ALTERRA stowa RIZA rivm onderzoek in dienst van mens en milieu

Architectuur Standaard Raamwerk Water

T. van der Wal (red.)

1999-16_architectuur-standaard-raamwerk-water

Alterra-rapport 072, ISSN 1566-7197
Stowa rapport 99-16, ISBN 90.5773.065.0
Riza rapport 99.063

Architectuur Standaard Raamwerk Water

Architectuur Standaard Raamwerk Water

T. van der Wal (red.)

Alterra Research Instituut voor de Groene Ruimte

Alterra-rapport 072
Stowa rapport 99-16
Riza rapport 99.063

maart 2000
Wageningen

Referaat

Wal, T. van der (editor), 2000. Architectuur Standaard Raamwerk Water; Wageningen, Alterra, Research Instituut voor de Groene Ruimte. Alterra-rapport 072. 116 blz. 45 fig.; 4 tab.; 10 ref. 7 bijlagen.

Dit rapport beschrijft de architectuur zoals die ontwikkeld is voor een Standaard Raamwerk Water. Dit raamwerk beoogt een generieke omgeving voor simulatiemodellen in het integraal waterbeheer te zijn. Dit rapport bevat een analyse van het domein 'simulatiemodellen in het integraal waterbeheer', een ontwerp van de informatiearchitectuur en aanbevelingen om tot een dergelijk generiek raamwerk te komen.

Trefwoorden: Aquest, domeinanalyse, hydrologie, informatiearchitectuur, integraal waterbeheer, modelkoppeling, raamwerk, simulatiemodellen, software ontwerp, standaardisatie

Alterra-rapport 072, ISSN 1566-7197
Stowa rapport 99-16, ISBN 90.5773.065.0
Riza rapport 99.063

Dit rapport kunt u bestellen door NLG 50,00 over te maken op banknummer 36 70 54 612 ten name van Alterra, Wageningen, onder vermelding van Alterra-rapport 072. Dit bedrag is inclusief BTW en verzendkosten.

Projectnummer 82025

Alterra-rapport 072, maart 2000

Voorwoord

Het project 'Architectuur Standaard Raamwerk Water' is uitgevoerd in opdracht van STOWA (*Stichting Toegepast Onderzoek Waterbeheer*) *en medegefinancierd door* Rijkswaterstaat, RIZA, RIVM en Alterra. Het project is uitgevoerd door een consortium van IT organisaties die actief applicaties ontwikkelen voor het integraal waterbeheer in nederland, te weten Alterra, WL|Delft Hydraulics, EDS, Geodan-IT en TNO-NITG. Door het belang dat zij hechten aan dit project is ook van uitvoerders kant een bijdrage geleverd aan de financiering.

Bij de uitvoering van het project zijn de volgende organisaties en personen betrokken geweest:

STOWA - Stichting Toegepast Onderzoek Waterbeheer	Jan Noort (Sepra) Ludolph Wentholt
Rijksinstituut voor Integraal Zoetwater-beheer en Afvalwaterbehandeling / RIZA	Michiel Blind Arthur Kors Wim de Lange Anne Ubbels Rolf van der Veen Harold van Waveren
Rijksinstituut voor Volksgezondheid en Milieu / RIVM	Tom Aldenberg Aldrik Bakema Lowie van Liere
Wageningen Universiteit en Researchcentrum / Alterra	Jandirk Bulens Piet Groenendijk Ab Veldhuizen
Waterbedrijf Gelderland	Peter Salverda
Alterra Group Software Engineering	Mark van Elswijk (SERC) Tonny Otjens Tamme van der Wal (Projectleider)
WL | Delft Hydraulics	Klaas-Jan van Heeringen Peter Schrier Jaco Stout
TNO - NITG	Frans van Geer Neno Kukuric Ipo Ritsema Frank Waardenburg
EDS International BV	Bas van Adrichem Johan Tacke
Geodan - IT BV	Barend Gehrels Theo Thewessen

Inhoudsopgave

Samenvatting

Inleiding (hoofdstuk 1)
Momenteel worden er legio modelprogramma's ontwikkeld, gebruikt en beheerd voor modellering van het integraal waterbeheer. Deze programma's komen veelal van verschillende leveranciers. Door de toenemende digitalisering van gegevens en de toenemende mogelijkheden op het gebied van capaciteit en rekentijd van computers ontstaat een grote behoefte aan koppeling van al deze programma's. Echter, de programma's zijn nooit voor dit doel ontworpen, waardoor koppeling een proces vol hindernissen en ergernissen is.
Ook worden er door de toenemende computermogelijkheden steeds meer uitbreidingen en verfijningen verwacht van al deze programma's. Ook hier blijkt dat dit niet eenvoudig is, omdat de programma's niet met het oog op uitbreidbaarheid ontworpen zijn.

Het Standaard Raamwerk Water (SRW) moet gaan voorzien in een generieke beschrijving, geformaliseerd in software, waarmee simulatiemodellen en datasets ontwikkeld, gebruikt, gekoppeld, beheerd en verbeterd kunnen worden. Door zich te conformeren aan een raamwerk maken ontwikkelaars programma's die wel afgestemd zijn op andere programma's. Daarnaast hoeft niet ieder programma meer de ballast van gebruikersinteractie of visualisatie van gegevens met zich mee te torsen: dit wordt als generieke oplossing door het SRW geboden.

Het project 'Architectuur Standaard Raamwerk Water' komt voort uit het Aquestprogramma, dat gericht is op ontwikkeling van de beleids- en planvorming in het Nederlandse waterbeheer door goede informatievoorziening en communicatie. De architectuur voor het SRW beschrijft de regels waaraan modelprogramma's en tools moeten voldoen om de doelstellingen van het SRW te waarborgen. De keuzen t.a.v. techniek en implementatie worden zoveel mogelijk uitgesteld tot de fase waarin het SRW daadwerkelijk gerealiseerd kan worden in een vervolgproject. Om te zorgen voor uitbreidbaarheid, flexibiliteit en aanpasbaarheid binnen de architectuur, is gekozen voor een moderne aanpak van het ontwerptraject dat is gebaseerd op losse componenten. Deze vormen tezamen het domein van het raamwerk.

Nadere achtergronden (hoofdstuk 2)
De werkgroep 'Generieke Tools' stelde in 1998 een Programma van Eisen op, waarbij het SRW en de architectuur ervan als afzonderlijke producten worden gezien. Beide hebben als doel een aantal knelpunten op te lossen: arbeidsdruk, operationaliseren, aansluiting, ontoegankelijkheid, presentatie, tijdsdruk, en model- en gegevenskoppeling. Uitgangspunten voor de architectuur zijn: openheid, centrale metagegevens, grafische gebruikersinterface, platformonafhankelijkheid en distributie.

De medewerkers van de belanghebbende organisaties vervullen binnen het SRW allen een verschillende rol. Bij het ontwerp en ontwikkeling van het SRW moeten de belangen meegenomen en afgewogen worden, zodat specifieke tools voor bijvoorbeeld modelontwikkelaars en adviesbureaus kunnen worden ontworpen.

De hiervoor opgesomde knelpunten en wensen t.a.v. de architectuur zijn direct te koppelen aan bepaalde kwaliteitseisen/-eigenschappen. Deze zijn overgenomen uit het Extended ISO Model of Software Quality. In latere stadia kan worden vastgesteld in hoeverre hieraan wordt voldaan.

Voor een gestandaardiseerde samenwerking tussen modelprogramma's in het vakdomein water kan voor de opbouw van modelprogramma's aangesloten worden bij reeds geaccepteerde standaarden. Het systeem Adventus van de Unie van Waterschappen standaardiseert definities van gegevens, gegevensmodellen en gegevensopslag. Voor het beschrijven en uitwisselen van ruimtelijke objecten kan aangesloten worden op OpenGIS.

Nadere uitgangspunten zijn geformuleerd voor het onderhavige project: Het Programma van Eisen is richtinggevend; bestaande software moet door migratie (her)gebruikt kunnen worden; meta-gegevens van modelprogramma's en tools worden niet centraal opgeslagen en tenslotte worden technologiekeuzen pas gemaakt voorafgaand aan de bouw van het SRW.

Domeinverkenning (hoofdstuk 3)
Een belangrijk aspect van dit project vormt de domeinverkenning: hoe breed kunnen we het domein veronderstellen waarbij het SRW nog wel functioneel, zinvol en toepasbaar is? In eerste instantie is het gedefinieerd als het domein van modellering in het integraal waterbeheer.
Dit domein is vanuit drie invalshoeken beschreven: fysica, simulatie en omgeving.
De domeinverkenning kwam tot stand in een vijftal workshops waarin gebruikersdeskundigen en informatici zitting hadden.

De fysica van het integraal waterbeheer bestaat uit het beschrijven van de modellering van de hydrologische kringloop. We onderscheiden in het domeinmodel vier gescheiden systemen, die ieder een eigen verantwoordelijkheid dragen voor bepaalde processen: mens, atmosfeer, land en water. Op de raakvlakken van deze systemen vindt overdracht van informatie plaats (interactie). Het systeem land kan nader worden verdeeld in een verzadigde zone, een onverzadigde zone en een terrestrisch ecosysteem. Het watersysteem kan nader worden verdeeld in oppervlaktewater en een aquatisch ecosysteem. Verdere detaillering is mogelijk maar binnen dit project niet relevant. Geconstateerd is dat bestaande modelprogramma's meerdere subdomeinen beslaan en dat koppeling van modelprogramma's alleen plaatsvindt als duidelijk gemaakt wordt welk programma voor welk subdomein verantwoordelijk is.

Vanuit de invalshoek 'modelleren en simuleren' neemt het wiskundig oplossingsmechanisme dat in waterbeheer-modelprogramma's wordt toegepast een centrale plaats in. Het oplossings-mechanisme bepaalt in sterke mate de manier waarop de werkelijkheid wordt geschematiseerd. Binnen de architectuur van het raamwerk bestaat een modelprogramma uit model-componenten. Daar zijn twee vormen van: modelelementen en connectoren, waarbij een connector altijd twee modelelementen verbindt. Hiermee wordt elk model beschreven als een netwerk van modelelementen. Elk modelelement is verantwoordelijk voor een stukje informatie in het model.

De derde invalshoek behelst de omgeving waarin modelprogramma's gebruikt worden. Hierbij is gekeken naar een aantal 'generieke tools' die in allerlei omgevingen voorkomen. Voorbeelden hiervan zijn visualisatietools, data-editors en procescoördinatoren. Vanuit deze invalshoek zijn modelprogramma's en generieke tools hetzelfde: een systeem bestaat uit modelprogramma's en tools in een voor die toepassing wenselijke configuratie. Beide typen componenten zijn ontwikkeld voor gezamenlijk gebruik, zodat zij op eenvoudige wijze kunnen worden gekoppeld en uitgewisseld.

Ontwerp van een architectuur (hoofdstuk 4)
Met de kennis opgedaan in de domeinverkenning is een architectuur ontworpen. Deze is bekeken vanuit drie verschillende, complementaire gezichtspunten. Deze zijn de statische structuur, de dynamische structuur en de technologisch infrastructuur.

De architectuur bevat, volgens de statische benadering, een beschrijving van de verschillende bouwstenen die een rol spelen in het raamwerk, van framework tot aan modelelement. Het SRW wordt gezien als een 'container' voor de componenten tool en modelprogramma. De modelprogramma's en tools moeten zich aanmelden bij het raamwerk, zodat aan het raamwerk bekend wordt gemaakt welke raamwerkcomponenten aanwezig zijn.

Naast registratie van een component binnen het raamwerk moeten componenten onderling gekoppeld kunnen worden om een configuratie van modelprogramma's en tools samen te stellen. Hierbij worden de gevraagde en te leveren diensten van deze twee componenten vergeleken. Indien bijvoorbeeld een modelprogramma voor het oppervlaktewater alleen kan functioneren als de drainageflux wordt aangeleverd, zal koppeling moeten plaatsvinden met een component die deze drainageflux kan leveren.

Binnen het SRW moeten bestaande modelprogramma's kunnen worden opgenomen. Om bestaande modelprogramma's te laten communiceren met andere componenten in het raamwerk moet een schil rond de bestaande modelprogramma gemaakt worden. De schil zorgt voor de vertaalslag van de eigen modelprogramma-structuur naar de structuur van SRW. In de architectuur is extra aandacht besteed aan de rol van generieke tools. Hierbij wordt onderscheid gemaakt tussen linked tools en in-process tools. Generieke tools zijn, evenals modelprogramma's, raamwerkcomponenten. Naast registratie van een generieke tool binnen het raamwerk moeten generieke tools gekoppeld kunnen worden aan modelprogramma's en onderling.

Bij de dynamische benadering is gelet op de werking van het raamwerk en de wijze waarop de componenten samenwerken. Modelcomponenten worden in het raamwerk opgenomen om daarin op een gegeven moment uitgevoerd te worden. Behalve dat je een modelcomponenten kan 'runnen' is het voor dynamische koppelingen geschikter om als gebruiker te 'vragen om uitvoer'. Als het modelprogramma zelf weet dat deze uitvoer nog berekend moet worden, zal het model aan het rekenen gaan. Dit gebeurt dus min of meer automatisch. Deze manier van uitvoering is geschikt voor dynamische koppelingen: model 1 vraagt model 2 om output, model 2 gaat rekenen en levert output, model 1 kan weer verder en kan later model 2 opnieuw om (andere) output vragen.

Migratie van software (hoofdstuk 5)
Teneinde bestaande modelprogramma's te kunnen gebruiken in een SRW moeten deze applicaties 'gemigreerd' worden, zodat ze aan de door het SRW gestelde eisen voldoen. Uitgangspunten voor een succesvolle migratie zijn:

- De applicatie speelt een rol in de nieuwe situatie;
- Bekend is waar het systeem in de nieuwe situatie aan moet voldoen;
- De applicatie kan worden omgezet naar een nieuwe situatie;
- De migratie van de software levert een voordeel op.

Om te bepalen hoe zinvol en voordelig een migratie is, moet een 'assessment' worden uitgevoerd. De methode Software Reengineering Assessment van het Amerikaanse Ministerie van Defensie biedt een houvast welke migratiestrategie het meest van toepassing is. Eerst wordt daarbij de technische haalbaarheid bepaald waarbij vastgesteld wordt of de migratie haalbaar, uitvoerbaar en zinvol is op technische gronden. Het resultaat is een lijst applicaties of software-componenten die voor reengineering in aanmerking komen en de wijze waarop dat moet geschieden. Vragen over de organisatie en te migreren systemen zijn daarvan van belang. Bij een positief resultaat wordt bekeken welke strategieën van toepassing kunnen zijn.

Daarna wordt de economische haalbaarheid vastgesteld. Hier wordt gekeken welke strategieën economisch gezien het meest zinvol zijn. De belangrijkste vraag is: hoeveel tijd kost het om de bestaande programmatuur opnieuw op te bouwen en hoeveel om bestaande software te behouden en te voorzien van een 'schil'? Uitgaande van de doelstellingen van het SRW wordt ook samenwerking tussen de onderzoeksinstellingen als economisch voordeel beschouwd.

De besluitvorming wordt eveneens bepaald door de omgeving en organisatie waarbinnen de software wordt gebruikt, de ervaring van programmeurs en lange termijndoelstellingen van de software. In een later stadium van het project zal de migratie meer aandacht moeten krijgen.

Aanbevelingen (hoofdstuk 6)
Een aantal technische ontwikkelingen zijn dermate belangrijk dat aansluiting daarbij onafwendbaar lijkt. De rol van OpenGIS bij de implementatie van geometrie en structuren is cruciaal voor toekomstige aansluiting bij andere systemen. Distributie van componenten over meerdere platforms vereist communicatie tussen die verschillende platforms en dus een methode om te weten waar een component aanwezig is, en op welk platform die uitgevoerd kan worden.

Voor het werken met componenten zijn verschillende industrie-standaarden ontstaan. Naast communicatie protocollen tussen componenten is ook gestandaardiseerd op de definitie van een component, het zogenaamde interface. Het is noodzakelijk om goede afspraken te maken over hoe een component er uitziet en hoe een component aanspreekbaar is.

Wil modelkoppeling in het SRW tot een succes komen, dan dienen modelbouwers en -beheerders rekening te houden met de afgesproken standaarden, hetgeen consequenties heeft voor de interne organisatie van zo'n component. Het gaat hierbij niet alleen om de technische eenduidigheid, maar ook om de semantiek: beide componenten moeten hetzelfde 'verstaan'. Hetzelfde geldt voor de generieke tools. De afspraken voor Adventus, waarbij al zoveel mogelijk uniformiteit in informatie nagestreefd wordt, zijn een goed begin.

Realisatie van het SRW m.b.v. de architectuurbeschrijving vereist kennis van het domein en de toepassingen. Zaken die nu nog onbekend zijn bij de ontwikkelaars en domeinexperts kunnen in de toekomst tot complicaties leiden.

1. Inleiding

1.1 Korte historie

In het Nederlandse waterbeheer wordt de integrale benadering steeds vaker toegepast. Voor beleidstudies, operationeel beheer en toegepast onderzoek zijn modelinstrumenten beschikbaar. Deze zijn veelal opgebouwd uit zeer uiteenlopende modelprogramma's die de verschillende werkvelden in het waterbeheer vertegenwoordigen (ecologie, economie, ruimtelijke ordening, morfologie, waterkwantiteit, waterkwaliteit, enz.). Daarbij wordt samengewerkt door zeer veel partijen. Het hele proces van de onderlinge koppeling van de verschillende modelprogramma's en gegevensbestanden, de aansturing van de modelprogramma's, het definiëren van de invoer en visualisatie van gegevens en modelresultaten is hierdoor een tijdrovende en lastige onderneming geworden [IGT97].

Deze ontwikkeling heeft geleid tot een groeiende vraag naar een flexibele, modulaire, gestandaardiseerde structuur voor een modelinstrumentarium ten behoeve van het integraal waterbeheer, een zogenaamd Standaard Raamwerk Water, hierna afgekort als SRW. In een dergelijk RSW moeten gemakkelijk verschillende modelprogramma's en databases - ook afkomstig van verschillende organisaties - gekoppeld en aangestuurd kunnen worden. De ontwikkeling van een SRW sluit ook aan bij de wens van veel in het waterbeheer betrokken organisaties, om steeds intensiever samen te gaan werken en op een efficiënte manier met kennis en middelen om te gaan. Het project 'Architectuur Standaard Raamwerk Water' is geïnitieerd vanuit het Aquest-programma. Aquest is gericht op de verdere ontwikkeling van de beleids- en planvorming in het Nederlandse waterbeheer, door goede informatievoorziening en communicatie.

1.2 Aquest en drie werkgroepen

In het kader van Aquest is in februari 1997 een discussie opgestart over de mogelijke ontwikkeling van een Standaard Raamwerk voor modelprogramma's in het waterbeheer, later SRW genoemd. Dit heeft in eerste instantie geresulteerd in de oprichting van een drietal werkgroepen: 'Generieke Tools', 'Good Modelling Practice' en 'IT-koppeling van modellen'.

De werkgroep 'Generieke Tools' heeft een eerste structuur van het SRW opgezet en een inventarisatie uitgevoerd naar de tools die beschikbaar zijn als mogelijke bouwstenen voor een SRW. De resultaten van de inventarisatie zijn gerapporteerd in [IGT97].

De werkgroep 'Good Modelling Practice' heeft een project in gang gezet, met als doelstelling 'het stimuleren van het op een juiste manier omgaan met modellen'. De studie heeft geresulteerd in het 'Handboek Good Modelling Practice' [GMP99].

De werkgroep 'IT-koppeling van modellen' is formeel geen werkgroep onder het SRW, maar een al bestaande groep die zich bezighield met de LWI-projecten 'OMT-case' en 'Architectuur-ontwerp complexe modelsystemen'. Centraal staat hier de koppeling van de modellen SIMONA van Rijkswaterstaat en Delft-3D van WL | Delft Hydraulics. Via deze groep vindt afstemming plaats over standaardisatie binnen het SRW en 2/3-dimensionale hydraulische modellen.

Voor het SRW is inmiddels een vierde groep actief die zich bezighoudt met de problematiek van eigendoms- en gebruiksrechten, onderhoud en beheer van software en gegevens-bestanden, die voor uitwisseling in aanmerking komen.

1.3 Het project Architectuur Standaard Raamwerk Water

Dit project beoogt een architectuur te beschrijven voor een SRW. De architectuur vormt de grondslag voor een vervolgproject, dat een eerste aanzet tot de ontwikkeling het SRW gaat geven. Een eerste vereiste is, dat de architectuur onderbouwd en breed gedragen moet zijn. Omdat gebruikers verschillende eisen stellen aan het instrumentarium, maar ook veel overlappende (en generieke) eisen hebben ten aanzien van de structuur en functionaliteiten, wordt voor een aanpak gekozen op basis van componenten. De architectuur beschrijft het samenwerken van, en communiceren tussen verschillende componenten (in verschillende constellaties).

Het project is uitgevoerd door een kernteam van IT specialisten uit het waterbeheer die de het architectuur-ontwerp gemaakt hebben (A-Team). Daarvoor is de bijdrage van een grote groep van Gebruikers (modelleurs, onderzoekers, beleidsmedewerkers) uit het waterbeheer onontbeerlijk geweest (G-Team). De kwaliteit en voortgang van het project is bewaakt door een groep begeleiders (B-Team).

Door de werkgroep 'Generieke Tools' is in november 1998 tevens een Programma van Eisen opgesteld, dat onderscheid maakt tussen een de Standaard Raamwerk Architectuur en de Standaard Raamwerk Water, die daarop is gebaseerd. De doelstellingen worden nader beschreven in hoofdstuk 2. Op enkele punten is het architectuurteam hiervan afgeweken.

1.4 Aanpak bij het ontwerpen van de architectuur

Om de architectuur breed inzetbaar en generiek te maken, is in het ontwerptraject voor een architectuur bewust niet gekozen voor het standaardstramien van informatieanalyse, functioneel ontwerp en technisch ontwerp. Ervaringen met deze 'traditionele' aanpak leren dat dit veelal starre, inflexibele structuren oplevert. Om ervoor te zorgen dat de architectuur uitbreidbaar, flexibel en aanpasbaar is, is gekozen voor een moderne aanpak als grondslag voor een op componenten gebaseerde ontwerptraject voor verdere ontwikkeling. De kern van deze aanpak bestaat uit het abstraheren van het domein, een zogenaamde 'domeinanalyse', in plaats van het opsommen van functionele eisen. Door het gehele domein in abstractie te beschrijven wordt een generiek beeld van de applicaties van dat domein gegeven [WAL98]. Hiermee ontstaat een architectuur die aanpasbaar en uitbreidbaar is. Het ontwerp van de architectuur is een eerste technische uitwerking van de domeinmodel. In een vervolgproject zal deze architectuur nader gespecificeerd moeten worden.

De beoogde werkwijze bij de uitvoering van het project 'Architectuur Standaard Raamwerk Water' is in detail beschreven in het werkplan [WAL98]. Centraal in de aanpak staat de participatieve modellering. Dit heet met name participatief, omdat de gebruikers een belangrijke rol hebben gekregen, niet alleen om het domein 'water' aan de automatiseerders te vertellen, maar ook om het domeinmodel steeds te verifiëren. Het resulterende model is dan de afspiegeling van de gemeenschappelijke visie van de gebruikers en de automatiseerders op het domein.

Het participatieve karakter is verder tot uitdrukking gekomen in de projectopbouw, die bestond uit een vijftal opeenvolgende workshops; de verslagen van de workshops (ochtendsessies met het G(ebruikers)-team en A(utomatiseerders)-team, middagsessies van het A-team) zijn opgenomen in Appendix B. Na afronding van de workshops is door het A-team een eerste beschrijving gemaakt van de architectuur en de componenteninterfaces van het SRW. Hierin is op basis van het domeinmodel een indeling gemaakt in componenten. De samenhang van de componenten, de communicatie daartussen en daarmee (dus de interfaces) zijn hierin meegenomen.

Dit rapport is voorgelegd aan een groot aantal, zoveel mogelijk onafhankelijke, reviewers voor toetsing van kwaliteit en haalbaarheid van de voorgestelde aanpak. De resultaten van die toetsing [ELSWIJK99] zijn voor een groot deel in dit rapport verwerkt.

1.5 Onderwerpen van de hoofdstukken van dit rapport

In dit rapport wordt een beschrijving van de architectuur en de componenten ten behoeve van het SRW gegeven. De inleidingen bij de hoofdstukken geven een uitvoerige beschrijving van de onderwerpen.

Hoofdstuk 2 gaat nader in op de achtergronden van dit project. Achtereenvolgens komen aan de orde: het Programma van Eisen van de werkgroep Generieke Tools (2.1), een beschrijving van de belanghebbenden en hun rol in het geheel (2.2), de knelpunten en kwaliteitseisen voortvloeiend uit het Programma van Eisen (2.3), relevante standaarden waarbij kan worden aangesloten (2.4) en nader geformuleerde uitgangspunten van het team dat deze aanzet tot een architectuur heeft ontwikkeld voor het SRW heeft ontwikkeld.

In hoofdstuk 3 wordt aangegeven welke vakdomeinen in het waterbeheer een rol spelen en hoe deze er uit zien. Uit een vijftal workshops waarin over verschillende invalshoeken (3.2) op het probleemdomein werd gediscussieerd komen een aantal domeinmodellen met basis-elementen en compartimenten naar voren. Deze invalshoeken blijven gescheiden bij modelprogramma's (3.3).

Het hoofddoel van dit project, de beschrijving van de architectuur van de SRW, is uitvoerig beschreven in hoofdstuk 4, vanuit drie verschillende, complementaire gezichtspunten. Deze zijn de statische structuur (4.2), de dynamiek (4.3) en de technologische infrastructuur (4.4). Bij de beschrijvingen, die onvermijdelijk een sterk informatica-technisch karakter dragen, wordt er vanuit gegaan dat de lezer voldoende kennis heeft van Object Oriëntatie (OO), Unified Modelling Language (UML) en Interface Definition Language (IDL). Paragraaf 4.5 besteedt kort aandacht aan opslag van bronnen, zoals bestanden, databases en gebruikers.

Om een rol in het SRW te kunnen spelen moeten modelprogramma's en tools aan specifieke voorwaarden voldoen; vaak betekent dit dat bestaande software moet migreren (hoofdstuk 5). De voorwaarden hiervoor worden besproken in 5.2, de technische haalbaarheid door reengineering in 5.3, en de economische haalbaarheid in 5.4. Paragraaf 5.5. geeft enkele voorbeelden.

De voorstellen voor vervolgacties staan in hoofdstuk 6, de aanbevelingen.

In de appendices bij dit rapport is relevante achtergrondinformatie opgenomen, waaronder een begrippenlijst in Nederlands en Engels (App. A), het mogelijke gebruik van de stekkerdoos water (App. B), een beschrijving van het wereldwijde consortium OpenGIS ten behoeve van standaarden (App. C), voorbeelden van distributie van objecten (CORBA, DCOM, OpenDoc en OLE) (App. D), de indicatoren die aansluiten bij de kwaliteitseigenschappen (Engels) behandeld in 2.3) (App. E), een lijst van veel gestelde vragen (App. F) en een overzicht van gebruikte symbolen (App. G).

1.6 Terminologie

In dit rapport worden sommige termen door elkaar gebruikt, soms wordt met hetzelfde woord iets heel anders bedoeld, soms worden verschillende termen voor hetzelfde gebruikt. De woordkeuze en terminologie moet dan ook vaak in de context begrepen worden.

Alhoewel de redactie van dit rapport gepoogd heeft zich te conformeren aan gangbare terminologie, zoals bijvoorbeeld staat in het handboek Good Modelling Practice [GMP99], is het gebruik van terminologie onmiskenbaar niet eenduidig en homogeen. Bijvoorbeeld de termen model, modelprogramma en modelapplicatie liggen heel dicht bij elkaar en worden afwisselend gebruikt. We hebben geprobeerd om modelprogramma voor de huidige perceptie van een software-product dat geldt als model te reserveren, en modelapplicatie voor een software-product zoals dat in het Standaard Raamwerk Water wordt beschouwd. Het woord model wordt voor beide gebruikt hetgeen meestal voortkomt uit het dagelijkse taalgebruik. Hoe dan ook, bij voorbaat onze excuses voor de eventuele ontstane verwarring.

In appendix A is een begrippenlijst opgenomen, met daarin de nederlandse termen en engelse vertalingen. Het engels is in principe gereserveerd voor de concrete onderdelen van het Standaard Raamwerk Water.

In appendix G is een symbolenlijst opgenomen ten behoeve van de interpretatie van de UML diagrammen.

2. Nadere achtergronden

2.1 Programma van eisen

Door de werkgroep 'Generieke Tools' is in november 1998 een Programma van Eisen [PvE98] opgesteld waarin de voorwaarden aan het Standaard Raamwerk op globaal niveau zijn beschreven. Het programma van eisen maakt onderscheid tussen twee producten: de Standaard Raamwerk Architectuur en het Standaard Raamwerk Water (SRW). De architectuur ligt ten grondslag aan het SRW. Het SRW wordt ontwikkeld om een aantal knelpunten, zoals in [PvE98] genoemd, op te lossen. Deze knelpunten zijn:

- *Arbeidsintensief:* Er is veel tijd en energie nodig om modellen te gebruiken (K1);
- *Operationaliseren:* Het operationeel maken van modellen kost veel tijd (K2);
- *Aansluiting:* Door beperkte mogelijkheden van koppeling en aansluiting tussen modellen kunnen vragen uit de praktijk niet goed of niet snel genoeg beantwoord worden (K3);
- *Ontoegankelijkheid:* De gebruiksvriendelijkheid en toegankelijkheid van modellen (applicaties) laat te wensen over (K4);
- *Presentatie:* De bestaande modelprogramma's bieden weinig presentatiemogelijkheden (K5);
- *Tijdsdruk:* De tijd voor analyse en het beantwoorden van vragen wordt steeds korter (K6);
- *Modelkoppeling:* Technische koppeling tussen modelprogramma's onderling verloopt slecht (K7);
- *Gegevenskoppeling:* Technische koppeling tussen modelprogramma's en gegevensbestanden verloopt moeizaam (K8).

Daarnaast noemt het Programma van Eisen een aantal uitgangspunten die in de onderstaande opsomming zijn samengevat; de genoemde uitgangspunten hebben invloed op de uiteindelijke architectuur.

- *Openheid:* Het SRW is een zelfstandige applicatie[1] met een open structuur waar functionaliteit in de vorm van modelprogramma's en tools aan toegevoegd kan worden;
- *Centrale metagegevens:* Meta-informatie over modelprogramma's, modelketens, tools en gegevens wordt centraal (op één plaats) binnen het SRW bewaard[2];
- *GUI:* Het SRW biedt een grafische gebruikersinterface;
- *Platformonafhankelijk:* Het SRW moet (zo veel mogelijk) platformonafhankelijk zijn[3];
- *Distributie:* Er is ondersteuning voor het gebruiken van modelprogramma's en tools die zich op andere locaties en andersoortige platformen bevinden.

In het Programma van Eisen wordt een aantal 'standaard' tools beschreven die tezijnertijd vast onderdeel moeten gaan uitmaken van het SRW. Het betreft hier een procesketenbeheerstool, een gegevensverwerking en -beheertool, een selectie- en definitietool, een calibratietool, een analysetool en een presentatietool (zie [PvE98] voor een toelichting op deze tools).

---

1 Traditioneel is een raamwerk een 'applicatieskelet' dat als basis dient voor het bouwen van applicaties. Het SRW gebruikt een iets andere definitie: het raamwerk is een skelet voor het koppelen van applicaties (met name modellen en tools). Het koppelen van applicaties kan gezien worden als het maken van een nieuwe applicatie.

2 Dit uitgangspunt is omwille van flexibiliteit en onderhoudbaarheid aangepast. Zie paragraaf 'Uitgangspunten architectuurteam'.

3 In [PvE98] wordt als meest ideale geval beschreven dat het SRW een applicatie is die volledig platformonafhankelijk is (dus één executeerbaar programma dat draait op elk platform). Dit legt sterke beperkingen op aan de manier waarop de applicatie geïmplementeerd zal worden; in feite biedt alleen de Java-omgeving dergelijke mogelijkheden.

2.2 Belanghebbenden

De architectuur van het SRW kent meerdere belanghebbenden (stakeholders), die op verschillende manieren te verdelen zijn, bijvoorbeeld naar de rol die ze spelen (ontwikkelaar, onderzoeker, etc.) of naar soort organisatie:

- Onderzoeksinstituut;
- Adviesbureau;
- Waterbeheerder (overheid);
- niet-waterbeherende overheid;
- opleidingsinstituut - hieronder vallen, onder andere, het wetenschappelijke onderwijs (universiteit) en het hoger-beroepsonderwijs (HBO);
- softwarebureau - bij een softwarebureau wordt op commerciële basis en op basis van een opdracht, bijvoorbeeld in de vorm van een productspecificatie, programmatuur ontwikkeld;
- belangenorganisatie, en;
- samenwerkingsverband.

De diverse medewerkers van de belanghebbende organisaties zijn onder te verdelen in rollen of functies. In het onderstaande overzicht zijn deze rollen opgenomen. Bij de lijst dient opgemerkt te worden dat het heel goed mogelijk is, of zelfs voor de hand liggend, dat personen binnen een organisatie meer dan één rol vervullen.

- *bestuurder:* de persoon die (financiële) bevoegheid heeft beslissingen te nemen over het laten uitvoeren van opdrachten;
- *beleidsmedewerker:* de persoon die de resultaten van de simulaties gebruikt voor het formuleren van beleid;
- *modeltoepasser:* de persoon die de uiteindelijke applicatie gebruikt (eindgebruiker), maar geen precieze kennis hoeft te hebben over hoe de applicatie werkt;
- *student:* de persoon die verbonden is aan een opleidingsinstituut en simulaties gebruikt om kennis over het domein te verwerven;
- *onderzoeker:* de persoon die net als de modeltoepasser het modelprogramma gebruikt, maar met kennis over de werking van het programma en de besturing;
- *modelontwikkelaar:* de persoon die het model maakt;
- *softwareontwikkelaar:* degene die in opdracht software ontwikkelt;
- *gegevensbeheerder:* de persoon belast met de beschikbaarheid en betrouwbaarheid van gegevens (die gebruikt worden in simulaties);
- *applicatiebeheerder:* degene die de ontwikkeling en evolutie van een applicatie beheert;
- *systeembeheerder:* de persoon belast met het operationeel houden van het systeem; het 'systeem' is in dit verband het SRW met alle gekoppelde programma's en gegevensbronnen.

De rollen komen op diverse plaatsen voor binnen de genoemde organisaties, maar niet overal hoort daar een direct belang bij met betrekking tot het SRW. De meest relevante combinaties van rollen en organisatie zijn met een '√' weergegeven in Tabel 2-1.

Tabel 2-1. Rollenmatrix van belanghebbenden bij het SRW.

