Overdraagbare feiten

Software werkt met allerlei soorten gegevens. Soms zijn dat werkgegevens en soms zijn dat op een of andere wijze verworven gegevens of eindresultaten. Soms zijn de gegevens sterk met elkaar gerelateerd. In dat geval vormen zij samen een complex object. In dat geval is het vaak zinvol om de acties die op deze gegevens werken ook bij dat object te betrekken. Dat gebeurt dan bij gelijkwaardige objecten op eendere wijze. De objectgeoriënteerde wijze van programmeren maakt intensief gebruik van deze aanpak.

In softwareprogramma's worden gegevens regelmatig tussen plaatsen uitgewisseld en vaak worden zij op een of andere wijze opgeslagen en later weer teruggehaald. Het opslaan van gegevens kan in bestanden, in documenten of in databases. Om een complex object te versturen of op te slaan moet het uiteengerafeld worden zodat de losse gegevens apart behandeld kunnen worden. Dit komt onder andere omdat elk type gegeven een eigen behandeling nodig heeft. Bovendien hoeft niet elk onderdeel van het object meegenomen te worden. Na aankomst op de bestemming of bij het uit de opslag halen moet het object weer uit zijn onderdelen opgebouwd worden. De niet meegenomen onderdelen moeten een verstandige initiële waarde krijgen. Vaak wordt een groot deel van de programmatuur aan dit proces van uiteenrafelen en weer samenvoegen gewijd. Daar komt nog bij dat het opslaan in bestanden, het opslaan in databases, het omzetten in documenten en het versturen via netwerken op sterk verschillende wijze gebeuren. Vrijwel steeds worden daar specifieke oplossingen voor verzonnen. De basishulpmiddelen die de programmeertalen ter beschikking stellen bieden namelijk geen standaard oplossing voor dit probleem. Een grotere uniformiteit is zeer wenselijk en kan significante besparingen opleveren.

Als er in de toepassingen die men bouwt regelmatig complex gestructureerde objecten voorkomen, dan kan het erg winstgevend zijn om in een dergelijke standaardoplossing te voorzien. Het blijkt dat met een dergelijke aanpak al snel een flinke factor aan tijdwinst geboekt wordt. Ook blijkt de robuustheid van het resultaat gunstig beïnvloed te zijn. Dat kan echter alleen als er een dergelijke vorm van standaardisatie door programmeurs geaccepteerd wordt. De standaardisatie wordt acceptabel als het toepassingsgebied voldoende groot is terwijl tegelijkertijd de standaard geen verregaande beperkingen oplegt.

De aanpak van het probleem maakt gebruik van de objectgeoriënteerde wijze waarop tegenwoordig vaak geprogrammeerd wordt. In deze werkwijze beschikken de complexe objecten over een reeks klassewijd gedefinieerde acties en een reeks eigenschappen. Een deel van de eigenschappen is klassewijd en de andere eigenschappen behoren bij het individuele object. Als het goed is worden de werkgegevens worden voor de buitenwereld van het object afgeschermd. Dat gebeurt door hen met de label 'private' te kenmerken. Deze privé eigenschappen kunnen dan alleen door de eigen routines van het object benaderd worden. Op die wijze kan het object zijn integriteit waarborgen. Sommige van deze privé gegevens kunnen ook door de erfgenamen van de klasse van het object gebruikt worden. In dat geval wordt 'private' vervangen door 'protected'. Wat overblijft, zijn eigenschappen die publiekelijk toegankelijk zijn. Deze openheid kan

gevaren voor de integriteit van het object opleveren. Daarom is het verstandig om de het publieke deel van de eigenschappen te beperken tot gegevens die toch al met de omgeving van het object uitgewisseld worden. Vaak kan een private eigenschap door een publiekelijk toegankelijk voorfront beschermd worden. Zo'n voorfront heet 'property'en biedt toegang via een get-set interface.

Er bestaat bij de huidige objectgeoriënteerde talen weliswaar een standaard wijze om aan te geven dat objecten overdraagbaar en archiveerbaar zijn, maar dat geldt niet voor de afzonderlijke eigenschappen van dat object. Zo bestaat er zowel in Java als in .net een attribuut met de naam 'serializable'. Dit attribuut kwalificeert een complete klasse als overdraagbaar en archiveerbaar. Deze aanpak gaat grotendeels buiten de controle van de programmeur om en heeft vaak nog extra beperkingen. Het ondersteunt de opslag in bestanden en de omzetting in XML documenten. Alle publiekelijk toegankelijke eigenschappen worden daarbij zonder uitzondering opgeslagen of omgezet. De ondersteuning van opslag in databases ontbreekt hierbij.

De moderne objectgeoriënteerde talen maken het mogelijk om een attribuut in te voeren dat aan gekozen velden van een object speciale activiteiten toekent die in werking treden als er door het object bepaalde acties uitgevoerd worden. Bovendien kunnen aan het attribuut nog parameters gehangen worden die de specifieke acties nader definiëren. Laten we dit attribuut 'transferable' noemen. Een object dat meedoet, kan de opdracht krijgen om zichzelf over te sturen of zichzelf op te slaan. Als eerste actie zal het dan zichzelf uiteenrafelen en de 'transferable' gegevens van zijn onderdelen opnemen in de overdracht of opslagactie. Bij de omgekeerde operatie, het ontvangen of terughalen, gebeurt het omgekeerde.

