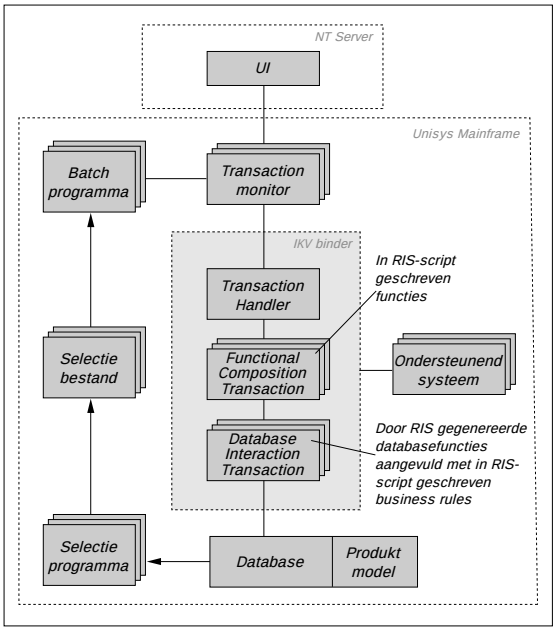
Voor veel bedrijven is het ontwikkelen van nieuwe systemen, met een op flexibiliteit gerichte architectuur, noodzakelijk om in de toekomst tijdig in te kunnen springen op wijzigende omstandigheden. Maar wat te doen met bestaande, vaak nog volop in gebruik zijnde systemen? De gedane investeringen en de impliciet in de systemen aanwezige kennis maken het volledig opnieuw beginnen een onaantrekkelijke optie. Bij de Westland/Utrecht Hypotheekbank (WUH), een kredietinstelling gespecialiseerd in de financiering van onroerende zaken, is daarom onderzoek gedaan naar de mogelijkheden om een bestaand systeem te migreren naar een nieuwe architectuur. In een tweedelig artikel wordt beschreven welke ervaringen zijn opgedaan in dit onderzoek. Het tweede deel verschijnt in het volgende nummer van Software Release Magazine.

Migratie bij de Westland/Utrecht Hypotheekbank (1)

Analyse van een monoliet

De WUH biedt een uitgebreid pakket financiële diensten aan zowel particuliere als zakelijke klanten. Eén van de informatiesystemen die deze dienstverlening ondersteunen is het IKV-systeem. IKV, wat staat voor Integratie Kredietverlening/Verzekeringen, is in de periode van 1991 tot 1994 door WUH ontwikkeld om tegemoet te komen aan de behoefte aan een verregaande flexibilisering van het toenmalige hypotheeksysteem en de integratie van kredietverlening met verzekeringen. Speerpunt bij het ontwerpen van IKV was, naast de integratie van hypotheken met verzekeringen, de introductie van een produktmodel. Dit model stelt de gebruiker van het systeem in staat on-line nieuwe produkten samen te stellen uit bestaande produktcomponenten. Het IKV-systeem is geïmplementeerd met het ontwikkeltool RIS van het bedrijf Infra Design. RIS biedt een 4GL-omgeving gebaseerd op COBOL; ontwikkelaars kunnen in een COBOL-achtige scripttaal functionaliteit beschrijven die door het ontwikkeltool wordt gecompileerd naar COBOL. Belangrijke componenten in de RIS-omgeving zijn FCT's (Functional Composition Transactions), modules waarin functionaliteit wordt beschreven in de RIS-scripttaal, en DIT's (Database Interaction Transactions), door RIS gegenereerde databasefuncties aangevuld met in RIS-script geschreven business rules. IKV bestaat uit respectievelijk 1200 en 1900 van deze FCT's en DIT's, terwijl de IKV-database ongeveer 180 entiteiten omvat. Afbeelding 1 geeft een globale indruk van de technische architectuur van IKV.