	Bestuurder	Beleidsmedewerker	Modeltoepasser	Student	Onderzoeker	Modelontwikkelaar	Software-ontwikkelaar	Gegevensbeheerder	Applicatiebeheerder	Systeembeheerder
Onderzoeksinstituut			√		√	√	√	√	√	√
Adviesbureau			√		√	√	√		√	√
Waterbeheerder	√	√	√		√	√		√		√
Niet-waterbeherende overheid	√	√	√					√		√
Universiteit/HBO			√	√	√	√	√	√	√	√
Softwarebureau							√		√	√
Belangenorganisatie	√	√	√					√		√
Samenwerkingsverband	√	√	√		√	√	√			√

De belangen kunnen variëren per soort organisatie, maar ze verschillen vooral per rol. In de onderstaande tabel zijn de belangen per belanghebbende rol opgenomen; waar nodig is nuance naar soort organisatie aangebracht.

Tabel 2-2. Belang bij een SRW van de verschillende rollen.

Rol	Belang
bestuurder	**minimaliseren van de kosten** - het grootste belang van de bestuurder is het zo efficiënt mogelijk gebruiken van de beschikbare financiën voor ontwikkeling, gebruik en integratie van modellen. **eenduidig antwoord (consensus)** - de resultaten van een simulatie waarop de bestuurder zijn beslissing moet baseren moeten eenduidig en reproduceerbaar zijn. **onderbouwing** - de resultaten moeten voldoende onderbouwd zijn.
beleidsmedewerker	**eenduidigheid van de gegevens** - resultaten en andere gegevens moeten inzichtelijk en begrijpelijk zijn. **onderbouwing** (zie 'bestuurder') **snel antwoord** - de tijd tussen vraag en antwoord moet zo kort mogelijk zijn; dit heeft te maken met de snelheid waarmee nieuwe modellen worden samengesteld en ontwikkeld, maar ook met de beschikbaarheid van modellen. **betrouwbaarheid** - van de resultaten moet bekend zijn of ze betrouwbaar zijn of niet (met andere woorden: hoe groot is de kans dat de werkelijkheid radicaal anders is dan voorspeld?). **presentatie** - de resultaten moeten mooi, duidelijk en helder gepresenteerd kunnen worden.
modeltoepasser & student	De belangen van de modeltoepasser en de student zijn vrijwel gelijk aan die van modelontwikkelaar (zie verderop), maar daarnaast zijn er nog wat belangen: **eenduidige gebruikersinterface** - modellen en tools moeten zoveel mogelijk overeenstemming vertonen qua gebruikersinterface; dit vergroot de begrijpelijkheid en voorspelbaarheid van applicaties en verlaagt de inleertijd. **beperkte complexiteit** - de applicatie moet zo 'simpel' mogelijk zijn. Dat betekent bijvoorbeeld het afschermen van overbodige complexiteit.

Vervolg tabel 2-2.

Rol	Belang
onderzoeker & modelontwikkelaar	***beperkte (zichtbare) invloed van technologie*** *- de gemaakte technologische keuzen moeten zo min mogelijk van invloed zijn op de wijze waarmee met het Raamwerk gewerkt kan worden.* ***koppelen van modellen en tools*** *- koppelingen tussen modellen onderling en met tools moeten eenvoudig te maken zijn; het maken van de koppelingen zal meestal nog veel domeinkennis vragen (bijvoorbeeld met betrekking tot verschaling), maar de techniek (het Raamwerk) mag het niet in de weg staan.* ***eenvoudige gegevensverwerking*** *- gegevens die nodig zijn voor het draaien van simulaties moeten eenvoudig ingevoerd en bewerkt kunnen worden; dit vereenvoudigt het werken en verkleint de kans op fouten.* ***beïnvloeden van rekensnelheid*** *- de prestatie van modellen zal afnemen als modellen onderling worden gekoppeld; er moet zoveel mogelijk invloed uitgeoefend kunnen worden op de snelheid van de simulatie.* ***onafhankelijkheid van leveranciers*** *- het Raamwerk en de componenten, die in de loop van de tijd zullen ontstaan, moeten zoveel mogelijk onafhankelijk zijn van leveranciers.*
software-ontwikkelaar	***uitwisselbaarheid van gegevens*** *- gegevens die gebruikt of geproduceerd worden door modellen moeten uitwisselbaar zijn tussen verschillende modellen en verschillende tools.* ***minimaliseren ontwikkeltijd*** *- de snelheid waarmee nieuwe modellen worden ontwikkeld (of samengesteld uit bestaande modellen) moet zo klein mogelijk zijn.* ***verbeteren herbruikbaarheid*** *- de gebouwde software moet zoveel mogelijk nogmaals te gebruiken zijn.* ***goede functionele specificaties*** *- om software (componenten) te kunnen baseren op een architectuur moeten de specificatie helder en eenduidig zijn.* ***duidelijkheid over organisatie*** *- als onderdelen van het Standaard Raamwerk (zowel modellen als tools) niet naar behoren presteren moet duidelijk vastliggen wat hiervoor het aanspreekpunt is.*
gegevensbeheerder	***uitwisselbaarheid van gegevens*** *- gegevens die gebruikt of geproduceerd worden door modellen moeten uitwisselbaar zijn tussen verschillende modellen en verschillende tools.*
systeembeheerder	***minimaliseren van beheers- en onderhoudskosten*** *- het beheer en onderhoud aan modelapplicaties, gegevens(bronnen) en tools moet zo min mogelijk kosten.* ***platformonafhankelijkheid*** *- de software moet zoveel mogelijk platform-onafhankelijk zijn. Introductie van een nieuw platform kan eventueel wel leiden tot hercompilatie.*

Alle personen in de genoemde rollen zullen op een of andere manier direct of indirect in aanraking komen met het SRW. Onder de 'eindgebruikers' vallen de modeltoepasser, onderzoeker, student, modelontwikkelaar en de software-ontwikkelaar. Bij het ontwerp en de ontwikkeling van het Standaard Raamwerk zullen de belangen, waar relevant, meegenomen moeten worden in de afwegingen.

De verdeling naar organisatie en rollen geeft de mogelijkheid voor toekomstige ontwikkelingen om te werken aan specifieke tools. Hierbij kan bijvoorbeeld worden gedacht aan tools specifiek voor modelontwikkelaars of voor adviesbureaus.

2.3 Kwaliteitseisen

De voorwaarden en uitgangspunten uit het Programma van Eisen kunnen worden vertaald naar een aantal (meer concrete) kwaliteitseisen. Preciezer gezegd: om een knelpunt te verwijderen moet (zo goed mogelijk) worden voldaan aan één of meer kwaliteitseisen.
In de onderstaande tabel zijn deze kwaliteitseisen opgenomen. De lijst is aangevuld met kwaliteitseigenschappen die direct uit de belangen zijn af te leiden. De eisen zijn overgenomen uit het Extended ISO Model of Software Quality [QUINT2] (zie Appendix E voor een overzicht van en toelichting op de kwaliteitseigenschappen).

Tabel 2-3. Kwaliteitseisen voor SRW.

Knelpunt/Belang	***Kwaliteitseigenschappen***
arbeidsintensief (K1)	*operability, understandability, helpfulness*
operationaliseren (K2)	*installability, adaptability*
aansluiting (K3)	*compliance*
ontoegankelijk (K4)	*user-friendliness, clarity, understandability*
presentatie (K5)	*customisability, attractivity*
tijdsdruk (K6)	*time behaviour, interoperability, adaptability, changeability*
modelkoppeling (K7)	*compliance, traceability*
gegevenskoppeling (K8)	*compliance*
openheid (uitgangspunt)	*compliance*
platformonafhankelijk (uitgangspunt)	*adaptability*
distributie (uitgangspunt)	*adaptability, interoperability*
eenduidig antwoord, onderbouwing, betrouwbaarheid	*traceability, accuracy*
beperkte complexiteit, beperkte zichtbare invloed van technologie	*user-friendliness, clarity*
beïnvloeden van rekensnelheid	*adaptability, changeability, customisability*
onafhankelijkheid van leveranciers	*compliance*
verbeteren herbruikbaarheid	*reusability*
minimaliseren van beheers- en onderhoudskosten	*stability, manageability*

Het Extended ISO Model of Software Quality[4] geeft bij elke kwaliteitseigenschap één of meer indicatoren (metrieken) en protocollen voor het vaststellen van deze indicatoren. In latere stadia kan aan de hand van metingen worden vastgesteld in welke mate aan de eisen (kwantitatief) is voldaan. Zie [QUINT2] voor een compleet overzicht.

De kwaliteitseigenschappen waaraan de architectuur, het SRW en te bouwen raamwerkcomponenten moeten voldoen, komen niet allemaal tot uitdrukking in dit ontwerp van de architectuur. User-friendliness, clarity en helpfulness, bijvoorbeeld, komen pas tot uiting bij de realisatie van de software (het raamwerk, tools en model-applicaties). In tabel 2-4 zijn de kwaliteitseigenschappen opgenomen waar in de architectuur aandacht aan is besteed.

[4] Het ISO Model of Software Quality is een wereldwijde standaard op het gebied van het specificeren van de kwaliteit van softwareproducten.

Tabel 2-4. Kwaliteitseisen die in de architectuur tot uitdrukking komen.

Kwaliteitseigenschap	***Architectuur***
Operability	*Installatie en registratie van componenten is onderdeel van de architectuur. Koppelingen tussen modelapplicaties en tools kunnen grotendeels automatisch worden gemaakt en gecontroleerd.*
Installability	*Installatie en registratie van componenten is onderdeel van de architectuur; de installatie van het SRW zelf is niet uitgewerkt (geen onderdeel van de architectuur).*
Adaptability	*In de architectuur wordt geen keuze gemaakt voor een technische infrastructuur, daarom is niet aan te geven in welke mate het raamwerk 'portable' is. Aan de andere kant zijn de aannames over de uiteindelijke infrastructuur beperkt en kan het binnen verschillende infrastructuren worden geimplementeerd.* *Met behulp van in-process tools kan de werking van het totale model (de configuratie) op generieke wijze worden beinvloed. Als het aanbod van deze tools groot genoeg is, kan dit automatisch met zo min mogelijk inspanning van de gebruiker.*
Compliance paragraaf 2.4)	*Als basis voor domeinconcepten (bijvoorbeeld voor het beschrijven van diensten) kan gebruik gemaakt (zie worden van Adventus.* *Voor het beschrijven van interfaces wordt de standaard OMG Interface Definition Language (IDL) gebruikt.* *Het uitwisselen van geometrische informatie sluit aan op OpenGIS.*
Time behaviour	*Het is onmogelijk om in de architectuur het tijdsgedrag van het samengestelde model te beschrijven. Dit hangt uiteraard af van de componenten waaruit het model is samengesteld. De architectuur biedt wel een aantal gereedschappen om de prestatie te beïnvloeden:* • *Tools die deel uitmaken van een configuratie kunnen aan- en uitgezet worden;* • *In-process tools kunnen gebruikt worden voor optimalisatie;* • *In de configuratie kan worden aangegeven dat bepaalde modellen parallel kunnen rekenen;* • *Modellen kunnen als batch gedraaid worden.*
Interoperability	*Deze eigenschap hangt samen met de keuze voor een bepaalde technische infrastructuur (middleware). Deze keuze wordt niet in dit document gemaakt. De architectuur legt wel sterk de nadruk op interfaces en vergroot daarmee ook de koppelbaarheid met andere systemen.*
Reusability	*De generieke tools, zoals beschreven, zijn nog niet gebouwd. Met behulp van de communicatiemechanismen, diensten en interfaces kunnen wel eenvoudig(er) herbruikbare bouwstenen (componenten) worden ontwikkeld.* *In de architectuur wordt duidelijk onderscheid gemaakt tussen modelapplicaties en tools. In het verleden was het gebruikelijk om één enkele applicatie te ontwikkelen waarin zowel het rekenhart als de gebruikersinterface zaten. De applicatie als zodanig was niet herbruikbaar. Door de belangrijke verantwoordelijkheden te scheiden, neemt de herbruikbaarheid sterk toe.*

2.4 Standaarden in het domein

Dit project beschrijft een architectuur voor de gestandaardiseerde samenwerking tussen modelapplicaties in het vakdomein water. Het raakt daardoor ook de IT-opbouw van modelapplicaties. Waar mogelijk kan daarbij aangesloten worden bij bestaande standaarden die reeds geaccepteerd zijn in dit vakdomein. Er zijn wereldwijd diverse organisaties op het gebied van standaardisering (ISO, ANSI). Ook in Nederland wordt er gestandaardiseerd, door het Nederlands Normalisatie Instituut - NNI (bv. NEN 1878 of NEN 3610).

Al genoemd in het plan van aanpak is de aansluiting bij GMP (Good Modelling Practice) en Adventus. Verder is als uitgangspunt gesteld dat het SRW dient aan te sluiten bij de Stekkerdoos Water (zie paragraaf 4.5).

Veel van de modellen in het domein water gebruiken ruimtelijke gegevens. Sommige modellen zijn zelfs gemaakt in een GIS (Geografische Informatie Systeem). Een internationale organisatie die zich bezighoudt met het standaardiseren van ruimtelijke gegevens en ruimtelijke modellen is het OpenGIS Consortium (zie ook Appendix C).

Adventus

Door het ontbreken van een standaard was dat bij de bouw van informatiesystemen voor waterschappen veel tijd en geld nodig voor de onderlinge aansluiting van informatiesystemen. Gegevensuitwisseling tussen informatie-systemen ging gepaard met een veelheid aan uitwisselingsrelaties, interfaces en maatwerk.

Daarom heeft de Unie van Waterschappen samen met de waterschappen een gegevens-standaard ontwikkeld ten behoeve van de realisatie van informatiesystemen. Hiertoe is tevens een logisch gegevensmodel en een technisch model ontwikkeld. Dit heet het Adventusstelsel en het bestaat uit:

- Gegevensstandaard water;
- Gegevensmodel Adventus;
- Technisch Model Adventus (TMA);
- Stekkerdoos Water;
- Beschrijving van de invoering in organisaties.

Adventus onderscheidt standaardisatie van:

- *definities van gegevens*, waardoor de gelijkbaarheid wordt vergroot en uitwisseling met derden eenvoudiger wordt;
- *gegevensmodellen*, waardoor een integrale benadering mogelijk is en gegevens-uitwisseling eenvoudiger wordt.
- *(centrale) gegevensopslag*, wat toegankelijkheid maximaliseert met een minimale dubbele opslag.

Voor het onderhavige project geldt dat tenminste bij het interbestuurlijke deel van Adventus moet worden aangesloten; dit is dat deel van het Adventus-stelsel waarover afstemming tussen STOWA en Rijkswaterstaat heeft plaatsgevonden.

Bij Adventus dient tot slot te worden opgemerkt dat het zich richt op één-dimensionale modellen.

OpenGIS

Het OpenGIS Consortium is opgericht om afstemming te bereiken in de opslag en het gebruik van ruimtelijke gegevens. Het bestaat uit leveranciers en gebruikers van GIS systemen. Door deze brede samenstelling is OpenGIS in staat om industriestandaarden te zetten. De afstemming geschiedt door 'OpenGIS approved' standaarden af te spreken. OpenGIS onderscheidt veertien onderwerpen voor standaardisatie (zie appendix C) waarvan een aantal betrekking heeft op het onderhavige project. Sommige onderwerpen hebben direct of indirect te maken met het beschrijven en uitwisselen van ruimtelijke objecten (o.a. punten, lijnen, vlakken).

De gegevens die in het SRW gebruikt worden zijn voor een groot deel ruimtelijke gegevens. Hiervoor lijkt het logisch om aan te sluiten bij de OpenGIS standaarden. Op dit moment is met name de definitie van ruimtelijke objecten van belang, omdat hiervan al implementatie specificaties zijn. Belangrijke GIS leveranciers als ESRI en Oracle hebben inmiddels de door OpenGIS goedgekeurde definities in hun gegevensmodellen verwerkt.

Andere OpenGIS onderwerpen die voor het SRW gebruikt kunnen worden zijn:

- *Ruimtelijke variatie*, dit beschrijft hoe de variatie van een variabele in een gebied verloopt met behulp van een functie (vaak wordt hiervoor een grid gebruikt). Bij b.v. de analytische elementen methode kan deze modellering direct gebruikt worden.
- *meta-informatie*, wat de informatie over modellen, tools en gegevens beschrijft. Dit kan ook in het SRW gebruikt worden. Overigens sluit het OpenGIS Consortium zich voor metadata aan bij de ISO standaarden op dit gebied die in september 2000 gereed komen.
- *Informatie gemeenschap*, dit beschrijft een geo-informatie gemeenschap als een verzameling systemen of individuen die met succes ruimtelijke informatie uit kunnen wisselen. Het domein water kan één van deze informatie gemeenschappen zijn.

2.5 Uitgangspunten voor het architectuur-team

Bij de ontwikkeling van het ontwerp voor de architectuur van het SRW heeft het team de volgende uitgangspunten gehanteerd:

1. *Programma van Eisen is niet dwingend* - In overleg met de opstellers van het Programma van Eisen is afgesproken dat dit niet dwingend is, met name daar waar oplossingen worden gesuggereerd. Eén concrete oplossing die wordt genoemd, is een drie-lagenarchitectuur. Hoewel deze architectuur conceptueel precies past op dat model, wordt de drie-lagenarchitectuur niet met name genoemd;
2. *Hergebruik van bestaande software* - Bestaande modellen (modelapplicaties) en gegevens moeten in het Standaard Raamwerk gebruikt kunnen worden. Aansluiting gaat niet vanzelf, maar vereist inspanning van ontwikkelaars en eventueel onderzoekers. In hoofdstuk 5 wordt aandacht besteed aan het migreren van bestaande systemen;
3. *Centrale metagegevens* - In het Programma van Eisen wordt als uitgangspunt gesteld dat alle meta-informatie over modellen en tools centraal is opgeslagen. Centraal opgeslagen gegevens over verschillende componenten kunnen leiden tot een slecht te onderhouden omgeving. Om die reden heeft het architectuurteam nuance aangebracht. Het SRW weet welke componenten (modellen en tools) beschikbaar zijn en hoe meta-informatie over deze componenten te verkrijgen is; de componenten zelf bieden diensten die de juiste meta-informatie opleveren. Zo blijft de meta-informatie gelocaliseerd op de juiste plaats;
4. *Technologiekeuze* - De ontwerpers van dit architectuurconcept doen nog geen uitspraak over de te kiezen infrastructuur. De eerste reden hiervoor is dat door het opnemen van technologische details dit document snel zal verouderen. Ten tweede is het binnen het gegeven tijdsbestek niet mogelijk om een gedegen onderzoek te doen naar de mogelijke toepassing van technologie. Tot slot vergt het kiezen van één of meer platformen goed overleg tussen de verschillende participerende instituten;
5. *Calibratietool* - In de eerste fase van oplevering wordt niet gewerkt aan een calibratietool, hoewel een dergelijk tool wel gespecificeerd is in het Programma van Eisen;
6. *OpenGIS* - Waar mogelijk zal aansluiting worden gezocht bij de OpenGIS standaarden. Dit heeft vooral betrekking op hoofdstuk 4 waarin de componenten worden beschreven. Deze componenten wisselen geo-informatie uit.

3. Domeinverkenning

3.1 Inleiding

In het SRW moeten diverse simulatiemodellen en databases gemakkelijk aan elkaar gekoppeld kunnen worden. Om dit te kunnen is het nodig dat de architectuur voor het SRW zeer flexibel en tegelijkertijd robuust is.

Flexibiliteit, met name uitbreidbaarheid en aanpasbaarheid, wordt in dit project nagestreeft door een generieke beschrijving van het vakdomein te maken [WAL98]. In het project is het vakdomein omschreven als 'Simulatiemodellen in het Integraal Waterbeheer'. De generieke beschrijving ontstaat door verregaande abstractie van de manier waarop simulatiemodellen de processen beschrijven. Hierdoor ontstaat een beeld van de concepten en hun onderlinge relaties. De aldus onderscheiden concepten zijn in het algemeen statischer en stabieler dan de concepten waarop applicaties zijn gebaseerd. Immers, concepten in een vakgebied wijzigen minder snel dan de manier waarop gebruikers die concepten willen gebruiken voor het software-matig oplossen of analyseren van problemen (traditioneel vastgelegd in functionele specificaties voor applicaties).

Koppelingen tussen simulatiemodellen zijn technisch relatief eenvoudig op te lossen; de 'semantische koppeling' tussen de modellen is echter veel complexer[5]. Modellen wisselen onderling informatie uit en moeten daarvoor dezelfde taal spreken. Met andere woorden, de concepten waarin de simulatiemodellen redeneren moeten dezelfde zijn. Domeinverkenning (of domeinanalyse) biedt hiervoor een oplossing doordat de focus ligt op het probleemdomein en de concepten die daarin een rol spelen in plaats van één of meer specifieke applicaties.
Een opsplitsing van het domein, zoals verderop beschreven in paragraaf 3.2.1, biedt de mogelijkheid om modellen te categoriseren. Om koppeling tussen modellen van verschillende oorsprong mogelijk te maken, is een dergelijke 'landkaart' een handig, zo niet onmisbaar, hulpmiddel. Bij het categoriseren van modellen kunnen verantwoordelijkheden over de modellen worden verdeeld.

In de voorliggende domeinverkenning wordt vanuit een aantal invalshoeken naar het probleemdomein 'Simulatiemodellen in Integraal Waterbeheer' gekeken en wordt nagegaan welke concepten daarbij een rol spelen. In verband met het bereiken van consensus is het van belang dat de geïdentificeerde concepten een voldoende mate van abstractie bezitten. De relevante concepten dienen als basis voor de architectuur van het SRW, welke in hoofdstuk 4 verder zal worden uitgewerkt.

De domeinverkenning is uitgevoerd door het houden van een vijftal workshops waarin door gebruikersdeskundigen en informatici gediscussieerd is over de verschillende invalshoeken op het probleemdomein. Vanwege de grote overeenkomsten en terwille van de efficiëntie is gebruik gemaakt van het bestaande raamwerk Framework Integraal Waterbeheer (FIW) van Alterra (voorheen DLO-Staring Centrum) [GROENENDIJK98]. Met name zijn uit het FIW het domeinmodel en het klassendiagram gebruikt.

---

[5] Een technische koppeling tussen twee programma's voorziet erin dat er gegevens (een bepaald aantal bytes) uitgewisseld kunnen worden. Bij een semantische koppeling wordt betekenis (semantiek) aan deze gegevens toegevoegd. Als twee modellen onderling bijvoorbeeld waterhoogtes uitwisselen, moeten deze voor beide modellen dezelfde betekenis hebben.

3.2 Invalshoeken

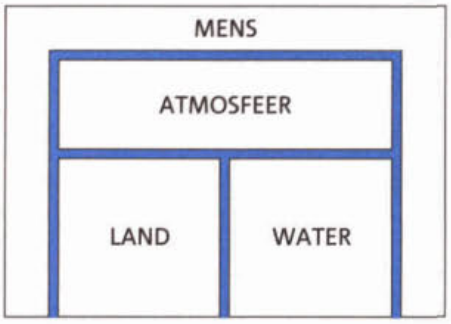
Teneinde de gewenste generieke beschrijving te maken wordt het vakdomein vanuit verschillende invalshoeken bekeken. Elk beschreven domeinmodel speelt een rol in de architectuur van het SRW, die in het volgende hoofdstuk wordt gespecificeerd. In het onderstaande overzicht staan kort de aspecten van die architectuur die beïnvloed zijn door de domeinverkenning:

- *Fysica in Integraal Waterbeheer* - De wijze waarop de fysische wereld is onderverdeeld in afzonderlijke delen ('subdomeinen') is een richtlijn voor de wijze waarop modellen geclassificeerd kunnen worden. Zo is uit het diagram (figuur 3-2) af te lezen dat de onverzadigde zone interactie heeft met de verzadigde zone. Binnen het SRW zou voor elk van deze twee subdomeinen een apart model gebruikt kunnen worden, die volgens een afgesproken manier met elkaar communiceren. Als samenwerking en koppeling tussen modellen op dit niveau gespecificeerd is, is het ook eenvoudiger om een gekoppeld model te vervangen door een soortgelijk. De verdeling is dan ook niet meer verbonden met bepaalde implementaties van modellen.
- *Modelleren en Simuleren* - Gekoppelde modelprogramma's moeten onderling gegevens kunnen uitwisselen. Deze gegevens kunnen complex van aard zijn, zoals bijvoorbeeld schematisaties. Om die reden is het goed een gemeenschappelijk beeld te hebben van hoe simulatieprogramma's rekenen. Uiteraard is het mogelijk (en waarschijnlijk) dat een modelprogramma een eigen representatie van gegevens heeft, maar voor uitwisseling zijn afspraken en standaarden nodig; hiervoor dient het domeinmodel als basis.
- *Omgeving voor Ontwikkeling* - Om nieuwe modellen te kunnen samenstellen of configureren is het belangrijk dat de gebruiker de beschikking heeft over de juiste bouwstenen (componenten). In het bijbehorende domeinmodel zijn de verschillende soorten bouwstenen aangestipt en deze zullen in de architectuur nader worden ingevuld.

3.2.1 Invalshoek 'Fysica in het Integraal Waterbeheer'

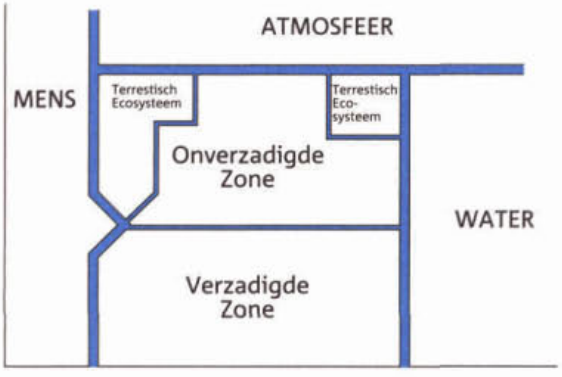
In de workshops is naar voren gekomen dat modelprogramma's in het Integraal Waterbeheer zeer divers zijn: er zijn programma's voor o.a. waterbeweging, waterkwaliteit, morfologie, ecologie, economie en gebruiksfuncties. Al deze modelprogramma's beschrijven een stuk van de (fysische) werkelijkheid. Daarom is als eerste belangrijke invalshoek de 'Fysica in het Integraal Waterbeheer' gekozen.

Vanuit de invalshoek 'Fysica in het Integraal Waterbeheer' neemt de kringloop van water en stoffen een centrale plaats in. In de kringloop vormen de milieucompartimenten bodem, het open water en de atmosfeer de basiselementen waartussen water wordt uitgewisseld. De bodem en het water wisselen middels neerslag en verdamping water uit met de atmosfeer. Onderling wisselen open water en bodem water en stoffen uit via de processen infiltratie en kwel. De mens laat zijn invloed alom gelden op de drie milieucompartimenten. Voorbeelden: via beregening wordt extra water aan de atmosfeer toegevoegd dat als neerslag op de bodem terecht komt; door ingrepen aan het bodemoppervlak wordt de infiltratie van neerslag in de bodem bemoeilijkt of gemakkelijker gemaakt (bv. verharden of scheuren van de toplaag van de bodem); via het onttrekken van water uit of het toevoeren van water naar het open water worden kwel- en infiltratieprocessen beïnvloed. Door bemesting worden stoffen toegevoegd.



Figuur 3-1
Basiselementen van de hydrologische kringloop

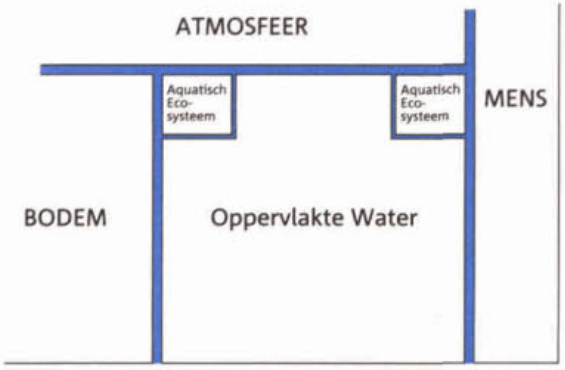
In figuur 3-1 zijn de milieucompartimenten en de mens weergegeven. Op de raakvlakken tussen de compartimenten en de mens vinden de genoemde processen plaats. Van de compartimenten mens en atmosfeer is in de workshops besloten ze niet verder uit te werken.
Voor de compartimenten land (= bodem) en water zijn wel verdere detailleringen gemaakt. Een onderverdeling van het compartiment land heeft geresulteerd in figuur 3-2.



Figuur 3-2
Detaillering van het compartiment land

Het compartiment is verdeeld in het verzadigde en onverzadigde deel.
Dit vanwege het fundamentele verschil in waterstroming dat optreedt. Naast de mens is vastgesteld dat ook het terrestrische ecosysteem zijn invloed op het compartiment bodem laat gelden en tegelijkertijd ook interacties heeft met de compartimenten atmosfeer en water. Ook de mens heeft weer invloed op het terrestrische ecosysteem.

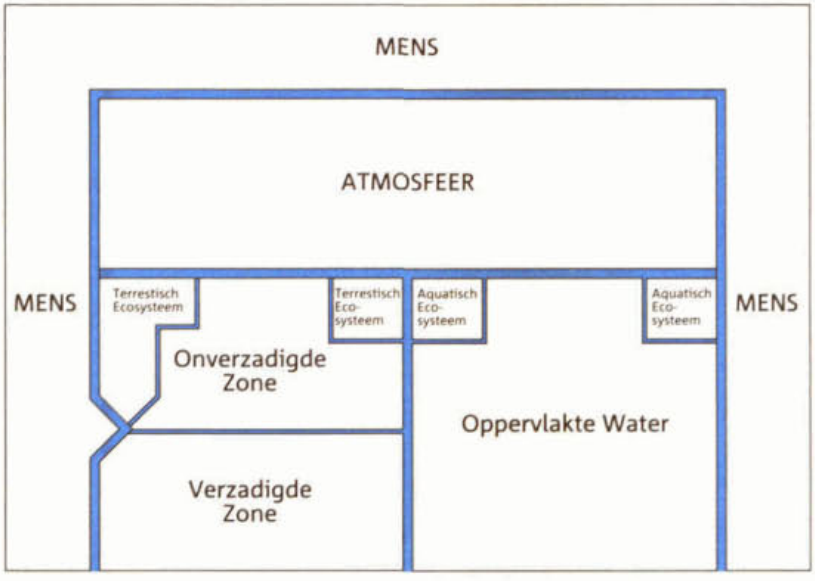
Voorbeelden van de interacties van het terrestrisch ecosysteem met de andere elementen: begroeiing vangt een deel van de neerslag op en laat een deel hiervan vertraagd doorvallen naar de bodem. Het andere deel verdampt. De mens heeft interactie door bijvoorbeeld begroeiing te verwijderen middels maaien. De begroeiing neemt grondwater uit de onverzadigde en verzadigde zone op en laat een deel hiervan verdampen naar de atmosfeer. Andere mogelijke interacties zijn: de mens kan via drainage de verzadigde zone beïnvloeden en via ingrepen in de bouwvoor de onverzadigde zone.



Figuur 3-3
Detaillering van het compartiment water

Uitwerking van het compartiment water heeft geleid tot figuur 3-3.
De detaillering bestaat uit het toevoegen van het aquatisch ecosysteem in het open water.

Door het samenvoegen van de detailleringen tot een geheel is figuur 3-4 ontstaan.



Figuur 3-4 Basiselementen van de hydrologische kringloop, na detaillering van de compartimenten land en water

Door middel van deze figuur is het fysische domein onderverdeeld in subdomeinen en zijn de raakvlakken daartussen gedefinieerd. Een verdere detaillering van de subdomeinen is nog mogelijk maar heeft voor de architectuur van het raamwerk geen toegevoegde waarde. Belangrijker is de constatering dat modelprogramma's gecategoriseerd kunnen worden daar aan te geven welk(e) subdomein(en) ze beslaan. Het toepassingsgebied van modelprogramma's die in het raamwerk worden opgenomen dient zich altijd te beperken tot (een deel van) een subdomein. Koppeling met andere modelprogramma's wordt dan mogelijk wanneer er geen overlap bestaat in de subdomeinen. Deze constatering wordt verder in hoofdstuk 4 uitgewerkt.

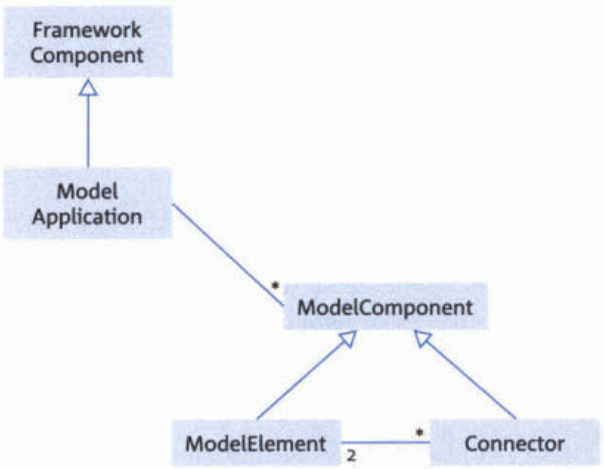
In de workshops is vastgesteld dat bepaalde specialistische modellen zich in de witte ruimte tussen de subdomeinen bevinden, zoals golfbewegingen als gevolg van wind-water-interactie, oevers en waterbodems.

3.2.2 Invalshoek 'Modelleren en Simuleren'

Modelprogramma's zijn op vele manieren te beschouwen. Naast een beschrijving van de fysica die een modelprogramma beschrijft, bekijken we in deze paragraaf de wiskundige oplossingsmechanismen. Het wiskundig oplossingsmechanisme heeft een sterke invloed op de manier waarop de werkelijkheid gemodelleerd wordt. Hieronder vallen tevens aspecten als de ruimtelijke en temporele schaal.

Vanuit de invalshoek 'Modelleren en Simuleren' neemt het wiskundig oplossingsmechanisme dat in waterbeheer modelprogramma's wordt toegepast een centrale plaats in. Het oplossingsmechanisme bepaalt in sterke mate de manier waarop de werkelijkheid wordt geschematiseerd. In de workshops zijn de volgende manieren van schematisatie aan de orde gekomen: eindige elementen methode, eindige differenties, analytische elementen en random-walk.

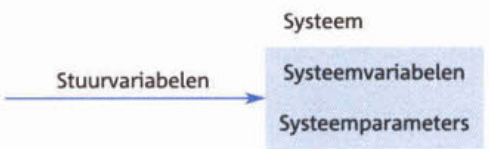
Binnen de architectuur van het SRW wordt uitgegaan van een conceptuele opbouw van modelapplicaties zoals die is vastgelegd in het uit het FIW afkomstige klassendiagram (zie figuur 3-5) [GROENENDIJK98].



Figuur 3-5
Klassendiagram van de opbouw van modelapplicaties uit modelcomponenten

De simulatie van de werkelijkheid vindt plaats in een modelapplicatie (ModelApplication), een computerprogramma waarin de berekeningen worden uitgevoerd. Het SRW beheert deze modelapplicaties, die een speciaal soort FrameworkComponent zijn. Modelelementen zijn bouwstenen die de gemodelleerde werkelijkheid beschrijven. Modellering van de fysica gebeurt door modelelementen met elkaar te verbinden. Een verbinding tussen twee modelelementen is een Connector.

