

Globale Aanpak UML Modellingering

Hieronder een aantal stappen voor het modelleren van de requirements (in de Use Case View van Rational Rose) en het ontwerp (in de Logical View van Rose). In het algemeen gaat het om een iteratief proces, dwz de hieronder beschreven stappen worden vaak een aantal maal herhaald om tot een goed resultaat te komen.

Stappen voor de Use Case View

1. Maak een *Use Case Diagram*, met
 - a. Een *package* voor hele systeem; er binnen komen de use cases en daarbuiten de actoren (als in een context diagram).
 - b. *Use Cases*: samenhangende delen functionaliteit, doelen van het systeem, moet zichtbaar effect hebben op omgeving van systeem (actoren)
 - c. *Actoren*: omgeving van het systeem, waar heeft de software die gemaakt moet worden interactie mee?
 - d. *Relaties* ertussen actoren en use cases (en eventueel tussen actoren of use cases onderling). Bv wie initieert use case, of hoe is de informatie flow (pijlen ..); gebruik stereotypes (<< .. >>) om de specifieke betekenis van de relaties aan te geven.
Voorzie alle use cases en actoren van commentaar.
2. Geef *details van actoren*; geef *interface* (operaties/events); gebruik eventueel een toestandsdiagram voor het gedrag van de actor. Denk ook aan foutgedrag.
3. Maak *sequence diagrammen* per use case om typische interacties tussen actoren en systeem aan te geven (maak klasse "The System").
4. Maak *toestandsdiagram(men)* voor het globale gedrag of specifieke use cases; vooral om de volgorde van de belangrijkste activiteiten of bijvoorbeeld de veranderingen van modes weer te geven.

Zie ook [artikel over de ROPES methode](#), Requirements Engineering, p 20 t/m 24 (tot Systems Analysis).

Stappen voor de Logical View

1. Identificeer essentiële objecten, abstraheer daarvan om *klassen* te identificeren. Dit kan bv door te zoeken naar zelfstandige naamwoorden in de informele specificatie; ook hebben actoren in de use case view vaak een representatie in het klasse diagram. Geef aan wat de verantwoordelijkheid is van elke klasse.
2. Maak *sequence diagrammen* voor typische interacties tussen objecten; dit levert *operaties* van klassen.
3. Maak ook de bijbehorende *collaboratie diagrammen*; dit geeft inzicht in de *associaties* tussen klassen.
4. Verfijn het klasse diagram met *attributen*, *rollen*, *multipliciteiten*, *aggregaties*; zoek ook naar *generalisaties*.
5. Zoek naar geschikte *patterns*.
6. Maak *sequence diagrammen* ter verdere illustratie van de interactie tussen objecten.

7. Maak *toestandsdiagrammen* bij “reactieve” klassen, dwz klassen die voortdurend reageren op inkomende events of calls door zelf ook weer events te genereren of calls naar andere objecten te doen.
8. Identificeer *active klassen*, threads of control.

Doel van deze stappen is een stabiel klasse diagram, met toestandsdiagrammen voor dynamisch gedrag van objecten en sequence diagrammen als illustratie van de interactie tussen objecten. Om te komen tot een flexibel, onderhoudbaar, aanpasbaar, etc., systeem, is het belangrijk om te zorgen voor stabiele, herbruikbare klassen met high cohesion (grote samenhang) en low coupling (zwakke/weinig koppeling). Van belang daarbij zijn abstractie en encapsulatie: de klasse biedt geschikte publieke operaties, terwijl interne details (bv attributen en interne operaties) verborgen blijven. Door middel van generalisatie kan voorkomen worden dat code gedupliceerd moet worden (en het “write once” principe geschonden wordt).

Zie ook [artikel over de ROPES methode](#), Object Analysis en Design (p 27 – 35). Voor info over patterns zie bv een [WWW-pagina met patterns info](#) en voor robots bijvoorbeeld de pagina van [Orocos \(Open RObot COntrol Software\)](#).

Jozef Hooman, 10 maart 2003