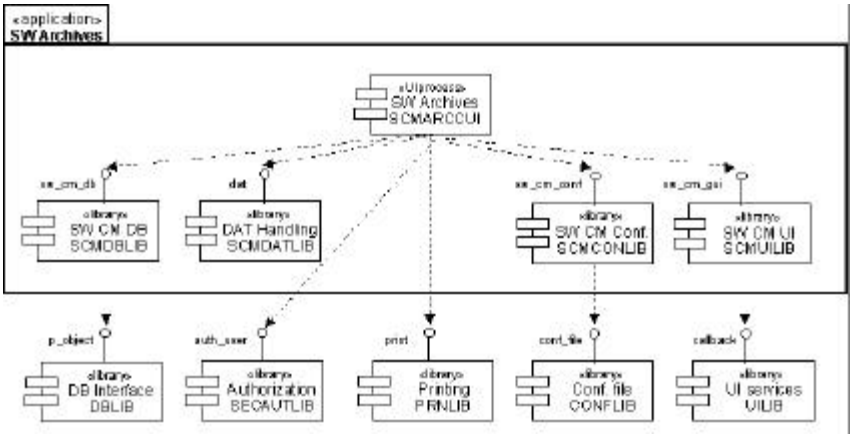
Kehitysnäkymä

Ajonaikaisten riippuvuuksien lisäksi varsinkin laajoissa projekteissa on syytä kiinnittää huomiota ohjelmistokomponenttien[2] riippuvuuksien hallintaan. Kehitysnäkymä kuvaa, miten ajettavat ohjelmat rakennetaan komponenteista. Kehitysnäkymän esittämiseen käytetään UML:n *komponenttikaaviota* (component diagram). Kuvassa 4 on esimerkki kehitysnäkymästä. Siinä esitetään kuvassa 3 esiintyneen käyttöliittymäprosessin *SW Archives* jako komponenteiksi.

Tässä tapauksessa sovellusta varten kehitetään pääohjelma (SCMARCCUI) sekä neljä siihen linkitettävää kirjastokomponenttia (*SW CM Database, DAT Handling, SW CM Configuration File ja SW CM User Interface*). Näiden lisäksi sovellus

[1] Tämän artikkelin esimerkit perustuvat Ari Jaaksin, artikkelin kirjoittajan, Ari Aallon ja Kimmo Vätön juuri ilmestyneessä kirjassa *Tried & True Object Development – Industry-Proven Approaches with UML* [Jaaksi-Aalto-Aalto-Vättö 99] käytettyyn GSM Manager –esimerkkiin.

[2] *Komponentilla* tarkoitetaan tässä toiminnallista koodikokonaisuutta, kuten kirjastoa tai ajokelpoista ohjelmaa, jota käytetään sellaisenaan osana laajempaa ohjelmistoa.



Kuva 4. Kehitysnäkymä kuvaa komponenttien välisiä riippuvuuksia ja rajapintoja.

käyttää viittä valmiskomponenttia, jotka on kuvassa piirretty *SW Archives* –pakkauksen ulkopuolelle. Riippuvuustiedon lisäksi kuvataan kustakin komponentista rajapinta, johon riippuvuus muodostuu.Kun kaavio organisoidaan riippuvuuksien mukaisesti kerroksiksi kuten kuvassa 4, huomataan mahdolliset integrointiongelmia tuottavat sykliset riippuvuudet jo suunnitteluvaiheessa. Esimerkin *SW Archives* –sovellus on esitetyn kehitysnäkymän perusteella hyvin suunniteltu. Jos komponenttien välillä olisi paljon syklisiä ja muita "vääriä" riippuvuuksia, kuva olisi sekavampi nuolien risteillessä komponentista toiseen.

Kehitysnäkymä on tässä esiteltävistä arkkitehtuurinäkymistä keskeisin projektityön kannalta. Komponenttijako muodostaa pitkälti pohjan työnjaolle projekteissa, ja teknisiin komponenttirajapintoihin liittyy useimmiten myös rajapinta ihmisten välillä. Komponenttiriippuvuudet ohjaavat kehitys- ja integrointijärjestystä ja siten projekti- ja testisuunnittelua. Myös jatkokehitykselle ja ylläpidolle kehitysnäkymä on arvokas, sillä se auttaa muutosten laajuuden ja testaustarpeen arvioinnissa.