De definitie van de modelelementen en de verbindingen ertussen (connectoren) is vastgelegd door gebruik te maken van de systeemtheorie: ieder systeem bevat systeemvariabelen (die de toestand van het systeem beschrijven) en systeemparameters (de onveranderlijken in het systeem). Door beïnvloeding door externe invloeden (de zogenaamde stuurvariabelen) verandert de toestand van het systeem (zie figuur 3-6). Aan modelelementen kunnen systeemvariabelen en stuurvariabelen worden toegekend. Een modelelement beschrijft dus een externe invloed op een systeem of de toestand van een systeem. Aan verbindingen kunnen systeemparameters worden toegekend. Deze beschrijven daarmee de onveranderlijken in het systeem.



Figuur 3-6
Schematische weergave van de relatie tussen systeemvariabelen, systeemparameters en stuurvariabelen

Modelelementen en connectoren kunnen worden samengesteld tot complexe eenheden waarmee een modelapplicatie kan worden gerepresenteerd.
Deze complexe eenheden zijn modelcomponenten genoemd.
De connectoren vormen hierbij ook de verbindingen tussen de verschillende modelcomponenten. In de connectoren moeten dan de verschillen in tijd- en ruimteschaal worden overbrugd.
Tijdens de workshops is nagegaan in hoeverre de bovengenoemde oplossingsmechanismen zijn te beschrijven met het conceptuele model.

Netwerken van 1D-lijnelementen
Dit oplossingsmechanisme is bekeken aan de hand van de modelapplicatie SOBEK. De modelelementen zijn hier terug te vinden als de rekenpunten in de schematisatie. De verbindingen tussen de rekenpunten zijn de connectoren.

2D- en 3D-roosters
Dit type rooster is niet bekeken aan de hand van een concrete modelapplicatie. Wat modelelementen en connectoren zijn, is afhankelijk van de gekozen oplosmethode en de locatie van de onbekenden ten opzichte van het rooster.

Eindige elementen
Dit oplossingsmechanisme is bekeken aan de hand van DUFLOW. De modelelementen zijn hier terug te vinden als de elementen in de schematisatie. De connectoren worden gevormd door de zijden van de elementen.

Analytische elementen
Dit oplossingsmechanisme is bekeken aan de hand van de modelapplicatie MLAEM. De modelelementen zijn hier terug te vinden als de punten waaraan een invloedsfunctie kan worden toegekend. De invloedsfunctie zelf vormt de connector.

Random walk
Dit oplossingsmechanisme is niet bekeken aan de hand van een modelapplicatie, maar vanuit de algemene theorie. De afhankelijke variabelen hierbij zijn positie (x, y, z) en de massa m. De vector van deze afhankelijke variabelen is te zien als een modelelement. Deze vector is ook te definiëren als het product van een onbekende vector maal een snelheidsvector, waarbij een ruisterm wordt opgeteld. Omdat de modelelementen geen interactie met elkaar hebben, zijn er geen connectoren.

Een ander belangrijk aspect van de invalshoek 'Modelleren en Simuleren' is dat van de algoritmes. Elk oplossingsmechanisme kan op verschillende manieren in software, oftewel in een algoritme, worden uitgedrukt. Algoritmes voor het oplossen van stelsels vergelijkingen bijvoorbeeld kunnen op diverse manieren worden opgesteld en kunnen nuttig zijn voor meerdere oplossingsmechanismen.

De algoritmes veranderen de toestand van het softwaresysteem en vaak ook het gemodelleerde systeem. Veranderingen in het softwaresysteem zijn 'traditioneel' merkbaar aan gewijzigde waarden van variabelen. Als een variabele een andere waarde krijgt, is echter nog niet per se een nieuwe toestand van de gemodelleerde werkelijkheid gerealiseerd, maar uiteindelijk zal deze nieuwe toestand bereikt worden door het gebruiken van één of meer algoritmes.

De toestand van het gemodelleerde systeem is beschreven door systeemvariabelen, die onderdeel zijn van de modelelementen. Algoritmes veranderen deze systeemvariabelen en daarmee die elementen.

3.2.3 Invalshoek 'Omgeving voor ontwikkeling'

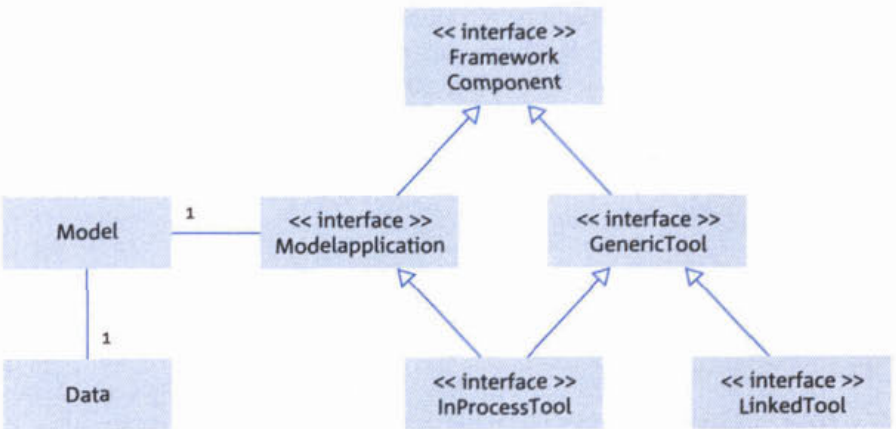
Het SRW is bedoeld om het koppelen van modelprogramma's en tools te faciliteren. Het maken van een nieuwe configuratie van modellen kan worden gezien als het ontwikkelen van een nieuw model. Daarnaast is het mogelijk dat een nieuw modelprogramma ontwikkeld moet worden om te voorzien in ontbrekende functionaliteit.

Vanuit de invalshoek 'Omgeving voor ontwikkeling' nemen de componenten waarmee een model geconfigureerd, uitgevoerd en geanalyseerd kan worden, een centrale plaats in. Binnen het SRW wordt onderscheid gemaakt tussen twee soorten componenten: modelprogramma's (rekenkernen plus de vereiste gegevens) en generieke tools.

In figuur 3-7 zijn de verschillende soorten componenten als klassendiagram afgebeeld. Aan de basis van alle componenten staat de FrameworkComponent, met andere woorden: alle componenten die in het SRW een rol spelen, zijn afgeleid van FrameworkComponent.
De componenten worden globaal, zoals hierboven reeds gemeld, onderverdeeld in twee groepen: modelprogramma's (interface ModelApplication) en tools (interface GenericTool).

Tools kunnen enerzijds hulpmiddelen zijn die in het simulatieproces direct een rol spelen (in-process tools). Anderzijds kunnen het hulpmiddelen zijn die onafhankelijk van het simulatieproces kunnen 'inhaken' op onderdelen van het simulatieproces (linked tools). Een voorbeeld van een 'in-process tool' is een tijdsynchronisatie-tool. Een voorbeeld van een 'linked tool' is een presentatie-tool.

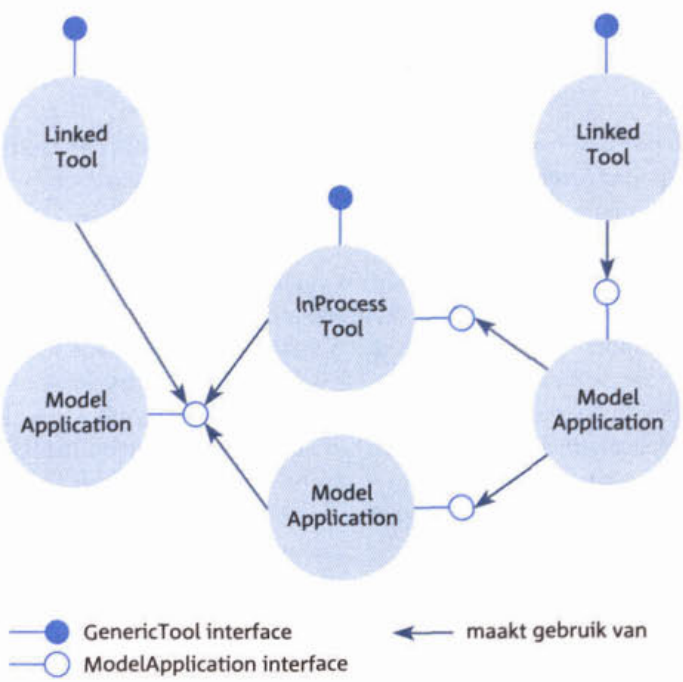
Een in-process tool verwerkt in de regel resultaten van het ene model en maakt die geschikt voor invoer in het andere. Dit laatste model heeft er echter geen benul van dat het een tool is - het tool 'gedraagt' zich als model. In het klassendiagram (fig. 3-7) is dat weergegeven door de interface InProcessTool, die een uitbreiding is van zowel de GenericTool-interface als ook de ModelApplication-interface.



Figuur 3-7
Klassendiagram van componenten van het SRW

Een linked tool wordt aan de keten gekoppeld ('gelinkt'), maar heeft geen invloed op de berekende resultaten. Een presentatie-tool kan, bijvoorbeeld, gekoppeld zijn aan een modelprogramma om de rekenresultaten te tonen; het tool oefent geen invloed uit op de berekeningen.

In figuur 3-8 is schematisch een voorbeeld weergegeven waarin de rollen van componenten en hun interfaces zijn afgebeeld. Componenten worden getoond als cirkel, interfaces als bolletje.



Figuur 3-8
Schematisch voorbeeld van de rollen van componenten in het SRW. De verschillende componenten maken gebruik van het interface van andere componenten (en dus niet van het component zelf)

In het Programma van Eisen [PvE98] zijn de volgende generieke tools reeds gedefinieerd: procesketenbeheerstool, data-editor en -managementtool, selectie- en definitietool, calibratietool, analysetool en presentatietool.

De belangrijkste taak van een tool voor procesketenbeheersing is het vastleggen van de volgordelijkheid waarin verschillende modelprogramma's uitgevoerd moeten worden en de uitwisseling van gegevens tussen die programma's. Hier wordt dus gedefinieerd welke modelprogramma's met elkaar moeten gaan samenwerken en in welke volgorde ze elkaar aanroepen. Hiertoe moet het tool controleren of het wel toegestaan is dat de applicaties samenwerken en, zo ja, de feitelijke koppeling tussen de applicaties tot stand brengen.

De belangrijkste taken van een data-editor en -managementtool zijn het wijzigen en vastleggen van gegevens voor modelprogramma's.

De belangrijkste taak in een selectie- en definitietool is het vastleggen en bewaren van een bepaalde consistente set van invoer die in het data-editor en -managementtool zijn ingegeven.

De taak van de analysetool is het bewerken van de uitvoergegevens van modelprogramma's. De calibratietool, bedoeld voor het calibreren/afregelen van de modellen, kan beschouwd worden als een specifiek soort analysetool.

De taak van het presentatietool is het visualiseren van gegevens.

3.3 Van domeinverkenning naar architectuur

In de voorgaande paragrafen is het probleemdomein 'simulatiemodellen in integraal waterbeheer' vanuit verschillende invalshoeken bekeken. De domeinverkenning is uitgevoerd door domeindeskundigen (onderzoekers op het gebied van waterbeheer, ecologie, waterkwaliteit, etc.) en IT-deskundigen. De bovenstaande domeinmodellen geven dus een beeld van het complete probleemdomein, waarbij volledigheid belangrijker is dan een grote mate van detail[6].

Zoals in de aanpak betoogd is, dienen de domeinmodellen vanuit de verschillende invalshoeken ter ondersteuning van een generieke beschrijving van het domein. Deze generieke beschrijving vindt uiteindelijk zijn weerslag in de architectuur van het SRW.

De opslitsing in de fysica van het water heeft geleid tot het inzicht dat modelapplicaties zich nadrukkelijk moeten uitspreken over welke subdomeinen ze verantwoordelijk zijn. De informatie van zo'n subdomein kan dan aan die modelapplicatie gevraagd worden. Het gaat er hierbij niet om dat de hier beschreven opslitsing volledig en de enige ware is, maar dat een dergelijke opdeling de basis vormt waarop modellen onderling afgestemd kunnen worden. Een belangrijk aspect van de architectuur is de scheiding van verantwoordelijkheden. In dit geval betekent dit ook duidelijkheid in verantwoordelijkheden. Binnen een keten zou het niet toegestaan kunnen worden dat twee applicaties zich voor hetzelfde verantwoordelijk achtten. Dit zou tot conflicten leiden.

---

[6] En gezien de diversiteit van de deelnemers is overeenstemming over die volledigheid veel eenvoudiger te bereiken dan over de details.

Bij de invalshoek 'modelleren en simuleren' is eveneens een scheiding van verantwoordelijkheden ontstaan. Deze scheiding uit zich in de schematisatie. De schematisatie waarmee een modelapplicatie de werkelijkheid benadert bestaat uit modelelementen en connectoren. De modelelementen zijn verantwoordelijk voor (een stukje van) de fysica. De connectoren verbinden modelelementen. Een modelelement hoeft dus geen kennis te hebben met welke andere elementen die verbonden is. Een connector is daarvoor verantwoordelijk. Een connector hoeft op zijn beurt zich niet druk te maken over het bijhouden van de status van de modelelementen.

Bij de invalshoek 'omgeving voor ontwikkeling' tenslotte wordt een scheiding tussen modelapplicaties enerzijds (zeg maar rekenkernen) en generieke tools anderzijds gemaakt. Belangrijk aspect hierbij is dat zowel modelapplicaties als generieke tools bouwstenen zijn binnen het SRW.

4. Architectuur van het SRW

4.1 Inleiding

De architectuur van een softwaresysteem beschrijft dat systeem in abstractie, uitgedrukt in de onderdelen en de samenhang ertussen. Het belangrijkste doel van een architectuur is het (zoveel mogelijk technologie-onafhankelijk) vroegtijdig vastleggen van ontwerpbeslissingen. De abstracte representatie van de software maakt het mogelijk later verschillende technische invullingen daarvoor te geven. De kennis uit het domein 'Simulatiemodellen in het Integraal Waterbeheer' zoals die in het vorige hoofdstuk is beschreven, wordt in dit hoofdstuk uitgewerkt tot een architectuur.

De architectuur van het SRW wordt hieronder vanuit drie verschillende gezichtspunten (of 'views') bekeken, die onderling complementair zijn. Deze drie zijn: de statische structuur, de dynamiek en de (technologische) infrastructuur.

De eerste invalshoek is de statische structuur van het SRW (zie paragraaf 4.2). Dit omvat de componenten (bouwstenen) waaruit het raamwerk (inclusief de modelprogramma's en de tools) is opgebouwd. De aandacht gaat hierbij uit naar de structuur van deze componenten, waaronder de diensten die de componenten leveren. De keuze van componenten en hun structuur is direct afgeleid uit de domeinanalyse (hoofdstuk 3).

Het tweede gezichtspunt is de dynamiek van het systeem (zie paragraaf 4.3). Het gaat hier met name om de werking van het SRW, hoe de componenten onderling samenwerken. De dynamiek heeft betrekking op het 'runtime-gedrag' van de componenten en omvat dus ook bijvoorbeeld concurrency en parallellisme.

De derde en laatste invalshoek beschrijft de technologische infrastructuur (paragraaf 4.4) die nodig is voor het realiseren van het SRW. Hierbij worden voor- en nadelen opgesomd, maar een keuze voor een bepaalde technologie wordt in dit rapport niet gemaakt.

Bij de uitwerking van de invalshoeken is gebruik gemaakt van UML (Unified Modelling Language: de de facto standaard voor het noteren van objectgeoriënteerde ontwerpen) en de Interface Definition Language (IDL) zoals die door de Object Management Group (OMG) is gedefinieerd. Bij de beschrijvingen van de verschillende views wordt ervan uitgegaan dat de lezer voldoende kennis heeft van objectoriëntatie, UML en OMG-IDL.

Omdat gegevensopslag en -uitwisseling een belangrijke rol speelt bij het bouwen en koppelen van simulatiemodellen wordt daaraan in paragraaf 4.5 aandacht besteed.

In dit hoofdstuk worden veel termen uit het IT vakdomein gehanteerd. Voor de onderdelen van het SRW worden engelse termen gebruikt. In de tekst wordt zoveel mogelijk deze engelse termen gebruikt waar verwezen wordt naar een concreet stukje architectuur ontwerp. Echter omwille van de zinsloop en leesbaarheid wordt hier in sommige gevallen een consessie aan gedaan.

4.2 Statische structuur van componenten

4.2.1 Onderscheiden componenten

Er bestaat een groot aantal definities voor de term component. Binnen dit project wordt onder een component verstaan:
Een samenhangend 'pakket' van herbruikbare software dat onafhankelijk als eenheid kan worden ontwikkeld en geleverd en dat ongewijzigd kan worden samengevoegd met andere componenten om een groter geheel te maken.

Er zijn in het vorige hoofdstuk verschillende soorten componenten onderscheiden binnen het SRW:

- *Framework:* verantwoordelijk voor het beheer van alle (geregistreerde) FrameworkComponents;
- *FrameworkComponent:* verantwoordelijk voor het bij elkaar houden van ModelApplications en GenericTools;
- *GenericTool:* verantwoordelijk voor het uitvoeren van een generieke taak;
- *ModelApplication:* verantwoordelijk voor het aggregeren van ModelComponents;
- *ModelComponent:* verantwoordelijk voor het bundelen van de gemeenschappelijkheden van modelelementen en connectors;
- *ModelElement:* verantwoordelijk voor het beschikbaar stellen van waarden van attributen van een stuk(je) gemodelleerde fysica;
- *Connector:* verantwoordelijk voor het verbinden van modelElements;
- *Model:* verantwoordelijk voor het verbinden van ModelApplication met Data;
- *Data:* verantwoordelijk voor het opslaan van de statische gegevens van een ModelApplication.

De relaties tussen deze componenten is zichtbaar in figuur 3-5 en figuur 3-7. In de volgende paragrafen wordt vanaf het hoogste niveau (Framework en FrameworkComponent) naar de lagere niveaus toegewerkt.

4.2.2 Framework en FrameworkComponent

Interfaces
De diensten die de componenten Framework en FrameworkComponent beschikbaar stellen worden in deze paragraaf beschreven. Omdat de componenten GenericTool en ModelApplication specialisaties zijn van FrameworkComponent is van deze componenten eveneens de basale interface vastgelegd.

Het SRW kan gezien worden als een 'container' voor de componenten Tool en ModelApplication. Componenten (modelapplicaties en tools) die in het SRW gebruikt moeten worden (raamwerk-componenten) moeten zich aanmelden bij het SRW. Op deze manier wordt aan het SRW bekend gemaakt welke Tools en ModelApplications aanwezig zijn.

Het SRW biedt de operatie *register (FrameworkComponent c).* Door deze methode aan te roepen wordt de aanmelding van een raamwerkcomponent geïnitieerd. Het SRW zal vervolgens controleren of dit ook daadwerkelijk mogelijk is (Is de raamwerkcomponent reeds aanwezig? Zo ja, is dit een oudere of nieuwere versie?) met behulp van de operatie *check(FrameworkComponent c).* Voor het goedkeuren van raamwerkcomponenten zijn verschillende beslissingscriteria en -strategieën denkbaar.

Alle raamwerkcomponenten zullen deze operaties gebruiken. Derhalve is het interface Framework gedefinieerd, die al de operaties rondom het registreren van raamwerk-componenten bevat. De beschrijving van dit interface is in onderstaand kader uitgewerkt.

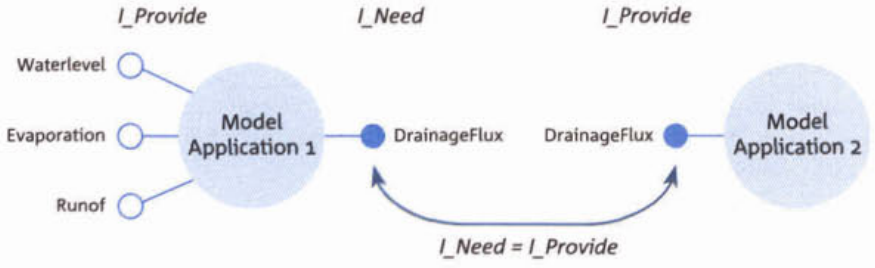
```
interface Framework {
    RegisterResult register(in FrameworkComponent c);
        Via deze methode kan een raamwerkcomponent zich aanmelden bij het Standaard Raamwerk.
        Deze methode zal worden geïnitieerd als een gebruiker van het Standaard Raamwerk een model
        of generieke tool wil toevoegen aan het raamwerk. Indien de registratie, na controle van de
        component, daadwerkelijk wordt uitgevoerd wordt 'rrOK' als resultaat gegeven.
    RegisterResult check(in FrameworkComponent c);
        Deze methode controleert of de raamwerkcomponent toegevoegd kan worden aan het Standaard
        Raamwerk. Het Standaard Raamwerk controleert hierbij of de component niet reeds eerder is
        geregistreerd. Hiervoor vergelijkt het raamwerk de naam, het type en het versienummer van de
        component met alle aanwezige componenten in het raamwerk. Indien bij vergelijking met een
        component blijkt dat alleen het versienummer verschillend is zal de gebruiker moeten beslissen
        of de aanwezige component geregistreerd mag worden.
    RegisterResult unregister(in FrameworkComponent c);
        De methode unregister verwijdert de component c uit de lijst van geregistreerde componenten.
        Als dat met succes is uitgevoerd, geeft de operatie 'rrOK' als resultaat, anders een code voor de
        opgetreden fout.
    RegisterResult unregisterAll();
        Deze operatie verwijdert alle geregistreerde componenten uit het raamwerk. Het resultaat van de
        operatie geeft het succes aan.
    sequence<FrameworkComponent> getRegisteredComponents();
        De methode getRegisteredComponents levert een opsomming van alle geregistreerde
        componenten op.
    boolean instantiate(in FrameworkComponent c);
        Deze operatie instantieert een exemplaar van FrameworkComponent c in de 'huidige'
        configuratie. Het instantiëren wordt verderop preciezer beschreven.
};

enum RegisterResult {
    rrOK,
    rrComponentNotAllowed,
    rrComponentAlreadyRegistered,
    ...
};
```

De operaties in het interface Framework zullen gegevens uit de raamwerkcomponenten willen hebben om de registratie te kunnen uitvoeren. Voor het aanvragen van deze gegevens moeten operaties in de raamwerkcomponenten beschikbaar zijn. Om ervoor te zorgen dat alle raamwerkcomponenten dit op een uniforme manier doen is het interface FramworkComponent gedefinieerd. Dit interface bestaat uit de volgende methoden:

```
interface FrameworkComponent {
    Name getName();
        Levert de naam van de component.
    ComponentType getType();
        Resulteert in het type waartoe de component behoort. Dit levert dus de informatie of een
        component een model of een tool is (en eventueel wat voor model of tool).
    VersionInfo getVersion();
        Geeft het versienummer van de component.
    sequence<ServiceDescriptor> getProvidedServices();
        Via deze methode kan opgevraagd worden welke diensten deze component levert.
    sequence<ServiceDescriptor> getRequiredServices();
        Via deze methode kan opgevraagd worden welke diensten deze component nodig heeft van
        andere componenten om zijn eigen diensten uit te kunnen voeren.
    ConnectResult connect(in FrameworkComponent c, ServiceDescriptor sd);
        Deze methode wordt aangeroepen indien de gebruiker (of het raamwerk) twee componenten wil
        koppelen, waarbij sd de dienst is die de aangeroepen component vraagt en component c zou
        moeten leveren. De component c wordt vervolgens geraadpleegd naar de gevraagde en te
        leveren diensten. Indien daadwerkelijk koppeling plaatsvindt van componenten wordt 'crOK' als
        resultaat gegeven.
    sequence<ServiceDescriptor> getUsedServices(FrameworkComponent c);
        Het resultaat van het aanroepen van deze operatie is een lijst van alle diensten die component c
        aan de aangeroepen component biedt.
    sequence<FrameworkComponent> getConnectedComponents();
        Deze operatie levert alle componenten die verbonden zijn met de aangeroepen component. De
        connecties zijn met behulp van de operatie connect tot stand gebracht.
    boolean areRequiredServicesComplete();
        Voordat er daadwerkelijk een simulatie gedraaid kan worden met het Standaard Raamwerk met
        een bepaalde configuratie van modelapplicaties moet gecontroleerd kunnen worden of de
        gecreëerde configuratie compleet is (d.w.z. of alle vereiste diensten door componenten worden
        ingevuld). Binnen deze methode worden alle vereiste externe diensten voor een component
        afgelopen en gecontroleerd of deze diensten daadwerkelijk door een ander component voldaan
        worden.
};

typedef string Name;
enum ComponentType {
    ctModelApplication,
    ctInProcessTool,
    ctLinkedTool
};

struct VersionInfo {
    double number;
};

enum ConnectResult {
    crOK,
    crConnectionNotAllowed
    ...
};
```

```
interface ServiceDescriptor {
    string getName();
        Deze operatie levert de naam van de dienst. In dit document zijn geen concrete richtlijnen
        opgenomen voor naamgeving. Het is in elk geval belangrijk dat aan de hand van de naam twee
        diensten met elkaar vergeleken moeten kunnen worden. Voor zover mogelijk wordt voor de
        namen gebruik gemaakt van de Adventuswoordenlijst.
    ServiceType getType();
        Deze operatie geeft het type van de dienst terug. Momenteel zijn twee soorten diensten
        onderscheiden: stData en stEvent. Een stData-dienst kan gebruikt worden om gegevens op te
        vragen; om een grondwaterflux uit een modelapplicatie te krijgen heeft een
        oppervlaktewatermodel een dergelijke dienst nodig. Als een dienst van het type stEvent is, kan de
        component bepaalde events genereren. Een presentatie-tool die 'live' resultaten toont van een
        berekening heeft een modelapplicatie nodig die events kan genereren wanneer nieuwe
        (tussen)resultaten beschikbaar zijn.
    ModelComponent getModelComponent();
        Deze operatie geeft het ModelComponent-object terug dat geassocieerd is met de dienst. In de
        volgende paragraaf worden modelcomponenten verder behandeld.
    Attribute getAttribute();
        Deze operatie geeft het Attribute-object terug dat geassocieerd is met de dienst.
};
enum ServiceType {
    stData,
    stEvent
};
```

Naast registratie van een component binnen het SRW moeten componenten onderling gekoppeld kunnen worden om een configuratie van modelapplicaties en tools samen te stellen. Het koppelen van twee componenten komt neer op het vergelijken van gevraagde en te leveren diensten van deze twee componenten. Indien bijvoorbeeld een modelapplicatie voor het oppervlaktewater alleen kan functioneren als de drainageflux wordt aangeleverd, zal koppeling plaats moeten vinden met een component die deze drainageflux kan leveren. In figuur 4-1 is de vergelijking van gevraagde en te leveren diensten schematisch weergegeven.



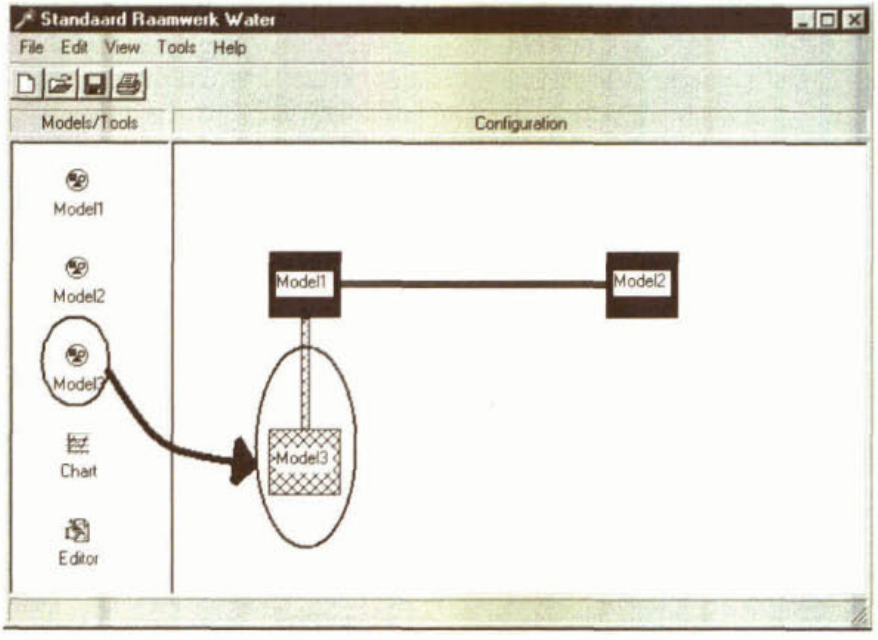
Figuur 4-1 *Schematische weergave van de koppeling van twee ModelApplications*

Met behulp van de operatie *getRequiredServices()* kan aan een component worden gevraagd welke diensten hij nodig heeft om te kunnen werken. Het resultaat van deze operatie is een lijst van ServiceDescriptor-objecten. Een ServiceDescriptor beschrijft de dienst die wordt geboden of gevraagd. Momenteel worden diensten onderling vergeleken op basis van hun naam en hun type (ServiceType); het is niet uitgesloten dat deze criteria in de toekomst nog worden uitgebreid wanneer preciezer bekend is wat een dienst omschrijft. Voor de naamgeving van de diensten zal de terminologie zoals die in het Adventusstelsel is gedefinieerd, worden gebruikt.

De operatie *getProvidedServices()* geeft een lijst van aangeboden diensten terug. Dit is ook een verzameling ServiceDescriptor-objecten.

Configureren en instantiëren van componenten

In de praktijk zou het configureren van modellen in het SRW er kunnen uitzien zoals in figuur 4-2. Een palet met alle opgenomen modellen en tools kan gebruikt worden voor het slepen van de afzonderlijke modellen en tools op het canvas van de configuratie.



Figuur 4-2
Voorbeeld van een ConfigurationEditor met Models/Tools-Palet

Figuur 4-2 illustreert de situatie waarbij de gebruiker Model3 wil koppelen aan Model1 (Model1 is reeds eerder gekoppeld aan Model2). Na deze aanvulling in de modelketen zal gecontroleerd worden of Model3 een of meer diensten nodig heeft die door Model1 geleverd kunnen worden (of andersom). Wanneer deze invulling van een of meer diensten door het nieuwe model mogelijk is, zal de uiteindelijke koppeling tot stand gebracht worden. Het SRW zal de gebruiker ondersteunen bij het maken van de koppelingen. Als dit niet automatisch kan, zal de applicatie aan de gebruiker een keuze moeten voorleggen.

Het controleren of een bepaalde configuratie met modelapplicaties en generieke tools compleet is voor het draaien van simulaties komt neer op het aflopen van alle opgenomen componenten waarbij gecontroleerd wordt of alle vereiste externe diensten van een component worden vervuld door een gekoppelde component binnen de configuratie. Hiervoor biedt een component de methode *areRequiredServicesComplete()*.

Als de gebruiker een component uit het palet naar de configuratie sleept, wordt een nieuw exemplaar van de gekozen component geïnstantieerd. Er is dus een duidelijk verschil tussen de componenten in het palet en in de configuratie. Elk component in het palet is uniek, maar kan meer malen geïnstantieerd zijn in de configuratie; een component in het palet is te vergelijken met een blauwdruk. Een tool die bijvoorbeeld bedoeld is om schalen tussen twee modellen op elkaar af te stemmen, kan meer dan één keer in de configuratie voorkomen.

Het maken van koppelingen tussen modellen is niet altijd eenvoudig. In het bovenstaande voorbeeld wordt een (simpel) principe beschreven, maar de details zullen het in de praktijk

moeilijk maken. Verschillende zaken moeten bij de koppeling aan bod komen, zoals verschaling van ruimte en tijd en het koppelen van modellen aan specifieke modelelementen (bijvoorbeeld voor het aangeven van laterale instroming).

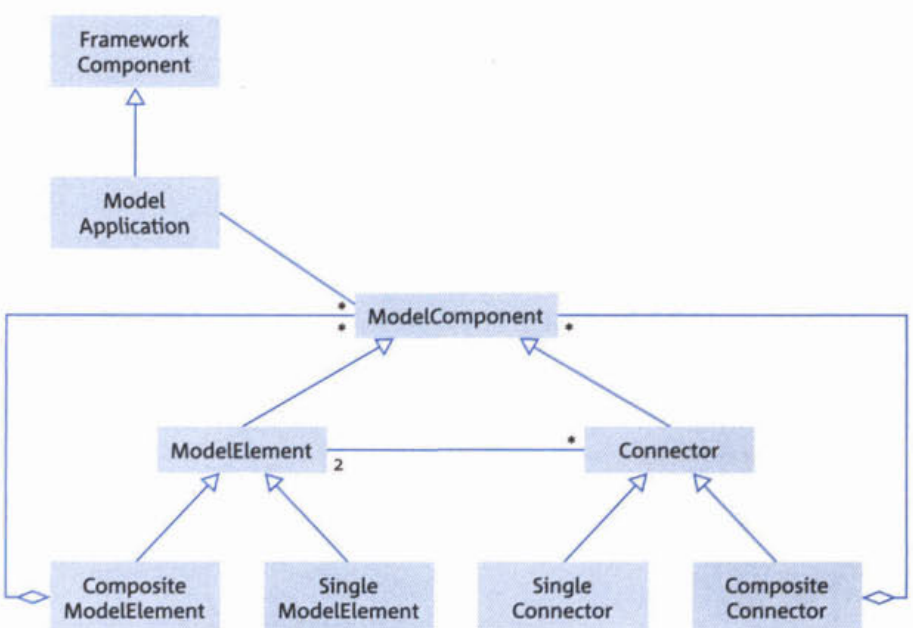
4.2.3 ModelApplication

Bij de beschrijving van modelapplicaties binnen de architectuur van het SRW spelen composities een belangrijke rol. Binnen modelapplicaties wordt gebruik gemaakt van een schematisatie waarbij 'bestaat uit'-relaties veelvuldig voor kunnen komen. Zo zal gemodelleerd moeten kunnen worden dat een bodemlaag bestaat uit een aantal horizonten. Voor deze composities kan gebruik gemaakt worden van het Composite Design Pattern [GAMMA95].

Figuur 4-3 Klassendiagram ter illustratie van het gebruik van het Composite Pattern voor het beschrijven van bodemlagen, naar analogie van een file-systeem

In figuur 4-3 zijn twee voorbeelden gegeven waarin het Composite Pattern gebruikt kan worden. Het belangrijkste van deze structuur is dat de 'kleine' elementen (in de voorbeelden de files of de horizonten) en de 'containers' (in de voorbeelden de directories of de soillayers) beide subklassen zijn van dezelfde superklasse (de node of het soilelement), waardoor zowel de 'kleine' elementen als de 'containers' een zelfde interface bieden. Het voordeel hiervan is dat de gebruikers (andere objecten) van zo'n object niet hoeven te weten of ze te maken hebben met een samengesteld danwel een enkelvoudig object.