FLEXIBILITEIT Hoewel IKV voor eindgebruikers nog steeds voldoet, leidt een aantal (markt)ontwikkelingen tot onvoldoende flexibiliteit van het huidige systeem. Ontwikkelingen in de huidige financiële markt zijn elkaar steeds sneller opvolgende nieuwe produkten, uitgebreide produktvariaties en mogelijkheden om produkten samen te stellen tot klantspecifieke oplossingen die weliswaar in het begin van de jaren negentig niet konden worden voorzien maar vandaag de dag wel hun eisen stellen aan de flexibiliteit van informatiesystemen. Daarnaast vereist de integratie van nieuwe technologieën als datawarehousing en e-commerce een andere vorm van flexibiliteit dan die de ontwerpers van IKV oorspronkelijk voor ogen hadden. Het vervangen



AFBEELDING 1: Technische architectuur van IKV

van bepaalde delen van het systeem door componenten uit de markt, of het nog verder parameteriseren van het systeem, is binnen de huidige architectuur bijvoorbeeld niet goed mogelijk.

Een nieuwe architectuur voor IKV moet deze problemen oplossen. Het uitgangspunt voor deze nieuwe architectuur wordt gevormd door het bekende drie-lagen model. Dit model beschrijft een structuur waarbij softwarecomponenten op basis van te verwachten veranderlijkheid worden verdeeld over drie lagen. Communicatie tussen componenten vindt in principe alleen plaats tussen componenten binnen dezelfde laag, of met componenten uit de laag direct eronder, waardoor aanpassingen in de meest veranderlijke, bovenste lagen geen invloed hebben op de basis(business)functionaliteit van het systeem. Soms zijn de lagen ook daadwerkelijk fysiek gescheiden (bijvoorbeeld over clients en servers), voor IKV is echter gekozen het drie-lagen model te implementeren binnen de bestaande RIS-architectuur. Naast de horizontale segmentatie van het systeem is in de nieuwe architectuur ook speciaal aandacht besteed aan de twee volgende punten:

- *Segmentatie binnen lagen*: Binnen lagen moeten componenten duidelijk afgebakende (en dus goed herbruikbare) deelfunctionaliteit implementeren.
- *Extractie van kennis*: Een gedeelte van de in het IKV-systeem aanwezige kennis is vervat in een productmodel. Het is de bedoeling ook alle andere kennis uit het systeem zodanig te extraheren, dat het mogelijk wordt de kennis met parameterbestanden en beslissingsmodules expliciet te implementeren binnen de nieuwe architectuur.

Hoewel de nieuwe architectuur fundamenteel anders is dan die van het bestaande IKV-systeem, moet veel van de gebruikersfunctionaliteit van het systeem gehandhaafd blijven. De vraag is dan ook of hier een migratie kan worden toegepast, waarbij de bestaande programmatuur en daarin vastgelegde kennis wordt hergebruikt bij het ontwikkelen van een nieuw IKV-systeem. De snelle ontwikkelingen op de hedendaagse hypotheekmarkt en de verregaande systeemintegratie binnen de ING-groep (waarvan de WUH deel uitmaakt) vragen veel van een automatiseringsafdeling. Dit, samen met het feit dat IKV momenteel nog weinig acute problemen oplevert, maakt dat slechts een minimale hoeveelheid interne capaciteit vrij gemaakt kan worden voor de uitvoering van een migratietraject. Een migratie moet daarom dusdanig gefaseerd uitgevoerd worden, dat het de lopende ontwikkelingen niet verstoort.

ARCHITECTUURMIGRATIEPROJECT Op basis van deze randvoorwaarden is, na het opstellen van het globale architectuurmodel, onderzocht in welke mate een migratietraject (geautomatiseerd) zou kunnen worden ondersteund. Eén van de kernproblemen bij een migratietraject is dat de onderdelen van de doelarchitectuur niet altijd direct herkenbaar zijn in het bestaande systeem en omgekeerd. Plastisch uitgedrukt: een specifieke FCT kan bijvoorbeeld code bevatten die eigenlijk thuishoort in componenten uit verschillende lagen van de nieuwe architectuur, of, omgekeerd, functionaliteit van een component uit een specifieke laag van de nieuwe architectuur is in het huidige systeem bijvoorbeeld verspreid over verschillende modules. Het oplossen van dit probleem is een eerste noodzakelijke stap op weg naar een architectuurmigratie.

