Formele methoden worden al succesvol toegepast.

gestart. Is de hardware volwassen genoeg dan wordt deze geïntegreerd met de software, waardoor geleidelijk een eerste prototype ontstaat. Door middel van verschillende systeemtesten (integratie, stress, smoke, error, performance) worden diverse aspecten van het systeem geanalyseerd. Naar aanleiding van de simulatieresultaten komen vaak foute ontwerpen of conflicterende eigenschappen aan het licht. Door de software en hardware ontwikkelingen te itereren, worden deze vaak in volgende slagen verholpen. Wanneer het systeem voldoende foutvrij blijkt gaat deze in productie. Kiest men voor de integratie van formele methoden in het ontwikkelproces, dan moet men meer tijd spenderen aan de design fase. De noodzaak om fysieke tests uit te voeren zal daarbij afnemen, doordat men in de beginfase gedwongen wordt om beter na te denken over het design.

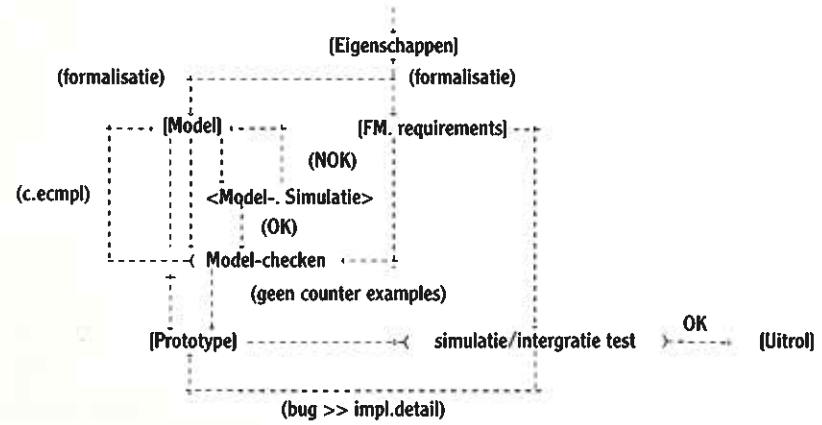
Formaliseren van requirements

De eerste stap is het formaliseren van requirements. Deze vormen de uitgangsbasis voor het te beschrijven model. Voor het beschrijven van functionele eigenschappen moet gedacht worden aan het opstellen van modale formules. Het opstellen van deze formules is niet eenvoudig, waardoor de designer gedwongen wordt om hier goed over na te denken. Het is niet ongebruikelijk dat bij verificatie van de requirements op het model, de requirements niet correct blijken.

Na het opstellen van de requirements kan een model van het gedrag worden gemaakt. Dit gaat overigens vaak tegelijkertijd met het opstellen van de requirements. Modellen kunnen buitengewoon omvangrijk worden. Tientallen pagina's is niet buitensporig.

Als het model beschikbaar is, kan het worden gebruikt om te toetsen of het model voldoet aan de verwachting van een designer en klant. Doordat formele modellen een eenduidig gedrag beschrijven, zijn de simulatoren vaak generiek en de modellen specifiek. In de praktijk blijkt dat de koppeling met grafische simulatoren in de richting van klanten uitstekend blijkt te werken, terwijl gedragsdiagrammen als communicatiemedium nog niet erg serieus worden genomen.

Bij de integratie van formele methoden in het ontwikkelproces, moet men meer tijd spenderen aan de design fase.



Zodra men er van overtuigd is dat het model een correcte abstractie van het beoogde systeem beschrijft, kan men het model verifiëren. In deze fase wordt getracht alle geformaliseerde requirements correct te bewijzen (door al het mogelijke gedrag te beschouwen). Indien een requirement niet geldt voor een model, wordt deze ontkracht door middel van een tegenvoorbeeld, wat betekent dat de designers terug moeten naar de tekentafel. Wordt er geen tegenvoorbeeld gevonden, dan is het beoogde systeem correct in termen van de opgestelde requirements.

Correct gedrag

Als aaneensluitend de implementatie van het prototype conform het gedrag is van het model, dan is daarmee bereikt dat het gedrag van het prototype ook correct is in termen van de opgestelde requirements. Om hierbij mankracht en bugs te minimaliseren kan het voordelen bieden om delen van de code te generen. Een uitrol kan nu plaatsvinden, maar het is natuurlijk wel verstandig om altijd tests uit te voeren. Het is namelijk mogelijk dat er discrepanties zijn opgetreden tussen het model en de implementatie. Echter de fouten die in deze fase optreden zijn, zijn vaak terug te voeren op fouten in de implementatie of niet beschreven gedrag in de requirements.

