Astin de Zeeuw

HET NIEUWE OBJECTGEORIËNTEERDE PROGRAMMEREN IN DE VERPAKKINGSWERELD

"WE ZIJN NIET INGEHAALD DOOR DE CONCURRENTIE, MAAR WE WILLEN WEL DE VINGER OP DE ZERE PLEK BLIJVEN LEGGEN", ZEGT KEES BOOTSMAN, GENERAL MANAGER VAN ROBOXIS IN BARNEVELD. "ONZE ROBOT IS SNEL, MAAR HET SAMENSPEL VAN ELEMENTEN IS BEPALEND VOOR DE UITEINDELIJKE CAPACITEIT VAN HET ROBOTSYSTEEM, EN DUS BEPALEND VOOR HET SUCCES."

Programmeur moet dan komen de voordelen vanzelf

Na de introductie van de T-robot in 2006 is het hard gegaan, voorts is vast te stellen dat de behoefte aan automatisering in Nederland nog steeds toeneemt. Ook Bootsman ziet de totale marktomvang groter dan verwacht. "Er zijn industrieën waar de hele week regulier geproduceerd moet worden om de afzet naar de eindgebruiker te kunnen garanderen. Ik heb gesprekken gevoerd met directeuren die echt voor de keus stonden hun fabriek te sluiten, of te investeren in automatisering." Bij Roboxis ziet men een behoorlijke verschuiving. "Het zijn niet alleen de grote bedrijven die automatiseren, maar ook kleine ondernemingen die relatief weinig uren per week produceren en moeite hebben personeel te vinden voor het eentonige werk," zegt Kees Bootsman. Bij Roboxis worden applicaties gemaakt waarbij met een hoge capaciteit per inuut moeilijk hanteerbare producten, zoals bijvoorbeeld gesneden vlees, gladde worsten, makrelen, koekjes, kaasjes of appelflappen, rechtstreeks in een flow wrapper of in een doos of tray worden verpakt. Daarbij wordt gebruik gemaakt van vision voor de productherkenning en plaatsbepaling, tezamen met een unieke en breed inzetbare robot. Deze T-robot is niet alleen snel, maar kan ook omgaan met grote massa's tot dertig kilogram. Daarnaast beslaat de robot een groot werkgebied voor het beladen van containers, waarbij de afmetingen van containers met 1200 x 1000 x 800 millimeter geen uitzondering vormen. De T-robot is van roestvrij staal, en is dus met name geschikt voor toepassingen in de natte sector. Hierbij moet men denken aan de vlees-, vis-, diepgevroren levensmiddelen- en snackindustrie. Met behulp van de ontwikkeltool Lasal Class en Lasal Screen van de Oostenrijkse producent Sigmatek, in Nederland vertegenwoordigd door SigmaControl uit Barendrecht, wordt de complete productielijn snel en efficiënt geprogrammeerd.

Overstap

René Rauw, Manager Control Engineering is overtuigd van de voordelen van het objectgeoriënteerd programmeren, zoals dat nu bij Roboxis met de T-robot als onderdeel van een verpakkingslijn gebeurt. Hij vertelt: "Wij hadden hier een aantal robots draaien met verschillende besturingen die niet de gewenste performance gaf. Wij waren gewend aan een eigen applicatiedeel, geprogrammeerd in C++, een motiongedeelte, een apart visiondeel en een apart visualiseringspakket. De problemen begonnen zich op te stapelen: de communicatie met de diverse systemen, de snelheid, de ondersteuning en uiteindelijk het zoeken naar storingen bij de verschillende pakketten. Toen hebben we een proef gedaan met de machinebesturing van Sigmatek. Tot onze verbazing kon het allemaal binnen anderhalve dag draaiend gemaakt worden. Wij waren oprecht verbaasd." René had het geluk al enigszins bekend te zijn met objectgeoriënteerd programmeren (OOP), omdat de eerste systemen destijds met pc-besturing werden uitgevoerd in C++. Na de overstap werd al vrij

Frank ten Velde van SigmaControl en René Rauw van Roboxis bij de productielijn voor het herverdelen van kaasjes.

anders gaan denken,

snel op de 'nieuwe manier' van programmeren een visverpakkingslijn in Volendam opgeleverd.

