Diversiteit met voorgedefinieerde onderdelen

# Consumentenelektronica: softwareontwerp volgens hardwaremodel

Software voor consumentenproducten en de ontwikkeling van die software, heeft een aantal karakteristieke kenmerken. Bij Philips is daarop ingespeeld met een eigen architectuur en ontwerpproces, samengevat onder de naam Koala. Het doel van Koala is om een grote diversiteit van producten te kunnen configureren met voorgedefinieerde onderdelen. Koala software, grafisch gerepresenteerd, lijkt nogal op een hardwareschema. Dat is volgens de Philips-ontwikkelaars geen toeval.

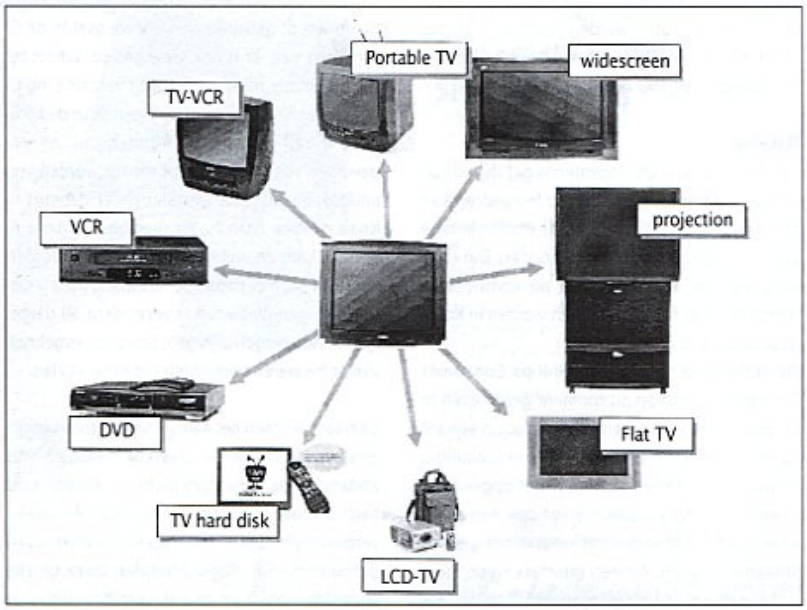


**Fig. 1. Diversiteit van Philips-producten waarvoor embedded software moet worden geschreven.**

**Een televisietoestel uit het duurdere marktsegment** bevat tegenwoordig 2 MB aan ROM en nog eens 2 MB aan RAM. Honderd ontwikkelaars zijn twee jaar bezig om de software te schrijven voor een dergelijk toestel (ref. 1). Softwareontwikkeling loopt hiermee duidelijk uit de pas met de hardware; bij nieuwe consumentenproducten denkt men in maanden, niet in jaren.

De programmatuur in geleverde apparaten moet ook meteen goed zijn. Bètatesten in de gebruikspraktijk, patches, of nieuwe versies om fouten te herstellen zijn allemaal uit den boze. Toch is de complexiteit van de embedded software niet minder dan van allerlei PC-applicaties.

Consumentenelektronica kenmerkt zich verder door de grote diversiteit, ook binnen een zelfde productgroep In figuur 1 staat een tiental producten die allemaal iets met televisie te maken hebben maar onderling ook sterk verschillen. Bij de softwareontwikkeling voor deze typen apparaten zal men er naar streven de overeenkomsten uit te buiten en de verschillen zo soepel