Een modelapplicatie wordt gezien als een component die bestaat uit een model en de vereiste gegevens (vaste, variabele en meetgegevens). Deze opbouw dient vooral als een fysieke opbouw gezien te worden. Bij het beschrijven van een architectuur voor modelapplicaties speelt echter de opbouw op conceptueel niveau een belangrijkere rol. Binnen de architectuur van het SRW wordt uitgegaan van het volgende conceptuele opbouw voor modelapplicaties:



Figuur 4-4 Klassendiagram van de conceptuele opbouw van modelapplicaties

Een modelapplicatie kan dus gezien worden als een verzameling modelcomponenten. Modelcomponenten representeren een onderdeel van het domein waarbinnen de modelapplicatie gebruikt wordt. Modelcomponenten representeren dus een onderdeel van de gesimuleerde werkelijkheid. Een modelapplicatie bevat tevens het rekenhart.

Binnen het SRW moeten bestaande modelapplicaties kunnen worden opgenomen. Uiteraard zijn deze modelapplicaties niet opgebouwd uit de hierboven beschreven bouwstenen (modelcomponenten). Dit betekent dat het migreren van bestaande modelapplicaties dus neerkomt op het maken van een schil (ook wel 'wrapper' genaamd) rond de bestaande modelapplicatie, zodat het lijkt alsof de modelapplicatie bestaat uit modelelementen en connectoren. De wrapper zorgt voor de vertaalslag van de bestaande modelapplicatiestructuur naar de hierboven beschreven structuur.

Interface ModelApplication

De interface ModelApplication is een uitbreiding van de interface FrameworkComponent en is hieronder in detail beschreven:

```
interface ModelApplication: FrameworkComponent {
    string getAdministrator();
        Deze methode geeft de naam van de beheerder van de modelapplicatie als resultaat.
    UseType getUseType();
        Via deze methode kan informatie verkregen worden over de gebruiksrol van de modelapplicatie
        binnen de huidige configuratie. De gebruikersrollen kunnen bijvoorbeeld verdeeld worden in
        sequentieel en dynamisch.
    ScaleInfo getScaleInfo();
        Het resultaat van deze methode is een structure waarin informatie is opgenomen over de
        minimale en maximale ruimteschalen, horizontaal en verticaal.
    Time getMinTimestep();
        De minimale tijdstapgrootte in seconden waarin de modelapplicatie kan rekenen levert deze
        methode als resultaat.
    Time getMaxTimestep();
        Analoog aan getMinTimestep() levert deze methode de maximale tijdstapgrootte.
    Time getOutputTimestep();
        Deze methode geeft de grootte van de tijdstap (in seconden) aan die gebruikt wordt bij het
        genereren van uitvoer van modelresultaten.
    ModelComponent whichElement(in Geometry g, in double z);
        Voor de opgegeven locatie (zie Geometrie) kan de overeenkomstige ModelComponent
        opgevraagd worden met deze methode.
    boolean hasBatchMode();
        Boolean-operatie geeft 'true' als resultaat indien de modelapplicatie een batch-mode bezit.
    void enableBatchMode();
        Deze methode schakelt de batch-mode in.
    void disableBatchMode();
        Deze methode schakelt de batch-mode uit.
};
typedef long Time;
enum UseType {
    utSequential,
    utDynamic
};
```

```
struct ScaleInfo {
    double minimumHorizontalScale;
    double maximumHorizontalScale;
    double minimumVerticalScale;
    double maximumVerticalScale;
};
```

Een modelapplicatie bestaat (conceptueel) uit modelcomponenten. In deze modelcomponenten vindt het feitelijke rekenen plaats.

Geometrie

De modelcomponenten wisselen onderling gegevens uit. Een deel van deze gegevens zijn ruimtelijke gegevens. Zoals in hoofdstuk 2 al is genoemd zal hierbij zo veel mogelijk gebruik gemaakt worden van OpenGIS-standaarden. Deze paragraaf geeft een summiere beschrijving van OpenGIS onderwerp 5 (the OpenGIS feature) en de voor dit project relevante interfaces en structuren. Er zijn implementatie specificaties beschikbaar voor dit onderwerp, welke betrekking hebben op zogenaamde Simple Features (twee-dimensionale punten, lijnen en vlakken, of verzamelingen van deze elementen).

Het OGC (OpenGIS Consortium) heeft een essentieel model vervaardigd van de ruimtelijke wereld. Volledige behandeling van dit model valt buiten het kader van dit project, zie hiervoor [http://www.opengis.org]. Basiseenheid van het model is de feature (Helaas is hiervoor nog geen gangbare Nederlandse term. Letterlijk vertaald is het kenmerk; in deze tekst wordt ook wel ruimtelijk object gehanteerd). Een feature bestaat uit attributen, waarvan één of meer attributen geometrische attributen zijn. Attributen worden ook verder in deze tekst beschreven, daar worden echter modelattributen bedoeld. Beide definities conflicteren niet, dat wil zeggen een modelattribuut kan een uitgebreid OpenGIS-attribuut zijn.

Geometrie kan binnen de OpenGIS-specificatie op een aantal manieren worden uitgewisseld:

- als Feature interface. Een Feature is een verzameling attributen, waarvan er één of meer geometrische attributen zijn.
- als Geometry Interface
- als zogenaamde Well-known Structure (WKS), een structuur
- als Well-known Binary (WKB), een reeks bytes
- als Well-known Text (WKT), een tekst waarin coördinaten zijn opgenomen, b.v. "POINT(1,2)"
- geometrie zal ook in tekstvorm als standaard XML (eXtended Markup Language) uitgewisseld kunnen worden (dit is op dit moment nog niet gestandaardiseerd).

Voor het SRW lijkt een interface niet de meest voor de hand liggende keuze. Ten eerste is voor modelelementen zelf al een interface gedefinieerd waardoor als we hier ook voor interfaces kiezen objecten onnodig extra met elkaar moeten samenwerken (wel zouden bepaalde interfaces van het interface Feature afgeleid kunnen zijn). Ten tweede gaat het vaak om de uitwisseling van locaties en niet zozeer om overige kenmerken. Als we moeten kiezen tussen structuren, bytes en tekst is de structuur (Well-known structure) een goede keuze. Binaries moeten door client en server uiteengerafeld worden, iets dat tijd kost en fouten met zich mee kan brengen (het is minder typesafe). Tekst moet ontleed worden door een parser, dit gaat nog meer ten koste van de performance.

Uitwisseling van geometrie door middel van Well-known structures lijkt dus de beste keuze. Hieronder wordt in IDL de definitie van een geometrie volgens de OpenGISspecificaties gegeven [OGC98, http://opengis.org/public/sfr1/sfcorba_rev_1_0.pdf].

```
struct WKSPoint {
    double x;
    double y;
};

typedef sequence<WKSPoint> WKSPointSeq;
typedef sequence<WKSPoint> WKSLineString;
typedef sequence<WKSLineString> WKSLineStringSeq;
typedef sequence<WKSPoint> WKSLinearRing;
typedef sequence<WKSLinearRing> WKSLinearRingSeq;
struct WKSLinearPolygon {
    WKSLinearRing externalBoundary;
    WKSLinearRingSeq internalBoundaries;
};
typedef sequence <WKSLinearPolygon> WKSLinearPolygonSeq;

struct Envelope {
    WKSPoint minm;
    WKSPoint maxm;
};

enum WKSType {
    WKSPointType, WKSMultiPointType, WKSLineStringType, WKSMultiLineStringType,
    WKSLinearRingType, WKSLinearPolygonType, WKSMultiLinearPolygonType,
    WKSCollectionType
};

union Geometry
switch (WKSType) {
    case WKSPointType:
        WKSPoint point;
    case WKSMultiPointType:
        WKSPointSeq multi_point;
    case WKSLineStringType:
        WKSLineString line_string;
    case WKSMultiLineStringType:
        WKSLineStringSeq multi_line_string;
    case WKSLinearRingType:
        WKSLinearRing linear_ring;
    case WKSLinearPolygonType:
        WKSLinearPolygon linear_polygon;
    case WKSMultiLinearPolygonType:
        WKSLinearPolygonSeq multi_linear_polygon;
    case WKSCollectionType:
        sequence<WKSGeometry> collection;
};
```

In onderstaande interfaces zullen we locaties dus als de structuur Geometry uitwisselen. Weliswaar hebben we hierin geen ruimte voor de derde dimensie omdat het om Simple Features ging, maar hiervoor kan later een nieuwe (nu nog niet bestaande) specificatie genomen worden. Wellicht is er in zulke toekomstige specificaties ook ruimte voor de temporele dimensie.

4.2.4 ModelComponent

Interface

De onderdelen waaruit de modelapplicatie bestaat, de modelcomponenten, bevatten een aantal attributen. Deze attributen zijn via de interface van ModelComponent te verkrijgen:

```
interface ModelComponent {
    sequence<Attribute> getAttributes();
        Deze operatie levert een lijst van attributen op die bij de modelcomponent horen.
};
```

Een attribuut is een voor het model relevante eigenschap (bijvoorbeeld waterstand, drainageflux of concentratie). De waarden van attributen worden in de regel berekend of uit een bestand gehaald.

Een samengesteld ModelElement (of Connector) zal binnen de implementatie van de aangeboden operaties gebruik maken van de kleinere elementen waaruit het is opgebouwd, de omgeving (de omliggende modelelementen) van het samengestelde element hoeft van deze interne opbouw geen kennis te hebben.

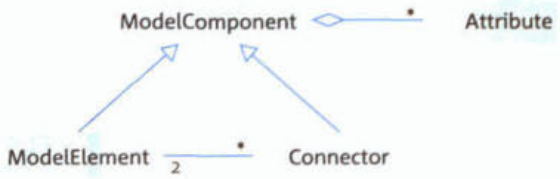
De klassen CompositeModelElement en CompositeConnector bieden voor de opgenomen aggregatie de volgende methoden:

```
interface CompositeModelElement/CompositeConnector: ModelElement/Connector {
    void addModelComponent(in ModelComponent mc);
        Met behulp van deze methode kunnen aan samengestelde modelelementen en connectoren
        (kleinere) elementen worden toegevoegd.
    boolean canAddModelComponent(in ModelComponent mc);
        Deze operatie zal door addModelComponent(ModelComponent mc) worden gebruikt om te
        controleren of het element mc daadwerkelijk mag worden toegevoegd aan de verzameling. Deze
        methode geeft 'true' als resultaat indien na de controle blijkt dat toevoeging toegestaan is aan de
        verzameling. Alleen bij dit resultaat zal ook daadwerkelijk het element mc toegevoegd worden.
    void removeModelComponent(in ModelComponent mc);
        Deze operatie verwijdert de modelcomponent mc uit de samengestelde component.
};
```

Attribute

Uit de bespreking van het FIW-model bleek dat waarden van systeemvariabelen, stuurvariabelen en systeemparameters niet altijd berekend hoeven te worden. De herkomst van deze waarden kan ook zijn: een database, een invoerschermpje etc. Daarom zijn de attributen van model-componenten (deze stellen de systeemvariabelen, stuurvariabelen en systeemparameters voor)

binnen een modelapplicatie als aparte objecten gemodelleerd. Hierdoor kan de gewenste flexibiliteit met betrekking tot de herkomst van de waarde van een attribuut gewaarborgd worden.



Figuur 4-5
Klassendiagram voor de samenhang tussen model-componenten en attributen

Een modelcomponent bevat dus een verzameling Attribute-objecten. In figuur 4-5 is dit in UML-notatie weergegeven.

De interface van Attribute-objecten bestaat uit de onderstaande methoden:

N.B.: het woord 'Attribute' is gereserveerd in IDL, vandaar dat hier ModelAttribute gebruikt wordt.

```
interface ModelAttribute {
    string getName();
        Levert de naam van het attribuut.
    string getUnitOfMeasurement();
        Deze methode geeft de eenheid waarin waarden van dit attribuut worden gegeven.
    AttributeType getType();
        Geeft als resultaat het type waartoe het attribuut behoort. Er kan bijvoorbeeld een verdeling
        gemaakt worden van attributen naar systeemvariabelen, stuurvariabelen en systeemparameters.
    SourceType getSource();
        Deze methode geeft als resultaat het soort bron waaruit het attribuut de waarden verkrijgt. Een
        attribuut kan als bron bijvoorbeeld van een formule of database gebruik maken.
    double getValue(in Geometry g, in double z, in Time t);
        Levert de waarde voor het attribuut voor de opgegeven locatie en tijdstip.
    DoubleSeq getValueSequence(in Geometry g, in double z, in Time t1, in Time t2);
        Levert een verzameling waarden voor het attribuut voor de opgegeven locatie het tijdsinterval
        [t1, t2]. Het aantal waarden waaruit de verzameling bestaat is afhankelijk van het uitvoer-interval
        van de betreffende bron.
    double getQuality(in Geometry g, in double z, in Time t);
        Deze operatie geeft een indicatie van de betrouwbaarheid van de attribuutwaarde op de
        opgegeven locatie en het tijdstip t.
    boolean isCalculated(in Geometry g, in double z, in Time t);
        Boolean-operatie geeft 'true' als resultaat indien de verkregen waarde op de opgegeven plaats en
        tijd berekend is, 'false' indien interpolatie heeft plaatsgevonden om de waarde te kunnen
        bepalen.
    Time getMinTime();
        Deze methode geeft het minimum tijdstip vanaf wanneer waarden te verkrijgen zijn voor dit
        attribuut.
    Time getMaxTime();
        Deze methode geeft het maximum tijdstip tot wanneer waarden te bepalen zijn voor dit attribuut.
    Time getTimeScale();
        De methode getTimeScale levert de tijdschaal waarmee gerekend wordt voor dit attribuut.
```

```
    double getHorizontalScale();
        De methode getHorizontalScale levert de horizontale ruimteschaal (x, y) waarmee gerekend
        wordt voor dit attribuut.
    double getVerticalScale();
        Analoog aan getHorizontalScale levert deze methode de verticale ruimteschaal (z) waarmee
        gerekend wordt voor dit attribuut.
};

typedef sequence<double> DoubleSeq;

enum AttributeType {
    atSystemVariable,
    atExternalVariable,
    atSystemParameter
};

enum SourceType {
stDatabase,
stFormula,
stConstant
};
```

4.2.5 Generieke tools

Generieke tools zijn componenten met algemene functionaliteit die niet model-applicatie specifiek is. In het Programma van Eisen worden de volgende generieke tools genoemd:

- *procesketenbeheertool (PKBT)* - Met behulp van een PKBT kunnen de onderlinge relaties tussen modelapplicaties en generieke tools worden gedefinieerd, waarna het PKBT ervoor zorgt dat de verschillende modelberekeningen in de juiste volgorde en op een consistente wijze worden uitgevoerd.
- *editor & data-management tool* - Onder een editor wordt verstaan een tool waarmee gegevens (zoals attributen van modelcomponenten) kunnen worden toegevoegd, verwijderd en worden gewijzigd.
- *selectie- en definitietool* - Dit tool is gedefinieerd als een tool waarmee cases en maatregelen kunnen worden gedefinieerd. Cases en maatregelen kunnen worden opgeslagen, geselecteerd en gecombineerd
- *calibratietool* - Een calibratietool ondersteunt de gebruiker bij de calibratie van de in het raamwerk opgenomen modelapplicaties met behulp van parameter-optimalisatie, gevoeligheidsanalyses, etc. Hoewel beschreven in het Programma van Eisen wordt verder (nog) geen aandacht besteed aan dit tool.
- *analysetool* - Een analysetool stelt de gebruiker in staat om gegevens om te zetten in bruikbare informatie door gegevens uit verschillende bronnen te combineren en onderling te vergelijken, te bewerken, te analyseren, etc.
- *presentatietool* - Presentatietools zijn er om gegevens te visualiseren in grafieken, kaarten, tabellen, etc. Visualisatie kan zowel op het scherm als op papier.

Generieke tools zijn, evenals modelapplicaties, raamwerkcomponenten. Het registreren van raamwerkcomponenten binnen het SRW is reeds beschreven in paragraaf 4.2.1. Naast registratie van een generieke tool binnen het raamwerk moeten generieke tools gekoppeld kunnen worden aan modelapplicaties en onderling.

Generieke tools implementeren alle operaties die de GenericTool-interface (zie hieronder) bevat. De interface GenericTool is een uitbreiding van de Framework-Component-interface (zie paragraaf 4.2.1).

```
interface GenericTool: FrameworkComponent {
    void pause();
        Deze operatie stopt (tijdelijk) het tool. Dit kan bijvoorbeeld gebruikt worden om de rekentijd te
        beïnvloeden. Zowel de gebruiker als het raamwerk zelf kunnen opdracht geven om een tool in
        'pauzestand' te zetten.
    void resume();
        Deze operatie laat een tool in pauzestand weer verder werken. Het effect van deze operatie
        verschilt per tool. Een presentatietool zal de visualisatie van de gegevens moeten bijwerken, maar
        een edittool hoeft geen speciale acties te ondernemen.
};
```

Om een tool te kunnen koppelen met een model of een ander tool wordt het eerder beschreven mechanisme gebruikt. Een tool verwacht bepaalde diensten van de componenten waarmee hij kan worden gekoppeld en biedt daarnaast zelf ook diensten aan.

Een tool is generiek wanneer het volledig onafhankelijk is van de te koppelen of ondersteunen modelapplicatie. Om hieraan te kunnen voldoen is in de architectuur veel aandacht besteed aan de interfaces van de tools en modelapplicaties. Een generiek tool zal altijd ondersteuning kunnen geven, zij het mogelijk tot op een beperkt niveau.

Niet alle tools hoeven generiek te zijn; het is bijvoorbeeld niet ondenkbaar dat voor bepaalde modelapplicaties specifieke tools gebouwd zullen moeten worden. Ook deze tools moeten aan de opgestelde interfaces voldoen, maar kunnen van een model heel specifieke diensten (services) verwachten.

Linked tools

Linked tools zijn generieke tools die aan modelapplicaties of andere tools gekoppeld kunnen worden, maar niet deelnemen aan de berekeningen. De in het Programma van Eisen beschreven tools vallen vrijwel allemaal in deze categorie. Sommige tools, zoals een analysetool en editor, zouden als gegevensbron in een configuratie kunnen optreden (dus als soort modelapplicatie). Dergelijke functies zijn echter in het Programma van Eisen niet beschreven. De procesketen-beheertool neemt een bijzondere plaats in het geheel in en wordt later in meer detail beschreven.

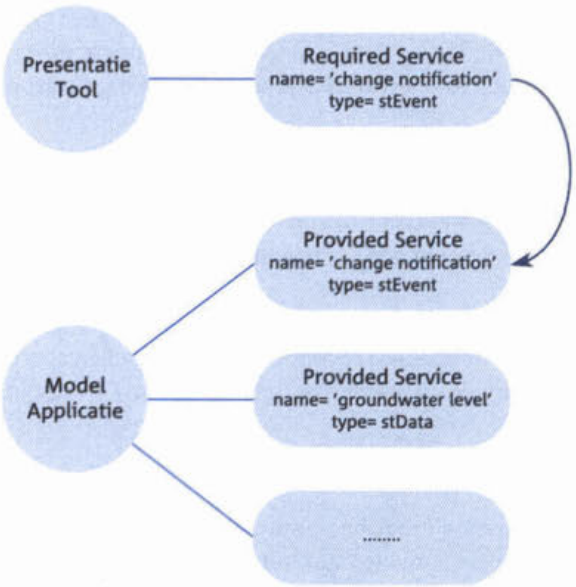
Het meest typische voorbeeld van een linked tool is de presentatietool. Dit tool spreekt het meest tot de verbeelding en wordt hieronder als voorbeeld gebruikt. De andere tools vertonen soortgelijk gedrag in het raamwerk.

In voorbeeld in figuur 4-6 wordt een presentatietool, dat een grafiek toont, gebruikt om de resultaten van een modelapplicatie[7] te visualiseren. De functionaliteit die gewenst is, is dat de grafiek up-to-date blijft met de resultaten van de berekening; met andere woorden: de gebruiker wil 'live' meekijken met de simulatie (en deze mogelijk onderbreken indien onverwachte, vreemde resultaten verschijnen).

---

[7] Er staat hier 'modelapplicatie', waarmee in principe élke component wordt bedoeld die de ModelApplication-interface implementeert (dus mogelijk ook een ander tool).

Zoals eerder beschreven moeten te koppelen raamwerkcomponenten het eens worden over gevraagde en geleverde diensten. De dienst die een presentatietool nodig heeft is van het type 'stEvent', dat wil zeggen dat het tool niets doet totdat er een event optreedt in de gekoppelde modelapplicatie. Uit het ServiceDescriptor-object moet duidelijk worden over welk soort event het gaat.



Figuur 4-6 Schematische weergave van de koppeling van een PresentationTool aan een Model-Application

In figuur 4-6 is schematisch weergegeven op basis waarvan de koppeling tussen model en tool wordt gedaan. Het zal duidelijk zijn dat de naamgeving van diensten aan regels gebonden moet zijn. In dit geval is de dienst "change notification" een standaard dienst.

De feitelijke koppeling wordt gemaakt door de operatie *connect* van het tool aan te roepen. Na interne administratie ontvangt vanaf dat moment het tool een bericht als er een verandering (een nieuwe waarde) optreedt. Bij de communicatie speelt de modelapplicatie de rol van EventSource en het presentatietool die van EventListener (zie paragraaf 4.2.6). De feitelijke informatie zit in het Event-object.

In-process tools

Een in-process tool is een generieke tool die deelneemt aan het rekenproces. Zo'n tool gedraagt zich als een modelapplicatie (implementeert interface ModelApplication) en gebruikers van de diensten hoeven niet te weten dat ze met een tool te maken hebben.

In het Programma van Eisen zijn dergelijke tools niet expliciet beschreven[8]. Een deel van de gewenste flexibiliteit en koppelbaarheid zal in het aanbod van deze in-process tools zitten. In de onderstaande opsomming staan voorbeelden van in-process tools:

- *verschaling* - Een verschalingstool kan zorg dragen voor verschaling van ruimte en/of tijd tussen modellen.
- *conversie* - Een conversietool kan bepaalde gegevens, die tussen modellen worden uitgewisseld, worden geconverteerd. Hierbij kan worden gedacht aan conversie naar andere eenheden.

[8] Het procesketenbeheertool is binnen deze architectuur wel beschouwd als in-process tool.

- *opslag van tussenresultaten* - Een opslagtool kan alle gegevens die langs komen opslaan in een bestand of database. De diensten die het opslagtool aanbiedt, zijn gelijk aan die hij 'ontvangt' (hij stuurt alles gewoon door).
- *debugging* - Een debugtool kan gegevens die tussen modellen worden uitgewisseld controleren en in geval van fouten optreden.
- *optimalisatie* - Een optimalisatietool kan de voor een modelapplicatie benodigde gegevens optimaliseren (bijvoorbeeld door de volgorde te veranderen of een matrix te herordenen).
- *filter* - Een filtertool geeft de invoer door als uitvoer en past deze eventueel aan. De hierboven genoemde tools vallen binnen deze categorie. Er zijn daarnaast nog andere filters denkbaar.

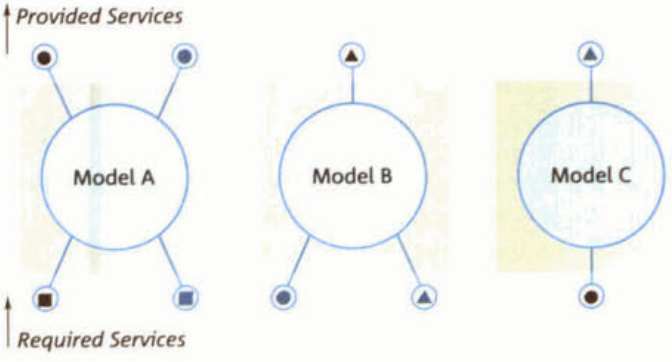
Doordat de in-process tools te koppelen zijn aan modelapplicaties en zichzelf weer gedragen als modelapplicaties kunnen ze onderling in een keten gekoppeld worden. Als de tools duidelijk gescheiden verantwoordelijkheden hebben, zijn al snel vele soorten bruikbare ketens te bouwen.

Net als bij de linked tools is het mogelijk dat voor bepaalde modellen specifieke in-process tools worden gemaakt (met name bij de genoemde optimalisatietools). Deze zijn meestal niet goed herbruikbaar.

Procesketenbeheertool

Een procesketenbeheertool heeft als taak het beheren van een configuratie van verschillende modelapplicaties en tools. Aan de keten (configuratie) die zo'n tool beheert, kunnen componenten worden toegevoegd of verwijderd.

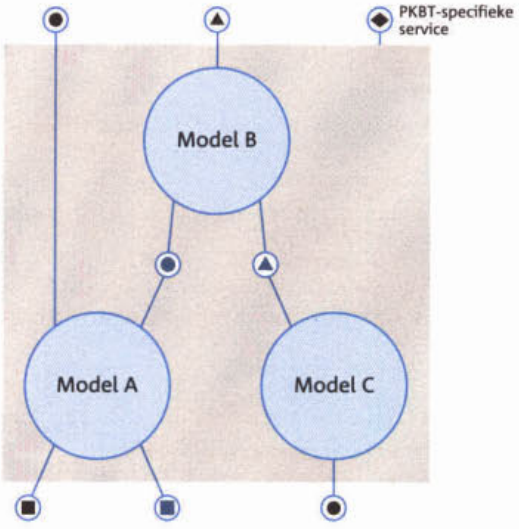
Een configuratie die een procesketenbeheertool bijhoudt, is zelf weer te beschouwen als een model-applicatie. Dit wordt afgedwongen door het feit dat een procesketenbeheertool de interfaces InProcessTool en ModelApplication implementeert. De hier beschreven constructie heeft veel weg van een samengestelde modelcomponent. Het SRW en het procesketenbeheertool bieden ondersteuning bij het maken van procesketens.



Figuur 4-7 *Schematische weergave van drie modelapplicaties die koppelbaar zijn (provided en required services zijn op elkaar afgestemd)*

Net als andere componenten biedt een procesketenbeheertool diensten aan en heeft het diensten nodig. Deze verzamelingen zijn afhankelijk van de onderdelen van de procesketen.

In figuur 4-7 zijn drie modelapplicaties (de blokken) getekend, elk met hun eigen geleverde en gevraagde diensten (in cirkels). Interfaces met hetzelfde patroon zijn gelijk. Als in een procesketenbeheertool een configuratie van deze drie componenten wordt gedefinieerd, kan de volgende component ontstaan:



Figuur 4-8
Schematische weergave hoe de drie modelapplicaties uit figuur 4-7 met behulp van een procesketenbeheertool tot één component geconfigureerd worden

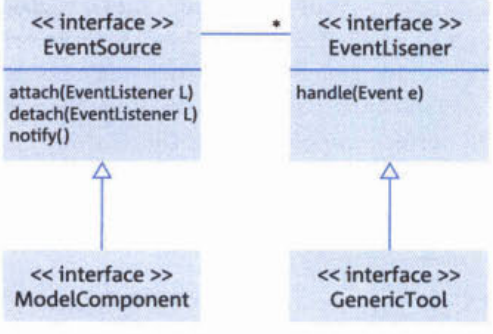
Het procesketenbeheertool als component biedt en vraagt dus zelf ook diensten.

4.2.6 Events

Generieke tools moeten binnen het SRW samenwerken met modelapplicaties. Dit betekent dat bijvoorbeeld een presentatiecomponent op de hoogte moet worden gebracht van het feit dat een nieuwe waarde voor een attribuut is berekend door een modelapplicatie. Om deze communicatie tussen modelapplicaties en inhakende tools (linked tools) mogelijk te maken wordt gebruik gemaakt van het zogenaamde publish-subscribe mechanisme. Dit mechanisme is ook bekend als het Observer Design Pattern (zie [GAMMA95]).

Componenten voor bijvoorbeeld presentatie of validatie worden gemodelleerd als een zogenaamde EventListener, een component die op de hoogte gebracht kan worden van gebeurtenissen tijdens simulaties (events). Modelcomponenten kunnen deze events genereren. Modelcomponenten worden daarom gemodelleerd als EventSource.
Een ModelComponent kan zich 'abonneren' bij één of meerdere EventListeners. Als er iets verandert in de modelcomponent, bijvoorbeeld de waarde van een attribuut, worden alle abonnees daarvan op de hoogte gesteld.

Een voorbeeld van een EventListener is een grafiekcomponent die in een grafiek de waarde van een attribuut, bijvoorbeeld grondwaterstand, in de loop van de tijd weergeeft. De EventListener heeft zich geabonneerd bij een modelapplicatie die grondwaterstanden uitrekent. Als de modelapplicatie een nieuwe waarde heeft berekend, worden de abonnees geïnformeerd. In het geval van de grafiekcomponent wordt een nieuwe waarde toegevoegd aan de grafiek.



Figuur 4-9
Klassendiagram van de Publish-Subscribe architectuur binnen het SRW

Uit het klassendiagram (figuur 4-9) is af te leiden dat een ModelComponent de EventSource-interface implementeert:

```
interface EventSource {
    attach(in EventListener L);
        Het abonneren van een ModelComponent op bijvoorbeeld een presentatiecomponent L vindt
        plaats met behulp van deze methode.
    detach(in EventListener L);
        Het opheffen van een abonnement op bijvoorbeeld een presentatiecomponent L vindt plaats met
        behulp van deze methode.
    void notify();
        Het melden van een gebeurtenis (event) aan alle EventListeners wordt gerealiseerd binnen deze
        operatie. Binnen deze methode wordt een event-object gecreëerd.
};
```

Uit het voorgaande klassendiagram is ook af te leiden dat generieke tools naast het GenericTool-interface de EventListener-interface implementeren.

```
interface EventListener {
    void handle(in Event e, in EventSource source);
        Door aanroepen van deze operatie worden generieke tools (en andere event listeners) op de
        hoogte gebracht van gebeurtenissen (events), gegenereerd door event sources (zoals
        modelcomponenten). Binnen de implementatie van deze methode kan bijvoorbeeld een punt
        toegevoegd worden aan een grafiek, als representatie voor een nieuw berekende waarde.
};
```

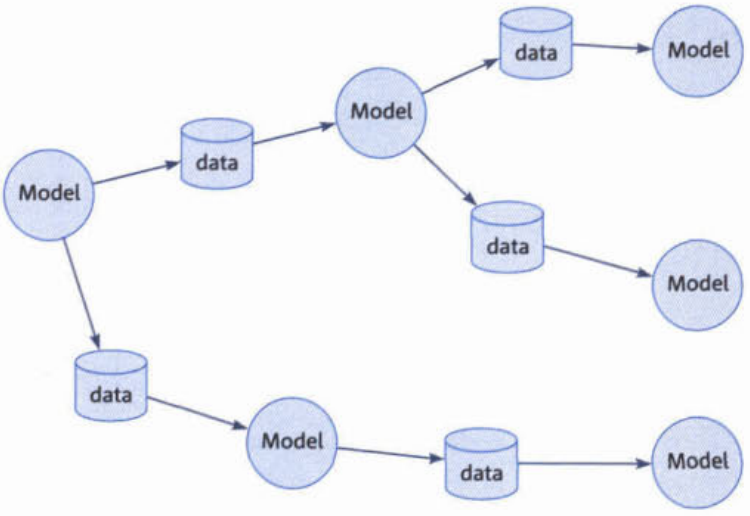
4.3 Dynamische structuur van componenten

In paragraaf 4.2 is aandacht besteed aan de statische structuur van componenten. Deze paragraaf beschrijft hoe componenten (modelapplicaties en tools dus) zich gedragen tijdens runtime. Als eerste zal de relatie tussen het domeinmodel en het koppelen van modellen worden beschreven. Daarna wordt beschreven hoe het SRW er uit zou kunnen zien en hoe het zich in verschillende situaties zou kunnen gedragen. Hierbij wordt gebruik gemaakt van de interfaces die beschreven zijn in paragraaf 4.2.

4.3.1 Modelkoppelingen en modelketens

In paragraaf 3.2.2 is conceptueel beschreven hoe modelapplicaties met elkaar communiceren. Modelapplicaties bestaan uit modelelementen en connectoren, speciale vormen van modelcomponenten. Modelcomponenten kunnen op verschillende manieren geïmplementeerd worden. Een modelcomponent kan een onderdeel van een modelprogramma zijn, maar een modelcomponent kan op zichzelf ook een compleet model zijn. (Zie ook paragraaf 4.2.3 over samengestelde componenten). Connectoren zijn dus als metafoor ook te gebruiken voor koppelingen tussen modellen. Door dit gedeelte van het domeinmodel voor zowel grote als kleine schaal te gebruiken, wordt de complexiteit verminderd en worden modelcomponenten uitwisselbaar ongeacht hun aard.