Hergebruik

Bij een systeem met vier identieke servomotoren kun je de software van de ene motor opnieuw gebruiken voor de andere servomotor. Het is wel van belang om de machine op een andere manier te benaderen. Rauw vervolgt: "Vroeger had je een systeem helemaal in de software zitten, en dat was het. Wil je echter alle voordelen van OOP benutten, dan zul je het verpakkingssysteem ook in objecten moeten verdelen. En dat is een andere manier van denken. Je moet meer nadenken over structuren. Je ziet dat wanneer het verpakkingssysteem goed in objecten verdeeld wordt, de software daar op kan worden afgestemd. Het aanvoerbandje voor een lege doos, de pusher die erop zit en het indexbandje zijn allemaal objecten waar je stukken software voor schrijft. Het is dan bij een volgend project eenvoudig om die stukken software opnieuw te gebruiken."
Op deze manier is de T-robot dus nu een object geworden in Lasal, welke bij elk nieuw project opgepakt en ingezet kan worden. Binnen een half uur een draaiende applicatie, in tegenstelling tot het opnieuw schrijven van applicatiesoftware. Simpelweg in een project slepen en aansluiten. Let wel: in het maken zit de winst niet, maar wel in het hergebruik.

Overzichtelijker

Naast het hergebruik van de softwarecode, de structurele opbouw en de uitbreiding van de machinefunctionaliteit zonder regelcodes te veranderen in bestaande objecten, wordt het project ook begrijpelijker binnen de organisatie. Rauw geeft het volgende voorbeeld. "Bij een productielijn waar drie doosjes met verschillende kaasjes met behulp van de T-robot moeten worden hergeschikt, heb ik onze projectleider heel goed kunnen uitleggen hoe de machine softwaretechnisch is opgebouwd en hoe die nu werkt. Het is inderdaad een hele andere benadering, omdat ik voorheen met een softwarecode aan kwam zetten, terwijl hij nu eerst de objecten ziet, waarbij het direct duidelijk is om welk machineonderdeel het gaat. Ik kan dan ook heel goed aangeven in welk deel het fout gaat. Daarnaast is de overdraagbaarheid ook eenvoudiger geworden. Daar waar de ene programmeur begint en duidelijk aangeeft hoe hij het verpakkingssysteem in objecten heeft verdeeld, kan een ander dit eenvoudig overnemen en afmaken. Je machine wordt zo gestructureerd opgezet dat het overzichtelijk wordt."

Vision

Rest natuurlijk ook de vraag hoe het zit met de koppeling tussen Lasal en vision. "De beeldverwerking zit in de camera en de posities krijgen we door. Wat je zelf moet maken, is de communicatie tussen de camera en Lasal. Dit softwareonderdeel kan weer als een object worden gezien. Bij een gelijke camera in een nieuw project kun je het cameraobject in het project slepen, zodat je direct de koppeling weer hebt met de camera. En dit alles zonder één nieuwe regel software te hoeven schrijven. We geven een trigger vanuit de controller aan de camera, waarop we de juiste gegevens van de camera over ethernet terugkrijgen. Voor elk type camera dat we toepassen, is een eigen herbruikbaar cameraobject geschreven. Een machineobject dat informatie nodig heeft, kan dit via client-server techniek bij het cameraobject opvragen. Wij zien Lasal echt als een totaaloplossing, die te gebruiken is voor het ontwerp van

Productielijn waar links de kaasjes onder de visionkast worden gescand, vervolgens worden ze gedraaid om juist gepositioneerd te zijn voor het oppakken met behulp van de zuigers van de T-robot.

de besturing, de visualisatie en sinds kort ook voor de veiligheid. Daar waar het noodstopcircuit voorheen puur hardwarematig aanwezig was, hebben we nu de mogelijkheid via de Lasal Safety Designer het noodstopcircuit softwarematig te configureren. Omdat veiligheid geheel in het besturingssysteem is geïntegreerd, biedt het als één van de voordelen dat alle informatie over het veiligheidssysteem automatisch bekend is in de machinebesturing."