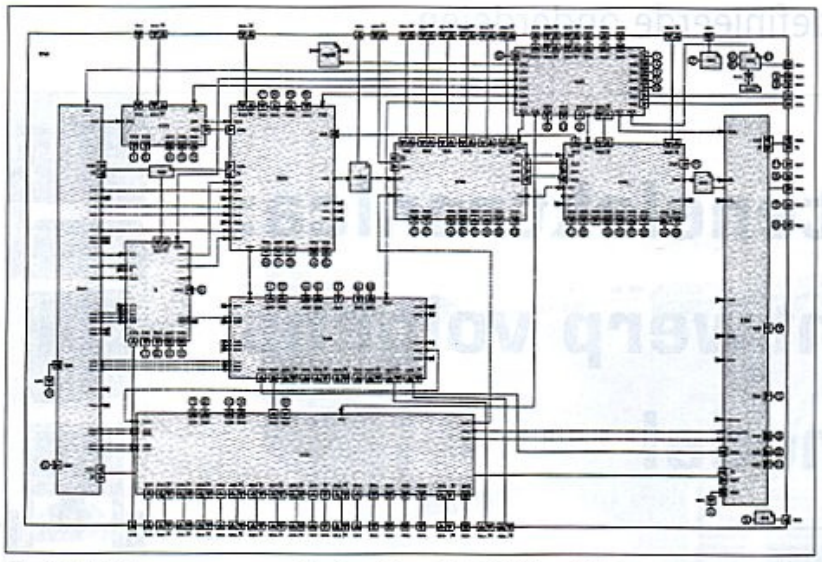

**Fig. 2. Grafische weergave van een Koala-configuratie. De blokken zijn de componenten met hun interfaces. Rechthoeken met een omgevouwen hoek representeren modules. (Bron: ref. 1)**

```
component CTvPlatform
{
 provides Iprogram pprg;
 requires II2c slow, fast;
 contains
   component CFrontEnd cfre;
   component CTunerDriver ctun;
 connects
   pprg      = cfre.pprg;
   cfre.rtun = ctun.ptun;
   ctun.ri2c = fast;
}
```

**Listing 1.: Een (onvolledige) componentbeschrijving. (Bron: ref. 2)**

mogelijk in te passen. Dat is het doel van de door Philips uitgedachte softwarearchitectuur en ontwerpmethode *Koala*.

## Componenten

Het probleem van beheersing van complexiteit en diversiteit speelt in elke technische wetenschap. De oplossing werd, en zeker ook in de elektronica, van oudsher gezocht in de toepassing van standaardcomponenten. Ook in de software-industrie wordt dit hoe langer hoe meer geprobeerd, om de al aangestipte en voor de hand liggende redenen. Een standaardfunctie in een programmeertaal of een procedure uit een toepassingsbibliotheek is eigenlijk al een component, maar een component in de moderne zin heeft veel meer functionaliteit. Het is echt een onderdeel dat op een gespecificeerde manier met andere onderdelen kan worden verbonden tot een configuratie. Een component kan zelf opgebouwd zijn uit kleinere componenten.

Op deze manier kan men dus niet alleen een complex systeem maar ook verschillende varianten daarvan samenstellen uit een beperkt aantal standaardcomponenten (bottom up). Anderzijds moet men een algemene architectuur beschrijven voor de hele groep of familie, om de juiste onderdelen te kunnen ontwikkelen (top down). Vanuit deze dubbele benadering is men bij Philips tot de conclusie gekomen dat Microsoft's Component Object Model (COM) het beste voorbeeld is van wat men op den duur wil bereiken. Maar daadwerkelijke embedded software volgens COM zou op dit moment te veel extra capaciteit vergen en niet alle aspecten van COM (zie hiervoor bijvoorbeeld ref. 5) zijn noodzakelijk. Dit componentenmodel voor software, en een algemene architectuurbeschrijvingstaal voor de productgroepen blijft als inspiratie dienen maar voor de praktijk is een eigen methode ontworpen: Koala. Deze wordt op dit moment toegepast door meer dan honderd ontwikkelaars van embedded software voor consumentenelektronica binnen Philips.

Bijkomend voordeel van componentgebaseerd ontwerpen is dan ook dat het gedistribueerde ontwikkeling ondersteunt. Software kan gelijktijdig, door meer ontwerpteams en eventueel op verschillende locaties worden geschreven en pas in een zo laat mogelijk stadium voor elk type product worden afgebouwd.

## Koala

Het principe van componenten is dat de gebruiker (systeemontwerper) zich op het gedrag kan concentreren en zich niet met de implementatie van dat gedrag hoeft bezig te houden. Dat is de zorg van de bouwer(s) van de component. Interacties tussen componenten worden in Koala gedefinieerd in *interfaces*.

De embedded software bij Philips Consumer Electronics wordt op dit moment geschreven in de programmeertaal C en een interface is een set functieprototypen in die taal. Die verzameling krijgt een algemene naam en wordt opgeslagen in een *repository*. Daarin komt ook een *datasheet*, een tekstdocument waarin het gedrag staat beschreven. Als een interface (type) eenmaal in de *repository* is ingevoerd, kan die niet meer worden veranderd. Er is geen versiebeheer.

Als een interface niet voldoet voor een bepaalde toepassing moet er een nieuw interface worden toegevoegd, ook met een algemene en uiteraard unieke naam.

De implementatie van functies zit in de componenten, maar dan op het aggregatieniveau van interfaces. Een component levert als het ware een, of meestal meer, interfaces en dit wordt aangegeven met *provides* (zie listing 1). Als een component juist een interface nodig heeft wordt dat aangegeven met *requires*. Een component kan ook verschillende instanties van een interface van een bepaald type bevatten, bijv. geluidssterkte voor de luidsprekers en geluidssterkte voor de koptelefoon. Elk *requires*-interface moet gebonden zijn aan precies één *provides*-interface. Een *provides* kan diverse, één, of geen verbindingen hebben. Componenten kunnen andere componenten bevatten en de interne verbindingen worden dan aangegeven zoals in listing 1.