De in het onderzoek verkende oplossing is het gebruik van een meta-model voor het leggen van een koppeling tussen de bestaande en de nieuwe architectuur. Zijn er in de huidige programmatuur structuren te herkennen die (misschien na een aantal abstractie-slagen) te gebruiken zijn als componenten voor het IKV-systeem nieuwe stijl? In het project zijn twee aspecten van het gebruik van een meta-model als basis voor een architectuurmigratie onderzocht:

- *Meta-model*: Hoe ziet een meta-model dat de oude en nieuwe architectuur aan elkaar koppelt eruit? Welke abstracties zijn nodig en hoe zijn ze gerelateerd?
- *Toolondersteuning*: Hoe kan het invullen van een meta-model (het ontdekken en maken van abstracties vanuit de bestaande programmatuur) met tools worden ondersteund?

Beide aspecten worden hierna toegelicht.

META-MODEL Om een architectuurmigratie mogelijk te maken moet op één of andere manier de relatie kunnen worden gelegd tussen de verschillende soorten componenten waaruit de oude en het nieuwe systeem bestaan. Het logische datamodel van deze componenten en hun relaties vormen het meta-model. De opbouw van het meta-model moet het mogelijk maken vragen te beantwoorden als:

- *Is het mogelijk delen van het huidige systeem af te bakenen?* Afgebakende deelsystemen maken een gefaseerde overgang mogelijk en kunnen helpen het nieuwe systeem netjes verticaal gesegmenteerd op te zetten.
- *Kan laagspecifieke functionaliteit worden onderkend?* In het drie-lagen model is bijvoorbeeld bedrijfsfunctionaliteit strikt gescheiden van de eigenlijke data. Is dit soort functionaliteit die past binnen de gedefinieerde lagen in het systeem te onderkennen? Wat gebeurt er met bepaalde data en wat is procesafhankelijk? Is er data in het systeem die altijd als een logisch eenheid wordt benaderd?
- *Welke in het systeem geïmplementeerde kennis kan worden onderkend en geëxtraheerd?* Veel kennis en beslissingsregels zijn impliciet in de huidige implementatie aanwezig. Kan deze kennis expliciet worden gemaakt? Welke beslissingsregels of beslissingsvariabelen kunnen worden onderkend?
- *Welke concepten zijn waar geïmplementeerd?* Het bewaren van de relatie tussen eventueel herkende abstracte

concepten of functionaliteit en de huidige implementatie daarvan is belangrijk, bijvoorbeeld voor het kunnen maken van impactanalyses voor migratiewerkzaamheden of het kunnen migreren van deelsystemen.

Bij al deze punten vormt het inzichtelijk maken van de relatie tussen abstracte structuren en de wijze waarop deze in het systeem is geïmplementeerd het belangrijkste uitgangspunt. Het meta-model beschrijft een aantal componenttypen en relaties die het mogelijk maken het huidige systeem te beschrijven en te abstraheren naar componenten die zinvol zijn binnen de nieuwe architectuur, zonder daarbij de relatie met de huidige implementatie te verliezen. Afbeelding 2 geeft een globale indruk van de manier waarop dit meta-model is opgezet.

Het meta-model bestaat uit verschillende soorten componenten, op verschillende abstractieniveaus:

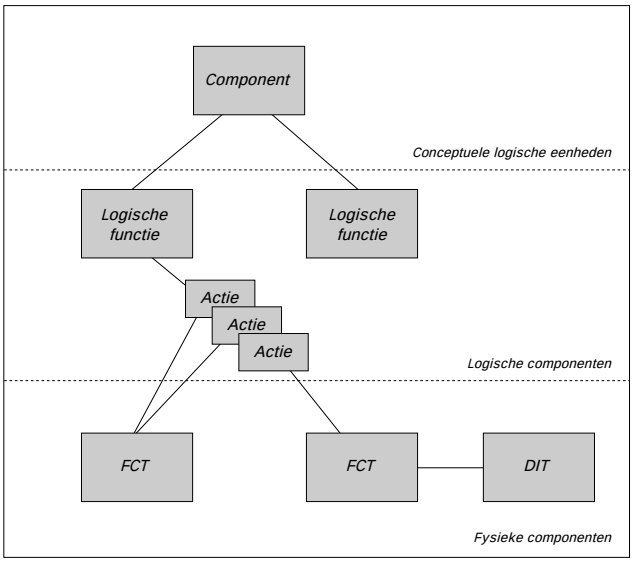
- *Fysieke componenten*: Componenten op dit niveau beschrijven de geïmplementeerde werkelijkheid. Hier zijn alle relevante IKV-componenten en hun onderlinge samenhang opgenomen. Voorbeelden van componenttypen op dit niveau zijn FCT's en DIT's. Een ingevuld meta-model beschrijft op dit niveau dus de componenten en relaties zoals ze in het huidige IKV systeem aanwezig zijn.
- *Logische componenten*: Geïmplementeerde werkelijkheid zegt niet altijd iets over logische deelfunctionaliteit. Een technische eenheid/module kan soms verschillende, onafhankelijke functionaliteit implementeren. Ook kan bepaalde deelfunctionaliteit juist verspreid over meerdere technische eenheden geïmplementeerd zijn. Logische componenten zijn abstracties van de implementatie zónder deze onvolkomenheden. Er is onderscheid gemaakt tussen statische en dynamische logische componenten. Voorbeelden van dynamische logische componenten zijn logische functies en acties. Een constraint is een voorbeeld van een statische logische component.
- *Conceptuele logische componenten*: Conceptuele logische componenten zijn bedoeld om op een hoger abstractieniveau logische functies te kunnen clusteren tot componenten met een duidelijk afgebakende functionaliteit en verantwoordelijkheid.

