Bits&Chips online

Space4U voegt fouttolerantie toe aan softwarecomponenten

Rico Kind
Bits&Chips - Nieke Roos

14 april 2005 - Om de time-to-market te verkorten, overwegen steeds meer CE-bedrijven om softwarecomponenten in te kopen. Die externe modules werken echter niet altijd even goed samen, wat de prestaties van het systeem niet ten goede komt. Onder leiding van Rico Kind (Philips TASS/Philips Research) heeft het ITEA-project Space4U een raamwerk ontwikkeld dat de interoperabiliteit van componenten verbetert door ze fouttolerant te maken.

Steeds meer ontwikkelaars van consumentenelektronica gaan voor softwarecomponenten buiten de deur shoppen. Een groeiend aantal CE-bedrijven overweegt om programmamodules van derden in te kopen en deze als zwarte dozen in te passen in hun eigen producten. Een groot probleem bij deze integratie is interoperabiliteit. Een kleine verandering in de samenstelling van componenten kan onverwachte errors introduceren. Deze kunnen op hun beurt voor een drastische reductie van de functionaliteit zorgen of zelfs het hele systeem laten crashen.
Een mogelijke oplossing is om de essentiële delen van de software minder kwetsbaar te maken voor errors. Het ITEA-onderzoeksproject Space4U heeft een raamwerk gebouwd om functionaliteit voor fouttolerantie toe te voegen aan zwarte softwaredozen. De leiding over deze ontwikkeling is in handen van Rico Kind, door Philips TASS gedetacheerd bij Philips Research. ‘Space4U is een vervolg op het Robocop-initiatief’, vertelt Kind (zie ook kader). ‘Binnen dat project is een platform opgezet voor het toepassen van softwarecomponenttechnologie in resource-constrained sys-temen. In Space4U zijn we een stapje verder gegaan en hebben we die architectuur uitgebreid met energiemanagement, foutmanagement en remote terminalmanagement.’
Kind geeft leiding aan de Space4U-activiteit die onderzoek doet naar foutmanagement. ‘Foutmanagement voorkomt dat optredende errors het hele systeem onderuit kunnen halen. Wanneer je een fout tegenkomt, dan lap je die op. Je zorgt dat je systeem zo goed mogelijk blijft draaien. Overigens is dat geen structurele oplossing, want daarvoor zul je toch die bug uit de code moeten halen en een nieuwe softwareversie moeten installeren. Met foutmanagement ben je veroordeeld tot de programmatuur die je op dat moment op je systeem hebt.’
‘Toen we begonnen, konden we nog alle kanten uit. Het doel was duidelijk, een systeem met zo weinig mogelijk fouten, maar de weg daarnaartoe lag nog niet vast. Je zou ervoor kunnen zorgen dat je systeem volledig foutloos is. Met goede designs en reviews kom je een heel eind, maar garanderen dat je geen fouten hebt gemaakt, dat kun je nooit. Op het moment dat het systeem draait en je krijgt ellende, dan moet je daar verstandig mee omgaan. Het is de kunst om een systeem te maken waarvan je weet dat er fouten in zitten, maar dat toch niet crasht.’
Het raamwerk dat Kind en de zijnen binnen Space4U hebben gebouwd, voegt functionaliteit toe om fouten af te vangen zonder dat de rest van het systeem daar last

van heeft. De architectuur doet dat door het verkeer van en naar een ‘gewone’ component te laten lopen via een zogeheten ‘middleman’. Deze grijpt in zodra er iets misgaat en lost het probleem ongemerkt op. ‘Merk op dat we niets kunnen doen als er iets fout gaat binnen een component. Ons uitgangspunt is dat het allemaal zwarte dozen zijn. Daarom kunnen we alleen ingrijpen in de communicatie tussen de modules. We kunnen de rotte appels er wel in hun geheel uitpikken, maar alleen de beurse plekken wegsnijden, dat kunnen we niet.’

