Oliot ja komponentit sulautettujen järjestelmien kehittämisessä

Heikki Honkela,
Software Engineering
Center SEC Oy

Oliopohjainen kehittäminen on valtaamassa alaa reaaliaika- ja sulautettujen järjestelmien kehittämisessä. Oliomenetelmien standardoituminen on luonut hyvän perustan työkalujen kehittymiselle tehostamaan suunnittelua, dokumentointia ja kommunikointia. Oliopohjaisuus luo perustan myös siirtymiselle komponenttipohjaisuuteen, joka parantaa edelleen uudelleenkäytettävyyttä.

UML (Unified Modeling Language) on saanut parissa vuodessa teollisuusstandardin aseman oliomallinnusmenetelmänä. UML syntyi eri oliomallinnusmenetelmien yhteensovittamisen tuloksena vuonna 1997. UML:n kehittäminen jatkuu OMG:n (Object Management Group) suojassa.

Oliopohjaisen kehittämisen mukana UML:sta on tullut myös reaaliaika- ja sulautettujen järjestelmien mallinnusmenetelmä. UML:n välineet eivät kuitenkaan riitä laiteläheisten järjestelmäelementtien suunnitteluun. Niinpä markkinoille onkin tullut koko joukko UML-pohjaisia välineitä, joihin on tehty sulautettujen järjestelmien suunnittelua palvelevia laajennuksia.

UML:n standardoinnista vastaava OMG on käynnistänyt vastikään hankkeen reaaliaika-UML:n standardoimiseksi. Standardointiprosessi vie oman aikansa. Sen kuluessa välinekehitys jatkuu voimakkaana. Standardointiprosessista onkin odotettavissa tiukka kädenvääntö, sillä eräät välinevalmistajat ovat jo nyt ehättäneet vauhdittamaan markkinointiaan kertomalla heidän välineensä laajennusten tulevan standardiksi.

Reaaliaikajärjestelmän erityispiirteet

Seuraava tarkastelu perustuu yhden työvälineen, Artisan Real-time Studio, tapaan toteuttaa UML:n reaaliaikalaajennukset. Kuvassa 1 on UML:ään sisältyvät tekniikat kuvattu ympyröinä ja Artisanin laajennukset suorakaiteina. Edellä mainitut asiat on ryhmitelty kuvassa eri tarkastelukulmien mukaisesti.

Reaaliaikajärjestelmän luonteeseen kuuluu reagointi ympäristön ärsykkeisiin. Reagointi edellyttää laitteen ja ohjelmiston välistä kommunikointia, joka ilmaistaan vasteaika- ja reaaliaikaisuusvaatimuksina.

Real-time Studiossa järjestelmän lähtökohdat kuvataan vaatimusarkkitehtuurina, jonka sisältää seuraavat osat:

- Rajauskaavio (scope, context diagram): Kuvaa järjestelmän käyttäjät ja näiden kommunikointitavat järjestelmän kanssa.
- Reunaehdot (constraints): Järjestelmälle asetetut vasteaika-, reaaliaikaisuus-, luotettavuus-, ym. ei-toiminnalliset vaatimukset.
- Järjestelmän käyttö (system usage): Kuvataan UML:n käyttötapausnotaatiolla.
- Järjestelmän tilakaavio (system modes): Tilakaaviotekniikalla toteutettu kuvaus järjestelmän tiloista ja tilasiirtymistä niiden välillä.

UML ei sisällä arkkitehtuuritasoisia kuvaustekniikoita. Vain sijoittelukaaviota (deployment diagram) voidaan pitää jossakin määrin sellaisena. Jotta laite- ja ohjelmistosuunnittelu voisivat tapahtua rinnakkain, tulee laitteiden ja ohjelmistojen rajapinnat kuvata yksityiskohtaisesti.

Artisanin ratkaisu perustuu 3-tasoiseen arkkitehtuuriin (kuva 2), joita ovat:

1. Olioarkkitehtuuri (rajapinnat, kontrollit, tietoluokat)
2. Ohjelmistoarkkitehtuuri (rinnakkaisuus, tallennusrakenne)
3. Sovellusarkkitehtuuri (järjestelmän toteutus laitetasolla).