Op de details van de verschillende componenttypen wordt hier niet nader ingegaan.

Een meta-model zoals hierboven omschreven beschrijft alleen componenttypen, géén instanties. Om uiteindelijk een migratie mogelijk te maken is juist een ingevuld meta-model nodig: uit welke technische componenten bestaat het huidige systeem, welke abstracties zijn daarin aanwezig en wat zijn uiteindelijk kandidaten voor componenten binnen de nieuwe architectuur?

Idealiter kan veel van de informatie nodig voor het invullen van het meta-model gehaald worden uit bestaande bronnen. Voor IKV zijn een aantal bronnen voorhanden:

- *Functioneel ontwerp*: Hierin wordt de systeemfunctionaliteit beschreven, opgesplitst in invoer, uitvoerprocessen en functies. Er is ook een logisch Entity Relation Diagram (ERD) met aanvullende beschrijvingen.
- *Technisch ontwerp*: Alle zelf ontwikkelde RIS-componenten zoals FCT's en Rules zijn beschreven. Er is ook een technisch ERD met aanvullende beschrijvingen.
- *RIS repository*: De RIS-ontwikkelomgeving beschikt over een repository waarin veel informatie over de implementatie aanwezig is. Alle (informatie over) technische componenten is in deze repository opgeslagen en het is zelfs mogelijk een aantal relaties tussen deze componenten te bekijken. Een voorbeeld hiervan is een transactieflow; de RIS repository kan (transitief) bepalen welke componenten een bepaalde component aanroept, of door welke componenten een bepaalde component wordt aangeroepen.



AFBEELDING 2: Voorbeeld van een aantal componenten en relaties uit het meta-model

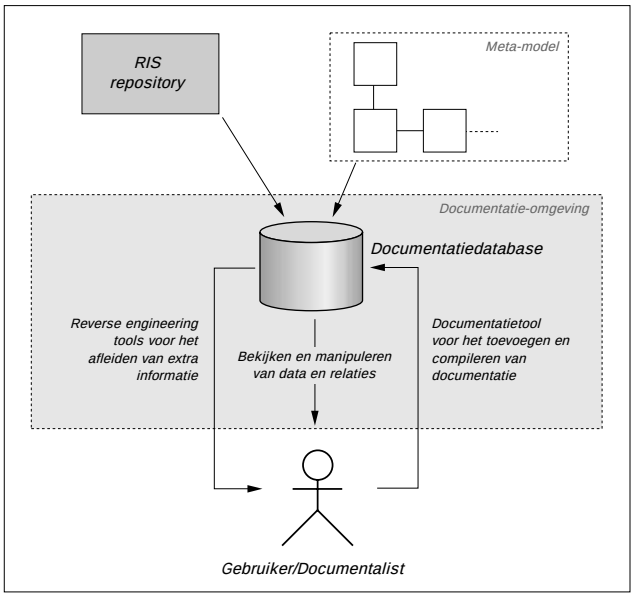
MONOLITHISCH Ondanks het feit dat er dus relatief veel informatie beschikbaar is over de huidige implementatie van IKV, blijkt dat deze informatie alleen toch niet voldoende is om als basis voor een migratietraject te dienen. Een belangrijke oorzaak hiervoor is de omvang van IKV: met 1100 logische functies (FCT's) en 1800 databasetransacties (DIT's) worden overzichten als transactieflows en calltrees al snel ondoorzichtig. Beschrijvingen van 'functies die functies aanroepen die functies aanroepen', verliezen met elke 'laag' een stuk inzichtelijkheid. De abstractieniveaus uit het meta-model zijn noodzakelijk om het benodigde overzicht te verkrijgen, maar juist voor het afleiden van deze abstractieniveaus is de bestaande informatie lastig te gebruiken.

