Verktøykasser for formell spesifikasjon og programvareutvikling:

Et middel til forbedret programvarekvalitet

Utvikling av programvare for avanserte datamaskinsystemer er meget krevende og involverer ofte store resurser. Databaserte verktøykasser for formell spesifikasjon og programvareutvikling kan være et viktig hjelpemiddel til forbedret programvarekvalitet og reduserte kostnader.

Ketil Stølen, Ife Halden

1 Formelle Teknikker

Innen telekommunikasjon, prosesstyring og objektorientert design har det i de senere år blitt utviklet en rekke språk, teknikker og metoder for formell spesifikasjon av programvare. Slike formalismer understøtter programvareutvikling ved å

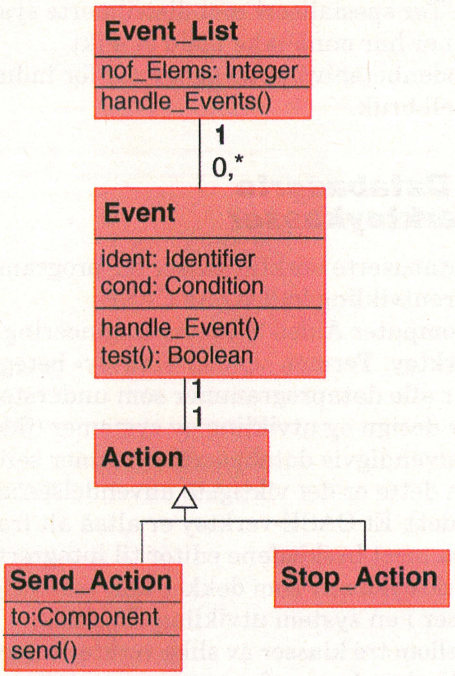
- tilby klare regler for hvordan en spesifikasjon skal se ut;
- tilordne spesifikasjoner en klar og entydig mening;
- forenkle konsistens- , fullstendighet- , og konsekvensanalyse for spesifikasjoner;
- øke mulighetene for bruk av dataverktøy ved design og implementering av spesifikasjoner.

Det finnes et betydelig antall slike formalismer på markedet. De fleste er spesialisert mot en bestemt type programvare eller et begrenset anvendelsesområde. Noen formalismer er for eksempel rettet mot kravspesifikasjon. Disse egner seg vanligvis ikke til design spesifikasjon, ei heller til å spesifisere tilordning av programvare til maskinvare. En teknikk som for eksempel har blitt utviklet med tanke på å formalisere krav til telekommunikasjonssystemer, er vanligvis lite egnet til å spesifisere krav til numeriske algoritmer. For en ikkespesialist kan det derfor være vanskelig å skaffe seg den oversikt som skal til for å velge de riktige formalismer for sin spesielle utviklingsoppgave. Det er vanlig å skille mellom tre ulike kategorier av slike formalismer.

- Semi-Formelle Spesifikasjons Teknikker (SFSTer).
  En SFST kalles semi-formell fordi den har en noe uklar grammatikk eller fordi spesifikasjoner uttrykt ved hjelp av denne teknikken ikke har en entydig mening.
- Formelle Spesifikasjons Teknikker (FSTer).
  En FST har en veldefinert grammatikk og tilordner spesifikasjoner en entydig mening uttrykt ved hjelp av en veldefinert matematisk struktur.
- Formelle Utviklings Teknikker (FUTer).
  En FUT inneholder en eller flere FSTer. I tillegg tilbyr den en kalkyle for matematisk deduksjon av nye mer forfinede spesifikasjoner eller implementasjoner fra gitte spesifikasjoner.

