MetaEdit+:n käyttö kehitysmenetelmän mallintamisessa

Miika Nurminen (minurmin@jyu.fi)
Annemari Auvinen (annauvi@jyu.fi)

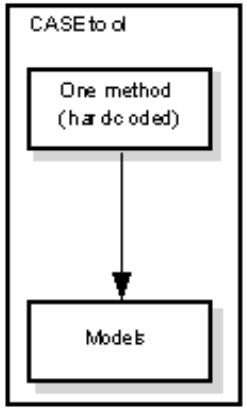
TJTST10 Tietojärjestelmien kehittämismenetelmät -seminaarityö
3.11.2008

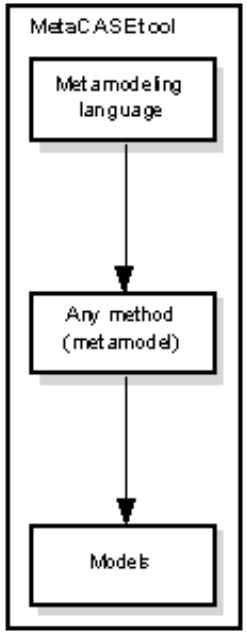
Sisältö

- Kehitysmenetelmistä ja CASE-työvälineistä
  - CASE vs MetaCASE-välineet
  - Näkökulmia menetelmiin
  - Menetelmäkehitys vs DSM
- MetaEdit+
  - GOPRR-metametamalli
  - UML-esimerkki
  - MetaEdit+:n arviointia

CASE vs MetaCASE-välineet

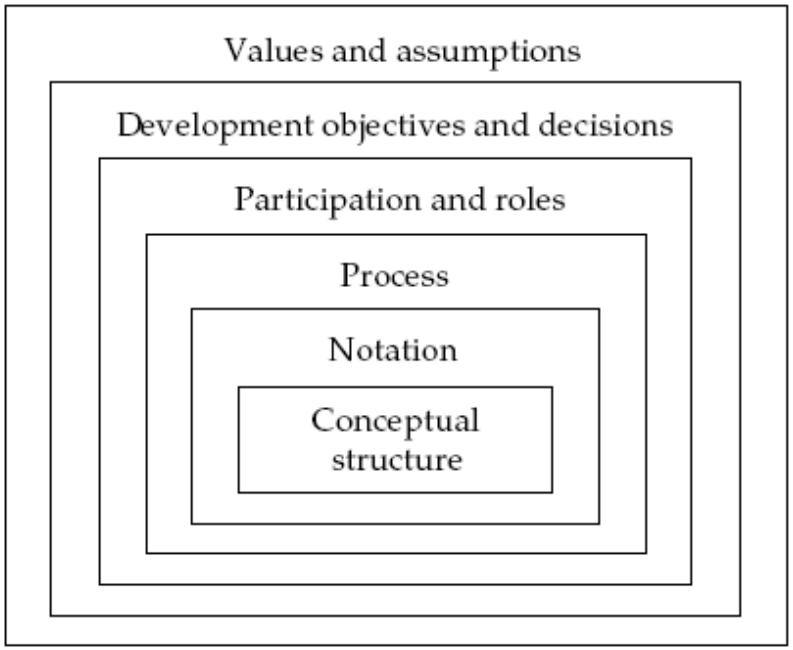
- CASE-työkalut yleensä suunniteltu vain tiettyä suunnittelumenetelmää silmälläpitäen
  - Tietty CASE-ympäristö hankittaessa kehitysorganisaation oli sopeuduttava sen käyttämään malliin.
  - CASE-ympäristöjen integrointiaste ja tietojen siirrettävyys huono
- UML-kieli on auttanut asiaa yhtenäistämällä notaatioita, mutta
  - työkalujen yhteensopivuus on XMI:sta huolimatta edelleen puutteellista
  - UML:n käyttö sovellusalueen mallinnuksessa voi olla ongelmallista
- MetaCASE-työkaluissa ideana sisällyttää samaan työkaluun sekä menetelmän mukaisten kuvausten editointi että muokattava metamalli, joka kuvaa malleissa käytettävän menetelmän (=pääosin notaation).
  - MetaEdit oli ensimmäinen graafisen metamallinnus-ympäristön sisältävä MetaCASE-työväline.
  - Metamallit helpottavat menetelmien integrointia ja ositusta komponentteihin.