Als we de traditionele procesgang vergelijken met die van een formele systeemontwikkeling, dan biedt de laatste de volgende voordelen. Door validatie te verplaatsen van de prototype fase naar de design fase, worden de ontwikkelaars gedwongen de requirements eerder concreet te maken. Relatief ingewikkelde en moeilijk reproduceerbare bugs, als gevolg van racecondities tussen parallelle processen, worden in de design fase ontdekt. Dit leidt in de eindfase tot minder gedetecteerde problemen. Daardoor wordt er minder "gesleuteld" aan de software in de eindfase wat het process voorspelbaarder maakt en de software onderhoudbaarder. Helaas kent de formele systeemontwikkeling ook uitdagingen die essentieel zijn om het gehele proces te doen slagen.

Het formaliseren van modellen uit stakeholder requirements is een specialistische taak, waarbij kennis van het systeem, de in te zetten technieken, en de graad van abstractie essentieel zijn. In modellen treedt het probleem van zeer grote statespaces op. Een modelleur moet de juiste keuze weten te maken tussen een voldoende abstracte systeembeschrijving (met weinig toestanden) en

een beschrijving die voldoende met de werkelijkheid overeenkomt (maar te veel states heeft om alle eigenschappen te kunnen verifiëren). Het is ook een uitdaging om te werken aan (semi-)automatische technieken om een groot model terug te brengen tot een kleiner model dat nog voldoende informatie bevat om correctheid vast te stellen. Veel van het onderzoek in de formele methoden gaat over het ontwerpen van methoden om grote toestandsruimtes door te kunnen rekenen. Het inzetten van dergelijke methoden vergt een gedegen training.

De praktijk

Er zijn inmiddels vele voorbeelden waar formele gedragspecificaties succesvol zijn toegepast. Een mooi voorbeeld is de verificatie van de automatische parkeergarage, waarin wagens door middel van shuttles, liften en schachten worden gemanoeuvreerd naar hun parkeerpositie. Heeft men de auto weer nodig, dan wordt deze automatisch terug gemanoeuvreerd naar de uitgang.
Ondanks dat het systemen getest was, bleek er sprake te zijn van autoschade. Na een grondige (niet formele) analyse werd het veroorzaken van schade in verband gebracht met de safety afhandeling in de control software. Om verdere schade te voorkomen, werd ervoor gekozen om het design van de safety controller met behulp van formele verificatietechnieken te herontwikkelen. Hierbij werd de bestaande software als leidraad genomen. Om de analyse te ondersteunen werd daarbij een op maat gesneden visualisatie tool ontwikkeld. Door beide technieken te combineren, bleek men in staat een correct bewezen aansturingsysteem voor de parkeergarage te ontwerpen.

Een recentere toepassing is te vinden in het grootste sterrenobservatorium van astronomische organisaties binnen Europa, Noord-Amerika en Japan, genaamd Atacama Large Millimeter Array (ALMA).

Een voorbeeld van succesvol toegepaste formele gedragsspecificaties is de verificatie van een automatische parkeergarage, waarin wagens door middel van shuttles, liften en schachten worden gemanoeuvreerd naar hun parkeerpositie.

Bij oplevering zal dit systeem bestaan uit meer dan 50 telescopen, die opererend in millimeter marges. In deze applicatie is er gekeken naar de op CORBA gebaseerde software architectuur. Dit raamwerk biedt het ALMA software systeem een collectie van componenten en services waarop delen van het systeem vertrouwen. Met behulp van de specificatie taal en de bijbehorende modelchecker van mCRL2 is er ingezoomd op de controller, welke verantwoordelijk is voor het opstarten van componenten en geassocieerde containers. Door te praten met verschillende ontwikkelaars, het doorlezen van documentatie en code analyse is er een model gemaakt, waarna de analyse zicht richtte op potentiële deadlocks. Tijdens de analyse is gebleken dat time-outs van essentieel belang zijn voor een deadlock vrij systeem. Verder bleek dat het geanalyseerde systeem correct werkte.

Een andere noemenswaardige casus is de verificatie van een prototype printer voor het fabriceren van printed circuit boards. Met de huidige fabricage van PCBs wordt veelal gebruikt gemaakt van masks, die door middel van een lithografisch proces op het substraat worden afgebeeld. Met behulp van een PCB-printer kan de dure productie van het opstellen van de mask worden overgeslagen en kan ieder gewenst patroon direct op een substraat worden geprint. Om ervoor te zorgen dat het systeem geen onvoorspelbaar gedrag vertoond, werd het gedrag van de controller formeel geverifieerd. De controller bestond uit 245 multi-threaded taken geïmplementeerd in C#. In totaal besloeg de controller (zonder het onderliggende concurrent framework) 170.000 regels code. Door te abstraheren van interne variabelen, verkregen we een model dat alleen de communicatie tussen componenten behield. Doordat de requirements in termen van deze communicaties waren opgesteld, waren we in staat om de safety eigenschappen van het systeem te verifiëren. «

Formele methoden bewijzen vooral hun nut bij constructies, die niet mogen falen.

Het grootste sterrenobservatorium in Europa, Japan en Noord-Amerika is een voorbeeld van een recente toepassing.