Praktische programmatuurmodulariteit door metaprogrammering

Martijn Houtman en Pieter Wisse

1. korte inleiding in algemene modulariteitsleer

Er kunnen allerlei redenen zijn voor indeling in ruimte (object) en/of tijd (proces). Zo kunnen bepaalde ònderdelen (lees ook: componenten, modulen e.d.) eenvoudiger te ontwerpen, ontwikkelen, gebruiken, onderhouden, vervangen en/of noem maar op zijn. Als ònontkoombare voorwaarde geldt uiteraard dat modulen eenduidig op elkaar aansluiten voor optimaal gebruiksresultaat.

Gangbaar is functionele indeling van een systeem met zijn elementen. Dat werkt zolang functies zich herkenbaar lenen voor classificatie – vrijwel – zònder onderlinge overlap. Zo heeft een woning doorgaans aparte kamers voor verschillende gedragingen door de bewoners en hun bezoekers. Overigens bestaat invloed ook òmgekeerd, per saldo versterkend; hun aangeleerd gedrag doet gebruikers zo'n functiescheiding en dienovereenkomstig passende indeling van 'voorzieningen' verwachten.

Hoe ruimer de schaal waarop gedragingen tellen, kan inderdaad om nòg meer redenen modulaire opzet gauw des te zinvoller zijn. Dat komt dan neer op infrastructuur. Wat de openbare weg heet, bijvoorbeeld, omvat zoiets als samenhangende verkeersmodulen die willekeurige burgers kunnen benutten.

Op de basisschool leren kinderen o.a. rekenen. Voor welke oplossingsmethoden dan ook is in dit geval overwegend procesmatige modulariteit kenmerkend. De oplossing van het probleem volgt stapsgewijs.

Afhankelijk van de opgave loont het om digitale verwerking van informatie tot een resultaat vergelijkbaar te laten verlopen, dus met inzet van 'samenwerkende' modulen. Opnieuw telt schaaleffect, zeg ook maar zoiets als infrastructuralisering. Voor verschillende verwerkingsopgaven kùnnen dezelfde modulen bruikbaar zijn. Sterker nog, naarmate de dekking van een bepaalde 'functie' stelselmatig moet zijn, daarvoor is autorisatie exemplarisch, is modulaire opzet zelfs noodzakelijk.

Vrijwel vanaf de eerste computerprogramma's is zulke indelingspraktijk gevolgd. Hoewel terminologie verschilt, is en blijft het beginsel ook voor structurering van zulke programma's hetzelfde. De aanpak heet vaak functionele decompositie of, in het Engels, (re)factoring. Zeg ook maar compartimentering, modulariteit. De (programmatuur)delen stonden ooit vooral bekend als subroutines. Tegenwoordig worden ze veelal, opnieuw in het Engels, services genoemd. Nogmaals, systeembenadering is niets nieuws.

2. voor informatieverwerking naar modulariteit van de tweede orde

Vanwege de eis van samenhang moet het aantal modulen ook weer redelijk beperkt – kunnen – gehouden worden. Elke module èxtra vermenigvuldigt immers het aantal eventuele intermodulaire relaties, enzovoort. Echter vooral om pràktische gebruiksredenen valt er meestal aan verschillende modulen niet te ontkomen. Hoewel de meeste mensen een gevarieerd dieet volgen, hebben zij in hun huis als regel één keuken voor bereiding van àl hun maaltijden; bijvoorbeeld slapen doen zij in (een) àndere kamer(s). Een module is multifunctioneler, het woord zegt het al, naarmate meer functies ermee worden gefaciliteerd.

De associatie van evenzovele verschillende modulen met verschillende functies wordt hier aangeduid als modulariteit van de eerste orde.

Zoals gezegd kan ook digitale informatieverwerking opbouwend onderhevig zijn aan functionele ontleding. Het aantal modulen kan echter véél kleiner blijven dat het aantal functies. Dat heet hier modulariteit van de tweede orde. Idealiter volstaat één module voor àlle functies. Zulke radicale multifunctionaliteit is in de praktijk weliswaar onbereikbaar, maar uiteraard telt èlke sterke

