Luchtkastelen?

Modelleren is het nieuwe programmeren. Het voegt abstractie toe en laat de computer de saaie details berekenen. Dat geeft lucht aan projecten, ruimte voor echt belangrijke zaken. De benodigde tools, zo is de belofte, zijn er bijna – maar we weten allemaal wat 'bijna' betekent in een softwareproject. Dat is overigens niet negatief bedoeld. Ik denk echt dat modelleren een belangrijke stap voorwaarts is. Ik heb echter ook geleerd dat de werkelijkheid ietsje complexer is dan de mooiste folders ons doen geloven.

Want hoe zit het bijvoorbeeld met systeemsoftware? Of met andere complexe hightechsoftware? Is het mogelijk om Linux-drivers te modelleren?

Column

Albert Mietus

Linux heeft de vervelende eigenschap dat de interface tussen driver en kernel regelmatig wordt geüpdatet. Dat geeft veel saaie detailveranderingen. Bovendien zijn er heel veel drivers die sterk op elkaar lijken. Daarmee hebben drivers veel weg van een productlijn: diverse producten die net anders zijn, maar vooral veel overeenkomsten vertonen. Met 'ouderwets' programmeren is dit altijd veel ontwikkel- en testwerk.

Modelleren belooft dit efficiënter te doen door eenmalig één model te maken, waaruit de code automatisch wordt gegenereerd. Het maken van zo'n model lijkt mogelijk: zaken als hardware-eigenschappen (registers, interrupts, DMA en dergelijke), controle-informatie voor de dev-, proc- en sys-pseudobestandssystemen en gewenst gedrag zijn eenvoudig te beschrijven. Weliswaar moeten we de codegeneratie soms aanpassen voor een nieuwe kernelversie, maar dat is eenvoudig, aldus de folders. Zelfs als het meer werk is dan beloofd, zijn er veel meer drivers dan interfaceveranderingen. Het voordeel staat als een huis, nee, als een kasteel.

Toch zijn er geen drivers gemodelleerd. Wellicht nog niet. Of zou het toch een luchtkasteel zijn? Ik denk het niet, eerder de uitzondering die de regel bevestigt. Het combineren van 'legacy' code met 'modern' modelleren is immers niet het meest voor de hand liggende voorbeeld. Als we alles van de grond af modelleren, gaat het vast veel beter.

Die uitdaging heb ik omgezet in een privéresearchproject toen ik even tussen twee banen in zat. De uitdaging was om een domeinspecifieke taal te maken voor dit soort hightech systemen. Dit heeft 'Castle' opgeleverd, waarmee een heel OS ontwikkelen eenvoudig is. Maar ook embedded software modelleren is revolutionair eenvoudig. En natuurlijk zet de bijbehorende codegeneratie-engine dit om in compacte, efficiënte code, zowel voor eenvoudige 8 bit CPU's als voor een modern multicore Arm-on-a-chip-systeem. *Concurrency* afbeelden op multicore systemen is een makkie. En Castle is geweldig om een compiler in te schrijven. Het ontwikkelen van én in Castle gaat erg snel, en toch is de code efficiënt. Het is zelfs mogelijk om er realtime systemen in te maken. Ik zou Castle zelfs willen gebruiken om Castle in te programmeren, pardon, te modelleren. Ik ben helemaal enthousiast. Er is niets dat niet kan in Castle. In de toekomst gaan we allemaal modelleren met Castle! Echt waar, Castle geeft projecten veel meer lucht.

U had het waarschijnlijk al opgemerkt: u mag deze column lezen als zo'n fraaie folder. Castle bestaat en is *bijna* af, maar nu ik weer een baan heb, is er wat lucht uit het Castle-project gelopen.

Albert Mietus werkt als systeemarchitect bij Sogeti High Tech (albert.mietus@sogeti.nl), maar schrijft deze column om zijn persoonlijke missie uit te dragen.