Kuva 1: ARTiSAN Real-Time Studio tarjoaa kattavan, yhteisen työpöydän.

Moniajotehtävien rinnakkaisuuden kuvaamiseen käytetään DARTS-notaation mukaista tieto- ja tapahtumavirtakaaviota, jota nimitetään Concurrency Diagramiksi. Kaavio täydentää UML:n yhteistyökaaviota (Collaboration Diagram) kuvaten tehtävien väliset yhteydet. Kuvaus laaditaan yksittäisen prosessorin tasolla.

Suunnittelijan työpöytä

Oikein valittuna suunnittelutyökalut muodostavat suunnittelijan työpöydän tai oikeastaan ohjelmistotuotannon yhteisen työpöydän. Jotta työpöytä olisi yhteinen, sen tulee tarjota työkalut myös eri näkökulmien edustajille.

Seuraavassa on tarkasteltu suunnittelijan

Kuva 2: Artisanin ratkaisu reaaliaikajärjestelmien arkkitehtuurin kuvaamiseen.

työpöydän sisältöä.

Suunnittelutyökalut

UML:n myöstä oliomallinnustekniikat ovat standardoituneet ja suunnittelutyökalujen menetelmäpohja yhtenäistynyt. Hyvän suunnittelutyökalun tulee tarjota tuki erilaisille lähestymistavoille. Yksi tykkää piirrellä, toinen jäsennellä. Edellistä työskentelytapaa tukee mahdollisuus laatia kaavioita, jälkimmäistä mahdollisuus tallentaa jäsentelyn tulokset kuvauskantaan ja generoida niistä kaaviot.

Järjestelmien koon ja monimutkaisuuden kasvaessa on kohteen jakaminen pienempiin suunnittelu- ja toteutuseriin tullut entistä tärkeämmäksi. Toteutuksen tekeminen pienemmissä erissä parantaa hallittavuutta ja vähentää siten epäonnistumisen riskiä. Hyvän välineen tulee tarjota tuki inkrementtipohjaiselle kehittämiselle. Kuvassa 3 on Real-time Perspectiveen perustuva esimerkki inkrementtisuunnittelun tehtävistä.

Generaattorit ja reverse-työkalut

Lennokkaimmat markkinointimiehet ovat intoituneet esittämään, että heidän työkalullaan ohjelmakoodi voidaan generoida yhdellä napin painalluksella ja siten nostaa sovelluskehityksen tuottavuutta yli satakin prosenttia. Tuo nappi odottaa edelleen keksijäänsä. Toki koodia on voitu ja voidaan generoida koneellisesti. Yhteensopivuudessa suunnittelutyökalujen tuotoksen kanssa ja kaksisuuntaisuudessa on kuitenkin ollut toivomisen varaa kuten myös generoidun koodin tehokkuudessa.

Tänä päivänä sekä koodin generointi että reverse engineering toimivat. Toteutusvälinekehitys on kuitenkin johtanut siihen, että kilpajuoksu jatkuu. Generaattorien kehityksen painopiste on tänä päivänä C++:n ohella CORBA- ja Java-puolella. Edellä mainitut eivät ole ainakaan tähän mennessä olleet yleisemmin käytössä sulautettujen järjestelmien toteutuksessa. Kehitys on kuitenkin saanut yritykset tarkistamaan kantaansa.

Työkalujen integrointi

Oleellisena osana yhteistä työpöytää on työkalujen integroitavuus muihin työkaluihin. Parhaiten tämä tapahtuu avointen rajapintojen avulla. Tahdistimet (synchronizer) tarjoavat automatisoidut rajapinnat. Niiden avulla saadaan erikoistyökalut integroiduksi; ei välttämättä parhaalla mahdollisella tavalla mutta toimivasti. Harvoinhan yhdeltä valmistajalta on saatavissa sopivimmat työkalut koko työpöydälle.