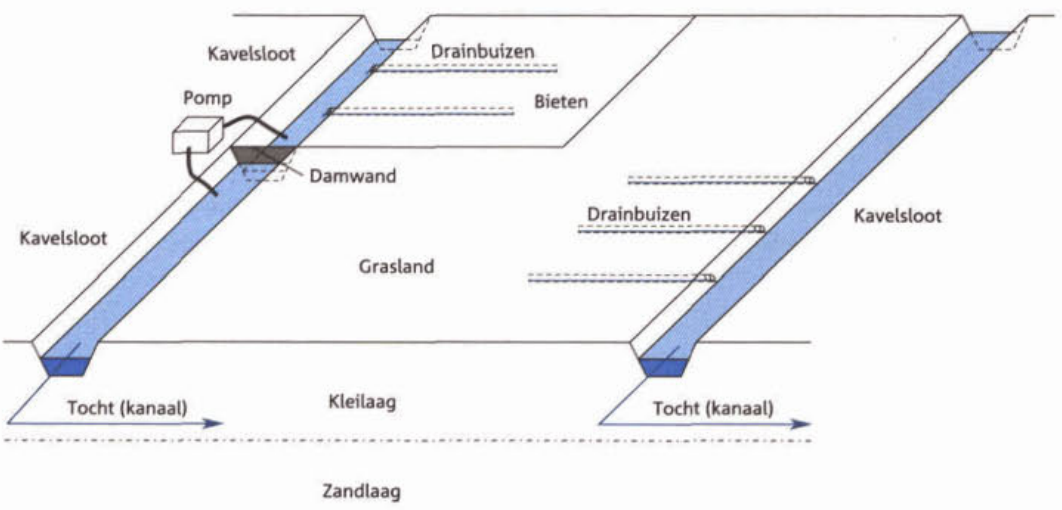
Een modelketen wordt traditioneel (bijvoorbeeld binnen het DSS Winbos) vaak voorgesteld als een verbonden keten van blokjes (gegevens, input, output) en bolletjes (modellen):



Figuur 4-10 Schematische weergave van een modelketen, bestaande uit modellen die via gedeelde data-bronnen aan elkaar gekoppeld zijn

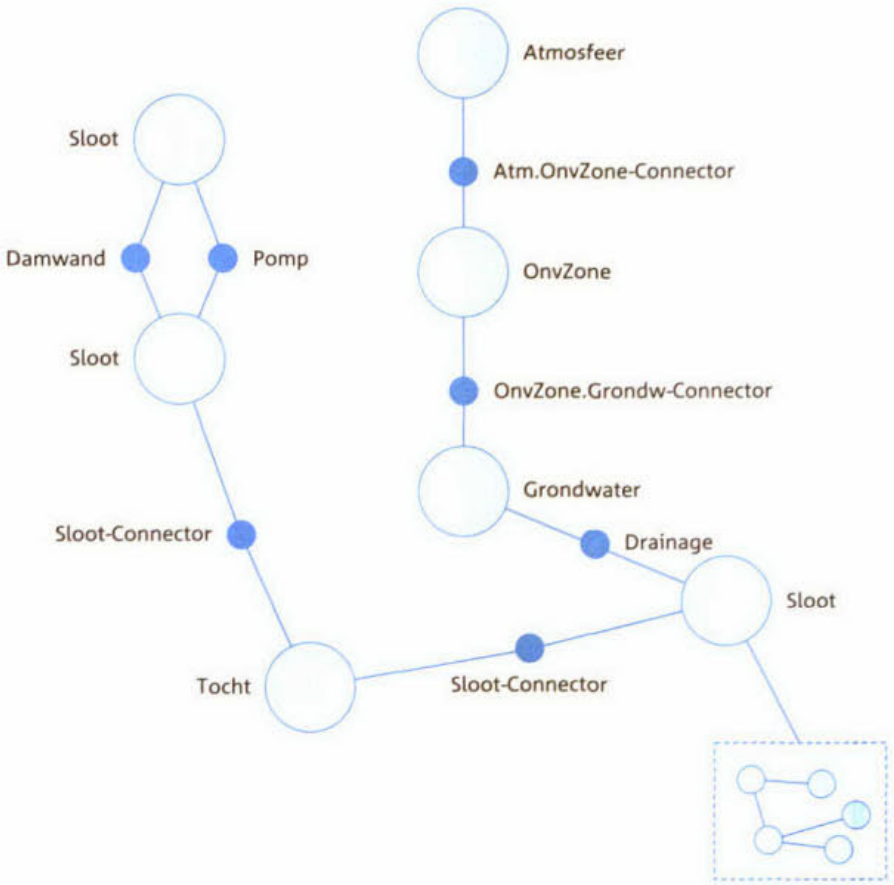
Binnen figuur 4-10 is duidelijk af te leiden wat de sequentie is van de keten: een model kan pas rekenen tot een bepaald tijdstip als alle voorafgaande modellen ook tot dat tijdsstip hebben gerekend en hun resultaten weggeschreven zijn.

In een van de workshops voorafgaand aan het schrijven van dit rapport is een aantal figuren ontwikkeld waarin modellen als model-elementen werden gekoppeld en zo één model vormen. Eén van die figuren bevat het zogenaamde poldermodel (figuur 4-11).



Figuur 4-11 Schematische weergave van een polder (zgn. poldermodel)

Aan de hand van figuur 4-11 is een objectmodel gemaakt in termen van model-elementen en connectoren. Hierbij is er vanuit gegaan dat een waterafvoermodel toegepast gaat worden.



Figuur 4-12 Schematische weergave van het poldermodel (fig. 4-11) met verschillende modelcomponenten (ModelElement of Connector)

In dit geschematiseerde poldermodel is voornoemde sequentie in de modelketen niet terug te vinden. Er zijn blijkbaar twee vormen van modelkoppelingen: modellen worden onderling gekoppeld, of binnen één model worden modelelementen gekoppeld. De relatie tussen beide wordt hieronder in een aantal fases ontwikkeld.

Fase 1

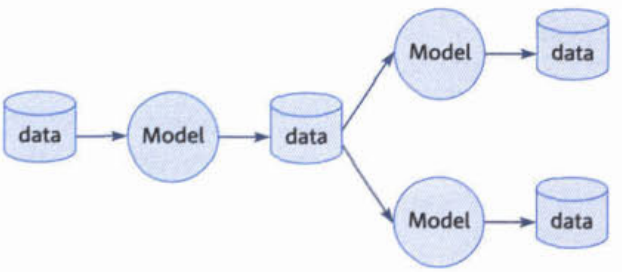
Elk model heeft input en output. Dit kan als volgt afgebeeld worden (modellen worden afgebeeld als rondjes, gegevens worden afgebeeld als tonnetjes).

Fase 2

Als er modellen aan elkaar gekoppeld moeten worden dan is de output van het eerste model de input van het tweede model. Of gegevens input of output zijn blijken dus meer een rol te zijn van de gegevens dan dat het een vast gegeven is. We schrijven daarom in de tonnetjes data.

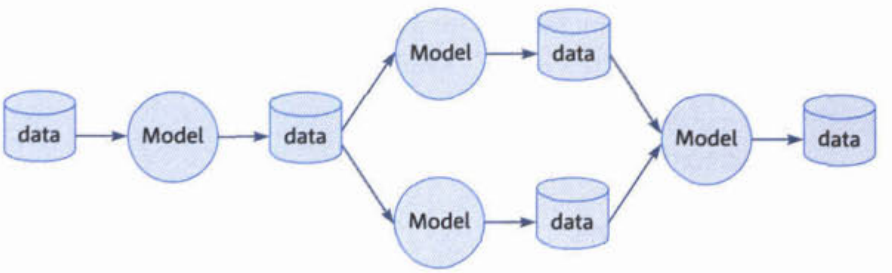
Fase 3

In bovenstaande simpele modelketen kan het tweede model pas uitgevoerd worden als het eerste model klaar is. Modelketens kunnen ook iets ingewikkelder zijn, bijvoorbeeld wanneer na een eerste model twee andere modellen uitgevoerd worden. In zo'n modelketen kunnen de twee modellen aan de rechterkant tegelijkertijd uitgevoerd worden. We kunnen deze modelketen dan ook 'parallel' noemen. Dit in tegenstelling tot de vorige modelketen (fase 2), deze heet 'serieel'.



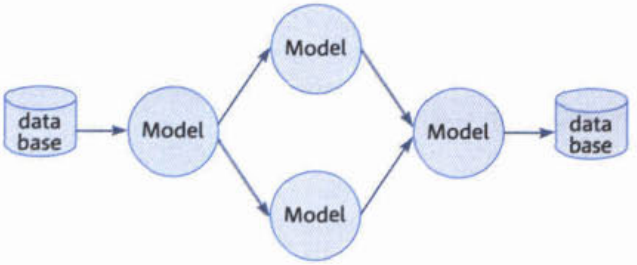
Fase 4

In deze vierde keten koppelen we rechts nog een eindmodel dat de data van zowel het tweede model als het derde model als input heeft. Deze modelketen is nog steeds parallel, maar het vierde model kan slechts uitgevoerd worden als het tweede en het derde model klaar zijn. Dit wordt synchronisatie genoemd.



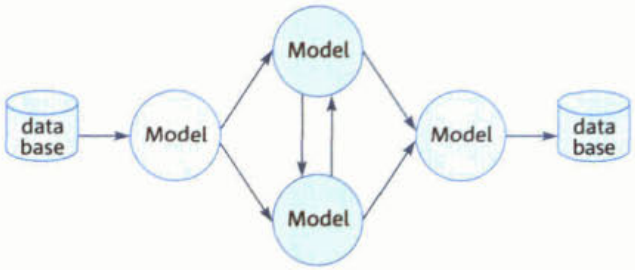
Fase 5

Bovenstaande ketens wisselen hun gegevens uit middels bestanden. Dit houdt in dat een model een uitvoerbestand heeft dat een ander model weer als invoerbestand leest. Dit kan vaak problemen veroorzaken. Beter zou het zijn dat de modellen de gegevens op een meer rechtstreekse manier met elkaar uitwisselen. De tonnetjes vallen dan weg en we krijgen de volgende keten:



Deze modellen kunnen overigens wel gegevens lezen of wegschrijven; daarom zijn de databases in deze modelketen getekend. Deze zouden ook bij de middelste modellen getekend kunnen worden. Belangrijk is vooral dat de modellen hun gegevens niet middels bestanden met elkaar uitwisselen. Voor de modelketen is het niet zo belangrijk waar de data zelf precies staan. Elk model beheert zijn eigen data, een model ziet er dus eigenlijk als volgt uit:

Fase 6
Modellen kunnen ook tijdens uitvoering met elkaar gegevens uitwisselen. We spreken dan van een dynamische koppeling. Dit is afgebeeld in de volgende modelketen:



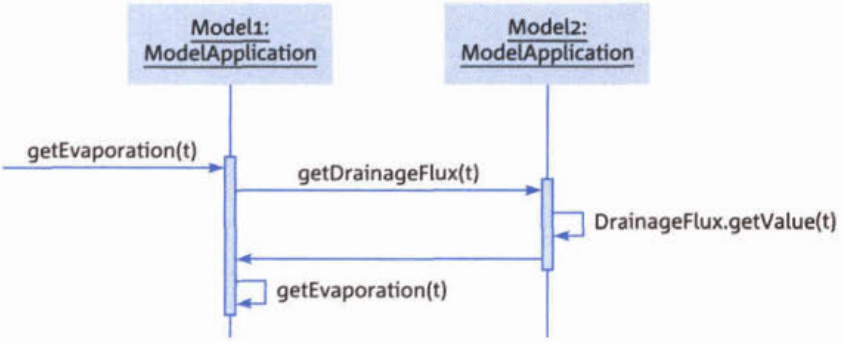
Deze laatste modelketen komt eigenlijk op hetzelfde neer als de keten afgebeeld in 'modellering van een polder' (fig. 4-12). Elk modelelement (dat een klein onderdeel van een model kan zijn, of een model zelf) kan data uitwisselen met andere modelelementen.

4.3.2 Uitvoering van modellen

Modellen worden in het raamwerk opgenomen om daarin op een gegeven moment uitgevoerd te worden. De uitvoering van zo'n model kan op verschillende manieren worden geïnitieerd. Ten eerste zou je een methode *Run* van de modelcomponent kunnen aanroepen, waarna het model zijn uitvoering begint. Dit werkt goed bij modellen die geheel onafhankelijk van andere modellen draaien. Maar bij dynamische koppelingen (fase 6) gaat dit niet zo makkelijk: het ene model heeft output nodig van het andere model en het andere model weer van het ene model (maar b.v. voor een ander tijdstip).

Een andere manier die geschikter is voor dynamische koppelingen is het vragen van uitvoer van een model ('pull'). Als het model zelf weet dat deze uitvoer nog berekend moet worden dan zal het model aan het rekenen gaan. Dit gebeurt dus min of meer automatisch. Deze manier van uitvoering is ook geschikt voor dynamische koppelingen: model 1 vraagt model 2 om output, model 2 gaat rekenen en levert output, model 1 kan weer verder en kan later model 2 opnieuw om (andere) output vragen.

In het sequence-diagram van figuur 4-13 is een voorbeeld uitgewerkt waarbij model 1 de evaporatie pas kan berekenen voor een tijdstip t wanneer de drainageflux op tijdstip t bekend is. Voor het berekenen van de drainageflux wordt gebruik gemaakt van model 2.



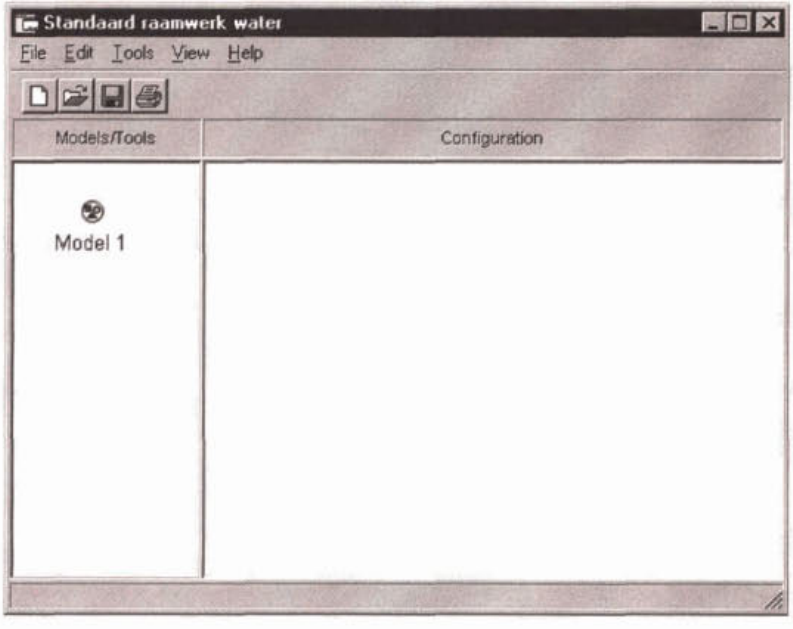
Figuur 4-13
Sequence-diagram voor de uitvoering van de berekening bij dynamische koppeling

4.3.3 Registreren van componenten

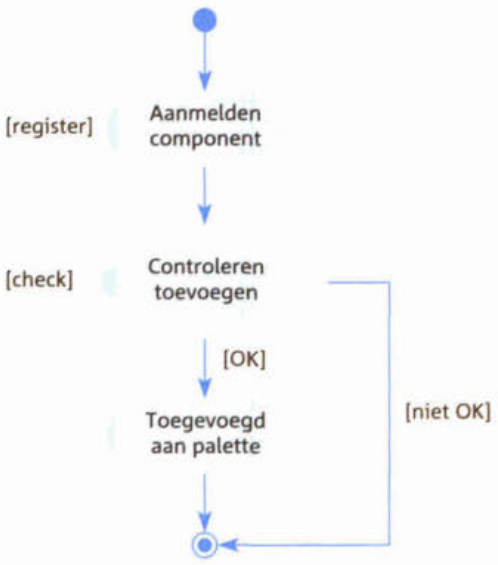
In de secties 4.3.3 - 4.3.6 wordt de uitvoering beschreven van twee gekoppelde modellen. Hierbij wordt gelet op wat de gebruiker zou kunnen doen om de koppeling tot stand te brengen. Verder wordt beschreven welke functies hiervoor door de verschillende componenten worden aangeroepen. De voorbeeldmodellen zijn de modellen, genoemd in paragraaf 4.2.2 (fig. 4-1):

- een oppervlaktemodel met als output de variabelen *Waterlevel*, *Evaporation* en *Runoff*. Input van dit model moet zijn de *DrainageFlux*;
- Het tweede model heeft als output *DrainageFlux*.

Gebruikers die het SRW voor de eerste keer opstarten beginnen (afhankelijk van implementatie) met een leeg scherm, verdeeld in twee lege gedeelten. In figuur 4-14 is zo'n voorbeeldscherm gegeven. Links bevindt zich het palet waarin modellen en tools kunnen worden geplaatst. Het rechtergedeelte bestaat uit een canvas waarin modellen en tools kunnen worden gesleept om daar aan elkaar gekoppeld te worden. Wanneer gebruikers componenten installeren worden deze in het palet opgenomen. Het registreren van componenten is weergegeven in een *activity diagram* in figuur 4-15. Het laat zien dat in run-time een component ter registratie aangeboden wordt. Wanneer dit goed verloopt dat wordt de geregistreerde component aan het palet toegevoegd, zodat het direct beschikbaar is binnen SRW.



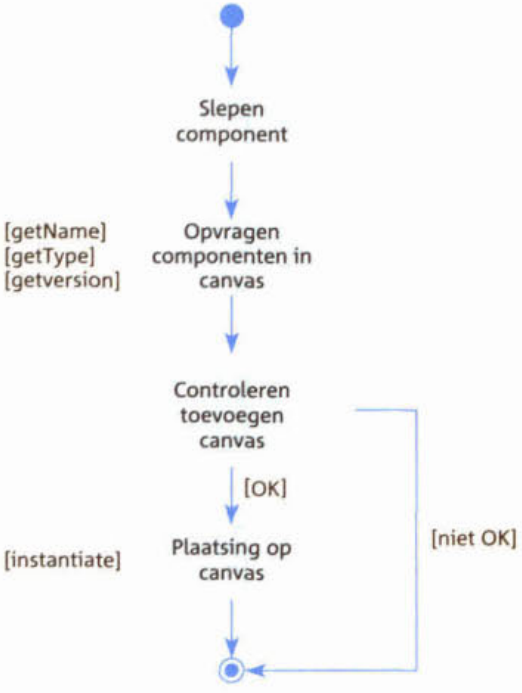
Figuur 4-14
Scherm van SRW met één geregistreerde component



Figuur 4-15
Activity diagram voor het registreren van een component

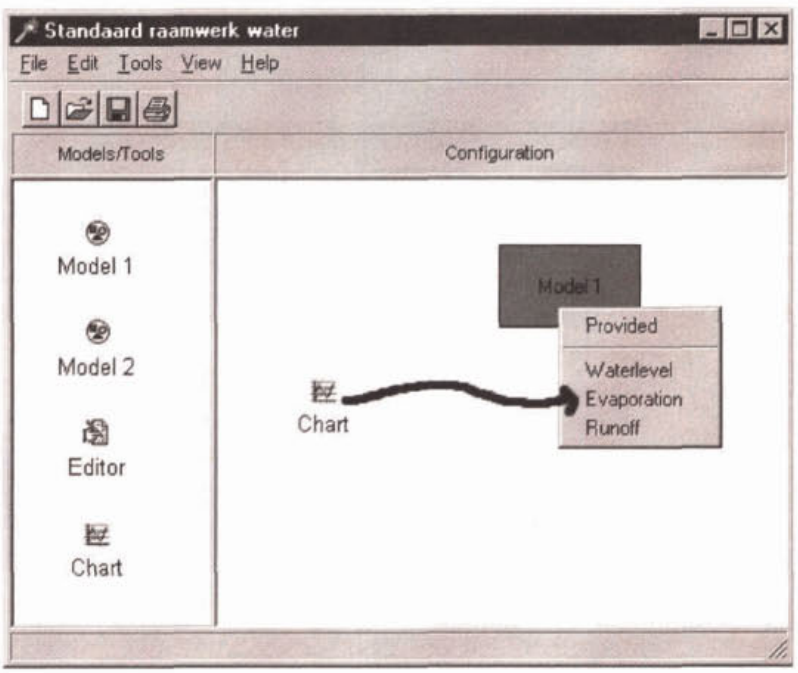
4.3.4 Instantiëren van componenten

Als de gebruiker modellen en tools aan het palet heeft toegevoegd, kan hij in het canvas (in de figuur rechts) een configuratie van modellen en tools maken. Hij kan een model in het canvas slepen. Dit wordt instantiëren genoemd omdat je een daadwerkelijke instantie van een geregistreerd model wilt hebben, met zijn eigen data en instellingen. Als de gebruiker een component instantieert, controleert het raamwerk of dit mogelijk en toegestaan is.



Figuur 4-16
Activity-diagram voor het instantiëren van een component aan het canvas

Voorts wil de gebruiker output zien van dit model. Hij dient hiervoor een *PresentationTool* aan het configuratie-canvas toe te voegen, bijvoorbeeld door een grafiek in het canvas te slepen. Dan moet de gebruiker deze PresentationTool verbinden aan het model (zie fig. 4-17). Hij kan hierbij één van de attributen die het model oplevert kiezen, hierbij wordt de functie *getRequiredServices* van de component aangeroepen en vervolgens wordt de functie *getAttributes* van de opgevraagde lijst *ServiceDescriptor* aangeroepen (zie fig. 4-18). De attributen kan door de user-interface worden vertaald naar een popup-menu waarin de opgeleverde attributen worden getoond. De gebruiker selecteert dan één van de attributen en de methode *connect* wordt aangeroepen. Het model is dan verbonden aan de tool.

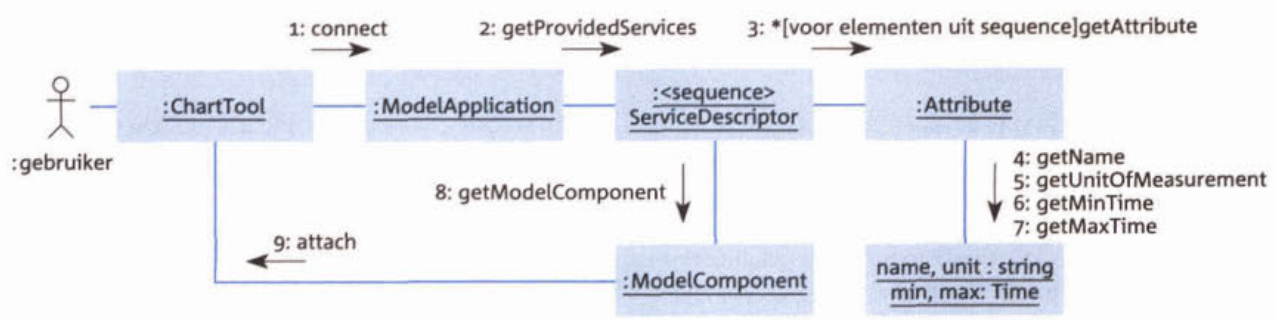


Figuur 4-17
Scherm waarmee het verbinden van een Tool aan een model in het SRW wordt geïllustreerd

In een grafiek worden gewoonlijk de naam en de eenheid van een weer te geven attribuut vermeld. Hiervoor worden de functies *getName* en *getUnitOfMeasurement* aangeroepen. Dan vraagt de tool aan het gekozen attribuut om de minimale en maximale tijd (functies *getMinTime* en *getMaxTime*). Hiermee kan de grafiek geschaald worden. De grafiek geeft aan dat het de output van een model wil weergeven door de functie *attach* aan te roepen. Hierdoor zal de grafiek bij wijzigingen in modelwaarden automatisch een bericht krijgen.

Figuur 4-18
Collaboration-diagram voor een ChartTool en ModelApplication bij het koppelen van een model en een grafiek

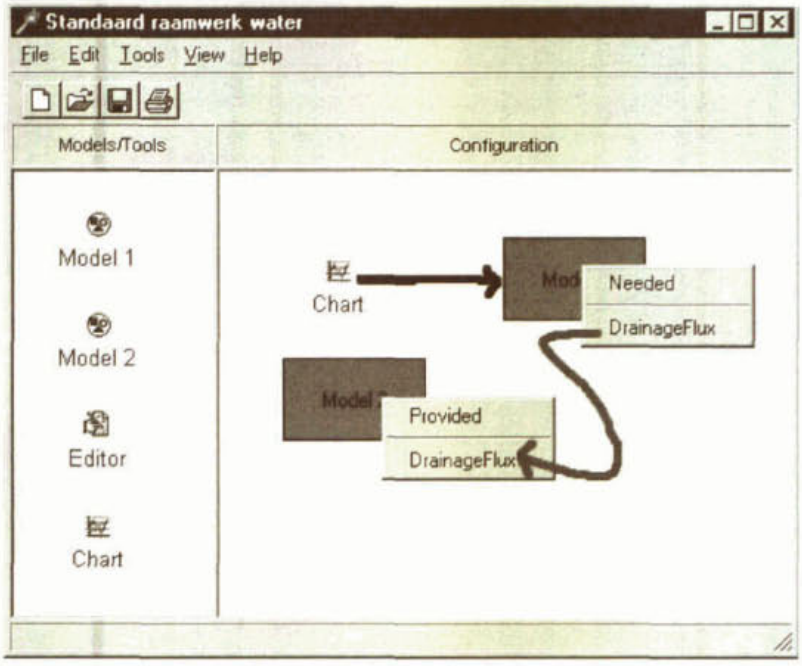
De tool vraagt om model output met de functie *getValueSequence*. In deze laatste functie moet ook een locatie worden opgegeven. Hiervoor wordt in eerste instantie de default-locatie gebruikt (b.v. links onder). Eventueel kan hier ook een kaartselectie-tool voor gebruikt worden.



Wanneer de grafiek tool nu daadwerkelijk om de output van het model vraagt (met *getValue* of *getValueSequence*) blijkt dat in dit huidige voorbeeld niet te gaan. Het model heeft als input nog het attribuut *DrainageFlux* nodig. Het model is niet verbonden aan een ander model dat dit attribuut oplevert. In de tot dusver geschetste configuratie is er ook geen model die dat attribuut kan opleveren.

4.3.5 Koppelen van modellen

In paragraaf 4.3.4 werd één model gekoppeld aan één tool. De output kon niet worden getoond omdat er niet aan alle inputvoorwaarden van het eerste model werd voldaan. In deze sectie wordt beschreven hoe er een model wordt toegevoegd en beide modellen worden gekoppeld (zie fig. 4-19).



Figuur 4-19
Scherm ter illustratie van het koppelen van twee modellen in het SRW

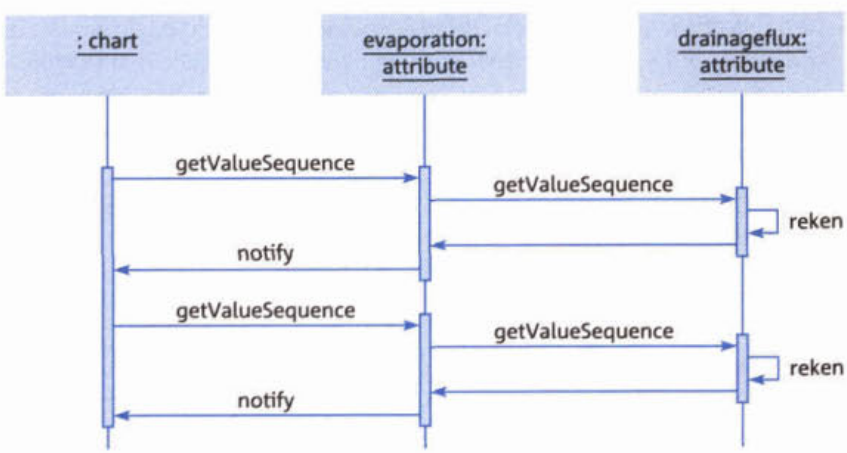
De gebruiker sleept een tweede model in het canvas (zie fig. 4-19). Het SRW probeert beide modellen automatisch te koppelen en roept voor alle modellen de functie *areRequiredServicesComplete* aan. Omdat dit niet geldt voor het eerste model wordt voor dat model *getRequiredServices* aangeroepen. Deze levert, zoals hierboven al is beschreven, een lijst met *ServiceDescriptor's* op, waarop de methode *getAttribute* wordt toegepast. Het eerste model blijkt behoefte te hebben aan een attribuut *DrainageFlux*. Dan kijkt het SRW of er andere modellen zijn die dit attribuut opleveren met de functies *getProvidedServices* en *getAttribute*. Het tweede model blijkt dit attribuut inderdaad op te leveren. De eenheden worden nu gecontroleerd door de functie *getUnitsOfMeasurement* van beide attributen aan te roepen. Als dit klopt kunnen beide modellen worden gekoppeld door de functie *connect* aan te roepen.

In veel gevallen zullen de modellen niet direct gekoppeld kunnen worden. De modellen sluiten bijvoorbeeld wel qua attribuut bij elkaar aan maar de eenheden kloppen niet. Ook kan de modelschematisatie niet bij elkaar aansluiten. In dit soort gevallen kan er een InProcessTool (zie paragraaf 4.2.5) tussen geplaatst worden. Zo'n InProcessTool kan de conversie van eenheden van het ene model naar het andere model verzorgen. Ook kan zo'n tool conversies in modelschematisaties of in tijdstappen van modellen verzorgen.

4.3.6 Uitvoeren van modellen: dynamische koppelingen

De tool (de grafiek in fig. 4-19) kan nu aan Model 1 opnieuw om output gaan vragen. Deze hernieuwde poging levert succes op want Model 1 kan nu gaan rekenen omdat Model 2 de gevraagde drainageflux levert.

Figuur 4-20 beschrijft hoe bij het uitvoeren van modellen gegevens bijvoorbeeld in een grafiek gepresenteerd worden. Bij aanroep van de methode *getValueSequence* gaat het model zelf rekenen. De berekening van het model is nog niet uitgevoerd dus het model zal moeten gaan rekenen. Aan de grafiek worden nul waarden teruggegeven omdat er nog geen waarden zijn. Het eerste model heeft als input *DrainageFlux* nodig. Dit wordt gevraagd aan het tweede modelattribuut. Deze slaat aan het rekenen voor één tijdstap. Als deze berekening klaar is dan wordt het resultaat aan het eerste model teruggegeven. Dit eerste model geeft de grafiek een bericht (*notify*) dat zijn waarden zijn gewijzigd. De grafiek vraagt dan de gegevens op met *getValueSequence* (zie fig. 4-20).



Figuur 4-20 *Sequence-diagram van het koppelen van een grafiek aan twee samenwerkende modellen in het SRW*

4.4 Technologische infrastructuur

In paragraaf 4.3 zijn componenten als abstracte, conceptuele blokken beschreven die onderling samenwerken en gebruik maken van elkaars diensten. Om het SRW te implementeren alsmede de verschillende tools en modelapplicaties, moet een vertaling plaatsvinden van conceptuele componenten naar 'fysieke' componenten.

Het SRW is een zelfstandig draaiende executable waarin componenten geïnstalleerd kunnen worden. Een fysieke component in het SRW is een pakket software dat zich enerzijds gedraagt volgens eerder genoemde regels en anderzijds zich in een binaire (executeerbare) vorm op een machine bevindt (op een PC, een mainframe, etc.). Bij een binaire component kan bijvoorbeeld worden gedacht aan een executable (een .EXE-bestand onder Windows) of een dynamische link library (DLL).
Om de fysieke componenten met elkaar te kunnen laten communiceren is aparte software nodig. De communicatie kan worden verzorgd door het besturingssysteem of door zogenaamde middleware, speciale software die belast is met het regelen van communicatie en distributie. In paragraaf 4.4.2 wordt verder ingegaan op het gebruik van middleware.

4.4.1 Installeren van componenten

Het installeren van componenten (generieke tools of modelapplicaties) kan, afhankelijk van de implementatie, op een aantal manieren plaatsvinden. De component kan voorzien zijn van een 'setup'-programma dat ervoor zorgt dat de component automatisch in het palet van het SRW verschijnt. Een andere mogelijkheid is dat het SRW beschikt over een 'add-in manager', waarmee componenten geïnstalleerd kunnen worden. Ook kan de component door de gebruiker vanuit de explorer in het palet gesleept worden. Tenslotte kan de gebruiker in een catalogus op Internet kijken en daar een component selecteren. De component kan dan naar zijn systeem toekomen (download) of de component kan remote blijven staan. In het laatste geval wordt er slechts een referentie naar de component geregistreerd (paragraaf 4.4.2). Als de component een model is zal deze dan ook op afstand (op een andere machine) draaien. Nogmaals, dit alles is afhankelijk van de mogelijkheden die de gekozen implementatie biedt.

4.4.2 Distributie

De applicatie moet, zoals in het Programma van Eisen en in dit document is aangegeven, gedistribueerd kunnen draaien. Dit houdt in dat de applicatie (het SRW) op één machine draait, maar de modellen op die machine maar ook op een andere machine kunnen draaien, verbonden door een netwerk. Ze worden dan aangestuurd door het raamwerk.

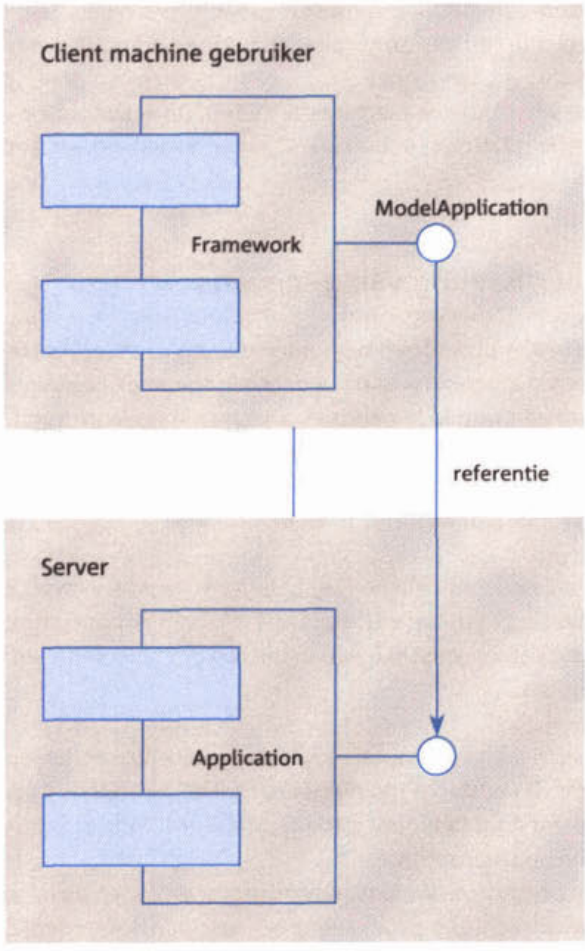
Er zijn een aantal technologieën beschikbaar om zo'n gedistribueerde omgeving te realiseren. Hier worden genoemd:

- Corba (Common Object Request Broker Architecture van de Object Management Group);
- DCOM (Distributed Component Object Model van Microsoft);
- Java (van Sunsoft) en met name zijn RMI (Remote Method Invocation) als middleware;
- andere technologieën voor netwerkcommunicatie zoals RPC (Remote Procedure Calls) en RMI (Remote Method Invocation uit Java), deze worden hier buiten beschouwing gelaten.

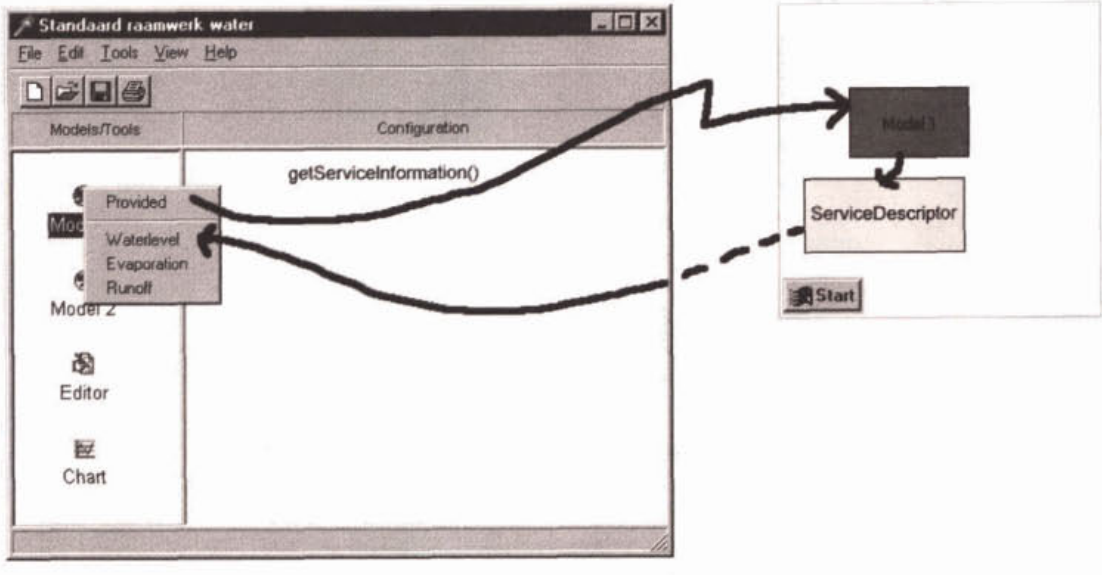
Genoemde omgevingen zullen hier niet uitvoerig worden besproken, hiervoor is genoeg referentiemateriaal beschikbaar (zie bv. Appendix D voor meer achtergrond-informatie). Bij de beschrijving in deze paragraaf van zo'n gedistribueerd systeem wordt Corba als voorbeeld genomen.