Näkökulmia menetelmiin

- Menetelmään liittyvästä tietämyksestä voidaan erottaa useita eri abstraktiotasoja tietämyksen tyypin mukaan
- Metamallipohjainen menetelmän kuvaaminen keskittyy pääosin menetelmän käsitteelliseen rakenteeseen ja jonkin verran notaatioon ja prosessiin, mutta muut menetelmätietämyksen tyypit jäävät useimmiten metamallintamisen ulkopuolelle
  - Käytännössä projektin omaksumilla käytännöillä ja muilla ei-teknisillä tekijöillä on perustavanlaatuisempi merkitys kuin dokumenteissa käytetyllä notaatiolla
  - Onko oikeutettua puhua MetaCASE-ympäristön malleista *menetelmä*kuvauksina?
- Metamallit mahdollistavat:
  - Abstraktiotason nostamisen
  - Syntaktiset tarkastukset (esim. mallin yhtenäisyys)
  - Mallien muunnokset
  - Semanttiset tarkastukset (esim. katselmoinnit)



Menetelmäkehitys vs DSM

- MetaCASE-työkalua voi käyttää sekä menetelmäkehitykseen että sovellusaluepohjaiseen mallinnukseen (DSM)
- Menetelmäkehityksessä käytetään yleisesti käytettyjä mallinnuskieliä (UML, ER, SA/SD, BSP jne) mahdollisesti yhdistäen tai muokaten mallinnuselementtejä
  - Perinteisessä menetelmäkehityksessä mallinnuselementit voivat ovat liian lähellä toteutusta
  - Esim. UML:n käyttö on luontevaa sovelluskehittäjille, mutta sovellusaluetta analysoitaessa sen geneerisyys aiheuttaa ylimääräisen tulkinnallisen kerroksen
  - UML:stä ei nykyisillä työvälineillä voida käytännössä (MDA:n lupauksista huolimatta) generoida merkittävää määrää sovelluksen koodia – ellei suunnittelumallista tehdä oleellisesti yhtä monimutkaista kuin ohjelmakoodista itsestään.
- Sovellusaluepohjaisessa kielessä metamalli luodaan alusta alkaen sovellusalueen pohjalta yhteistyössä sovellualueen asiantuntijoiden kanssa
  - Sovellusaluemallista on mahdollista generoida merkittäviä määriä lopullisen sovelluksen koodia, mikä parantaa ohjelmistotyön tuottavuutta
  - Mutta: ennen kuin koodia voidaan generoida, on toteutettava raporttigeneraattori, mikä on oleellisesti monimutkaisempaa verrattuna yksittäisen sovelluksen koodaamiseen perinteisin keinoin (vrt. kääntäjän vs. sovelluksen toteuttaminen). Hyödyt ovat olemassa, mutta näkyvät vasta, jos sovellusalueelle toteutetaan *useita* sovelluksia samalla generaattorilla
  - Lisäksi integraatiota varten abstrahoitava ajoympäristön ja ”tavallisten” komponenttien koodi toteuttamalla sovellus*alue*kehys, jota generoitu koodi kutsuu (vrt. sovelluskehys).