Een belangrijke oorzaak hiervan is de complexiteit van de functionaliteit. Maatwerk en uitzonderlijk veel productvariaties zorgen voor een zeer complex geheel. Het monolithisch karakter van de huidige implementatie is

een ander probleem: verwevenheid van functionaliteit door verschillende, elkaar aanroepende functies en veelvuldig hergebruik maken afbakening van deelsystemen en dus een gefaseerde overgang erg lastig. Ontwerpprincipes van 'samenhang' en 'koppeling' zijn niet altijd consistent toegepast waardoor technische componenten soms meerdere functies implementeren of juist andersom: meerdere technische componenten implementeren samen één functie. Mede hierdoor is ook 'hergebruik' van functionaliteit niet altijd duidelijk herkenbaar en komt bepaalde functionaliteit zelfs gedupliceerd voor.

Ook de beschikbare documentatie is niet direct bruikbaar, vooral door het ontbreken van documentatie gericht op concepten. In IKV bestaan domeinconcepten zoals 'rentevast periodes' of 'leningdelen'. In het functioneel ontwerp van IKV worden echter niet deze domeinconcepten als uitgangspunt genomen, maar de procesgang voor de eindgebruiker. De kennis met betrekking tot de diverse concepten is dus verdeeld over (de documentatie van) diverse processen en functies, waardoor het lastig is om een goed beeld te krijgen van de wijze waarop zo'n concept geïmplementeerd is. De beschikbare ERD's en in- en uitvoer procesbeschrijvingen geven weinig houvast voor een lagen architectuur doordat geen duidelijk onderscheid gemaakt is in functionaliteit rond data die altijd uitgevoerd moet worden en functionaliteit die afhankelijk is van een bepaald proces.

Dit alles betekent dat de bestaande documentatie en informatie misschien prima voldoen aan de normale eisen van een ontwikkelafdeling, maar een onvoldoende basis vormen voor een beheersbare migratie. Het fysieke niveau van het meta-model kan aan de hand van beschikbare informatie voor een groot deel worden ingevuld, maar daarmee houdt het op. Want voor het herkennen van abstracties, noodzakelijk om ondanks grote hoeveelheden en technische onvolkomenheden het overzicht te kunnen behouden, biedt de beschikbare documentatie geen houvast. Om tot het niveau te komen waarop een architectuurmigratie gerealiseerd kan gaan worden, lijkt abstractie van componenten uit code de enige mogelijkheid. De nieuwe abstracte componenten, de onderlinge relaties en de beschrijving van deze beiden vormen samen een nieuw soort documentatie die misschien wél kan dienen als basis voor een migratietraject.



AFBEELDING 3: Documentatieomgeving

TOOLONDERSTEUNING Volledig handmatige afleiding van de meta-modelabstracties uit de programmatuur is onbegonnen werk. In het kader van het project is daarom een aantal (prototypes van) hulpmiddelen gebouwd om het abstractieproces te ondersteunen (zie afbeelding 3). De basis hiervan is een database met alle meta-modelcomponenten. Via een grafisch tool (de IKV browser) kunnen gebruikers deze componenten invoeren, manipuleren en inspecteren. Om de gebruikers te ondersteunen bij het vinden en herkennen van de abstracties uit de hogere lagen van het meta-model zijn diverse 'reverse engineering' tools gebouwd. Hierbij gaat het zowel om het produceren van bepaalde (overzicht-)views als om het uitvoeren van (semi-)automatische analyses en compilatieslagen.