Figuur 4-21 geeft aan hoe een model op afstand draait van het SRW. Het SRW heeft een referentie naar een interface van *ModelApplication*. Zo'n referentie in Corba betekent niet dat het object zelf op dezelfde machine hoeft te staan. Het object kan wel direct benaderd worden. De benadering is zogezegd transparant.

Een *ModelApplication* kan worden geregistreerd in het SRW zoals in de vorige sectie is besproken. Het SRW weet dan hoe het model benaderd moet worden. Alle functionaliteit (in het canvas slepen, opvragen van beschikbare attributen door middel van de functie getProvidedServices, etc) werkt geheel transparant, dus alsof het object op dezelfde machine staat (zie ook fig. 4-22). Als er data aan het object wordt opgevraagd, b.v. met de functie *getValue*, kan het zijn dat het model moet gaan rekenen. Dit rekenen gebeurt dan uiteraard op de machine waar het object (het model) zelf staat. Als het model gebruik maakt van een database dan gebeurt dat ook lokaal. Alleen als het model andere modellen benadert dan gaat dat via het netwerk.



Figuur 4-21 *Deployement-diagram voor het koppelen van modellen op afstand met SRW*



Figuur 4-22 *Scherm ter illustratie hoe het benaderen van modellen op een andere machine werkt in het SRW*

Voor de plaatsing van componenten op systemen kunnen verschillende strategieën worden gebruikt. Zo kan het omwille van performance-verbetering nodig zijn om twee componenten op dezelfde computer te plaatsen, terwijl het ook mogelijk is twee componenten die parallel kunnen werken juist op verschillende machines te zetten. De keuze voor plaatsing is sterk afhankelijk van de situatie (platform, middleware, netwerksnelheid, etc.) en van de configuratie van modellen.

4.5 Opslag en uitwisseling van gegevens

Opslag en uitwisseling van informatie en het ondersteunen daarvan is essentieel voor het SRW. De aard van de gegevens is afhankelijk van het gebruik; sommige worden als invoer (randvoorwaarden) voor een simulatie gebruikt, andere zijn tussenresultaten voor een berekening, weer andere zijn eindresultaten gevraagd door de gebruiker. Van een verzameling gegevens zijn vaak de (toekomstige) vormen van gebruik niet bekend. Om die reden is het van belang dat de toegang tot deze data uniform en helder is.

Het koppelen van verschillende modellen is het belangrijkste doel van het SRW. Gekoppelde modellen wisselen onderling informatie uit op basis waarvan zij berekeningen uitvoeren. De overdracht van gegevens kan op verschillende manieren geïmplementeerd worden, evenals de (eventueel tijdelijke) opslag.

In het onderstaande overzicht is een aantal mogelijke gegevensbronnen opgenomen:

- *bestand* - Deze vorm van opslag is momenteel de meest gebruikte. Gegevens zijn vastgelegd in een tekstbestand dat door het modelprogramma wordt ingelezen of wordt geschreven. Dit is een persistente gegevensbron;
- *database* - Naarmate de hoeveelheid gegevens groter wordt en de nabewerkingen complexer van aard worden, zijn databases geschikter dan bestanden. Een database-managementsysteem (DBMS) stelt de gebruiker in staat om ingewikkelde zoekvragen te laten stellen en die efficiënt te beantwoorden. Net als bestanden is dit een persistente gegevensborn;
- *gebruiker* - Niet alle gegevens die in een simulatie worden gebruikt, hoeven uit een persistente gegevensbron (zoals een bestand of een database) te komen. Het is goed mogelijk dat de gebruiker, voordat de simulatie begint met rekenen, wordt gevraagd naar bijvoorbeeld stuurparameters. Dit is een niet-persistente gegevensbron.

Uitwisseling en overdracht van data kan op allerlei manieren gerealiseerd worden. Grofweg kunnen de manieren in twee groepen worden ingedeeld: overdracht via een persistente of niet-persistente gegevensbron.

Overdracht die niet via een persistente gegevensbron gaat, wordt ook wel 'on-line gegevensoverdracht' genoemd. Deze overdracht gaat gebruikelijk via parameters in een aanroep. Een model A roept een functie aan van een model B en geeft daarbij de benodigde gegevens. Belangrijk hierbij is dat model B precies voldoende gegevens krijgt om de gewenste berekening uit te voeren. De interface van model B moet nauwkeurig en volledig gedefinieerd zijn. Voorts is het voor model A niet van belang of het resultaat van de aanroep feitelijk berekend is of uit een persistente gegevensbron is opgehaald; met andere woorden, model B 'speelt' voor model A de rol van gegevensbron.

De overdracht van gegevens kan ook via een persistente gegevensbron gaan. In dat geval maakt een model gebruik van de bron om berekeningen uit te voeren en moet het model dus weten hoe deze bron aan te spreken. De oorsprong van de gegevens is niet van belang en kan variëren (ingevoerd door gebruiker, berekend door ander model, geleverd door derde, etc.). Om het gebruik en uitwisseling van bestanden en databases te vergemakkelijken (en mogelijk te maken) is het belangrijk dat ze aansluiten bij een gangbare standaard. Hiervoor kan gebruik gemaakt worden van de definities uit Stekkerdoos Water en SUF-OW.

Bovenstaande varianten van gegevensoverdracht, zijn, hoewel in detail en mogelijk in performance verschillend, binnen de architectuur op uniforme manier te beschrijven. Een model heeft voor berekening bepaalde gegevens, oftewel bepaalde diensten, nodig. Deze diensten kunnen worden geleverd door een component die enkel gegevens beheert (bijvoorbeeld een database of bestandenverzameling) of door een component die nieuwe waarden berekent (een modelprogramma).

In paragraaf 4.5 is enkel aandacht besteed aan opslag ten behoeve van uitwisseling tussen modellen en tools. Een modelprogramma kan, en dat is niet onwaarschijnlijk, daarnaast gebruik maken van een 'eigen' opslagmedium of opslagformaat. Dit gaat met name op voor bestaande modellen die aangepast of omgeschreven moeten worden voor integratie met het SRW.

5. Migratie van bestaande modelprogramma's

5.1 Inleiding

Het SRW biedt de mogelijkheid om modellen onderling en met tools te laten samenwerken. Om een rol te kunnen spelen moeten modellen en tools aan specifieke voorwaarden voldoen (ze moeten bijvoorbeeld als FrameworkComponent gebruikt kunnen worden); het is zeer onwaarschijnlijk dat bestaande applicaties zonder aanpassing die rol kunnen spelen. Dit betekent dat bestaande software aangepast moet worden, met andere woorden: de software moet 'migreren'. In dit hoofdstuk wordt ingegaan op software-migratie in algemene termen.

5.2 Voorwaarden voor migratie

Migratie is niet altijd zinvol of (technisch) haalbaar. De voorwaarden voor succesvolle migratie zijn:

- *een bestaand systeem* - Er moet een duidelijk afgebakende applicatie zijn die als geheel of in delen een rol moet gaan spelen in een nieuwe situatie. Om de juiste afwegingen te maken is nodig dat voldoende informatie beschikbaar is over het systeem. Het kan nodig zijn om door middel van 'reverse engineering' achter deze informatie te komen, indien die niet voorhanden is;
- *een helder doel* - Het moet eenduidig vastliggen waar het bestaande (te migreren) systeem in de nieuwe situatie aan moet voldoen. Om migratie mogelijk te maken moeten de architectuur van het SRW en de rollen die door softwarecomponenten gespeeld moeten worden, volstrekt helder zijn;
- *technische haalbaarheid* - Het moet technisch mogelijk zijn om de applicatie om te zetten naar de nieuwe situatie. Indien het onmogelijk blijkt, kan worden gedacht aan het opnieuw bouwen van de software. Dit laatste kan uiteraard alleen als de eisen aan de applicatie niet in strijd zijn met de eisen die door het SRW worden gesteld;
- *een voordeel* - De migratie van de software moet een voordeel opleveren. In de meeste gevallen zal vooral gekeken worden naar economische voordelen. Als het migreren van een systeem meer kost dan het opnieuw bouwen, is migratie niet erg zinvol.

Om te bepalen of migratie zinvol en voordelig is, moet een assessment worden uitgevoerd. Het Amerikaanse Ministerie van Defensie gebruikt een assessment-methode genaamd Software Reengineering Assessment. De methode is beschreven in het Software Reengineering Assessment Handbook (SRAH, [SRAH97]); met dit handboek kan worden vastgesteld welke migratie-strategie (of -strategieën) het meest van toepassing zijn. De methode gaat ervan uit dat nog niet duidelijk is óf er software gemigreerd moet worden . De verschillende strategieën zijn:

- herdocumenteren: het maken van documentatie over het systeem, waaronder gebruikersdocumentatie en commentaar in broncode;
- reverse engineering: het analyseren, begrijpen en abstraheren van het systeem tot een nieuwe vorm op een hoger abstractieniveau;
- vertalen van broncode: omzetten van broncode van de ene programmeertaal naar een andere;
- reengineering van gegevens: omzetten van gegevens van het ene formaat naar het andere;
- herstructureren: het veranderen van de representatie van het systeem naar een andere op hetzelfde abstractieniveau, waarbij de functionaliteit ongewijzigd blijft;

- retarget: het aanpassen van de software zodaning dat het in een nieuwe omgeving kan draaien - een ander besturingssysteem, een andere database, e.d.;
- redevelopment: het opnieuw ontwerpen en implementeren van het systeem;
- status quo: alles ongewijzigd laten.

De methode bestaat grofweg uit drie fasen: 'technical assessment' (paragraaf 5.3), 'economic assessment' (paragraaf 5.4) en 'management decision process'. De technical assessment is bedoeld voor het vaststellen van de technische haalbaarheid en de aanwezigheid van (technisch) voordeel van de migratie, de economic assessment is gericht op de economische aantrekkelijkheid en het management decision process dient ter ondersteuning van de uiteindelijke managementbeslissing. De methode is veelomvattend en heeft als nadeel dat veel gedetailleerde informatie over de organisatie voorhanden moet zijn (die er meestal niet is). In dit hoofdstuk 5 is een aantal overwegingen en beslissingscriteria overgenomen met aanvullingen uit het toepassingsdomein van het SRW; zie [SRAH97] voor een volledige beschrijving van de methode.

5.3 Technical assessment

In een technical assessment wordt vastgesteld of migratie haalbaar, uitvoerbaar en zinvol is op vooral technische gronden. Het resultaat van deze beoordeling is een lijst van applicaties of software-componenten die voor reengineering in aanmerking komen en de wijze waarop dat moet geschieden.

De eerste stap van de technical assessment is het vaststellen in hoeverre de organisatie (voor)bereid is om een migratietraject in te gaan. Hiervoor zijn antwoorden op de volgende vragen van belang:

- Is er ondersteuning uit het management voor reengineering?
- Hoe belangrijk is het reengineering project voor de organisatie?
- Is er financiële ondersteuning voor de inspanning?
- Is de reengineering in lijn met de bedrijfsstrategie?
- Is duidelijk wie het project gaat uitvoeren?
- Is vastgelegd hoe het beheer en onderhoud van de nieuwe software is georganiseerd?
- Is bekend hoe het succes van de migratie wordt bepaald?
- Is bekend welke middelen (tools, talen, etc.) gebruikt moeten worden?
- Is er voldoende kennis over de middelen beschikbaar?

In de volgende stap moeten eigenschappen van het te migreren systeem (of systemen) worden bepaald. De belangrijkste van deze eigenschappen zijn: leeftijd, complexiteit, programmeertaal, niveau van documentatie, beheersbaarheid, betrouwbaarheid, beperkingen in uitbreiding, het belang van het systeem (bijvoorbeeld voor klanten) en het platform waarop de software draait. Nadat eigenschappen van het systeem zijn vastgesteld kan nogmaals overwogen worden of migratie zinvol is of niet (migratie kan bijvoorbeeld zinvol zijn als de programmeertaal, het platform of de applicatie zelf sterk verouderd is, als er grote beperkingen voor uitbreidingen zijn of als het systeem niet meer te beheren is).

Naast de genoemde algemene kenmerken is voor toepassing in het SRW belangrijk om te weten of de software reeds geschikt is voor integratie met het SRW. In de rest van dit hoofdstuk 5 is aangenomen dat het te migreren systeem nog niet voldoet aan de eisen van het SRW. Daarnaast nemen we aan dat het een simulatieapplicatie betreft.

Als vaststaat dat het systeem moet migreren naar de nieuwe situatie kan worden vastgesteld welke strategieën van toepassing zijn. Hierbij wordt gekeken naar herdocumentatie , vertaling van broncode, reengineering van gegevens, retargeting en herstructurering. Software reengineering essessment beschrijft echter noodzakelijke acties en geen kant-en-klare oplossingen. In de context van het SRW zal meestal een combinatie van strategieën nodig zijn en dan met name:

- *reengineering van gegevens* - Het SRW stelt eisen aan het uitwisselingsformaat van gegevens. Daarnaast is het mogelijk dat de gegevens aangepast, aangevuld of gecombineerd moeten worden met een andere gegevensbron.
- *herstructurering* - Diverse generieke tools in het SRW bieden gebruikers toegang tot de modellen en resultaten; dit is functionaliteit die zeer waarschijnlijk ook in de applicatie al aanwezig is en die bij voorkeur daaruit verwijderd wordt. In de tweede plaats moeten modellen onderling samenwerken en moeten resultaten zoveel mogelijk op tijdstapbasis worden uitgewisseld, terwijl de meeste modelapplicaties zelf van begin- tot eindtijd doorrekenen. Simpel gezegd moet de 'tijdlus' uit de applicatie gehaald of instelbaar gemaakt worden.

Van de resterende strategieën zijn reverse engineering en redevelopment van belang. Indien uit de inventarisatie blijkt dat twee of meer van de bovengenoemde strategieën van toepassing zijn, kan het verstandig of zelfs noodzakelijk zijn om reverse engineering te gebruiken ter ondersteuning. Reverse engineering is echter een veel kostbaarder en complexer activiteit dan de vijf eerder genoemde.

Als de software op veel punten 'slecht' scoort en dus veel strategieën van toepassing zijn, is het opnieuw ontwikkelen (redevelopment) van de applicatie, of delen daarvan, het overwegen waard. In het SRAH [SRAH97] wordt geadviseerd om dit alleen te doen als de verwachte levensduur van de applicatie vijf jaar of langer is.

Nadat de toepasbare strategieën zijn vastgesteld, kan eventueel nauwkeuriger worden bekeken voor welke onderdelen ze van toepassing zijn. Het kan bijvoorbeeld voorkomen dat slechts enkele componenten of modules opnieuw ontwikkeld hoeven te worden.

5.4 Economic assessment

Het doel van een economic assessment is het vaststellen welke strategieën economisch gezien zinvol zijn. Software reengineering assessment biedt hiervoor een model waarvoor een groot aantal financiële details bekend moeten zijn, die meestal niet met grote nauwkeurigheid te achterhalen zijn. De belangrijkste overwegingen hebben betrekking op de kosten van de strategie in vergelijking met het onveranderd laten van de situatie. Met uitzondering van 'status quo' vragen alle strategieën tijd, geld en capaciteit. Te meer, daar het niet onwaarschijnlijk is dat tijdens het uitvoeren van de migratie de oude situatie ook tijdelijk gehandhaafd moet blijven.

Om te bepalen of een migratietraject economisch voordeel oplevert, moet op de een of andere manier ook een schatting gemaakt kunnen worden van de software in de nieuwe situatie. Aangezien één van de doelen van het SRW het faciliteren van samenwerking tussen instituten is, geldt dit, naast het verdienen van geld, ook als economisch voordeel.

Uitgebreide behandeling van de economic assessment valt buiten de scope van dit rapport. In de praktijk zullen de belangrijkste afwegingen zijn: hoeveel kost het om de bestaande programmatuur opnieuw te bouwen en hoeveel kost het om bestaande software te behouden

en te voorzien van een schil? Bij het maken van deze beslissing dient rekening gehouden te worden met toekomstige ontwikkeling. Als de makers van de modelapplicaties (onderzoekers en dergelijke) op de 'traditionele manier' moeten kunnen blijven werken, kan dat de beslissing beïnvloeden.

5.5 Voorbeelden van migratie

In de voorgaande paragrafen van dit hoofdstuk is aandacht besteed aan de te kiezen migratiestrategie. In de praktijk zal de vraag meestal zijn: bouwen we de applicatie opnieuw of niet? Als er niet voor nieuwbouw wordt gekozen, moet de bestaande software (inclusief de benodigde gegevensbronnen) worden ingepakt, zodanig dat alles in het SRW past. Hieronder wordt een aantal voorbeelden gegeven van migratietrajecten. Elk voorbeeld begint met een mogelijk uitgangssituatie. Er is niet geprobeerd om een uitputtende opsomming van situaties te geven.

Voorbeeld 1. Erfenis uit het verleden
De modelapplicatie is ontwikkeld op een oud platform en de broncode is in de loop van de jaren vele malen aangepast door verschillende personen. Bij de aanpassingen is verzuimd om deze nauwkeurig te documenteren; de onderhoudskosten zijn daardoor ook hoog. De inspanning die het kost om de applicatie geschikt te maken voor het SRW is (vrijwel) gelijk aan het opnieuw implementeren. Gegeven deze situatie wordt besloten om de applicatie opnieuw te bouwen. Daarbij wordt een 'modern' platform en een goed ondersteunende ontwikkelomgeving gekozen. Het SRW biedt een aantal bouwstenen en diensten die bij de ontwikkeling meegenomen kunnen worden - wat een besparing van inspanning oplevert.

Voorbeeld 2. Op de goede weg
Bij het bouwen van de modelapplicatie (niet zo lang geleden) is geprobeerd om die zo modulair mogelijk op te zetten. Gebruikersinterface en rekenhart zijn netjes van elkaar gescheiden; de impact van het kiezen van een andere gegevensopslag is zeer beperkt. In essentie passen de onderdelen van het model prima in het SRW, maar de interfaces wijken af. Verder worden hier en daar diensten door het SRW gevraagd (met name in zelfbeschrijvende operaties, de 'meta-informatie') die nog niet zijn voorzien. Vanwege de overeenkomsten wordt besloten tot gedeeltelijke herim-plementatie. Het is een betrekkelijk kleine moeite om de bestaande interfaces aan te passen en uit te breiden voor aansluiting met het SRW.

Voorbeeld 3. Een verpakking
De bestaande applicatie moet blijven bestaan. De belangrijkste reden is dat er gebruikers-groepen zijn die daarom vragen. Een deel van hen heeft het model geïncorporeerd in hun dagelijkse werk en zitten niet te wachten op een grote omschakeling. De organisatie die het model heeft gebouwd en geleverd zoekt echter aansluiting met het SRW. Om te voorkomen dat er twee parallelle ontwikkeltrajecten gaan ontstaan van dezelfde applicatie wordt besloten een schil te bouwen om het model. Op die manier kan het model in het SRW worden gebruikt, maar hoeft het model niet opnieuw te worden ontwikkeld. De koppeling met het SRW slaagt, maar er zijn concessies gedaan.

Er zijn verschillende manieren om migratie uit te voeren, die onder andere worden beïnvloed door de omgeving en organisatie waarbinnen de software gebruikt wordt, door de ervaring van de programmeurs, door de lange-termijndoelstellingen van de software en zo meer. In dit rapport worden hiervoor nog geen verdere aanbevelingen gedaan, maar in een later stadium verdient migratie zeker meer aandacht.

6. Aanbevelingen

6.1 Aansluiting bij technische ontwikkelingen

Alhoewel in dit rapport nog nauwelijks over technische implementatie gesproken is, zijn een aantal technische ontwikkelingen dermate belangrijk dat aansluiting daarbij onafwendbaar lijkt. De volgende ontwikkelingen (zouden) een rol moeten spelen bij realisatie van het SRW.

OpenGIS: Op het gebied van verwerking en manipulatie van geografische data is een industriestandaard aan het ontstaan, onder aanvoering van het OpenGIS Consortium. Grote software-leveranciers als Oracle (marktleider rdbms) en ESRI (marktleider GIS) hebben reeds OpenGIS compliant implementaties in hun pakketten verwerkt. De rol van OpenGIS bij de implementatie van geometrie en *structuren is cruciaal en noodzakelijk* voor toekomstige aansluiting bij 'de rest van de wereld'.

CORBA en DCOM: Distributie van componenten over meerdere platforms (al dan niet van hetzelfde type) vereist communicatie tussen die verschillende platforms en dus een methode om te weten waar een component aanwezig is en op welk platform die uitgevoerd kan worden. Industriestandaarden daarvoor zijn CORBA van de Object Management Group en DCOM van Microsoft. Alhoewel een keuze voor beide onlogisch is, maar een keuze tussen beide toch voeten in de aarde heeft, is het duidelijk dat de wijze van structurering van de run-time omgeving volgens de manier verloopt van deze zogenaamde 'middle ware solutions'.

XML: In het streven naar standaardisatie van documenten (in de breedste zin van het woord dus ook figuren, plaatjes en tabellen) hebben de ontwikkelingen rond het World Wide Web geleid tot de eXtended Mark-up Language XML. Dit is, net als het eenvoudige HTML, een afgeleide van de Standard Generalised Markup Language SGML. Het gebruik van XML biedt de mogelijkheid om op eenvoudige wijze data en de formaten daarvan in een structuur te vatten. XML wordt momenteel in de industrie als standaard erkend en de meeste grote software pakketten ondersteunen dit. Binnen het SRW kan XML als bestaand format meerwaarde bieden voor het op afstand koppelen met externe softwarepakketten.

6.2 Koppeling van componenten en tools door standaardisering

Een van de uitgangspunten van de ontwikkeling van het SRW is, dat het mogelijk is om willekeurige configuraties van beschreven modelcomponenten in een procesnetwerk te koppelen. De domeinmodellering en de uitwerking in de architectuur laat zien dat koppeling van willekeurige componenten uitsluitend werkt indien de uitwisseling gestandaardiseerd, zowel voor wat betreft het formaat als de semantiek. Modelcomponenten moeten van elkaar 'weten' wat de een voor de ander 'kan betekenen' en vice versa. Zonder standaardisering daarvan, hetgeen nog niet het geval is, wordt koppeling een moeizaam proces.

Dit wordt ondersteund door recente pogingen tot modelkoppelingen in legio verbanden of projecten die keer op keer een moeizaam proces opleveren met vele onvoorziene valkuilen. Wil modelkoppeling in het SRW tot een succes komen, dan dienen modelbouwers en -beheerders rekening te houden met de afgesproken standaarden, hetgeen consequenties heeft voor de interne organisatie van zo'n component. Hetzelfde geldt voor de generieke tools.

De afspraken zoals die bijvoorbeeld voor Adventus gemaakt zijn, waarbij al zoveel mogelijk uniformiteit in informatie nagestreefd wordt, zijn een goed begin. Echter, Adventus is momenteel slechts uitgewerkt voor 1D-systemen.

6.3 Domeinkennis inzetten voor realisatie

Realisatie van het SRW kan een aanvang krijgen op basis van dit rapport. Echter, het vereist kennis van het domein en de toepassingen om met behulp van de architectuurbeschrijving tot een goed raamwerk te komen. Uiteraard bestaat er de mogelijkheid dat zaken die nu nog onbekend zijn bij de ontwikkelaars en domeinexperts tot complicaties in de toekomst kunnen leiden. Echter, met de kennis die het projectteam heeft ten aanzien van bestaande systemen lijkt realisatie van een generiek raamwerk binnen handbereik.

7. Literatuur

[IGT97]

Werkgroep Generieke Tools, Inventarisatie Generieke Tools, Aquest, november 1997.

[GAMMA95]

Gamma, E., R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addion-Wesley, Reading, MA 1995

[GMP99]

Werkgroep Good Modelling Practice, Vloeiend Modelleren in het Waterbeheer Handboek Good Modelling Practice, 1999

[JACOBSON92]

Jacobson, I., M. Christerson, P. Jonsson, G. Övergaard, Object Oriented Software Engineering, A Use Case Driven Approach, Essex: Addison Wesley Longman Limited, 1992

[PvE98]

Werkgroep Generieke Tools, Programma van Eisen, Standaard Raamwerk Water, Aquest, 30-11-1998

[QUINT2]

Kwaliteit van softwareproducten - Praktijkervaringen met een kwaliteitsmodel, B. van Zeist et al., Kluwer Bedrijfswetenschappen, 1996

[SRAH97]

Software Reengineering Assessment Handbook, v3.0, 1997, Technical Report.

[WAL98]

Wal, T. van der, Werkplan Architectuur Standaard Raamwerk Water, SC-DLO, december 1998.

[ELSWIJK99]

Elswijk, M. van, Reviewverslag Architectuur Standaard Raamwerk Water, versie 1.2, SERC, 30 Augustus 1999.

[GROENENDIJK98]

Groenendijk, P., M. Bierkens, J. van Dam, M. van Elswijk, R. Lokers, E. Moors, T. Otjens, R. Smit en A.A. Veldhuizen, Domeinanalyse Framework Integraal Waterbeheer, SC Interne Mededeling 512, DLO Staring Centrum, 1998.

Appendixes

Appendix A. Begrippenlijst

Onderstaande begrippenlijst is ontleend aan 'Handboek Good Modelling Practice' [GMP99] en 'Inventarisatie Generieke Tools' [IGT97]. De lijst is aangevuld met een aantal in OO-kader vaak gebezigde IT-termen.

Begrip	*Term*	*Betekenis*
afhankelijke variabele	*dependent variable*	*variabele die ten opzichte van één of meer onafhankelijke variabele(n) verandert.*
algoritme	*algorithm*	*lijst van stappen om in een computerprogramma een probleem op te lossen.*
analytische elementenmethode	*analytic element method*	*methode voor grondwaterstromingsberekeningen gebaseerd op het superponeren van analytische oplossingen van de Poisson-vergelijking (voor gebiedsvariabele verticale stroming en voor de bergingsterm voor niet-stationariteit) die gelden binnen oneindige of begrensde gebieden in meerdere al-dan-niet-gekoppelde lagen.*
applicatie	*application*	*computerprogramma*
architectuur	*architecture*	*een abstractie van een systeem die de ontwerpstructuren en -relaties en de daarbij geldende regels of principes bevat, en die gebruikt is voor meerdere ontwerpen. Anders: de architectuur van een systeem bestaat uit de structuur van haar onderdelen, de aard en relevante extern zichtbare eigenschappen van deze onderdelen en de relaties en condities ertussen.*
associatie	*association*	*een structurele relatie tussen twee klassen.*
attribuut	*attribute*	*een element van een klasse dat informatie bevat die door die klasse beheerd wordt.*
bericht	*message*	*boodschap: een verzoek van een object aan een ander object om een operatie uit te voeren.*
betrouwbaarheids-analyse	*uncertainty analysis*	*activiteiten om de betrouwbaarheid van een model te schatten na afloop van de calibratie (en/of validatie).*
bouwsteen	*building block*	*zowel objecten als subsystemen vormen bouwstenen voor een applicatie.*
calibratie	*calibration*	*activiteiten om een vooraf bepaalde mate van overeenkomst tussen model en metingen in het veld te verkrijgen door het (systematisch) veranderen van onzekere factoren (vaak parameters), gevolgd door een analyse van de restfouten.*
case	*case*	*(deel van een) maatregel vertaald naar één simulatie (run) voor een model of modelinstrumentarium. het totaal aan parameters waarmee een bepaalde situatie wordt doorgerekend. In de case zitten zowel instellingen die de specifieke beheerder of beleidsmaker bepaalt als instellingen die voor hem extern zijn en waar hij geen invloed op kan uitoefenen (bv. zeespiegelrijzing).*

Begrip	Term	Betekenis
component	component	een samenhangend 'pakket' van software dat onafhankelijk als eenheid kan worden ontwikkeld en geleverd en dat ongewijzigd kan worden samengevoegd met andere componenten om een groter geheel te maken (bijvoorbeeld een applicatie, systeem).
conceptueel model	conceptual model	beschrijving van de structuur van een systeem met kwalitatieve afhankelijkheden.
constante	constant	grootheid die nauwkeurig bekend is.
data	data	gegevens.
data-assimilatie	data assimilation	benaderingswijze waarbij data wordt geintegreerd met een fysisch/chemische procesbeschrijving, zodanig dat de informatie-inhoud van zowel de data als de proces-beschrijving expliciet wordt gemaakt en gewogen.
deterministisch	deterministic	zonder toeval (i.t.t. stochastisch).
dimensie	dimension	1. lengte, breedte, hoogte. 2. (dimensie-analyse) eenheid waarin een grootheid wordt uitgedrukt.
dimensie-analyse	dimension analysis	test of in de modelvergelijkingen alle dimensies correct zijn.
discretisatie	discretisation	het ombouwen van een continue model (in ruimte en tijd) naar een model dat het systeem beschrijft in discrete (niet oneindig kleine) stappen in de ruimte en tijd.
doelfunctie	object function	kwantificering van de modelfout met behulp van veldmetingen.
domein	domain	wetenschapsgebied, in het onderhavige verband: deelgebieden van het waterbeheer.
dynamisch model	dynamic model	model waarbij de tijd een onafhankelijke variabele is.
eenheid	unit	gedefinieerde maat om een grootheid in uit te drukken of te meten.
eigenschap	property	een karakteristiek kenmerk van een object.
eindige-differentiemethode	finite difference method	transformatie van partiële differentiaalvergelijkingen die continu zijn in de ruimte, naar discrete differentievergelijkingen om ze zo numeriek op te kunnen lossen, bv. met behulp van een discreet 'grid' of raster (in 1D, 2D of 3D).
eindige-elementenmethode	finite element method	transformatie van partiële differentiaalvergelijkingen die continu zijn in de ruimte naar discrete differentievergelijkingen om ze zo numeriek op te kunnen lossen, bv. met behulp van discrete elementen, d.w.z. ruimtelijke compartimenten (in 1D, 2D of 3D).
encapsulatie	encapsulation	inkapseling: het verborgen blijven van hoe een object zijn taak uitvoert omdat de methoden en gegevens van het object zijn afgescheiden van en onzichtbaar voor de buitenwereld. Wat een object kan, is bekend. Hoe het object die taak uitvoert, blijft verborgen en heeft ook geen gevolgen voor de andere delen van het systeem.

Begrip	Term	Betekenis
energiebalans	energy budget	balans van energiestromen.
entiteit	entity	zelfstandige grootheid met een eigen betekenis; een onderwerp of een gebeurtenis, waarover informatie in een database is opgeslagen.
fuzzy logic model	fuzzy logic model	model met beschrijvingen op basis van vage logica, bijvoorbeeld met tussenvormen tussen ja en nee (misschien).
gebruikersinterface	user interface	deel van een applicatie dat belast is met de interactie tussen gebruiker en applicatie.
gegevenswoordenboek	data-dictionary	begrippenlijst met een gedetailleerde en eenduidige omschrijving van de begrippen. In een gegevenswoordenboek zal bv. staan dat de water-stand het gemiddelde is van de gemeten waterstand t.o.v. NAP over een periode van 10 minuten en gegeven is in meters. Een gegevenswoordenboek legt geen implementatie of keuze voor een database op, maar legt afspraken vast.
geldigheidsgebied	scope	het geheel van voorwaarden waaronder een model mag worden toegepast.
generieke tools	generic tools	componenten met algemene functionaliteiten die niet modelspecifiek zijn en daardoor breed inzetbaar zijn.
gevoeligheidsanalyse	sensitivity analysis	onderzoek naar de relatie tussen veranderende factoren (vaak parameters) en modeluitvoer.
globaalgedragtest	global behavior test	test of de globale werking van het model overeenkomt met de verwachting.
grafische gebruikers-interface	graphical user interface, GUI	gebruikersinterface die gebruik maakt van grafische (schematische, symbolische) presentatie voor interactie. Applicaties die zijn ontwikkeld voor Windows 95 of X-Windows bieden een grafische gebruikersinterface.
heuristische methode	heuristic method	niet formele methode om een niet precies bekend doel te bereiken op een onderzoekende en voortdurend evaluerende wijze volgens een bepaald criterium.
identificatie	identification	calibratie met als doel eenduidige waarden van alle parameters en andere calibratiefactoren te bepalen.
ijking	calibration	calibratie (zie aldaar).
instantie	instantiation	een object dat gecreëerd is uit een klasse.
integratie algoritme	integration algorithm	algoritme om (numeriek) differentiaalvergelijkingen op te lossen.
integreren	integrate	oplossen van differentiaalvergelijkingen.
interface	interface	het geheel van de diensten die een object of een component aan de buitenwereld kan verlenen. Bestaat uit (een deel van) de operaties van het object of de component.

Begrip	Term	Betekenis
interpretatie	interpretation	verklarende uitleg.
Jakobiaan	Jacobian matrix	matrix van partiële afgeleiden van de individuele residuen naar de (model) parameters.
klasse	class	Een beschrijving van het type van een verzameling van objecten met dezelfde karakteristieken en de toegestane bewerkingen op en de beperkingen met betrekking tot individuele objecten. (2) Bij het object-georiënteerd programmeren is dit een definitie van de interne variabelen en methoden voor een groep objecten van dezelfde soort. Een klasse bevat zowel gegevens als algoritmen die op die gegevens werken. Jacobsen: een sjabloon voor gelijksoortige objecten. Het sjabloon beschrijft de interne structuur (operaties en attributen) van de gelijksoortige objecten.
maatregel	measure, action	omschrijving van een ingreep om een bepaald resultaat/verandering te bewerkstelligen.
massabalans	mass balance	balans van stofstromen.
meta-informatie	meta information	gegevens over gegevens (waar zijn de gegevens, hoe en door wie gemeten, welke nauwkeurigheid, enzovoort).
methode	method	A method is a programmed procedure that is defined as part of a class and included in any object of that class. A class (and thus an object) can have more than one method. A method in an object can only have access to the data known to that object, which ensures data integrity among the set of objects in an application. A method can be re-used in multiple objects. Een implementatie van een service welke door een object geleverd wordt.
model	model	verzamelbegrip voor representaties van essentiële aspecten van een systeem, waarbij kennis gepresenteerd wordt in een bruikbare vorm. N.B: in het Handboek GMP wordt hiermee veelal bedoeld een programma in de computer (een model-programma) met bijbehorende invoer. Het woord 'model' kan echter ook slaan op wat opmerkingen op papier, een wiskundige formulering, een schema of figuur. IT: (vaak) abstracte beschrijving van software, veelal geïllustreerd met diagrammen.
modelfout	model error	afwijking van het resultaat van een model ten opzichte van de werkelijkheid.
modelleerproces	modelling process	alle stappen die doorlopen moeten of kunnen worden bij het maken en werken met modellen.
modelleren	model	maken van een model, werken met een model.
modelleur	modeller	1. de ontwikkelaar van een model. 2. iemand die werkt met een model
modelprogramma	model program	wiskundig model in de vorm van een computer-programma, bedoeld om modellen mee te bouwen door middel van het invoeren van gegevens.