MetaEdit+

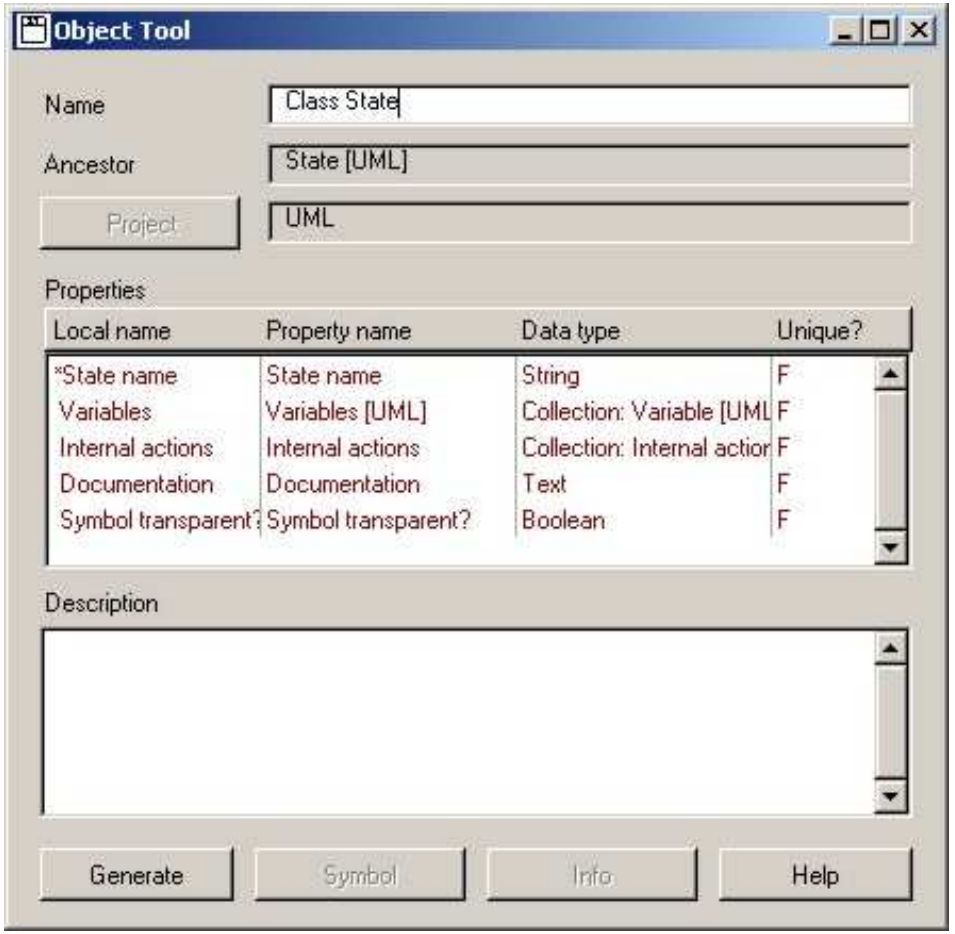
- Koostuu kahdesta osasta:
  - MetaEdit+
  - MetaEdit+ Method Workbench
- GOPRR-datamalli:
  - **Objekti** (*object*) on asia, joka on olemassa riippumatta suhteista tai rooleista, esimerkiksi *Class*.
  - **Ominaisuus** (*property*) on kuvaava tai määrittelevä ominaisuus, esim. *nimi*.
  - **Suhde** (*relationship*) on yhteys objektiryhmien välillä, esim. *Data Flow*. Suhde liitetään objekteihin roolien kautta
  - **Rooli** (*role*) määrittelee, kuinka objektit osallistuvat suhteessa, esim. rooleja ovat *Flows from* ja *Flows to.*
  - **Graafi** (*graph*) on joukko objekteja, suhteita, rooleja ja näiden liitoksia (*bindings)*. Graafi osoittaa, mitkä objektit suhde yhdistää minkäkin roolin kautta, esim. *Data Flow Diagram* ja *Object Diagram.*
  - GOPRR-metatyyppejä käytetään sekä tyyppi- että ilmentymätasolla.