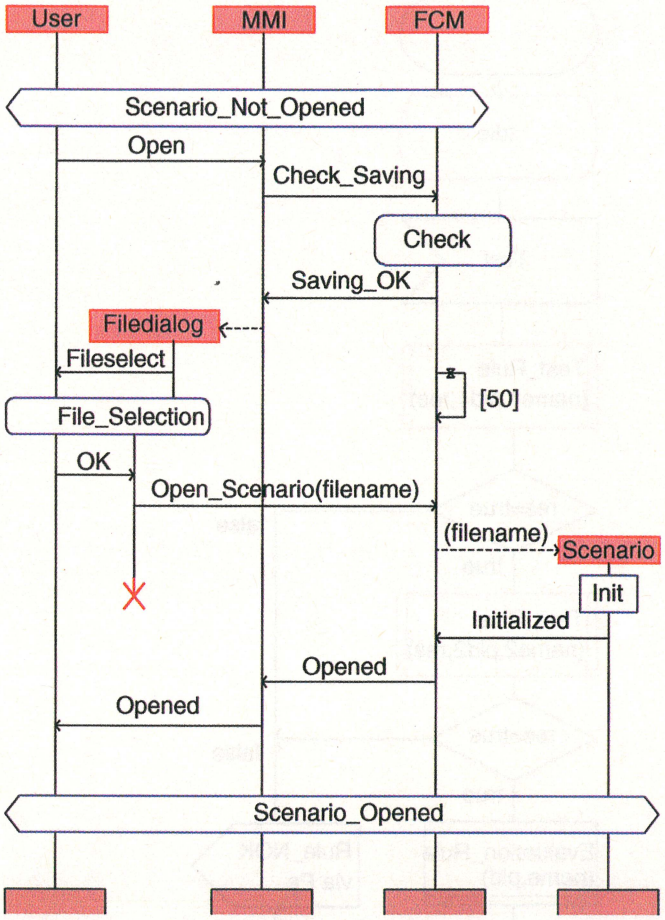
SFSTer brukes i stadig økende grad innen programvare industrien. UML (Unified Modelling Language), som nylig ble standardisert gjennom OMG (Object Management Group), er en SFDT hvis popularitet er sterkt tiltagende. UML er en samling av hovedsakelig grafiske spesifikasjonsteknikker hvis samlede uttrykkskraft kan beskrive de fleste programvarerelaterte aspekter. For eksempel, kravspesifikasjoner kan uttrykkes ved hjelp av sekvensdiagrammer og klassediagrammer. Sekvensdiagrammer (se Figur 1) egner seg til å beskrive kommunikasjon mellom ulike programvare komponenter og deres omgivelse, mens klassediagrammer (se Figur 2) er en objektorientert videreutvikling av entitetsrelasjons diagrammer som i flere tiår har blitt brukt til å beskrive relasjonen mellom ulike database entiteter. På

Fig. 2 Klassediagram i UML



Forklaring: Hver av de fem boksene representerer en klasse. Hver klasse har et navn. I tillegg kan den ha attributter og metoder. Klassene er relatert ved hjelp av assosiasjoner. For eksempel, klassene Event_List og Event er relatert ved en en-til-mange assosiasjon. Pilen representerer arv. Det betyr at Send_Action og Stop_Action er subklasser av superklassen Action.

Fig. 1 Sekvensdiagram i MSC

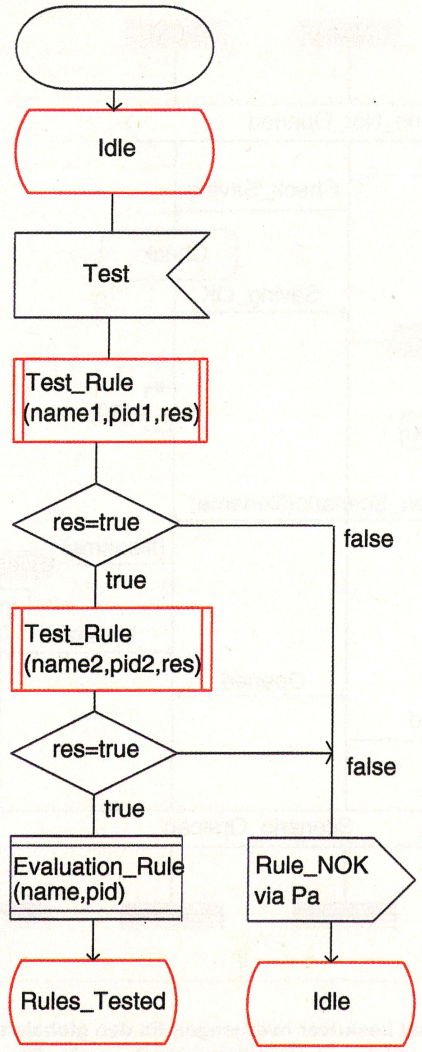


Forklaring: Diagrammet beskriver overgangen fra den globale systemtilstanden Scenario_Not_Opened til den globale tilstanden Scenario_Opened. Diagrammet har fem såkalte instansakser som hver representerer en komponent. Tre av disse komponentene eksisterer ved tilstandsovergangens begynnelse, nemlig User, MMI og FCM. Komponenten Filedialog oppstår og opphører å eksistere innenfor tilstandsovergangen, mens Scenario oppstår og fortsetter å eksistere. Hver stiplet pil representerer en komponent kreering. De øvrige pilene representerer forsendelse av en melding i pilens retning.

designnivå tilbyr UML blant annet tilstands-diagrammer sterkt inspirert av spesifikasjonsspråket Statecharts. Dette språket ble utviklet tidlig på åttitallet av David Harel mens han arbeidet som konsulent for den Israelske flyindustrien.