Begrip	Term	Betekenis
modelproject	modelling project	project waarbij het werken met een model een belangrijke rol speelt.
modelprojectformulier	model project form	formulier van het Handboek GMP om het model-project zo volledig mogelijk te beschrijven.
module	module	A module is a collection of dependent components such as an object or some looser collection of procedures and functions. A module encapsulates related components. [Sommerville].
modulair programmeren	modular programming	een algemene manier van programmeren, waarbij programma's worden opgedeeld in modules, die elk eigen procedures en gegevens bevatten. De bedoeling is om modules te maken die zo min mogelijk afhankelijk van elkaar zijn.
neuraal netwerk model	neural network model	model waarbij de relaties tussen input en output worden beschreven m.b.v. van een netwerk van knopen die ieder een gewicht hebben, zodanig dat het neurale netwerk bij een bekende input een bekende output oplevert.
object	object	de instantie van een klasse. Een object heeft een aantal operaties en een toestand (gerepresenteerd door de attributen) die het effect van de afzonderlijke operaties vasthoudt. (Jacobson).
onafhankelijke variabele	independent variable	variabele ten opzichte waarvan de veranderingen in een dynamisch systeem worden beschreven; voorbeelden: tijd, drie ruimtelijke dimensies.
operatie	operation	een beschrijving van (een deel) van het gedrag van een klasse. Wordt geïmplementeerd door een algoritme.
overerving	inheritance	principe dat bepaalt dat elke subklasse alle kenmerken van zijn superklasse heeft.
overige variabele	auxiliary variable	variabele waarvan de waarde niet afhangt van de waarde op een vorige waarde van de onafhankelijke variabele (dus bijvoorbeeld niet van de waarde op een vorig tijdstip).
parameter	parameter	grootheid die constant wordt verondersteld, maar die niet nauwkeurig bekend is.
partiële differentiaal-vergelijking	partial differential equation	differentiaalvergelijking met meer dan één onafhankelijke variabele.
probleembeschrijving	problem definition	een heldere, precieze (niet noodzakelijk kwantitatieve) specificatie van wat bekend is over het probleem en van wat berekend moet worden.
programmeur	programmer	iemand die computerprogramma's maakt of verandert. N.B: sommige modelleurs programmeren ook zelf (programmerende modelleurs), maar meestal is dit niet het geval.
raamwerk	framework	een raamwerk is een verzameling geprefabriceerde software-bouwstenen die programmeurs kunnen gebruiken, uitbreiden of aanpassen voor specifieke computeroplossingen (applicaties). OO framework: een raamwerk is een verzameling klassen die tesamen een abstract ontwerp vormen voor een familie van gerelateerde problemen (applicaties).

Begrip	Term	Betekenis
rekenkern	computational kernel	het gedeelte van een simulatieprogramma dat belast is met het uitvoeren van de berekeningen.
residu	residual	restfout.
restfout	residual	verschillen tussen de uitkomsten van een gecalibreerd model en veldmetingen.
restfoutenanalyse		(statistische) analyse van restfouten.
robuustheidtest	robustness test	test of het model bestand is tegen extreme invoergegevens.
scenario	scenario	combinatie van een stategie en één of meer autonome ontwikkelingen (droog jaar, etc).
schematisatie (modelschematisatie)	schematization	Een (ruimtelijk) geschematiseerde weergave van de werkelijkheid uitgedrukt in de voor een model relevante grootheden. Voor ééndimensionale waterkwantiteits-modellen is dit veelal een netwerk van knopen en takken, voor tweedimensionale waterkwantiteitsmodellen een (geografisch) grid, voor emissiemodellen districten, voor ééndimensionale grondwatermodellen bodemlagen, etc.
semi-stationair	semi-stationary	hiervan wordt aangenomen dat het stationair is.
simuleren	simulate	nabootsen m.b.v. een model (gedachtenmodel, fysiek model, model op de computer).
soft-hybride model	soft-hybrid model	data-georiënteerd model (bijvoorbeeld een neuraal netwerk) waarin (bijvoorbeeld via de calibratie) fysische concepten worden meegenomen.
stabiliteit	stability	eigenschap van differentiaalvergelijking en/of integratiemethode, waarbij de fout in elke integratiestap kleiner wordt.
standaard invoer	standard input	de invoer van een standaard test (bijvoorbeeld uit de handleiding van het model-programma of een eigen eenvoudige case), waarvan de bijbehorende uitvoer bekend is.
stationair (statisch) model	stationary (static) model	model dat niet dynamisch is; de veranderingen in de tijd worden niet onderzocht.
statistische verdeling	statistical distribution	kansverdeling van een steekproef.
stochastisch	stochastic	met toeval.
strategie	strategy	set van samenhangende maatregelen (zie aldaar) om een bepaald resultaat te bereiken.
(sub-)systeem	subsystem	groepering van objecten (of subsystemen) die als geheel weer als een object kan worden opgevat.
systeem	system	een geheel (vaak een deel van de werkelijkheid) bestaande uit entiteiten, waartussen relaties bestaan.

Begrip	Term	Betekenis
systeembeschrijving	*system definition*	*een tekstuele beschrijving van een systeem.*
toestandsvariabele	*state variable*	*variabele waarvan de waarde afhangt van zichzelf op een eerder punt in de tijd of ruimte.*
tool	*tool*	*algemeen: hulpmiddel.* *IT: Applicatie die dient als hulpmiddel; geautomatiseerd hulpmiddel.*
type	*type*	*een klasse wordt ook wel aangeduid als het type van een object.*
validatie	*validation*	*vergelijken van modeluitvoer met een onafhankelijke (dat wil zeggen nog niet in de calibratie gebruikte) set meetgegevens, teneinde te kunnen vaststellen of het model 'goed' is (of het concept goed is, het model het verleden kan reproduceren met de vereiste nauwkeurigheid en of het model geschikt is om alle vragen te beantwoorden).*
variabele	*variable*	*grootheid die verandert.*
veldwaarneming	*observation*	*veldmeting, dus waarneming aan het systeem waar het model een representatie van is.*
verificatie	*verification*	*controle of het wiskundige model correct geïmplementeerd is in de computer.*
wiskundig model	*mathematical model*	*de wiskundige vertaling van het conceptuele model.*

Appendix B. Data opslag en verwerking

Deze appendix beschrijft het mogelijke gebruik van de stekkerdoos in samenwerking met het SRW. De appendix is als discussiestuk opgenomen.

B.1 Inleiding

In dit document wordt voor vier soorten gegevensoverdracht in een SRW-applicatie beschreven:
- Wat de stand van zaken is;
- Wat mogelijke knel- of aandachtspunten zijn;
- Wat ons inziens de meest voor de hand liggende oplossingsrichting is.

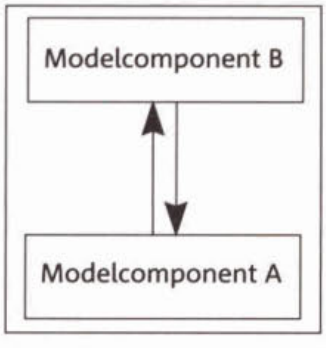
De vier soorten gegevensoverdracht zijn:
- 'On-line' gegevensoverdracht;
- Gegevensoverdracht d.m.v. bestanden voor gebruik binnen meerdere componenten;
- Gegevensoverdracht via import en export vanuit SRW;
- Gegevensoverdracht via bestanden voor intern gebruik binnen één component.

Het document heeft de status van discussiestuk.

B.2 'On-line' gegevensoverdracht

Stand van zaken

Onder 'on-line' gegevensoverdracht wordt hier de rechtstreekse gegevensoverdracht (zonder tussentijdse fysieke opslag) tussen de componenten in een SRW applicatie bedoelt. Deze aanpak vereist standaardisatie op het niveau van "calls". Met andere woorden, er moet worden vastgelegd hoe een component A aan een component B bijvoorbeeld een waterstand vraagt. De algemene syntax van dergelijke calls is in de huidige architectuurbeschrijving goed opgenomen.



Figuur B-1
'on-line' gegevens-overdracht tussen twee componenten

Mogelijke knel- of aandachtspunt

Naast hoe een component A aan een component B bijvoorbeeld een waterstand vraagt, moeten component A en B het ook met elkaar eens zijn over wat er wordt geleverd. Als model A voor een waterstand geheugenruimte reserveert, en bovendien uitgaat van bepaalde eenheden, dan zal model B dit in deze vorm moeten leveren. Het aspect wat er moet worden geleverd moet zodoende tot op het niveau van implementatie gespecificeerd worden.

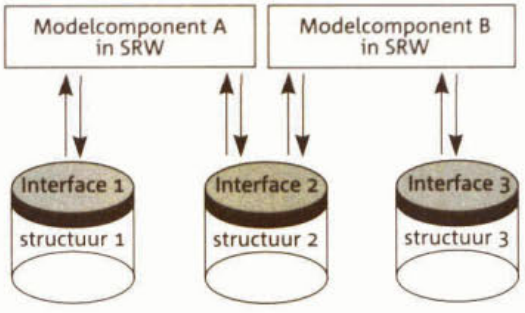
Oplossingsrichting

Conform het 'programma van eisen' worden de definities en beschrijvingen gebaseerd op het gegevenswoordenboek Adventus zijn. Voor alle gegevens (bijvoorbeeld 'waterstand') moet een gestandaardiseerde set "calls" worden vastgesteld die het mogelijk maakt dit data type te bevragen en bewerken.

B.3 Gegevensoverdracht d.m.v. bestanden voor gebruik binnen meerdere componenten

Stand van zaken

Een component kan ook gegevens halen uit een database via een interface. Met "database" wordt in deze context zowel een DBMS als een set van bij elkaar behorende bestanden bedoeld. Door de database via een interface te benaderen is deze op een uniforme wijze toegankelijk. De communicatie tussen modelcomponenten en interfaces is vergelijkbaar met de communicatie tussen componenten en binnen het standaard raamwerk gestandaardiseerd. Aangezien de interface geen deel uitmaakt van de component behoeft een component geen kennis te bezitten over de onderliggende fysieke opslagstructuur noch de wijze waarop deze moet worden benaderd. Een dergelijke database kan door meerdere componenten (binnen of buiten het SRW) gebruikt worden.



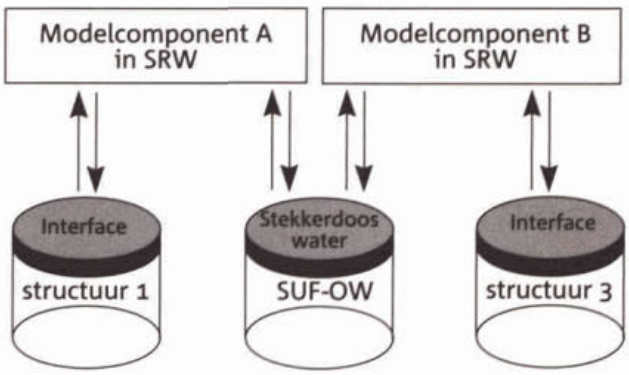
Figuur B-2 Gegevensoverdracht tussen modelcomponenten met behulp van databestanden

Mogelijk knel- of aandachtspunt

Vanuit het standaard raamwerk bezien is het aantal relevante (mogelijke) formaten van zowel databases als bestanden te groot om volledige ondersteuning te bieden.

Oplossingsrichting

In principe geldt dat het ontwikkelen van een passende interface de verantwoordelijkheid is van de leverancier van de database. Echter, voor de talloze kleine databestanden waarvoor het ontwikkelen van een interface niet haalbaar is, is het zinvol een interface naar een standaard uitwisselingsformaat (suf) vanuit (elke component van) het SRW te ondersteunen. Een dergelijke interface is in de vorm van de 'stekkerdoos water' al grotendeels beschikbaar. Het standaard uitwisselingsformaat behorende bij de stekkerdoos water is het SUF-OW. De stekkerdoos is Adventus conform. Zodoende sluit deze oplossingsrichting aan bij de oplossingsrichting genoemd onder 'On-line' gegevensoverdracht.

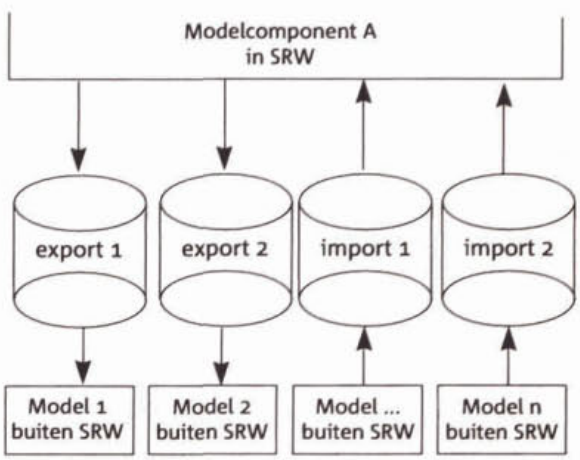


Figuur B-3
Gegevensoverdracht tussen modelcomponenten met gebruik van SUF-OW

B.4 Gegevensoverdracht via import en export vanuit SRW

Beschrijving
Gegevensstructuren opgeslagen in bestanden afkomstig van applicaties buiten het SRW moeten kunnen worden geïmporteerd door een SRW applicatie. Ook zal een SRW applicatie zelf gegevens voor "extern" gebruik beschikbaar willen kunnen stellen.

Mogelijk knel- of aandachtspunt
Bij afwezigheid van een gestandaardiseerde fysieke opslagstructuur is het niet mogelijk een algemeen toepasbare interface voor bestanden beschikbaar te stellen. Concreet: de modelcomponent in het SRW moet verschillende export- en importformaten kunnen lezen, afhankelijk van de gewenste koppelingen met de buitenwereld. Er zullen dus op ad-hoc basis aanpassingen aan componenten moeten worden uitgevoerd. Dit ad-hoc aanpassen van componenten voor koppelingen met niet SRW-modellen komt sterk overeen met de huidige manier van het koppelen van modellen. Bij deze vorm van modelkoppelingen is kennis nodig over de fysieke opslagstructuur en de toegang hiertoe, maar ook meer "inhoudelijke" kennis m.b.t. de gegevens in het bestand (Zijn bijvoorbeeld de eenheden "Adventus conform"?).
Dit in tegenstelling tot de situatie zoals beschreven in 'bestanden voor gebruik binnen meerdere componenten'.



Figuur B-4
Gegevensoverdracht tussen modelcomponenten in SRW en modelcomponenten buiten SRW

Oplossingen

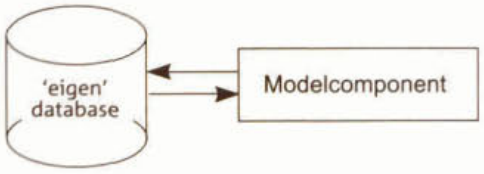
In paragraaf B-3, 'Gegevensoverdracht d.m.v. bestanden voor gebruik binnen meerdere componenten', is als oplossingsrichting de SUF-OW standaard voor de dataopslag en de stekkerdoos water als bijbehorende interface beschreven. Deze structuur kan ook als structuur voor import en export naar externe modellen gebruikt worden. Dit houdt in dat:

- als de oplossingssuggestie voor 'bestanden voor gebruik binnen meerdere componenten' wordt uitgevoerd, geen verdere aanpassingen binnen het raamwerk nodig zijn voor communicatie met modellen buiten het raamwerk.
- modellen buiten het raamwerk het SUF-OW formaat moeten kunnen ondersteunen. (Noot: dit zou in dit verband ook beschouwd kunnen worden als een eerste stap voor migratie)

B.5 Gegevensoverdracht via bestanden voor intern gebruik binnen één component

Beschrijving

Bij een component kunnen "eigen" gegevens behoren. Bedoeld worden gegevens die alleen voor de component zelf van belang zijn en niet behoeven te worden uitgewisseld en/of gedeeld met andere componenten. In het geval van een model kunnen dit bijvoorbeeld instellingen voor model specifieke parameters zijn of tijdelijke opslag van gegevens. Binnen de component is kennis aanwezig over de fysieke opslagstructuur van en toegang tot deze gegevens.



Figuur B-5
Gegevens voor een modelcomponent voor 'eigen' gebruik

Mogelijk knel- of aandachtspunt

Indien de gegevens die in de 'eigen' database staan niet interessant zijn voor andere toepassingen is er van een probleem geen sprake. Het kan echter voorkomen dat gegevens die in de 'eigen' database staan alsnog van belang zijn (of worden) voor andere componenten of, bijvoorbeeld in het geval van beschrijvingen van schematisaties, voor een presentatietool toegankelijk moeten zijn. Indien hiermee geen rekening gehouden wordt zal in vermoedelijk veel van dergelijke gevallen een specifieke tool moeten worden ontwikkeld, die de 'eigen' database toegankelijk maakt. Het is echter vooralsnog niet te overzien hoeveel van de data in 'interne' databases voor extern gebruik toegankelijk moet worden gemaakt.

Oplossingsrichting

Naar aller verwachting moeten bij de migratie van een applicatie naar een SRW-toepassing aanpassingen aan de broncode worden uitgevoerd. Evaluatie van het mogelijke belang van de 'eigen' databases voor andere componenten zou parallel uitgevoerd kunnen worden. Op basis van deze evaluatie kunnen aanpassingen worden uitgevoerd zodat waar noodzakelijk data ook voor andere componenten toegankelijk zijn. In navolging van de oplossingsrichting bij eerder genoemde knelpunten kan overwogen worden ook de interne dataopslag te standaardiseren (voor nieuw te ontwikkelen componenten). Een SUF-OW gegevensopslag ligt voor de hand indien al met betrekking tot de andere genoemde knelpunten voor dit formaat is gekozen.

B.6 Tot slot

In het onderhavig rapport is uitgebreid op de 'on-line' gegevensoverdracht ingegaan. Specificatie van wat er precies wordt gevraagd en geleverd (bijvoorbeeld eenheden) zal de vervolgfase (bouw van het prototype) aan bod komen. Andere vormen van gegevensoverdracht (via bestanden) komen binnen het SRW niet aan bod. Dergelijke vormen van gegevensoverdracht zullen wel optreden. Dientengevolge is consensus onder alle partijen over dit onderwerp gewenst.
In principe zijn alle vormen van gegevensopslag in het standaard raamwerk toegankelijk, mits een geschikte interface beschikbaar is. Het is niet strikt noodzakelijk dat één of meerdere dataformaten door het SRW standaard worden ondersteunt. Met ondersteunen wordt hier bedoeld dat de stekkerdoos en het SUF-OW formaat 'up to date' worden gehouden, zodat ook nieuwe typen data via de stekkerdoos kunnen worden benaderd. Echter, indien een dergelijke ondersteuning niet wordt bewerkstelligd zal dit (mogelijk) leiden tot:

1. Beperkte mogelijkheden voor het gebruik van (kleine) specialistische databestanden, waar de kosten voor het schrijven en bijhouden van een geschikte interface mogelijk niet opwegen tegen het veranderen van de databasestructuur naar een volledig ondersteunde structuur;
2. Ontoegankelijkheid van geëxporteerde resultaten voor verdere verwerking;
3. Ontoegankelijkheid van componenten 'interne' databestanden voor andere componenten;
4. Geen efficiency verbetering ten opzichte van de huidige situatie van grote aantallen specifieke interfaces en dataformaten.

Om bovenstaande gevaren het hoofd te bieden en een heldere vorm van gegevensoverdracht binnen en buiten het raamwerk te bewerkstelligen lijkt de volgende oplossing geschikt:

1. De ondersteuning van ten minste één gegevensformaat en bijbehorende interface vanuit het SRW;
2. Het (zoveel mogelijk) standaardiseren van import en export bestanden naar dit formaat;
3. Het (zoveel mogelijk) standaardiseren bestanden voor de overdracht van gegevens binnen het SRW naar dit formaat.

Indien consensus over standaardisatie bestaat, ligt voor de hand de stekkerdoos water interface/SUF-OW combinatie vanuit het SRW te ondersteunen: deze wordt inmiddels breed gedragen en veelvuldig gebruikt. Voor typen data die op geen enkele wijze in de stekkerdoos water interface / SUF-OW combinatie kunnen worden opgenomen worden zal een aparte standaard interface/gegevensformaat combinatie moeten worden gekozen respectievelijk ontwikkeld.

Appendix C. OpenGIS

C.1 Inleiding

Het OpenGIS Consortium is een wereldwijd consortium waarvan de leden komen uit alle delen van de wereld en uit diverse segmenten (overheid, universiteiten, bedrijfsleven). Alle grote GIS leveranciers (o.a. ESRI, Intergraph, SmallWorld, Oracle, MapInfo, Microsoft) zijn aangesloten bij het OGC.

De uitwisseling van gegevens tussen verschillende GIS systemen verliep in het verleden en verloopt nog steeds vaak erg moeizaam. Er bestaat een grote hoeveelheid bestandsformaten met verschillende opslagmethoden (bijvoorbeeld met topologie of zonder topologie). Verder worden er verschillende operating systemen gebruikt. En ten slotte worden voor het opslaan van dezelfde gegevens vaak verschillende concepten gebruikt (wegen kunnen b.v. als één of als twee lijnen worden opgeslagen). Het OGC is opgericht om orde te scheppen in deze chaos. Het OGC definieert de OpenGIS standaarden. Deze standaarden moeten er voor zorgen dat verschillende GIS systemen met elkaar gegevens uit kunnen wisselen. Die GIS systemen kunnen klein of groot zijn, van verschillende GIS leveranciers komen en op verschillende plaatsen in de wereld staan, verbonden door een netwerk.

C.2 Onderwerpen

OpenGIS heeft een veertiental onderwerpen gedefinieerd waarvoor standaarden ontwikkeld moeten worden. Deze onderwerpen zijn:

1. Geometrie van ruimtelijke objecten (feature geometry)
2. Projecties (spatial reference systems)
3. Geometrie van locaties (locational geometry)
4. Functies en interpolatie (stored functions and interpolation)
5. Ruimtelijke objecten (the OpenGIS feature and feature collections)
6. Ruimtelijke dekking (the coverage type)
7. Aard observatie (earth imagery)
8. Relaties tussen objecten (relations between features)
9. Kwaliteit (quality)
10. Overdracht technologie (transfer technology)
11. Metagegevens (metadata)
12. Service architectuur (the OpenGIS service architecture)
13. De Catalogus (catalog services)
14. Semantiek en informatie gemeenschappen (semantics en information communities).

De OpenGIS standaarden zijn nog niet af. Van de meeste van bovenstaande onderwerpen bestaan tot dusver slechts abstracte modellen. Om een systeem volgens de standaarden te maken zijn zogenaamde implementatie specificaties nodig. Deze implementatie specificaties zijn op dit moment (juni 1999) alleen voor onderwerp 5 (ruimtelijke objecten) gereed. De laatste hand wordt gelegd aan de specificaties voor een aantal andere onderwerpen (o.a. 6, 13).

Appendix D. Gedistribueerde objecten

(De onderstaande tekst is afkomstig uit een white paper van SERC en is bedoeld als achtergrondinformatie.)

D.1 Inleiding

Medio 1997 hebben gebruikers van informatietechnologie (IT) nog altijd te kampen met hoge ontwikkel- en onderhoudskosten. Het snel invoeren van nieuwe produkten en diensten wordt bemoeilijkt door de bestaande informatiesystemen. Ook het beschikbaar stellen van de juiste gegevens op het juiste tijdstip en de juiste plaats vergt een grote inspanning.

Deze problematiek wordt voor een deel veroorzaakt door de architectuur van de informatiesystemen. De ondersteuning van de organisatie door IT bestaat meestal uit een aantal monolithische systemen. Elk van deze systemen ondersteunt een specifiek deel van de organisatie. De presentatie-, functie- en gegevenslogica in deze systemen zijn meestal verweven. Daarnaast zijn de systemen ongestructureerd geprogrammeerd en moet kennis over de code telkens opnieuw worden opgebouwd. Ten slotte zijn er in de loop der jaren vele ad hoc koppelingen tussen de systemen gebouwd om gegevens te kunnen uitwisselen. Bij wijzigingen moeten ook deze koppelingen worden aangepast.

Dergelijke IT-oplossingen zijn een zware last voor organisaties. Wijzigingen in de systemen vergen grote inspanningen en kosten derhalve veel geld. De slagvaardigheid en flexibiliteit van de organisatie worden bedreigd. Er is behoefte aan een flexibele, onderhoudbare informatievoorziening die optimale ondersteuning biedt aan de continu veranderende omgeving. Het opsplitsen van de brei informatiesystemen in logische functionele brokken die op een gestandaardiseerde wijze met elkaar communiceren helpt bij het oplossen van de geschetste problematiek.

In dit artikel gaan we in op de verschillende aspecten van gedistribueerde informatieverwerking. We laten zien hoe informatiesystemen opgesplitst kunnen worden in samenwerkende componenten en beschouwen de twee communicatiestandaards: CORBA en DCOM. Ook kijken we naar een technologie die voor een belangrijk deel heeft bijgedragen aan de ontwikkeling van gedistribueerde objecten: compound documents. Ten slotte besteden we aandacht aan de hindernissen die genomen moeten worden om een gedistribueerde informatieverwerking op te kunnen zetten.

D.2 Gedistribueerde objecten

Veel bedrijven trachten bovenstaande problematiek op te lossen met behulp van traditionele client/server-technologie (C/S). De gegevenslogica wordt ondergebracht in relationele database servers (bijvoorbeeld Oracle of Sybase). De presentatielogica wordt ondergebracht in clients die met behulp van 4GL-tools (bijvoorbeeld PowerBuilder of Visual Basic) worden gerealiseerd. De functielogica wordt deels in de clients en deels in de servers ondergebracht. Dergelijke C/S-architecturen bieden slechts gedeeltelijk een oplossing voor de problematiek. Zo zijn de gegevens-, functie- en presentatielogica nog altijd onvoldoende gescheiden hetgeen de

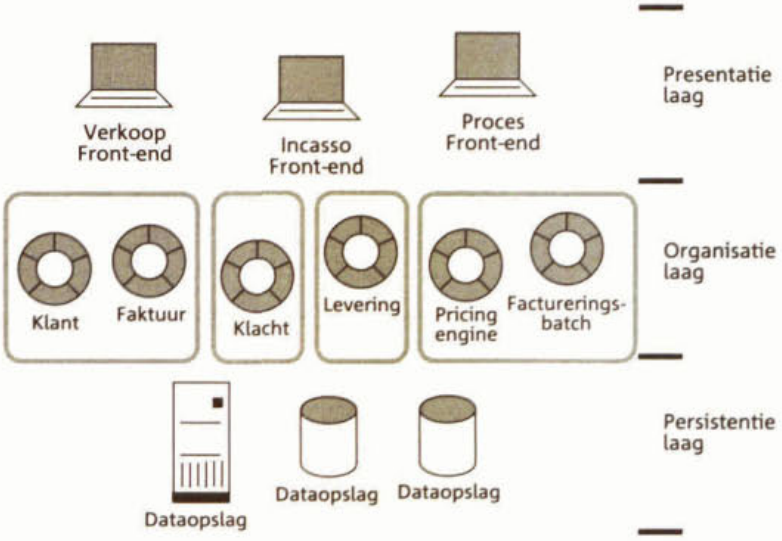
flexibiliteit van het systeem niet ten goede komt. Daarnaast is men afhankelijk van de leverancier: 4GL-tools hebben allen een eigen scripting language. Datzelfde geldt ook voor de stored procedures van de relationele databases.

Gedistribueerde objecttechnologie gaat verder dan de traditionele C/S-benadering. Het opdelen van een informatiesysteem in componenten geschiedt op basis van object-oriëntatie (OO). De ideeën achter OO zijn al ruim twintig jaar oud. Een groot draagvlak voor het OO-gedachtegoed is echter pas de laatste jaren ontstaan doordat het dankzij de voortschrijdende techniek, mogelijk is geworden de ideeën te realiseren. De benodigde krachtige machines, computernetwerken, besturingssystemen en standaards zijn beschikbaar.

Voordat we ingaan op gedistribueerde objecten kijken we eerst naar objecten zelf. De twee belangrijkste begrippen van OO zijn encapsulatie en overerving. Een object biedt diensten aan die door andere objecten kunnen worden gebruikt. De interne werking van een object is voor andere objecten afgeschermd, alleen de diensten zijn zichtbaar. Dit wordt encapsulatie genoemd. Een voorbeeld van een object is het object "Klant". De diensten van dit object zijn bijvoorbeeld "Wijzigen adres" en "Bepalen kredietwaardigheid".

Objecten kunnen zijn afgeleid van andere objecten. In OO-terminologie wordt dit overerving genoemd. Overerving houdt in dat alle diensten van het bronobject worden geërfd. Zo kunnen de objecten "Zakelijke klant" en "Particuliere klant" worden afgeleid van het object "Klant". Beide afgeleide objecten kennen dan ook de dienst "Bepalen kredietwaardigheid". Het afleidingsmechanisme laat toe dat beide objecten deze dienst op hun eigen manier implementeren terwijl de buitenwereld blijft uitgaan van die ene dienst "Bepalen kredietwaardigheid".

Gewone objecten bestaan alleen binnen programma's. De diensten van deze objecten kunnen alleen binnen het betreffende programma worden gebruikt. Buiten het programma is geen toegang tot de objecten. Een gedistribueerd object is een object dat wél kan samenwerken met andere objecten buiten het eigen programma. Een gedistribueerd object kan zelfs samenwerken met objecten op andere machines ongeacht besturingssysteem, netwerk, programmeertaal en hardware. Een gedistribueerd object is als een onafhankelijk pakket verpakt en beschikbaar voor alle aangesloten andere objecten. Het kan zowel als client als als server optreden. Figuur D-1 geeft een voorbeeld van een verzameling samenwerkende gedistribueerde objecten.



Figuur D-1 *Schematische illustratie van een architectuur van gedistribueerde objecten*

De gedistribueerde objecten in figuur D-1 zijn ingedeeld in drie categorieën. De middelste laag, de organisatielaag bevat de belangrijkste objecten. Deze objecten vormen tezamen een afbeelding van de objecten waar de organisatie mee te maken heeft. De afgeronde rechthoeken op de organisatielaag symboliseren machines. De objecten "Klant" en "Factuur" worden dus door dezelfde machine beschikbaar gesteld.

De presentatielaag bestaat uit drie gedistribueerde objecten. Elk van deze objecten implementeert een gebruikersinterface voor een specifieke gebruikersgroep. Zo ontsluit de "Verkoop Front-end" de objecten uit de organisatielaag op een zodanige wijze dat de verkoopmedewerkers hun werkzaamheden optimaal kunnen uitvoeren. Zij kunnen bijvoorbeeld inzicht krijgen in de leveringen geleverd aan een klant en de nog openstaande facturen.

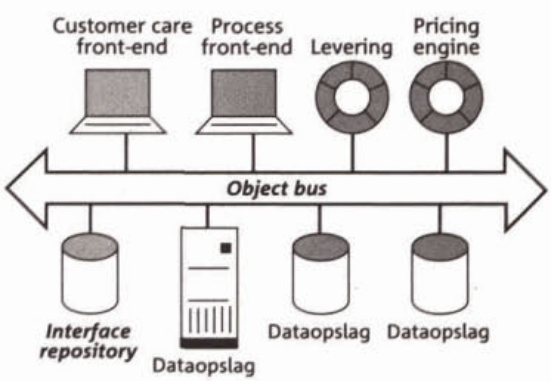
De persistentielaag bevat ten slotte gedistribueerde objecten die de opslag verzorgen van de verschillende organisatielaagobjecten. Dit zijn bijvoorbeeld object-georiënteerde of relationele databases.

Een informatie-architectuur opgebouwd uit gedistribueerde objecten is bijzonder flexibel. Objecten kunnen worden gewijzigd en toegevoegd zonder dat dit gevolgen heeft voor andere objecten; nieuwe produkten en diensten kunnen makkelijker worden gerealiseerd. Daarnaast is de oplossing schaalbaar: als er capaciteitsproblemen zijn kan er een extra machine worden ingezet. Wanneer de objecten organisatiebreed beschikbaar zijn, heeft iedereen in de organisatie dezelfde gegevens voorhanden. Ook kunnen diensten op andere manieren ontsloten worden. In het voorbeeld van figuur D-1 zouden bestellingen door toevoeging van één object op de presentatielaag ook via het World-Wide Web kunnen worden geplaatst.

Gedistribueerde objecten kunnen met elkaar communiceren zonder dat ze van elkaar weten waar ze zich bevinden, in welke taal ze geschreven zijn, etcetera: communicatie tussen gedistribueerde objecten is volledig transparant. Om deze transparantie te realiseren zijn communicatiestandaards nodig. Hieronder gaan we daar dieper op in.

D.3 Communicatiestandaards, CORBA en DCOM

Een communicatiestandaard biedt infrastructurele voorzieningen zodat objecten onafhankelijk van hardware, besturingssystemen, programmeertalen en netwerkprotocollen met elkaar kunnen communiceren. Dergelijke standaards worden ook wel object middleware genoemd. In figuur D-2 is de algemene oplossing schematisch weergegeven.



Figuur D-2 *Schematische illustratie van een infrastructuur voor gedistribueerde objecten*

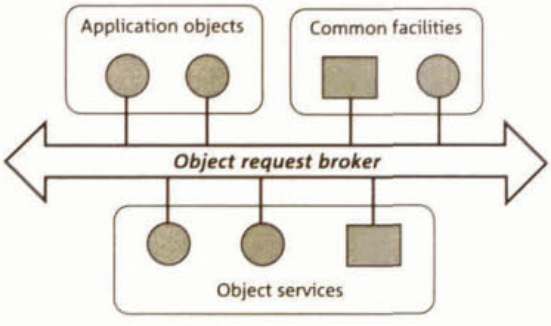
De oplossing bestaat uit twee delen. Ten eerste is er een intermediair nodig die de communicatie tussen de verschillende objecten verzorgd. Ten tweede moet er een gemeenschappelijke taal zijn waarmee de verschillende objecten beschreven kunnen worden.

De intermediair treedt op als een soort makelaar. Hij redigeert vragen van objecten naar andere objecten en verzorgt de terugkoppeling van het antwoord. Deze makelaar wordt ook wel een object bus genoemd, analoog aan de adres- en databus in een computer. De diensten die objecten aanbieden zijn beschreven in een gemeenschappelijke taal, een zogenaamde interface definition language (IDL). Ieder object in de infrastructuur is beschreven in IDL. Middels een zogenaamde interface repository houdt de object bus de beschrijvingen van de diverse objecten bij en kan hij vraag en antwoord afstemmen.