Menetelmäkehitys MetaEdit+:ssa

- menetelmiä voidaan muokata ja olemassa olevia metodimäärityksiä uudelleenkäyttää kehitettäessä uusia
  1. Identifioidaan ja määritellään menetelmän objektityypit ja niiden ominaisuudet.
  2. Identifioidaan menetelmän suhdetyypit ja määritellään niiden ominaisuudet.
  3. Määritellään menetelmän roolityypit ja niiden ominaisuudet.
  4. Määritellään tarvittavat symbolit objekteille, suhteille ja rooleille.
  5. Määritellään graafityypit ja lisätään niihin objekti-, suhde- ja roolityypit.
  6. Määritellään graafityypeissä suhteiden liitokset.
  7. Määritellään rajoitteet objekteille, jotka osallistuvat suhteisiin tai rooleihin.
  8. Määritellään graafityypissä objektityyppien tarkentavat linkit graafien välille sekä hajotelmat
  9. Määritellään tarkistukseen, mallien dokumentoimiseen ja koodin generointiin liittyvät raportit.
  - Näitä vaiheita voidaan suorittaa iteratiivisesti ja osittain samanaikasesti. Tyyppien määrityksiä voidaan myös muokata myöhemmin.

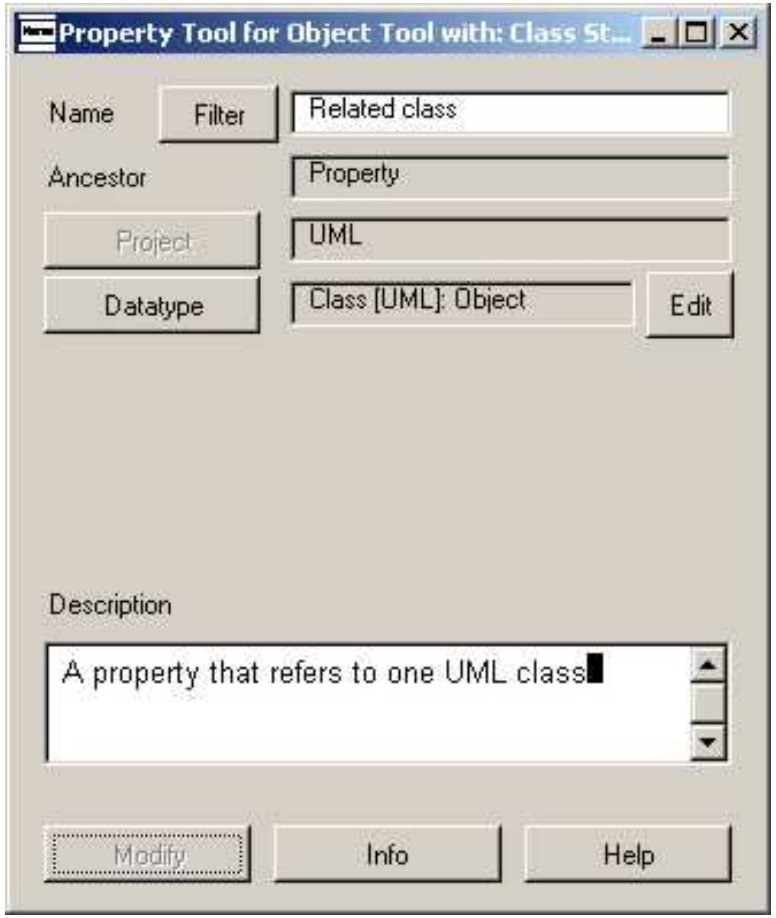
UML-esimerkki (1/6)

- Laajennetaan State Diagram –menetelmää laajentamalla State-tyyppiä niin, että se sisältää myös tiedon luokasta, jota tila kuvaa.
- *Objektityypin määrittely:*
  - Tehdään uusi State-objektityyppi tilakaavioon. Käytetään olemassa olevaa UML:ssä käytössä olevaa State-tyyppiä ja muokataan sitä. Objektityökalussa valitaan pohjaksi State [UML] ja määritellään objektille nimi Class State. Ominaisuuksissa näkyy kaikki State [UML] -objektin ominaisuudet.