Alle componenten staan in een component-*repository*, met twee algemene namen. De korte naam wordt gebruikt als voorvoegsel in de C-functienamen. Er is ook weer een datasheet ter documentatie maar nu ook een verzameling C- en header-files. Die worden alleen door de component zelf gebruikt en kunnen dus zonder gevolgen voor ander componenten worden veranderd. Bug fixes en versiebeheer gebeurt in Koala op lokaal niveau. Aan een bestaande component kunnen wel nieuwe interfaces worden toegevoegd, met bepaalde restricties, maar interfaces mogen niet worden verwijderd. Al dergelijke ontwerpregels zijn gebaseerd op hergebruik van software en gedistribueerd ontwikkelen.

Uiteraard wil men het aantal componenten enigszins beperkt houden en alleen die onderdelen als zodanig in de *repository* invoeren die ook echt herbruikbaar zijn. Voor initialisaties en unieke verbindingen tussen interfaces (lijm ofwel *glue*) gebruikt men in Koala *modules*. Deze unieke onderdelen gedragen zich als componenten maar hebben geen interfaces. In een grafische weer-

gave zijn het rechthoekjes met een omgevouwen bovenhoek en de lijnen geven aan met welke componenten zij zijn verbonden (zie figuur 2). Componenten worden grafisch gerepresenteerd door blokken met interfaces aan de randen. De driehoekjes binnen een interface geven de richting aan van de functieaanroep. De naar binnen gerichte interfaces zijn dus de *provides* van de component, de naar buiten gerichte de *requires*. De lijnen representeren de verbindingen.

Deze manier van weergave is duidelijk op de elektronica gebaseerd en vergemakkelijkt de communicatie tussen hardware- en softwareontwerpers in de projectorganisatie.

## Configuratie

De uit componenten opgebouwde software voor een bepaald type apparaat (zoals in fig. 1) heet een configuratie. Dit is ook op te vatten als een samengestelde component (zoals in figuur 2) die echter zelf geen (externe) interfaces levert of nodig heeft. Een gangbare configuratie bevat tientallen componenten en een gangbare component heeft een stuk of tien interfaces.

Om een configuratie op te bouwen uit interface- en componentbeschrijvingen is er een tool gemaakt die ook Koala heet. Om de verbindingen te verwezenlijken binnen een configuratie moeten bijvoorbeeld allerlei functienamen worden herbenoemd. Dit gebeurt door macro's Als component c bijvoorbeeld een functie f levert krijgt die de algemene naam c_p_f. In header-files krijgt dezelfde functie dan verschillende namen die in plaatselijke componenten van toepassing zijn. Het is ook mogelijk om met *switches* te kiezen tussen drivers, bijvoorbeeld tussen een kanaal of frequentiemodus van een televisie front end. Door parametrisering en voorwaardelijke compilatie van de C-code kan de software worden geoptimaliseerd voor elke configuratie. Koala kent voor dit doel zelfs objectgeoriënteerde spreadsheets van *diversity parameters*. Behalve door *diversity interfaces* kan diversiteit ook aangegeven worden door interfaces van componenten optioneel te maken en als zodanig aan te geven. Configuraties en ook de codebestanden vergen natuurlijk versiebeheer en alle *repositories* worden beheerd door een standaard systeem daarvoor. Daarin verschilt het embedded softwareontwerp bij Philips CE niet van de gangbare praktijk, maar alle productdiversiteit wordt nu beheerd door Koala.

**Referenties**


1. R. van Ommering: *'A Composable Software Architecture for Consumer Electronics Products,'*, http://members.brabant.chello.nl/~rvanommeringen/pubs/Xootic99_rvo.pdf
2. R. van Ommering, F. van der Linden, J. Kramer, J. Magee: *'The Koala Component Model for Consumer Electronics Software'*. Computer, March 2000
3. R. van Ommering: *'Beyond Product Families: Building a Product Population?'*, http://members.brabant.chello.nl/~rvanommeringen/pubs/Iwsapf3_rvo.pdf
4. R. van Ommering: *'Koala, a Component Model for Consumer Electronics Product Software'*, http://members.brabant.chello.nl/~rvanommeringen/pubs/Ares2_rvo.pdf
5. S.Williams, C. Kindel, *'The Component Object Model: A Technical Overview'*, http://msdn.microsoft.com/library/techart/msdn_comppr.htm