vermindering van het aantal.
Het belang om het aantal modulen te beperken is nog eens gestegen door vernetwerking van digitale voorzieningen. Door zulke infrastructuralisering houdt informatieverwerking immers steeds meer facilitering van informatieverkeer in en neemt het aantal functies zelfs sterk toe waarin nota bene samenhangend 'voorzien' moet zijn, denk wederom aan stelselmatige autorisatie en informatiebeveiliging.
Tweede-ordemodulariteit is logisch door besef dat aspecten van informatieverwerking als geheel op hùn beurt informatieverwerking vergen (met eventuele aspecten dáárvan, enzovoort). De term aspect geldt hier mbt indeling als synoniem van functie vanwege de voor programmatuuropzet gangbare aanduiding aspectgerichtheid (Engels: aspect orientation) voor onderscheid naar onderling zo onafhankelijk mogelijke invalshoeken (Engels: separation of concerns).
Door ontleding volgens de tweede orde gaat het dus om alsmaar vèrdere variaties op één thema. Met een aspect als functie lukt het om vergaand éénheid in verwerking aan te brengen voor verscheidenheid van functies. De noodzakelijkerwijs aspectspecifieke verwerkingsmodaliteit laat zich met parameters, ofwel óók informatie, variëren. Voor de eventuele afstemming/wisselwerking tùssen verschillende aspectbehandelingen als resp. informatieverwerkingen moet telkens een uitkomst volgens het ene aspect kunnen gelden als parameterwaarde(n) voor één of meer andere aspecten die relevant zijn, en zonodig omgekeerd.
Een goed idee ligt meestal pas achteraf voor de hand. Overigens is onder de noemer van wiskunde – wat hier heet – modulariteit van de tweede orde evenmin nieuw. De 'vertaling' naar werkende (computer)programmering is dat althans voorzover (on)bekend nog wèl.

3. voordelen
Indien modulariteit van de eerste orde zoveel voordelen biedt, zijn ze met modulariteit van de tweede orde alweer beduidend groter. Welke voordelen zijn er zoal?
Tegenovergesteld aan vermenigvuldiging van het aantal modulen resulteert modulariteit van de tweede orde in deling van het aantal. Zoals willekeurig herbruikbare modulen al een beperking van – de omvang van – programmatuur mogelijk maken vergeleken met voor-elk-probleem-een-compleet-apart-systeem, kan dus óók zelfs het aantal modulen zeer sterk beperkt blijven. Dat maakt borging van de kwaliteit ervan uiteraard véél eenvoudiger.
Maatregelen voor o.a. informatiebeveiliging zijn – als kwaliteitsaspect – overzichtelijker. De hoeveelheid programmatuur is zowel véél geringer als in formele zin gestructureerder. Voorts gebeurt facilitering van informatiebeveiliging dus grotendeels met dezèlfde module als middel dat doel van beveiliging is: reflexiviteit.
Door de module over het vermogen te laten beschikken om naar behoefte daadwerkelijke exemplaren te genereren (Engels: multithreading) is voorzien in willekeurige schaalbaarheid van verwerking.

4. aspecten annex functies van informatieverwerking
Voor een idee van wat zoal voor aspecten doorgaat, blijft het ook op de open stelselschaal van informatieverkeer nuttig allereerst erop te wijzen dat ordening van informatie (lees hier ook: data) voor opslag, verwerking enz. met digitale middelen gebeurt volgens typen (lees ook: klassen). Voor een bepaald type, nota bene èxpliciet contextueel verbijzonderd vanwege stelselmatig onontkoombare betekenissenvariëteit,[1] zeg hier gemakshalve bijvoorbeeld tòch simpelweg persoon, kan dan informatie over meerdere exemplaren, ofwel individuen als 'echte' personen, zijn verzameld. Ook voor verband (lees ook: relatie) tùssen exemplaren geldt verband tussen typen als voorschrift.

[1] Voor stelselmatig passende modelleermethode, zie Metapatroon, handboek stelselmatig informatieverkeer. Wiskundig geïnspireerd kenmerkt tweede-ordemodulariteit het recursieve verbijzonderingsbeginsel van Metapatroon (Engels: Metapattern) voor betekenis(duiding) als gedragsbeschrijving; betekenisverschillen kunnen voor willekeurig bereik samenhangend worden geordend (lees ook: stelselmatig geordend worden).

Een gangbare aanduiding van zulke ‘typische’ ordeningsdata voor ermee veronderstelde betekenissen van gebruiksdata luidt: metadata. De letterlijke vertaling van metadata is immers: data over data. Voorzover verwerking van zulke metadata functioneel vergelijkbaar is met verwerking van data, is dezèlfde module bruikbaar.
Aanvullende data over gebruiksdata is er niet alleen ‘over’ betekenissenordening. Allerlei aspecten tellen voor digitalisering, zoals presentatie en beveiliging met op hùn beurt vaak meerdere deelaspecten. Data dáárover geldt daarom óók als metadata, waarvoor dankzij passende parametersering dezèlfde module eveneens bruikbaar is.
Kortom, met gebruiksdata als uitgangspunt geldt data ‘erover’ als metadata. Maar metadata zijn verder ook gewoon weer data, inderdaad, waarvoor het nuttig kan zijn om expliciet metadata te bepalen (die uitgaande van de ‘oorspronkelijke’ data dan metametadata zijn, en eventueel zo verder).
Zo is er als het ware een Wet van Omgekeerde Evenredigheid van Metadata met Modulariteit van Programmatuur. Want hoe méér aspecten van ‘verwerking’ van gebruiksdata met metadata dekkend beschreven zijn, des te minder (programmatuur)modulen er nodig zijn. Nogmaals, idealiter ‘verzorgt’ één en dezèlfde module àlle aspecten met desbetreffende metadata als nodige en voldoende parameters voor sturing van aspectspecifiek verwerkingsverloop en aldus bepaling van het resultaat ervan.