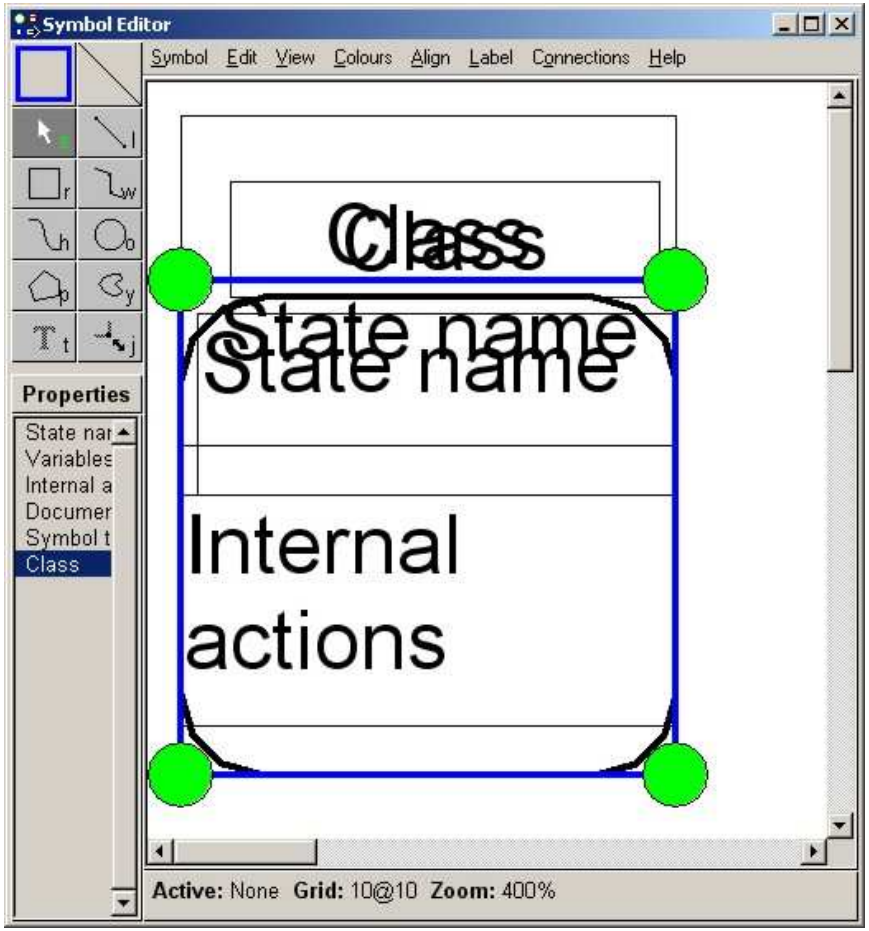
UML-esimerkki (2/6)

- *Lisätään uusi ominaisuus objektille*:
  - Lisätään ominaisuuslistaan ominaisuus, jonka arvo viittaa Class [UML] -objektiin. Annetaan ominaisuudelle nimi Related class ja määritellään ominaisuuden datatyypiksi **Object** ja Class [UML]. Kuvauksen määrittämisen jälkeen ominaisuus voidaan luoda **Generate**-painikkeella. Objektityyppi voidaan luoda **Generate**-painikkeella.