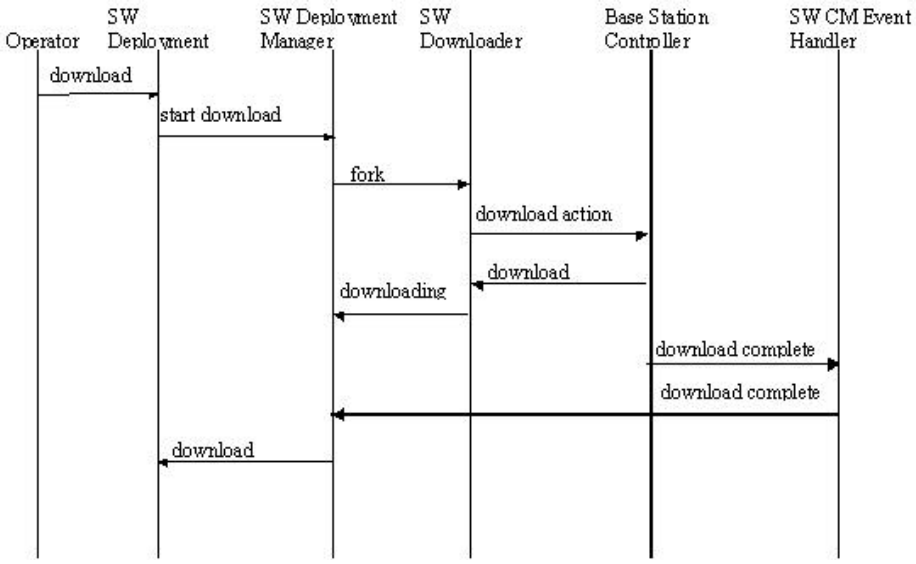
Skenaarionäkymä

Viimeinen, "+1" –näkymä, on skenaarionäkymä, jonka avulla muissa näkymissä olevien elementtien roolia, vuorovaikutusta, rajapintoja ja vastuita voidaan suunnitella ja arvioida. Skenaarionäkymässä käytetään UML:n *sekvenssikaavioita* (sequence diagram), joilla kuvataan järjestelmän osasten toimintaa valittujen käyttötapausten yhteydessä.

Kuvassa 5 on esimerkki sekvenssikaaviosta, jossa tukiasemaohjelmisto ladataan verkonhallintajärjestelmästä tukiasemaohjaimeen. Tämän skenaarion osallistujat ovat ajonaikaisesta näkymästä löytyviä prosesseja, ja nuolet kuvaavat prosessien välistä viestintää. Aika etenee kuvassa ylhäältä alaspäin.

Skenaarionäkymä on skaalautuva: sekvenssikaavioiden osallistujia esittävinä pystypalkkeina voi olla esim. C++ -luokkia, komponentteja kehitysnäkymästä, prosesseja ajonaikaisesta näkymästä tai sovelluksia ja jopa itsenäisiä järjestelmiä loogisesta näkymästä. Näin laajojenkin järjestelmän osien toimintaa voidaan esittää kuhunkin tilanteeseen sopivalla abstraktiotasolla.

Yhteenveto



Kuva 5. Sekvenssikaavio tukiasemaohjelmiston lataamisesta

Esitetyt 3 + 1 UML-pohjaista näkymää arkkitehtuuriin tarjoavat ilmaisuvoimaisen tavan kuvata laajojenkin ohjelmistojen keskeisimpiä arkkitehtuuriratkaisuita. *Loogisen näkymän* avulla voidaan havainnollistaa järjestelmän loogista ositusta kuten esimerkiksi tuoterakennetta. *Ajonaikainen näkymä* kuvaa järjestelmän ajonaikaiset elementit riippuvuuksineen sekä niiden sijoittelun tietokoneille. *Kehitysnäkymän* avulla voidaan hallita projektissa kehitettävien ja käytettävien komponenttien rajapintoja ja riippuvuuksia. *Skenaarionäkymä* auttaa muiden kolmen näkymän valmistelua ja iterointia tarjoamalla näkyvyyttä järjestelmän osien yhteistoimintaan keskeisten käyttötapausten kautta. Näiden näkymien avulla ohjelmistoarkkitehtuuria voidaan suunnitella ja hallita tehokkaasti, mikä luo hyvät edellytykset esimerkiksi iteratiiviselle ja inkrementaaliselle ohjelmistoprosessille.

Viiteet

[Jaaksi-Aalto-Aalto-Vättö 99]
Ari Jaaksi, Juha-Markus Aalto, Ari Aalto, Kimmo Vättö: *Tried & True Object Development – Industry-Proven Approaches with UML*. Cambridge University Press, 1999.

[UML]
http://www.rational.com/uml/index.jtmpl

[Kruchten 95]
Philippe Kruchten: *The 4+1 View Model of Architecture*. IEEE Software November 1995, sivut 42 – 50.

Juha-Markus Aalto,
Nokia Telecommunications,
Network Management Systems