Virtuoos kinderspel

Een korte uitleg over objectgeoriënteerd programmeren als eerbetoon aan Les Paul (13 augustus 2009). Objectgeoriënteerd programmeren, in de programmeursmond vaak OO ('object oriented') geheten, maakt – hoe kan het ook anders – gebruik van objecten en klassen. De methode die in de jaren zestig van de twintigste eeuw voor het eerst in de programmeertaal Simula te vinden was, kwam dertig jaar later pas echt in zwang. Inmiddels maken vele moderne programmeertalen als C#, C++, Java en Delphi er gebruik van. Bij OO worden in de eerste plaats klassen en objecten gedefinieerd. Klassen kunnen gezien worden als een soort blauwdruk voor de objecten, waarin variabelen, procedures en functies (ook wel methodes genoemd) zijn vastgelegd. Aan een object, dat tot een bepaalde klasse behoort, kunnen vervolgens aanvullende eigenschappen worden toegevoegd. Een klasse zou bijvoorbeeld een gitaar kunnen zijn: een instrument met een body, hals en snaren, die in trilling worden gebracht om muziek mee te maken. Een object dat tot die klasse behoort zou vervolgens een Gibson Les Paul kunnen zijn: een elektrische gitaar met zes snaren, twee humbuckers en een specifieke vorm die zorgt voor een warm en donker geluid. Objecten en klassen maken vervolgens gebruik van drie pijlers: inkapseling, overerving en polymorfie. Inkapseling houdt in dat een object precies die functie vervult waar het voor bedoeld is, ongeacht waar de input vandaan komt, en ongeacht waar de output naar toe gaat. Een Gibson Les Paul is om muziek mee te maken en niet om ramen mee te zemen. Daarbij maakt het Les Paul niet uit of de snaren met plectrum of tanden in trilling worden gezet, en of er een koptelefoon of zware Marshall-versterker aan de uitgang hangt.

Objecten vormen bouwstenen waar je afzonderlijk weinig mee aanvangt, maar die samen een goedwerkend systeem vormen. Bij overerving wordt er met basisklassen en subklassen gewerkt. Je gebruikt dit principe wanneer je meerdere klassen hebt die erg op elkaar lijken maar net iets afwijken. Nu kun je de ene kopiëren, plakken en iets aanpassen. Maar met de veelvoorkomende zelfde stukjes code loop je nu het risico dat wanneer je iets moet aanpassen, je het ook in alle kopieën moet aanpassen. Het is dus beter een hiërarchie op te bouwen, waarbij je de gemeenschappelijke kenmerken in een klasse onderbrengt en de aanvullende subklassen met specifieke eigenschappen definieert.

Tot slot is er polymorfie. Dankzij deze eigenschap van objectgeoriënteerd programmeren is het mogelijk al in een vroeg stadium een functie voor generieke objecten te definiëren, die later op specifieke objecten kan worden losgelaten, zonder dat de functie van het bestaan van de specifieke objecten weet. In het voorbeeld van de gitaar: een functie 'E aanslaan' zal voor de Gibson resulteren in een toon met een frequentie van 82,4 hertz, terwijl dit voor een basgitaar 41,2 hertz zal zijn.

Tot zover de theorie. Wat zijn nu eigenlijk de voordelen van een dergelijke werkwijze? Op de eerste plaats gebeurt het programmeren gestructureerder, waarmee het allemaal een stuk overzichtelijker wordt. Het denken in objecten is eenvoudiger dan het denken in ingewikkelde code. Dit betekent in de regel minder fouten en een snellere en flexibelere manier van werken. Objecten en klassen kunnen gemakkelijk worden aangemaakt, getest, worden vrijgegeven, in een bibliotheek worden opgenomen, om vervolgens te worden hergebruikt of aangepast. Met de huidige visualisatiemethoden doen de ingewikkelde stukjes code waaruit ze zijn opgebouwd onder het oppervlak hun werk. Daarnaast kan de engineering werkelijk parallel plaatsvinden. Terwijl voorheen eerst het mechanisch ontwerp plaatsvond, gevolgd door het elektrisch ontwerp en daarna pas de programmering, hoeft de specifieke hardware pas nu in een veel later stadium gekozen te worden, wat resulteert in veel kortere ontwikkelingstijden.

Bron: 'Meer objectgeoriënteerd programmeren in mechatronica', Machinebouw 2009