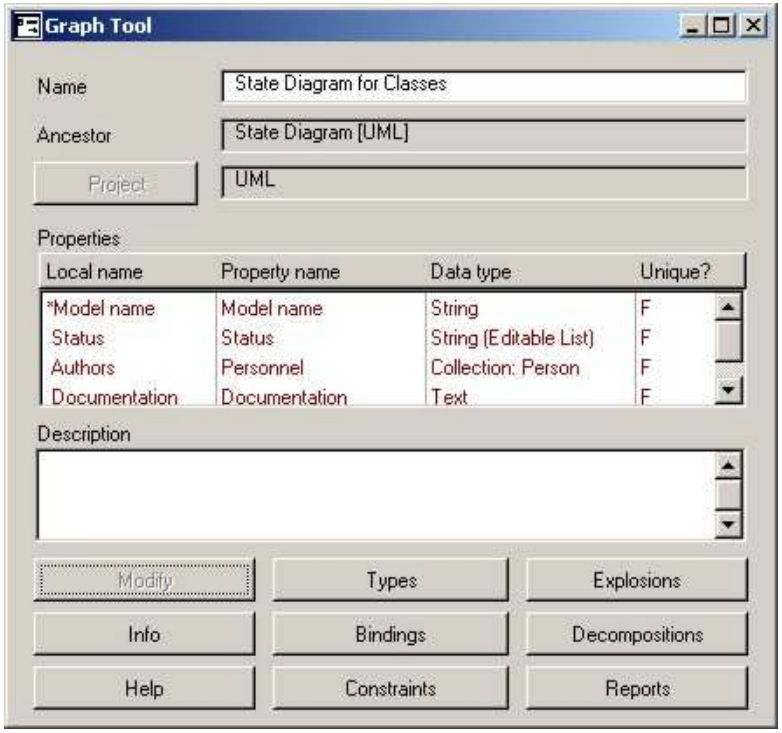
UML-esimerkki (3/6)

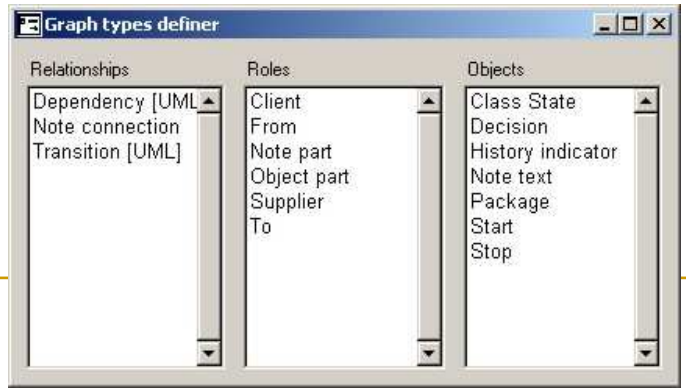
- *Määritellään objektille symboli:*
  - Objektityökalun **Symbol**-painikkeella päästään määrittelemään symboli. Vasemmasta reunasta löytyvästä työkalurivistä lisätään halutut kuviot ja siirretään kuvioon alhaalta löytyvät, tarvittavat ominaisuudet Class, State name ja Internal actions. Lopuksi määritellään yhteyspisteet.



UML-esimerkki (4/6)

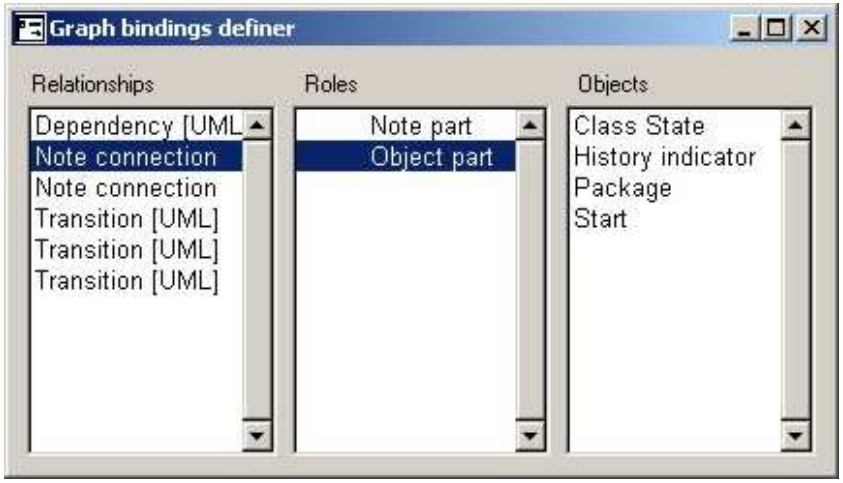
- *Määritellään suhdetyypit ja roolit:*
  - Suhteita tai rooleja ei tässä tapauksessa tarvitse muutella. Aukeava ikkuna on samanlainen kuin objektityypin määrittelyssäkin ja myös suhteille ja rooleille voidaan määritellä symbolit symbolieditorilla.
- *Määritellään graafityyppi:*
  - Graafityypin määrittelemiseksi aukaistaan graafityökalu.Valitaan tyypiksi State Diagram [UML]. Annetaan graafille nimi, kuvaus ja luodaan se **Generate**-painikkeella. Tämän jälkeen voidaan **Types**-painikkeella avautuvassa ikkunassa määritellä, mitä objekti-, suhde- ja roolityyppejä käytetään metodissa. Lisätään Class State -objekti ja poistetaan State [UML] objektilistasta.