Complexe objecten kunnen een hiërarchische structuur hebben. Dat betekent dat ze uit een serie objecten kunnen bestaan die zelf ook weer uit een of meer al dan niet complexe objecten bestaan. Een speciale vorm is een object dat uit een verzameling van nagenoeg gelijkwaardige objecten bestaan. Dat mogen ook weer complex gestructureerde objecten zijn. Die verzameling kan al dan niet geordend zijn.

De buitenste schil wordt gevormd door een object met speciale eigenschappen. Bij het versturen of bij het opslaan in een bestand wordt het gehele object geconverteerd in een bytestroom. Bij het versturen over netwerken wordt de bytestroom nog geconverteerd naar een base64 stroom. De binnen liggende objecten hoeven niets anders te doen dan zichzelf in een bytestroom om te zetten. Het kan zinvol zijn om voordat omgezet wordt naar een base64 stroom de resulterende bytestroom te versleutelen en vlak daarvoor ook te comprimeren. Comprimeren is alleen nodig wanneer de omvang groter is dan een bepaalde grensmaat. Bij versleutelen levert de opdrachtgever de sleutel aan. Verder gaat alles vanzelf. De hele operatie gebeurt zo dat bij het weer samenstellen van het omhullende object de hele operatie automatisch omgekeerd wordt. Ook dan moet de correcte sleutel door de opdrachtgever aangereikt worden. De programmeur hoeft zich geen zorgen te maken over het uiteenrafelen en het samenvoegen. Ook het comprimeren en versleutelen verloopt automatisch. De programmeur hoeft er alleen maar voor te zorgen dat er, indien nodig, een correcte sleutel voorhanden is.

Het geheel kan geïmplementeerd worden met vier klassen. De eerste is een klasse van objecten die als binnenobjecten kunnen functioneren. Deze klasse van objecten kunnen ‘transferable’ velden bevatten. Van de eerste klasse wordt een tweede klasse afgeleid die als omhullende objecten kunnen fungeren. De derde klasse bestaat uit de verzamelingen van gelijksoortige binnenobjecten. Ook zij zijn afgeleid van de eerste klasse. De vierde klasse bestaat uit referentieobjecten en deze zijn ook afgeleid van de eerste klasse. Het is niet de bedoeling dat de objecten uit de derde en vierde klasse zelf ‘tranferable’ velden bevatten. Daarnaast bestaat de implementatie nog uit een kleine routinebibliotheek die de benodigde hulproutines herbergt.

Het omhullende object is in staat om zijn eigen databasestructuur te definiëren. Nadat dit gebeurd is, kan het zichzelf geheel automatisch in de database opslaan. Bovendien kan het zichzelf en al zijn onderdelen uit de database verwijderen. Om te voorkomen dat daarbij geen loshangende referenties overblijven, mogen de objecten die meedoen aan de standaard maar eenmaal in de database opgeslagen worden. Voor de rest moet met referenties naar deze objecten gewerkt worden. Dat is de reden van het bestaan van de vierde klasse. Voordat een object verwijderd wordt moeten eerst alle referenties naar dat object verwijderd worden. Het systeem bewaakt dit. Het gevolg van dit alles is dat de databasemanager geen specifieke kennis van de softwareprogramma’s hoeft te hebben. De programma’s verzorgen zelf hun eigen databasemanagement.

Om de hiërarchische structuur van het object in een relationele database te kunnen behouden beschikt elk object over twee ID’s. Het eerste is een eigen globally unique identifier. Het tweede is het ID van het direct omvattende object. Verder kan een object beschikken over een volgnummer of een unieke naam die gebruikt wordt voor de ordening in een verzamelobject waarin dat object opgenomen is. Verzamelobjecten beschikken over een of meer sleutelrijen waarmee de ordening geregeld kan worden. De verzamelobjecten worden zelf niet in de database opgeslagen. Hun elementen wel. Zij bevatten een referentie naar het verzamelobject. De aanpak maakt van een relationele database een krachtige objectgeoriënteerde database.

De omhullende objecten ondersteunen de overdracht van dat object over netwerken, opslag in en terughalen uit binaire bestanden, opslag in en terughalen uit relationele databases en heen en terugconverteren naar XML bestanden. Het omhullende object kan zijn eigen XSD bestand creëren. Het zal niet moeilijk zijn nog andere capaciteiten aan dit systeem toe te voegen.

Het is zonder meer verstandig om toepassingen waarin complexe objecten gemanipuleerd moeten worden op de bovenbeschreven standaard te baseren. Dat geldt vooral als de toepassing zowel gebruikmaakt van opslag in databases, in binaire bestanden en in XML bestanden, terwijl de objecten ook over netwerken gestuurd worden. Als deze standaard eenmaal bestaat en men kan er mee werken dan kan met een grote tijdswinst boeken en zal de toepassing een relatief grote robuustheid tonen.

De schrijver van dit artikel heeft met succes enorme toepassingen met deze aanpak gebouwd. Het onderhoud ervan was een makkie.