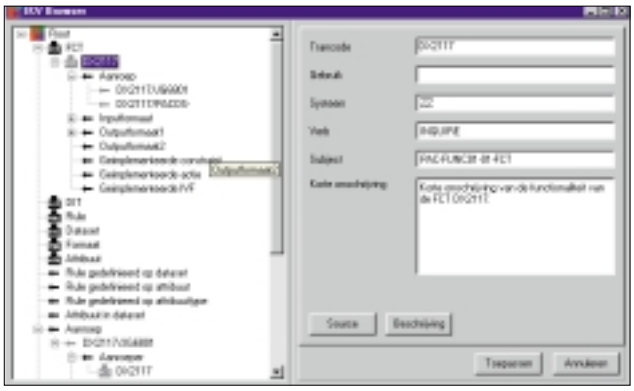
De kern van de documentatieomgeving is een database waarin alle componenten uit het meta-model worden opgeslagen. Voor het onderzoek is de database initieel eenmalig gevuld met een kopie van de inhoud van de RIS-repository. Dit betekent dat gegevens over alle implementatiecomponenten (FCT's, DIT's) in de database zijn opgenomen inclusief de broncode en additionele relaties die in de repository worden bijgehouden.

De inhoud van de database wordt ontsloten met een interactief tool, de IKV Browser (zie afbeelding 4). In feite maakt dit tool (de instanties van) de verschillende entiteiten en verbanden uit het meta-model zichtbaar. De browser werkt volgens de metafoor van een tree browser, bekend van onder meer de Explorer. Van elke instantie van een entiteit zijn de documentatierelaties zichtbaar gemaakt als subboom. Door zo'n boom te openen worden de, via die betreffende relatie, gerelateerde entiteiten zichtbaar gemaakt. In een apart paneel worden de detailgegevens van de betreffende documentatiecomponent zichtbaar gemaakt. Voor de componenten die niet automatisch uit de implementatie afleidbaar zijn, kunnen deze gegevens ook ingevoerd en opgeslagen worden.

De browser hanteert een aantal technieken om 'informatie overload' te voorkomen en de verbanden inzichtelijk te maken. Bij entiteiten of relaties met erg veel instanties (denk aan de 1100 FCT's) moet de gebruiker expliciet aangeven welke in de browser te zien zijn. Verder worden verschillende soorten iconen en verschillen-

Lees verder op pagina 46

Vervolg van pagina 28



AFBEELDING 4: De IKV Browser

de kleuren gebruikt om verbanden en entiteiten aan te geven. Tenslotte is het mogelijk om nieuwe browservensters te openen op subbomen.

De IKV Browser ontsluit de gegevens uit de documentatiedatabase. Naast de browser is ook een aantal prototypes ontwikkeld om het abstractieproces te ondersteunen:

- *Graafvisualisatie*: Relaties (en vooral ketens van gerelateerde componenten) uit de documentatiedatabase zijn in de IKV Browser lastig te bekijken. Visuele weergave maakt een beter overzicht van deze relaties mogelijk.
- *Documentatiecompilatie*: De documentatie van logische componenten kan nauw samenhangen met de documentatie van gerelateerde componenten. De documentatieomgeving biedt ondersteuning voor het samenstellen van beschrijvingen uit beschrijvingen van gerelateerde componenten.
- *Dataflowanalyse*: Aanroeppatronen alleen bieden vaak weinig houvast. Inzicht in de data die bij de verschillende aanroepen wordt meegegeven of teruggekregen kan soms uitkomst bieden.
- *Dotplotanalyse*: Hoewel het idee bestaat dat IKV geduplicceerde code bevat, is vaak niet direct duidelijk wáár duplicatie voorkomt. Dotplotanalyse is een manier om via een visuele representatie codeduplicatie op te sporen.

Ieder van deze vier soorten ondersteuning van het documentatieproces is in prototype vorm uitgewerkt. Hoe dat in z'n werk is gegaan komt aan bod in het volgende nummer van Software Release Magazine. Het slotartikel bevat tevens de eindconclusies en een uitgebreide literatuuropgave.

George Starke
is projectmanager bij de Westland/Utrecht Hypotheekbank.

Gert Florijn en Matthijs Maat
zijn werkzaam bij het Software Engineering Research Centre te Utrecht.