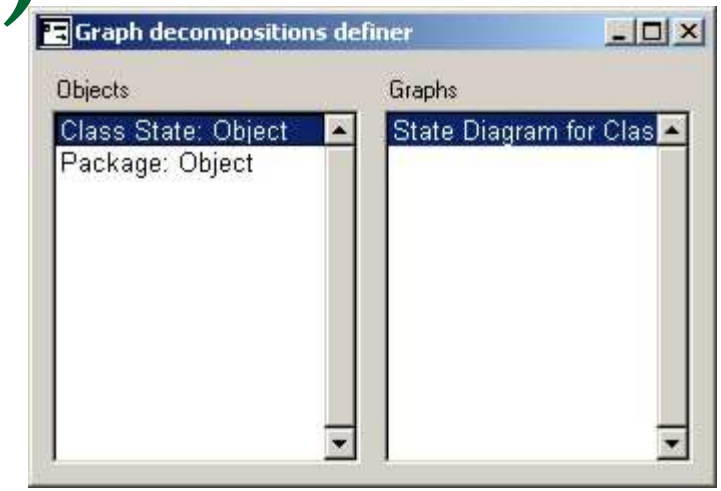
UML-esimerkki (5/6)

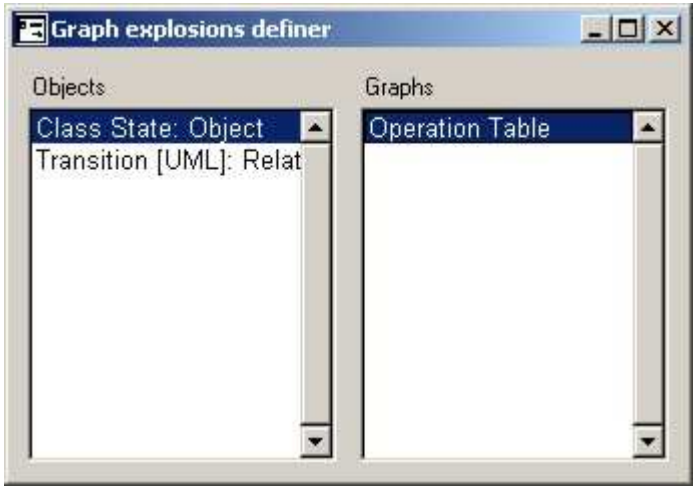
- *Määritellään liitokset eli yhteydet objektityyppien välillä:*
  - Jokainen liitos sisältää yhden suhdetyypin, vähintään kaksi roolityyppiä ja yhden tai useamman objektityypin jokaiselle roolille. Liitos siis määrittelee, mitkä objektit voivat olla missäkin roolissa tietyssä suhteessa.
  - Liitostyökalu saadaan auki graafityökalun **Bindings**-painikkeella. Lisätään Note connection –suhteen Object part -roolin objektiksi Class State ja poistetaan sieltä State [UML]. Nyt State [UML] on korvattu yhdessä liitoksessa uudella Class State -objektilla. Myös muut State [UML]-objektit on korvattava Class State -objektilla jokaisen liitoksen jokaisessa roolissa.



