Reverse Engineering

I dette dokumentet har vi blandet norsk og engelsk (kopiert fra brukermanualene).

Før man kan kjøre ”reverse engineering” på den eksisterende kildekoden (HSS), må følgende steg være gjennomført:

Java Code Generation og JDK 1.2 small

Dette må gjøres for at Tau UML skal skjønne at nå er det Java som gjelder, samt hvilke Java JDK klasser som gjelder. Ved at dere utfører følgende liste blir Tau UML Suite klargjort for både ”reverse engineering” og kodegenerering:

Activate the Java Code Generation module

The Java Code Generation module provides the commands, the properties, and the Tcl code required to use the Java Code Generator. For more information about modules, see “Activating Modules”. In your project add the following two phases in one configuration: ObjectDesign and Implementation.

To see which modules are currently active: In the Browser, move to the project. Select Utilities | Show Active Modules. The UML suite lists the currently active modules.

If the Java Code Generation module is not listed, you must activate it. To activate the Java Code Generation module: On project level in the xyzJava project, select Utilities | Customize. In the Category dialog, select Module Customization and click Next. In the Level dialog, select “Project:dittProsjektnavn” and click Next. Click Finish to create the customization file.

The Module Availability Editor opens. Select Edit | New. In the Select Module dialog, select Java Code Generation and click OK. In the Module Availability Editor, select File | Save. The Java Code Generation module is now active.

Close the Module Availability Editor. In the Browser, move to Corporate level, and then back to Project level in the xyzJava project. Doing so activates your Project-level customization file. Select Utilities | Show Active Modules to confirm that the Java Code Generation module is now active. Add the JDK classes: The JDK provides a number of classes that are necessary for writing Java code. You often need to reference these classes when modelling your applications.

In the UML suite, you can create each JDK class as you need it, or you can create all of the classes at once using a UML suite module.

To add the JDK classes to the ObjectDesign phase of your project: In the Browser, move to the project. On project level in your project, select Utitlities | Customize. In the Category dialog, select Module Customization and click Next. In the Level dialog, select Current Level (Project) (or “Project:dittProsjektnavn”) and click Next. Click Finish to create the customization file.

The Module Availability Editor opens. Select Edit | New. In the Select Module dialog, select JDK 1.2 Classes (Small) and click OK. In the Module Availability Editor, select File | Save. The JDK 2.0 (small) classes are now activated.

Close the Module Availability Editor. In the Browser, move to Corporate level, and then back to Project level in your project. Doing so activates your Project-level customization file. Select Utilities | Show Active Modules to confirm that the JDK 2.0 (small) module is now active.

Move to the ObjectDesign phase in the project. Select Java | Import JDK (small). Over the next several minutes, the UML suite adds a number of packages to the ObjectDesign phase. These packages define the JDK classes. Meanwhile, take a cup of coffee!

Nå er du klar for "reverse engineering"

Lag en ny pakke (package) i objectdesign fasen din med f.eks. navnet HSS. Gå til denne folderen. Gjør så følgende menyvalg:

Java | Reverse Engineering

I det nye vinduet som popper opp; navigér deg frem til katalogen "model". Valgte du å lage katalogen inf3120, er path'en til katalogen "model": *hom/brukernavn/inf3120/model*. Velg alle .java filene i denne katalogen slik at disse havner i Selection-feltet i popup vinduet. Trykk på "OK" knappen. I det vinduet som nå popper opp skal du gjøre følgende:

"Diagram prefix" skal være satt, skriv f.eks. HSS
"Generate" Cds and CDMs
"Placement algorithm" Grid placement
"Number of classes per cd": 20
"Reverse engineer association" skal være aktiv (dvs. på)
"Code injection" skal være aktiv (dvs. på)

I "Monitoring"-vinduet skjer det deretter mye. Det viktige er at det lages klasser og diagrammer. Om du ser i HSS-pakken i Tau UML vil du nå se at det har kommet mange diagrammer. Det er i et eller flere av disse diagrammene endringene til den eksisterende koden skal finne sted.

Generere kode

Når du har endret ferdig koden din og skal generere koden for det nye UML diagrammet gjør du følgende:

Først markerer du prosjektet ditt og velger så: File | Properties | Edit

I feltet "File System Path Part" legger du inn path'en til katalogen hvor de genererte .java filene skal lagres: f.eks. /hom/brukernavn/HSS

Gå til "Implementation" fasen i prosjektet ditt og gjør følgende:

Java | Generate | Specific New Packages..

I det vinduet som kommer opp velger du den pakken der klassediagrammene dine ligger. Trykk "OK" og "Monitoring" vinduet dukker opp. Om det ikke eksisterer noen feil i diagrammene dine vil det bli generert Java-kode. Hvis ikke må du rette opp feilene. En feil som kan oppstå er at Tau UML ikke skjønner at Serializable er et interface, feilmeldingen vil da se slik ut:

Illegal realization between class 'WardRegister' and class 'Serializable' that is not an interface.

For å fjerne denne åpner du klassediagrammet, høyre-klikker klassen "Serializable" og velger "Edit properties". I det nye vinduet som kommer frem velger du "Interface" under "Stereotype". (Om du ikke får valgt "Stereotype" trykker du på "Define Item" hvis denne finnes i den nåværende versjonen av Tau UML Suite. Det gjør det i så fall mulig å velge "Stereotype".)

Lagre diagrammet og generer kode på nytt ved å markere pakken HSS under "Implementation" og velg: Java | Generate | All

Husk å legge til ”package model;” i alle filene
Ingen er perfekte, selv ikke Tau UML generatoren. Ved ”reverse engineeringen” fjernet den en linje i alle .java filene som ble importert. Dette må dere fikse opp i. Øverst, dvs. før noen andre linjer i alle .java filene i katalogen ”modell” må dere legge til linjen: package model;
Dette må gjøres for å opprettholde 3-lags arkitekturen.