Kuvauskanta

Monipuolisimmat välineet perustuvat kuvauskantaan, johon kuvaukset tallennetaan. Parhaan tuen projektimuotoiselle työskentelylle tarjoavat välineet, joissa on aito monenkäyttäjän kuvauskanta. Kuvauskannan tulee olla skaalattavissa työasemakohtaisesta hajautetuksi verkkoratkaisuksi. Yhteinen kuvauskanta on keskeinen osa yhteistä työpöytää.

Kuvauskannassa kuvaukset säilyvät tulevaa tarvetta varten. Esimerkiksi uuden version suunnittelun perustaksi saadaan aiemman version mallit, jotka voidaan päivittää ohjelmista reverse-ominaisuudella. Näin suunnittelupohja saadaan vastaamaan ylläpidettyjä ohjelmia. Malleja voidaan kopioida myös toisten projektien perustaksi; pyörää ei ole pakko keksiä uudelleen.

Kuva 3: Inkrementtikehittämisen tehtävät Real-time Perspectiven mukaan.

Dokumentointi

Kuvauskanta on lähtökohta järjestelmädokumentaation ajan tasalla pitämiseen. Kun väline tukee jäsentyyppikohtaisten kuvausten laatimista ja tallentamista, niin dokumentointi ei vaadi kirjailijan taitoja. Riittää, kun kustakin kohteesta tallennetaan oleelliset asiat.

Yhteiskäyttöinen kuvauskanta mahdollistaa tarvittavan dokumentaation tuottamisen milloin vain juuri sen hetkisestä tilanteesta. Dokumentin voi tuottaa kuka tahansa, jolla on kuvauskannan käyttöoikeus.

Ei riitä, että kuvauksia voi selata työasemalla. Projekteissa on tilanteita, joissa tarvitaan paperidokumentteja esim. hyväksyttämistä varten. Tietyn tilanteen dokumentaatiota tarvitaan myös myöhemmän verifioinnin perustaksi. Dokumenttigeneraattorilla kuvauskannan sisältö voidaan halutuilta osiltaan siirtää teksinkäsittelyyn.

Komponenttikehittäminen

Komponenttipohjainen kehittäminen on seuraava ohjelmistotuotannon kehittämistrendi. Komponentin perusidea on kehittämisen nopeudessa sekä yksittäisen ohjelmiston yli menevässä yleistämisessä ja uudelleenkäytettävyydessä. Komponentteja voidaan liittää toisiinsa joustavasti ja siten käyttää kokonaisuuden osina joustavammin ja vähemmällä työllä kuin perinteisiä ohjelmistorakenteita.

Komponentit on kuitenkin ensin kehitettävä: suunniteltava, toteutettava ja julkaistava. Komponentteja voidaan hankkia myös ulkopuolelta. Hankittujen komponenttien liitettävyyden varmistamiseksi on järjestelmä ensin suunniteltava vaatimuslähtöisesti ja kuvattava yhtenäisesti.

UML soveltuu yhtenäiseksi kuvaustavaksi. Se tarjoaa kuitenkin parhaimmillaankin vain joukon toisiaan tukevia kuvaustekniikoita, joiden tavoitteena on olioiden määrittely, suunnittelu ja toteuttaminen. Komponenttikehittäminen tuo olioiden päälle lisäelementin, jonka avulla olioita voidaan tehokkaasti hallita. Onnistunut komponenttikehittäminen vaatii ajattelutavan muutosta - oliot eivät ole komponentteja.

Siirtyminen olioista komponentteihin tapahtuu joustavasti, mikäli käytettävä oliomallinnusväline tukee komponenttikehittämistä. Siten siirtyminen olioista komponentteihin voi tapahtua asteittain, yksinkertaisesti olioita paketoimalla.

Lopuksi

UML tarjoaa yhtenäiset kuvaustekniikat järjestelmän rajauksesta metatasoiseen koodin määrittelyyn. Näistä lähtökohdista tuotetun koodin pitäisi olla paljolti toteuttajasta riippumatonta. Toisiinsa sopivat kuvaustekniikat edesauttavat myös järjestelmän lähtökohtana olevien vaatimusten hallintaa ja vähentävät siten epäonnistumisen riskiä. Ainakaan se ei ole kiinni menetelmistä ja välineistä.

Heikki Honkela,
Software Engineering Center SEC Oy