UML-esimerkki (6/6)

- *Valitaan mahdolliset objektien, suhteiden ja roolien tarkentavat linkit graafien välille ja mahdolliset objektien hajotelmat:*
  - Tarkentavat linkit graafien välillä määrittelevät, mihin graafityyppiin tietyn tyyppiset objektit, suhteet ja roolit voidaan linkittää. Hajotelmat määrittelee, mitkä objektityypit voidaan toiminnallisesti osittaa.
  - Decomposition-ikkuna saadaan auki graafityökalun **Decompositions**-painikkeella. Lisätään luotu Class State objektilistaan ja oistetaan State [UML] objektilistasta. Lisätään graafit valitsemalla uusi Class State –objekti aktiiviseksi objektilistasta ja lisätään graafilistaan State Diagram for Classes. Muutetaan Package-objektin hajotelma osoittamaan uuteen State Diagram for Classes. Poistetaan vanha State Diagram [UML] graafilistasta.
  - Korvataan kaikki tarkentavat linkit vanhasta State [UML]-objektista uuteen Class State -objektiin. Määrittelyyn päästään graafityökalun **Explosions**-painikkeella. Poistetaan State [UML] objektilistasta ja lisätään Class State. Graafilistaan lisätään objektille Operation Table.
- *Lopuksi painetaan Modify-painiketta graafityökalussa ja hyväksytään muutokset.*
- *Menetelmä on valmis käytettäväksi:*
  - Käynnistetään jokin editoreista (Diagram, Matrix tai Table) ja luodaan uusi State Diagram for Classes –graafi.





MetaEdit+:n arviointia

- Etuja:
  - Selkeä ja ilmaisuvoimainen metametamalli, jota voidaan soveltaa sekä yleisiin menetelmäkuvauksiin että sovellusaluemalleihin
  - Ei-teknisten käyttäjien helposti ymmärrettävissä oleva sovellusaluemallinnus
  - Integrointimahdollisuudet: API- ja Web service –tuki, XML-tuonti/vienti
  - Graafeja voidaan kuvata kaavioina, taulukkoina ja matriiseina riippuvuudet säilyttäen
  - Monimuotoiset raporttipohjat eri formaateissa (mm. html, doc)
  - Ryhmätyön tuki
  - Omia käsitteitä ja sääntöjä voidaan päivittää myös kehitysaikana
- Heikkouksia:
  - Käyttöliittymä (sekä yleisen käytettävyyden että tuettujen kaaviotyyppien osalta)
  - Sovelluksen koodin generointi vaatii erityisasiantuntemusta ja on työlästä
  - 1-suuntainen generointiprosessi: ei käänteistekniikkaa tai ”roundtrip” engineeringiä
- Kritiikkiä MDE-työkalujen (Enterprise Architect, MetaSketch, MetaEdit+ ja Microsoftin DSL-työkalut) vertailussa (Saraiva & de Silva 2008) :
  - Vaikka MetaEdit+ perustuu yksinkertaiseen ja joustavaan GOPRR-meta-metamalliin, niin se ei sisällä käyttäytymiseen liittyviä ominaisuuksia, mikä vaikuttaa niiden metamallien joukkoon, joita työkalulla voidaan määritellä.
  - MetaEdit+ ei myöskään tue mallien transformaatioita, mutta se tarjoaa raportointitavan, jolla voidaan luoda tekstipohjaisia tuotoksia mallin kuvauskannan tiedoista.
  - Vertailluista työkaluista ainoastaan MetaEdit+ ja Enterprise Architect eivät tukeneet metamallimääritysten vientiä.