Typiske eksempler på FSTer er SDL (Specification and Description Language) og MSC (Message Sequence Charts) som har blitt standardisert av ITU (International Telecommunication Union). Også disse teknikkene er grafiske. MSC er, som navnet indikerer, basert på sekvensdiagrammer (se Figur 1). Sekvensdiagrammene i MSC har imidlertid en langt større uttrykkskraft enn de som finnes i UML. SDL (se Figur 3) oppsto allerede på syttitallet og →

Fig. 3 Tilstandsovergang i en SDL prosess



Forklaring: Diagrammet beskriver en tilstandsovergang i en SDL prosess, nemlig overgangen fra den lokale kontrolltilstanden Idle til den lokale kontrolltilstanden Rules_Tested eller tilbake til Idle. Tilstandsovergangen er kun mulig når meldingen Test mottas fra omgivelsen til den aktuelle SDL prosessen. De røde boksene representerer kall på prosedyren Test_Rule, mens den blå boksen kreerer en ny SDL prosess som i fremtiden vil løpe i parallell med den beskrevne. Hvis tilstandovergangen ender i Idle så blir meldingen Rule_NOK sendt langs kanalen Pa.

ble utviklet med tanke på telekommunikasjonsapplikasjoner, men kan også brukes til å spesifisere andre typer programvare hvor interaksjon og kommunikasjon spiller en viktig rolle. SDL er i likhet med Statecharts basert på kommuniserende tilstandsmaskiner og først og fremst rettet mot design.

Et eksempel på en ledende FUT er VDM (Vienna Development Method) som har blitt standardisert gjennom ISO (International Standardization Organisation). VDM ble utviklet allerede på slutten av syttitallet ved IBMs forskningslaboratorium i Vien. En langt yngre FUT hvis popularitet er sterk tiltagende er B-metoden. Både VDM og B-metoden er først of fremst rettet mot utvikling av sekvensiell programvare. FUTer spesialisert mot distribuerte systemer har ennå ikke nådd et slikt modenhetsnivå at de egner seg for industriell bruk.

2 Databaserte Verktøykasser

Databaserte verktøykasser for programvareutvikling kalles ofte CASE- (Computer Aided Systems Engineering) verktøy. Termen «CASE-verktøy» betegner alle dataprogrammer som understøtter design og utvikling av systemer (ikke nødvendigvis databaserte systemer selv om dette er det viktigste anvendelsesområdet). Et CASE-verktøy er altså alt fra den mest beskjedene editor til integrerte verktøykasser som dekker alle vesentlige faser i en system utvikling. Vi skiller mellom tre klasser av slike verktøykasser.

- Verktøykasser for programmering og implementering.
- Verktøykasser for kravspesifikasjon og design.
- Verktøykasser for en hel systemutvikling.

En verktøykasse for en hel systemutvikling kan forståes som en integrering av verktøy fra de to første klassene. I resten av denne artikkelen interesserer vi oss kun for slike integrerte verktøykasser.

Teknologi for integrerte verktøykasser har ofte blitt erklært «død». Ikke desto mindre spiller slike verktøykasser en stadig viktigere rolle, og nye verktøy av denne type kommer hele tiden på markedet. Og det er ikke uten grunn: Mange av de problemene forbundet med programvareutvikling, som motiverte introduksjonen av integrerte verktøykasser for mer enn ti år siden, kjennetegner også dagens programvareutvikling, men i motsetning til tidligere råder dagens verktøyleverandører over en teknologi som gjør det mulig fult ut å angripe disse problemene.

Bedre metodikk samt forbedrede og nyutviklede spesifikasjons og designteknikker er også viktige årsaker til økt satsning på integrerte verktøykasser. Ikke minst har utviklingen og standardiseringen av UML gitt bransjen en helt ny giv.

Fortsetter i neste utgave av Automatisering