5. metaprogrammering voor modulariteit van de tweede orde
Met KnitbITs is programmatuur beschikbaar die Information Dynamics volgens het beginsel van modulariteit van de tweede orde heeft ontwikkeld met één module voor relevante aspecten van het dataverkeer met de database(s).[2]
Door traditionele programmering, zeg ook maar gericht op eerste-ordemodulariteit, ontstaat een sturingsprobleem wanneer een component (lees hier dus ook: module) op meerdere wijzen geparameteriseerd is en parameters dienovereenkomstig verschillend ingevuld moeten worden. De besturende programmatuur moet daar rekening mee houden. Dat maakt directe koppeling onvermijdelijk tussen de besturende programmatuur en het variabel gebruik van de component.
Met metadata-sturing blijft sterke koppeling vermeden tussen de besturende programmatuur en variaties van de component. Dat is haalbaar door een abstractere wijze van programmeren: metaprogrammering. Het gaat hier om één (meta)service, die zich voor elk probleem on-the-fly programmeert, om een oneindige verzameling dataverwerkingsproblemen op te kunnen lossen. Het is een soort 'kameleonservice' die zich aan de omgeving van het betreffende probleem aanpast, zonder telkens opnieuw programmacode te hoeven schrijven.
Door de parameters van een component op generieke wijze met behulp van data en metadata in te vullen, vervalt het onderscheid tussen het gebruik van verschillende parameters door de component. Een flexibele opzet wordt bereikt door een gezamenlijke interface voor de variaties van de component vast te leggen en in de besturende programmatuur te gebruiken; de keuze voor de variatie van de te gebruiken component wordt via metadata aangestuurd. Dankzij metaprogrammering verschuift de aandacht van de inhoud van variërend dataverkeer naar het abstractere probleem van de beschrijving van de vorm van variërend dataverkeer.
Om een component te kiezen en de parameters van een component op generieke wijze via data en metadata in te vullen, wordt voor KnitbITs thans gebruik gemaakt van een volledige beschrijving van de component in XML (over de voordelen van variaties op één thema gesproken, de keuze is opzettelijk vooralsnog voor XML gemaakt vanwege de beschikbare mogelijkheden voor willekeurige transformaties). Met een XSLT-transformatie wordt de XML-metadata samen met XML-data samengevoegd door één andere module tot een volledige XML-beschrijving voor de component,

[2] KnitbITs is het platform voor inrichting van digitale facilitering van informatieverkeer met veranderlijke betekenissenvariëteit van dien, dwz volgens conceptueel ontwerp(model) opgesteld met Metapatroon. Met bestaande registers valt vaak reeds vergaand samenhang te vestigen door ze op een informatierotonde (IR) aan te sluiten. Dat hulpmiddel heet KnitbITs IR, zie Stelselmatig overzicht via informatiesleutels voor een algemene toelichting.

inclusief initialisatie. Om de geïnitialiseerde component te genereren, deserialiseert de besturende programmatuur deze volledige XML-beschrijving. Hierna kan de besturende programmatuur de te gebruiken interface opvragen. Doordat de component klaar staat voor gebruik inclusief geïnitialiseerde parameters, is programmeren van de initialisatie van de component in de besturende programmatuur niet meer nodig.
Zo beschouwd kunnen voordelen van de opzet alweer wat nader worden verklaard. De besturende programmatuur kent enkel de interface van de te gebruiken variaties van de component. Hierdoor is de programmering van de besturende programmatuur onafhankelijk van de programmering van de component. Dezelfde besturing kan verschillende variaties van een component aansturen zònder – mogelijk overlappende – programmering met verschillende besturing voor de componentvarianten. Dit houdt programmering optimaal compact. Dat maakt de bron van eventuele (programmeer)fouten prompt duidelijk. Zodra ze opgemerkt zijn, àls ze überhaupt nog optreden, kunnen ze eenvoudig en vlot worden verbeterd.