De ontwikkeling van een systeem op basis van het Space4U-framework begint bij het maken van een XML-interfacedefinitie voor elke component. Met de compiler afkomstig uit het Robocop-project zet de ontwikkelaar deze IDL-beschrijvingen (Interface Definition Language) om naar C-codesjablonen, waar hij vervolgens de functionaliteit van het systeem in stopt. Het compileren van de gevulde C-bestanden levert hem de binaire componenten. Bij gebruik van zwarte dozen kan de ingenieur dit traject overslaan. ‘Het is voldoende om de zwarte doos, de bijbehorende interface-definitie en een beschrijving van de functionaliteit te hebben.’

Parallel hieraan loopt de ontwikkeling van de functionaliteit voor foutmanagement. Voor elke component die de architect robuust wil maken, definieert hij in XML de fouttolerantieaspecten op basis van de bijbehorende interfacedefinitie. Het resulterende FXL-bestand (*Fault Tolerance Extension Language*) zet hij eerst om naar kant-en-klare C-code en vervolgens naar een binaire fouttolerantiecomponent, de ‘middleman’.

‘Omdat de middelman is gebaseerd op dezelfde interfacebeschrijving als de bijbehorende gewone module, heeft hij ook dezelfde in- en uitgangen’, legt Kind uit. ‘Daarnaast heeft hij de ‘omgekeerde’ aansluitingen, zodat hij perfect past op de software waarvoor hij het foutmanagement moet doen. Functionele component en fouttolerantie-eenheid samen hebben weer dezelfde interfaces als de eerste in zijn eentje, zodat de rest van het systeem niet in de gaten heeft dat de communicatie verloopt via een tussenpersoon.’

Een speciaal ontwikkelde runtime omgeving, eveneens afkomstig uit het Robocop-project en verbeterd in Space4U, combineert de functionele modules en de middelmannen ten slotte dynamisch tot een uitvoerbaar systeem. ‘Telkens als een gewone component vraagt om een koppeling met een foutgemanagede module, hangt de runtime er onder water de bijbehorende middelman tussen. Zo ontstaat een mix van functionele en fouttolerantie-eenheden.’ Het Space4U-raamwerk biedt verschillende manieren om om te gaan met errors die optreden. ‘Voor het project hebben we er een klein aantal uitgewerkt. Bij het *retry*-mechanisme zal de middelman bijvoorbeeld een gemanagede component die de fout ingaat opnieuw aanroepen, want misschien gaat het een volgende keer wel goed. Bij *reactivation* herstart hij de module om met een schone lei opnieuw te beginnen. Bij *graceful degradation* amputeert hij de manke poot met als doel het kernsysteem overeind te houden. Omdat het raamwerk nog in de onderzoeksfase verkeert, is de keuze op dit moment beperkt. Uiteindelijk moet er een heel arsenaal aan mechanismen komen, maar dat is iets voor een eventueel vervolgproject.’

Wie het Space4U-raamwerk wil gebruiken voor het inbouwen van fouttolerantie, moet precies weten wat voor vlees hij in de kuip heeft. ‘Een architect moet goed weten wat het systeem is, uit welke componenten het bestaat, hoe ze samenwerken, wat de zwakke plekken zijn, waar hij problemen verwacht, welke modules cruciaal en welke er minder cruciaal zijn. Dat moet allemaal terugkomen in het fouttolerantieontwerp.’

Kind wijst erop dat het gebruik van het framework geen garantie biedt voor een fouttolerant systeem. ‘Wij leveren slechts de tools waarmee een architect op een gestructureerde manier fouttolerantie kan inbouwen in zijn systeem. Het is aan hem om om te beslissen hoeveel robuustheid hij wil inbouwen, waar hij dat doet en wat hij bereid is daarvoor aan performance in te leveren. Want een systeem met foutmanagement moet extra werk verzetten en hoe meer er wordt gemanaged, hoe meer het kost. De architect moet daarin een verstandige keuze maken, want ook met ons framework kun je nog steeds een heel slecht systeem bouwen.’

Volgende week presenteert Rico Kind tijdens de Bits&Chips 2005 Embedded Conference de resultaten van het Space4U-project.

terug