Er zijn op dit moment twee standaards die invulling geven aan bovenstaande infrastructuur: CORBA van OMG en DCOM van Microsoft. Hieronder beschouwen we beide standaards.

CORBA

CORBA staat voor Common Object Request Broker Architecture. Deze communicatiestandaard wordt ontwikkeld door de Object Management Group (OMG). OMG heeft in 1994 de specificaties van versie 2.0 vastgesteld. De eerste CORBA 2.0 implementaties zijn in het najaar van 1996 uitgekomen. OMG is in mei 1989 opgericht door acht bedrijven, te weten: 3Com, American Airlines, Canon, Data General, Hewlett-Packard, Philips Telecommunications, Sun Microsystems and Unisys. Sinds oktober 1989 opereert OMG als onafhankelijke non-profit instelling. Medio 1996 zijn meer dan 600 bedrijven en instellingen uit de softwarebranche lid van OMG.



Figuur D-3 Schematische weergave van de samenhang tussen de verschillende componenten van CORBA

De Object Request Broker (ORB) is de object bus van CORBA. CORBA objecten kunnen tijdens de uitvoering aan de ORB vragen welke diensten beschikbaar zijn en daar vervolgens gebruik van maken. Een ORB beschikt over voorzieningen voor beveiliging van het gebruik van diensten en dataverkeer. Een ORB kan zowel stand-alone draaien als in een netwerk met andere ORB's.
De Object Services zijn verzamelingen elementaire diensten verpakt als gedistribueerde objecten. Ze kunnen gezien worden als een uitbreiding op de diensten van een ORB. OMG heeft op dit moment standaards gedefinieerd voor dertien Object Services. Enkele belangrijke zijn:

- *Life Cycle Service* - voor het creëren, kopiëren, verplaatsen en verwijderen van gedistribueerde objecten.
- *Persistence Service* - voor het persistent opslaan van objecten in opslagmedia. Er is ondersteuning voor object-georiënteerde databases, relationele databases en eenvoudige bestanden.

- *Naming Service* - hierdoor kunnen objecten aan elkaar refereren met behulp van een naam.
- *Concurrency Control Service* - met behulp van deze service kan voorkomen worden dat een object vanuit twee verschillende objecten wordt gewijzigd.
- *Event Service* - middels deze service kunnen objecten "run-time" hun interesse aan- en afmelden voor het optreden van specifieke gebeurtenissen.
- *Licensing Service* - voor het registreren van het gebruik van objecten door andere objecten. Verschillende wijzen van het berekenen van het gebruik kunnen worden gekozen.

De Application Objects zijn de objecten die de organisatie modelleren en die de gebruikers bij hun werkzaamheden ondersteunen. Deze objecten worden ook wel aangeduid met de term business objects. Application Objects maken gebruik van de Object Services.

Om de ontwikkeling van application objects te vereenvoudigen kent CORBA de zogenaamde Common Facilities. Dit zijn voorgedefinieerde raamwerken van samenwerkende objecten die kant-en-klaar ingezet kunnen worden in specifieke situaties. Common Facilities kunnen vergeleken worden met de insteekkaarten die voor personal computers beschikbaar zijn. Zoals de insteekkaart op de adres- en databus wordt ingeprikt, zo wordt een Common Facility op de object bus ingeprikt.

Er zijn twee categorieën Common Facilities: horizontale en verticale. De horizontale bieden algemene, domeinonafhankelijke ondersteuning. Hierbij kan gedacht worden aan gebruikersinterface-objecten, informatiemanagementobjecten en objecten ter ondersteuning van werkstroombesturing, e-mail en dergelijke. De verticale Common Facilities omvatten objecten voor de ondersteuning van specifieke bedrijfstakken, bijvoorbeeld banken (denk aan objecten voor kredieten en spaarhypotheken), telecommunicatie, gezondheidszorg, procesindustrie, etcetera.

CORBA is een open standaard: er kan gekozen worden uit verschillende leveranciers. De belangrijkste zijn: BEA systems met ObjectBroker, Expersoft met XShell, VisiBroker van Visigenics, Iona met Orbix en Sun met Distributed Objects Everywhere (DOE).

DCOM

DCOM is Microsofts communicatiestandaard voor gedistribueerde objecten. DCOM is gebaseerd op COM, welke geëvolueerd is uit OLE. OLE staat voor Object Linking and Embedding en werd in 1990 door Microsoft geïntroduceerd als een technologie om data van verschillende applicaties (bijvoorbeeld plaatjes en spreadsheets) in één document te kunnen integreren. Verderop zullen we nader ingaan op deze technologie.

In 1993 heeft Microsoft een verbeterde versie van OLE uitgebracht, gebaseerd op een object bus. Deze object bus kreeg de naam COM (Component Object Model) mee. Windows 3.1 en Windows 95 zijn beide gebouwd rond COM. OLE-objecten (maar ook bijvoorbeeld ActiveX objecten) communiceren via COM met elkaar. COM is een object bus die werkt op één machine. Microsoft zal in de volgende versie van Windows (gepland voor 1998) een gedistribueerde versie van COM leveren: DCOM. DCOM is beschikbaar voor Windows NT 4.0. Voor Windows 95 kan je een upgrade downloaden van de website van Microsoft. De strategie van Microsoft is om via de Windows desktop een groot aandeel te krijgen in de markt voor gedistribueerde objecten.

Een OLE-object kan via COM tijdens de uitvoering informatie verkrijgen over interfaces van andere OLE-objecten. Een OLE-object kan meerdere interfaces hebben. Deze interfaces worden met COM's IDL beschreven. Daarnaast kunnen interfaces met ODL (Object Description Language)

aan de Type Library (de interface repository van Microsoft) worden opgegeven. COM kent voorzieningen voor licensing, events, life cycle management en dergelijke. Via COM kunnen ten slotte ook Windowsdiensten als bestandsbeheer en geheugen allocatie worden aangeroepen.

D.4 Compound documents, OpenDoc en OLE

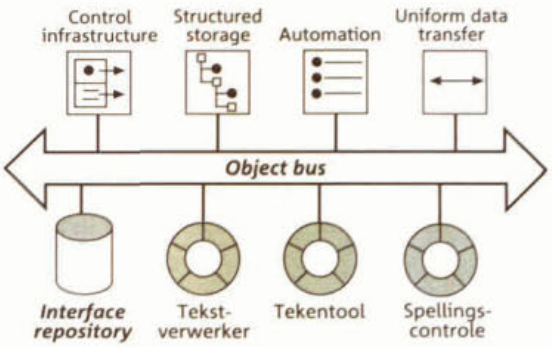
Ideeën over gedistribueerde objecten leefden al langer. Ook in het kamp van de desktopontwikkelaars is hierover nagedacht. Om te kunnen meten met de concurrentie werd de functionaliteit van desktopapplicaties (tekstverwerkers, spreadsheets, tekenpakketten en dergelijke) steeds verder uitgebreid. De desktopapplicaties werden grote monolithische applicaties. Ter illustratie: Microsoft Excel vereiste in 1990 nog 4Mb ruimte op de harde schijf, vandaag de dag is dat 16Mb geworden.

Deze groei heeft negatieve gevolgen voor de ontwikkeling van desktopapplicaties. Kostte de upgrade van WordPerfect versie 3 naar 4 nog 14 manjaar, de upgrade van 5 naar 6 kostte niet minder dan 250 manjaar. Ook voor de gebruiker heeft de groeiende functionaliteit een keerzijde: de bediening wordt steeds complexer. Dat terwijl gewone gebruikers slechts tien procent van de geboden functionaliteit benutten.

Om de ontwikkelingen van desktopapplicaties te blijven beheersen, is een aantal techno-logieën uitgedacht onder de verzamelnaam compound documents. Hieronder bespreken we de verschillende technologieën en beide implementaties: OpenDoc en OLE.

Compound documents

De compound documents technologie omvat twee ideeën: enerzijds de mogelijkheid verschillende typen data in één document op te kunnen nemen, anderzijds de mogelijkheid desktopapplicaties uit onafhankelijke delen (componenten) te kunnen laten bestaan.
Een compound document is een document dat meerdere typen data kan bevatten. Een document kan bijvoorbeeld tekst en plaatjes bevatten. Een control infrastructure zorgt ervoor dat de verschillende typen data correct op het scherm verschijnen en dat de gebruiker afhankelijk van de positie van de cursor de juiste bewerkingscommando's kan aanroepen. Als de cursor bijvoorbeeld boven een plaatje staat, verandert de menubalk in die van het bijbehorende tekenpakket.



Figuur D-4
Schematische weergave van Compound Documents

Om de verschillende typen data gestructureerd op te kunnen slaan, is er een structured storage mechanisme uitgedacht. Dit mechanisme gaat uit van containers en parts. Containers zijn samengestelde documenten: zij kunnen parts en andere containers bevatten. Parts zijn

elementaire data-objecten: bijvoorbeeld een stuk tekst, een tekening of een spreadsheet. Het mechanisme laat toe dat containers containers bevatten: in een tekstdocument opgenomen plaatje kan weer een stuk tekst bevatten, etcetera.

Er zijn twee technieken voor het opnemen van data-objecten: linking en embedding. Bij linking wordt een verwijzing naar het object opgenomen, bij embedding wordt het object zelf opgenomen. Voordeel van links is dat wanneer een object in meerdere documenten is opgenomen, in één keer alle documenten kunnen worden aangepast. Nadeel is dat links kunnen verbreken wanneer een gelinked object wordt verplaatst.

De compound documents technologie voorziet in een scripting language "over compo-nenten heen": automation. Basisoperaties van de verschillende componenten kunnen via een macrotaal worden aangeroepen. Hiermee kan bijvoorbeeld automatisch een grafiek van de kwartaalcijfers in een rapport worden opgenomen. De geschreven macro haalt de juiste gegevens op uit de database component, maakt met behulp van de plaatjes component een grafiek en meldt deze aan bij de tekstverwerker component.

Uniform data transfer zorgt ervoor dat de verschillende typen data op uniforme wijze kunnen worden uitgewisseld. Naast het knippen en plakken naar en van het klembord, wordt ook drag-and-drop ondersteund: objecten kunnen in andere documenten worden opgenomen door ze met de muis naar het andere document te slepen.

In plaats van monolithische applicaties bestaan desktopapplicaties uit kleine stukken software voor het leveren van een specifieke functionaliteit. Bijvoorbeeld een component om een plaatje te tekenen, een component om tekst te verwerken en een component om spelling te controleren. Om deze verschillende componenten met elkaar te kunnen laten communiceren bevat compound documents ook het idee van de object bus.

Zowel aan de gebruikers- als aan de ontwikkelzijde kan met behulp van de compound documents technologie een produktiviteitsverhoging worden gerealiseerd. Gebruikers kunnen eenvoudiger gegevens integreren en hebben voor dezelfde gegevens eenzelfde interface. Ontwikkelaars bouwen geen monolithische applicaties meer maar ontwikkelen of selecteren kleine componenten die ieder een specifieke rol vervullen.

Net als bij de object middleware kan ook bij compound documents gekozen worden uit twee standaards: OpenDoc en OLE.

OpenDoc
OpenDoc is de standaard die ontwikkeld is en wordt door Component Integration Laboratories (CI Labs). CI Labs is in 1993 opgericht door Apple, IBM, Novell, Oracle, Taligent SunSoft, WordPerfect en Xerox. Zoals de naam al doet vermoeden is OpenDoc een open standaard. CI Labs kent medio 1996 maar liefst 2.100 leden. Dit zijn voornamelijk softwarebedrijven die OpenDoc-componenten ontwikkelen.

Dankzij samenwerking tussen OMG en CI Labs sluiten CORBA en OpenDoc op elkaar aan. De object bus van OpenDoc voldoet aan de CORBA-standaard. Het Common Facilities framework van CORBA voor grafische gebruikersinterface-objecten is de OpenDoc-standaard.
OpenDoc is op dit moment beschikbaar voor OS/2 en Apple Macintosh. Voor Windows 95 en NT wordt dit najaar een beta versie verwacht.

Om te voorkomen dat OpenDoc-objecten volledig geïsoleerd zijn van OLE-objecten heeft Novell ComponentGlue ontwikkeld. ComponentGlue zorgt ervoor dat OpenDoc-objecten met OLE-objecten kunnen communiceren en vice versa.

OLE
Zoals beschreven in de voorgaande paragraaf is OLE de compound documents implementatie van Microsoft. OLE is beschikbaar voor Windows en Apple Macintosh. Microsoft Word 6.0 liet de eerste OLE-softwarecomponenten zien waaronder de spellingscontrole en de vergelijkingseditor. Met behulp van OLE kunnen deze componenten ook vanuit andere pakketten worden gebruikt.

Van de buitenkant bieden OLE en DCOM dezelfde functionaliteit als OpenDoc en CORBA. De verschillen zitten in de onderliggende filosofie en technologie. OpenDoc en CORBA zijn gebaseerd op het OO-gedachtegoed: encapsulatie en overerving zijn standaard. OLE en DCOM ondersteunen alleen encapsulatie. Daarnaast heeft CORBA een technologische voorsprong van minimaal twee jaar. De leden van OMG zijn sinds 1989 veel problemen, die bij gedistribueerde objecten een rol spelen, tegengekomen en hebben deze opgelost.

Alhoewel CORBA en OpenDoc technisch superieur zijn aan DCOM en OLE gaat het te ver om te zeggen dat Microsoft de strijd om de standaards voor gedistribueerde objecten zal verliezen. Sterker nog: Microsoft heeft op dit moment een aandeel van 85 procent in de desktopmarkt. Dit aandeel is een goede springplank om straks met het uitbrengen van Network OLE een substantieel aandeel te veroveren in de servermarkt waar nu CORBA domineert.

D.5 Drempels voor de invoering van gedistribueerde objecten

Hoewel een organisatie vanuit technologisch perspectief de informatievoorziening met gedistribueerde objecten kan realiseren, is er vanuit andere invalshoeken nog een aantal hindernissen te nemen. Zo vormt het kennisniveau van de eigen en externe automatiseringsdeskundigen een belemmering. Kennis rond OO is schaars, kennis rond gedistribueerde objecten nog schaarser. Het opbouwen van de juiste kennis vereist een aanzienlijke inspanning.

Daarnaast is de inrichting van de automatiseringsorganisatie niet altijd afgestemd op het ontwikkelen en beheren van gedistribueerde informatiesystemen. Ten eerste is het belangrijk om de omvang van projectteams klein te houden. Teams hebben dan niet de capaciteit om het wiel opnieuw uit te vinden maar zullen bestaande componenten moeten herbruiken. Daarnaast zouden projectteams kunnen worden beloond op basis van de hoeveelheid opgeleverde herbruikbare componenten. Een apart team zou de IT-ontwikkelingen moeten coördineren opdat architecturen zoals die van figuur D-1 kunnen ontstaan. Voor ieder domein in een organisatie zou er een dergelijk team moeten zijn. Ten slotte zou een overkoepelende organisatie verantwoordelijk moeten zijn voor de algehele regelgeving en algemene faciliteiten voor gedistribueerde objecten.

De korte-termijn visie van het management is een ander obstakel op de weg naar een flexibele architectuur. Men wil hier en nu resultaten. De extra investeringen die voor het realiseren van een gedistribueerde informatievoorziening nodig zijn, zullen pas in een later stadium hun vruchten afwerpen. Dit is moeilijk te verkopen.

Ten slotte geldt dat gedistribueerde objecten nog geen proven technology zijn. We staan aan het begin van een nieuw automatiseringstijdperk. Het aantal uitgevoerde projecten op dit gebied is nog bijzonder klein, daarnaast zijn de standaards nog in ontwikkeling. Het zal waarschijnlijk nog één of twee jaar duren voordat er een goed fundament is gelegd dat ook de grote massa over de streep kan trekken. Het is echter helder dat gedistribueerde objecten de oplossing vormen voor de problematiek van de hedendaagse automatiseringspraktijk. Organisaties doen er verstandig aan zich nu al vast voor te bereiden op de komende veranderingen.

Appendix E. Extended ISO Model of Software Quality

Het Extended ISO Model of Software Quality wordt in detail in [Quint2] beschreven. Hieronder volgt een opsomming van de indicatoren die aansluiten bij de in dit document genoemde kwaliteitseigenschappen. Deze zijn overgenomen uit [Quint2].

Operability
Attributes of software that bear on the users' effort for operation and operation control.

Indicators for operability:

1. *expert judgement on operability:* The extent to which the software product presents functionality to the user without hindrance, as judged by a team of experts in this field.
2. *operability compared with sample:* The operability of the software product is compared to the operability of a predetermined sample product.
3. *operability in practice:* The extent to which the software product presents functionality to the user without hindrance, as judged by users after a period of use.
4. *set-up installation time:* The sum of time required for the installation process, including preparations, execution of environment set-up and verification.
5. *set-up installation procedures with human interaction:* The number of steps for set-up installation operation which require human interaction (for everyday use).
6. *availability of set-up performance:* The ratio of number of set-up performances available and the total number of performances: availability of set-up installation restart, availability of set-up installation confirmation functions.
7. *ease of set-up:* The average time required to set up the software or system.
8. *availability of set-up installation restart:* The number of points at which users can make a pause and from which the installation operation can be restarted.
9. *availability of set-up installation preparation:* The number of types and amount of computer resources including system, hardware, software and personnel to be employed for installation.
10. *availability of set-up installation confirmation:* The number of steps for installation confirmation operation which are necessary to validate that the software or system has been successfully installed and is ready to be used adequately.
11. *default value availability ratio:* The ratio of operating commands having default values to the total number of operating commands.
12. *command uniformity:* The proportion of operating commands having uniform formats, which are based on common-sense and comprehensible rules.
13. *consistency of terms in message:* The proportion of system message terms that are standardised.
14. *message clearness:* The proportion of system messages from software or system in which causes and corresponding action are clearly identified by the user who received those messages.
15. *skill level adaptability:* The proportion of functions for which operating methods can be selected to correspond to the user's level of skill.
16. *uniformity of screen manipulating operations:* The proportion of types of screen-manipulating operations using common basic conventions or patterns.
17. *stability of input/output areas on display:* The proportion of input/output screen formats designed with standardised formats in which the position and form of input/output fields are commonly laid out.

18. *number of keystrokes:* The number of keystrokes of operation required by the user to carry out the work. Those keystrokes include strokes of key click, button click, screen touch, pen move, mouse move, etc.
19. *availability of reduced effort for repeated operations:* The ratio of strokes required to repeat operation to the strokes required for the first operation to perform a specific task.
20. *mean time between human error operations:* Average time interval between one operation error (human error operation) and the next.
21. *ultimate operation time:* The time required for the operation, that is ultimately reduced and cannot be reduced anymore by further improvements.
22. *time for shutdown operation:* The average time between the input of the system shutdown command at the beginning of shutdown operation and the time when shutdown is completed.
23. *guide function availability ratio:* The ratio of available guide functions to the required ones for a given set of functions.
24. *human error operation cancellability ratio:* The ratio of command/data entries that can be cancelled to the total command/data entries.
25. *ability to emphasise expressions:* The ratio of number of actually implemented means to required ones which are provided to emphasise expressions to the user for a given set of functions. It is considered that colour, sound, brightness, and animation are means to emphasise expressions.
26. *response time for user:* The elapsed time from the user input request or command to the nearest response to that request by the software or system. The first response to the request may be a state or progress report expressing that software or system is processing the request. At least mean, minimum and maximum time should be measured.
27. *display time:* The elapsed time from the current display on screen after request to change, to the next complete display on screen. At least mean, minimum and maximum time should be measured.

Understandability

Attributes of software that bear on the users' effort for recognising the logical concept and its applicability.

Indicators for understandability:

1. *rated understandability:* The understandability of the instructions, menus, commands, pictograms, icons, help information, instructions, manuals, etc., of the software product as rated by the user.
2. *readability score:* Rating of the readability of the software product (on-screen messages, documents, pictograms, etc.).
3. *concept clearness:* The proportion of functions that can be explained by using clear, familiar models to illustrate concepts. This represents the degree to which the functions and conventions of a software product are explained through models using familiar concepts from the everyday world.
4. *availability of demonstration software:* The proportion of functions presented to the users through demonstration software.
5. *usage clearness:* The ratio of functions explained or by using clear models or presented to the user through demonstration software or anyway described.
6. *availability of input/output data items list:* For functions with input or output operations, the number of input/output items that are listed.
7. *recognisability of modifiable parameters:* The proportion of parameters that are identified as being either modified or fixed.

Helpfulness
Attributes of software that bear on the availability of instructions for the user on how to interact with it.

Indicators for helpfulness:
1. *ratio of expounding text:* The ratio of the amount of expounding text (including error messages) available in the software product to the total amount of text which can be presented on screen.
2. *normalised ratio of expounding text:* The ratio of the amount of expounding text (including error messages) available in the software product to the size of the software product.

Installability
Attributes of software that bear on the effort needed to install the software in a specified environment.

Indicators for installability:
1. *installation effort:* The installation effort in man-months.
2. *parameter change ratio:* Parameter correction ratio which is to be changed by transferring the software.
3. *recompile program ratio:* Program ratio which is to be recompiled by transferring the software.
4. *file change ratio:* File correction ratio which is to be changed by transferring the software.
5. *output list change ratio:* Output list correction ratio which is to be changed by transferring the software.

Adaptability
Attributes of software that bear on the opportunity for its adaptation to different specified environments without applying other actions or means than those provided for this purpose for the software in question.

Indicators for adaptability:
1. *effort for portability:* Mean effort needed to adapt a unit volume of the software product to a different specified platform.
2. *applicable ratio of hardware environment:* Applicable ratio of hardware environment without changing the software.
3. *applicable ratio of OS environment:* Applicable ratio of operating system environment without changing the software.
4. *applicable ratio of data environment:* Applicable ratio of data environment without changing the software.
5. *applicable ratio of operation environment:* Applicable ratio of operation manual and procedure without changing the software.

Compliance
Attributes of software that make the software adhere to application related standards, conventions or regulations in laws and similar prescriptions.

Indicators for compliance:
1. *standardised data format ratio:* The ratio of standardised data formats to the data formats to be standardised.
2. *standardised medium format ratio:* The ratio of standardised medium formats to the medium formats to be standardised.

3. *standardised character ratio:* The ratio of standardised graphic characters and control characters to those to be standardised.
4. *standardised interface ratio:* The ratio of standardised interfaces to the interfaces to be standardised.
5. *expression conformance ratio:* The ratio of functions that have specific expressions or methods of computation matched to the rules or standards.

User-friendliness
Attributes of software that bear on the users' satisfaction.

Indicators for user-friendliness:
1. *rated user-friendliness:* Ratio of users that prefer the new software product over the previous, after a certain period of practical use.
2. *expert judgement on user-friendliness:* User-friendliness, as judged by a team of experts on topics as: screen composition, vocabulary, application of colour and sound.
3. *user-friendliness compared to sample:* User-friendliness compared to a sample product, as judged by a team of experts. The experts decide to what extent user-friendliness of the software product matches the sample product.

Clarity
Attributes of software that bear on the clarity of making the user aware of the functions it can perform.

Indicators for clarity:
1. *function recognition ratio:* Ratio of functions that an average unpractised user distinguishes in the software product without help, within a period of time.
2. *function use ratio:* Ratio of functions of the software product that an average user actually employs after a certain period of time.

Customisability
Attributes of software that enable the software to be customised by the user to reduce the effort required for use and increase satisfaction with the software.

Indicators for customisability
1. *configurability ratio:* The ratio of parts of functionality that can be changed without changing the code to the total number of parts of functionality.
2. *configurability effort:* The effort required to change the functionality of the software product.

Attractivity
Attributes of software that bear on the satisfaction of latent user desires and preferences, through services, behaviour and presentation beyond actual demand.

Indicators for attractivity:
1. *user judgement on attractivity:* The equipment level of the software product, especially the availability of additional services, behaviour and presentation, as judged by the user.

Time behaviour
Attributes of software that bear on response and processing times and on throughput rates inperforming its function.

Indicators for time behaviour:

1. *batch turnaround time:* The time that passes between start and finish of background processing.
2. *batch capacity:* Number of information items which can be processed sequentially during background processing.
3. *processing time:* The average and maximum time a user needs for a certain processing task, with a certain usage load.
4. *processing capacity:* Number of processing tasks of a certain type that the user can perform during a certain period with a certain usage load.
5. *average internal transaction time:* Average time a certain internal processing task occupies with a certain usage load.
6. *maximum internal transaction time:* Maximum time a certain internal processing task occupies with a certain usage load.
7. *turnaround time:* Indicates the speed of processing by measuring the elapsed time between the beginning of process requirement and gaining the result of the process.
8. *response time:* Indicates the speed of processing by measuring the elapsed time between the end of inquiry or request to the computer system and the start of response.
9. *CPU elapsed time:* Indicates the speed of processing by measuring the elapsed time between the start of program execution and the end of program execution.
10. *CPU execution time:* Indicates the speed of processing by measuring the CPU time between the start of program execution and the end of program execution.
11. *I/O processing time:* The time required for I/O operation between main memory and external storage.
12. *waiting time:* The time between program stop by the interruption of O/S or resources, and program restart.
13. *network processing time:* Indicates the speed of network processing by measuring the time from the beginning of data transmission to the end of the transmission between host computer and terminal.
14. *terminal processing time:* The time between the beginning of terminal processing and the end of terminal processing.
15. *throughput:* Indicates the ability of the system by measuring the amount of jobs processed in a unit of time.
16. *number of processed transactions:* Indicates the ability of the system by measuring the number of transactions done in a unit of time.

Interoperability

Attributes of software that bear on its ability to interact with specified systems.

Indicators for interoperability:

1. *effort per interaction:* The effort needed to realise interoperability per unity of size of interoperability.
2. *matched data format ratio:* The ratio of data formats matched to those of the other system in the interoperation.
3. *matched character ratio:* The ratio of graphic characters and control characters matched to those of the other systems in interoperation.
4. *matched interface ratio:* The ratio of interfaces matched to those of the other systems in interoperation.
5. *observed standard ratio:* The ratio of observed standards to the standards made in the system or among the system.

Changeability
Attributes of software that bear on the effort needed for modification, fault removal or for environmental change.

Indicators for changeability:
1. *modification effort per unit volume:* Average amount of effort needed to modify the software product, per unit volume of the modification.
2. *correction effort per defect:* Mean effort needed to repair a defect in the software product.
3. *mean fault correction time:* Mean time from receiving the failure report to sending the corrected software in fault correction.
4. *mean failure treatment time:* Mean time from the failure occurrence to the restoration for end users.
5. *mean fault correction work time:* Mean work time for correcting the discovered software fault to be corrected in fault correction, in consequence of analysing the failure.
6. *mean revise work time per changed line of source code:* Mean work time for revision per updated and added source code line.
7. *fulfilment degree of maintenance document:* Fulfilment degree of product documents that are usable in the maintenance stage.

Traceability
Attributes of software that bear on the effort needed to verify correctness of data processing on required points.

Note: if the required point is only set at the end of the programming phase, this characteristic is essentially the same as accuracy.

Indicators for traceability:
1. *operation control effort:* The amount of time that is lost while processing due to operation control activities, manually or automatically.
2. *ease of operation control:* The amount of effort needed to perform the operation control.

Accuracy
Attributes of software that bear on the provision of right or agreed results or effects.

Indicators for accuracy:
1. *failure ratio:* The ratio of incorrect processed transactions to the total of presented transactions.
2. *significant digits ratio:* The ratio of the implemented significant digits to the required significant digits.
3. *manual conformance ratio:* The ratio of functions implemented and the matching product to the functions written in the user's manuals.
4. *rounding treatment ratio:* The ratio of functions with the required rounding treatment to the total number of implemented functions.

Reusability
Attributes of software that bear on its potential for complete or partial reuse in another softwareproduct.

Indicators for reusability:

1. *ratio reusable parts:* Ratio of reusable parts of the software product to the total number of parts of the software product.
2. *ratio reused parts:* Ratio of reused parts of the software product to the total number of parts of the software product.

Stability

Attributes of software that bear on the risk of unexpected effect of modifications.

Indicator of stability:

1. *Ratio of new faults at revision:* The ratio of new faults made at the revision.

Manageability

Attributes of software that bear on the effort needed to (re)establish its running status.

Indicators for manageability:

1. *Control effort ratio:* The ratio of effort put in controlling the software product (including maintenance) in man-hours to the number of hours the product is available to the users.

Appendix F. Veelgestelde vragen

In deze appendix wordt een aantal vragen beantwoord die tijdens de review en tijdens de workshop gesteld zijn. De antwoorden zijn hier opgenomen om verduidelijkingen te geven op een aantal punten die voor de verdere uitvoering van belang zijn.

Wat is het belang van de domein-analyse?
De domein-analyse is een onderdeel van de gehanteerde methode om tot een architectuur ontwerp te komen. Het resultaat daarvan is het domeinmodel: een vereenvoudigde, abstracte afspiegeling van de werkelijkheid. Hier is het domein omschreven als 'simuleren in het integraal waterbeheer'. De domein analyse heeft drie stellingen opgeleverd:

1. Het domein 'integraal waterbeheer' valt op te delen in sub-domeinen; Sub-domeinen zijn 'koppelbaar' wanneer ze elkaar aanvullen, elkaars 'taal' spreken en niet overlappen. Hetzelfde geldt voor modellen in het integraal waterbeheer;
2. Het domein 'integraal waterbeheer' laat zich voor simulaties en modelstudies opdelen in modelelementen. Ieder modelelement is verantwoordelijk voor een stukje informatie;
3. Systemen voor simulatie in het integraal waterbeheer bestaan uit modelapplicaties (ook wel rekenkernen) en generieke tools, zoals een visualisatietool en een dataeditor.

Het belang van de domein analyse is verder dat we kunnen concluderen dat een algemeen raamwerk dat modelprogramma's koppelt geen oplossing biedt: Het is immers niet de technische koppeling maar veelal de semantische koppeling die moeilijkheden veroorzaakt. Het SRW biedt een oplossing voor modellen die dezelfde semantiek hanteren (modelelementen).

Is er ook aandacht voor performance?
Bij de ontwikkeling van de architectuur voor het SRW is met name naar ontwerp- en bouw-performance gekeken: Het SRW biedt een versnelling van het ontwikkelen en bouwen van modelprogramma's. Ten aanzien van runtime performance is al geconcludeerd dat het SRW daar niet toe bijdraagt, maar ook geen beperkingen oplegt. Runtime performance is een kwestie van tuning. Het SRW biedt een aantal mogelijkheden om de runtime performance te verbeteren. Allereerst kunnen complexe problemen als geheel opgelost worden. Verder kan de systeem-configuratie aangepast worden. Bijvoorbeeld het weghalen van een in-line visualisatietool kan de performance verhogen. Aspecten als caching van data of efficiente oplossingsalgoritmen zijn denkbaar.

Hoe wordt in het SRW omgegaan met gegevens?
Een belangrijk aspect van de architectuur is de wens/eis dat rekenkernen (modelapplicaties) zoveel mogelijk onafhankelijk zijn van de opslag van de gegevens. Alle SRW-componenten, dus zowel modelapplicaties als generieke tools, kennen de modelelementen en weten daar mee om te gaan. Modelelementen zijn daarbij verantwoordelijk voor het opvragen en bewaren van gegevens. Hierbij kan natuurlijk gebruik gemaakt worden van standaards, zoals de stekkerdoos, of een opslag in een relationeel databasemanagement systeem.

Hoe wordt in het SRW omgegaan met bestaande modelprogramma's?
De kans is klein dat bestaande modelprogramma's één-op-één in het SRW opgenomen kunnen worden. Daarom zal per modelprogramma een migratiestrategie bepaald moeten worden. Dit kan variëren van het maken van een simpele 'wrapper' tot volledige herbouw. Een wrapper

voorziet in de koppeling tussen de eisen van het SRW en de specificaties van een bestaand modelprogramma. Wanneer bestaande modelprogramma's hun eigen datafiles nodig hebben, moet zo'n wrapper voorzien in de data-uitwisseling tussen SRW en die files. Het gaat natuurlijk niet alleen om de technische verplaatsing van gegevens, maar ook om de semantische afstemming. Modelprogramma's die uit meer dan alleen een rekenkern bestaan (maar ook bijvoorbeeld een user-interface omvatten) zijn moeilijker te wrappen. Hierbij is het isoleren van de rekenkern een oplossing.

Wordt er wel gelet op andere ontwikkelingen?
Het SRW moet geen nieuw eiland vormen en wil daarom aansluiten bij andere ontwikkelingen. Verschillende ontwikkelingen zijn reeds in het rapport genoemd als belangrijk, zoals het adventus-stelsel voor gegevens-definitie en de stekkerdoos voor gegevensopslag, Good Modelling Practice als leidraad voor modelontwikkeling (en SRW zou dat kunnen ondersteunen), OpenGIS voor ruimtelijke gegevens en modellen, DCOM en CORBA voor inter-component communicatie en XML voor uitwisseling met allerlei pakketten.

Er zijn ons momenteel geen vergelijkbare initiatieven bekend elders in de wereld. Het zoeken daarnaar dient echter een continue activiteit te blijven. Belangrijk is wel te constateren dat het SRW niet alleen de 'hooks' biedt voor componenten om erop aan te sluiten, maar het zou ook 'hooks' kunnen krijgen om op andere raamwerken aan te sluiten. Hierdoor worden componenten van SRW dus ook makkelijk koppelbaar met andere initiatieven.

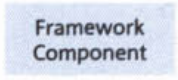
Gaat het SRW alleen over Water?
Nee, het SRW gaat over waterkwantiteit, waterkwaliteit, ecologie en milieu. Het domein-model blijkt ook een abstracte beschrijving te kunnen zijn voor allerlei aanpalende domeinen. Het is ook nadrukkelijk de bedoeling om breder dan alleen water in te steken. Echter er is in eerste instantie naar water gekeken omdat daar bij huidige modelkoppelingen de grootste problemen liggen. SRW biedt een structuur om modellen te ontwikkelen, te koppelen en te gebruiken. Alle modellen (dus ook niet-water modellen) die aan de specificaties van het SRW voldoen kunnen van die structuur gebruik maken.

Appendix G. Gebruikte symbolen uit UML

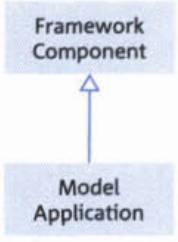
Hieronder staat een kort overzicht van symbolen die in de verschillende UML diagrammen gebruikt zijn.

1. Klassediagram

Een klassediagram toont de statische structuur in de vorm van klassen en de relaties tussen deze klassen.



Klasse:
Een klasse is een sjabloon voor gelijksoortige objecten.



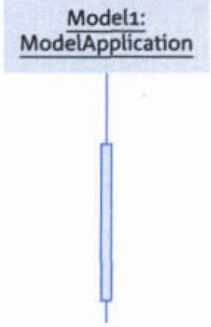
Overerving:
(ook aangeduid als specialisatie)
ModelApplication is een specialisatie van FrameworkComponent.

Associatie:
Een ModelElement heeft een relatie met nul of meer Connectoren; een Connector heeft een relatie met twee ModelElementen.

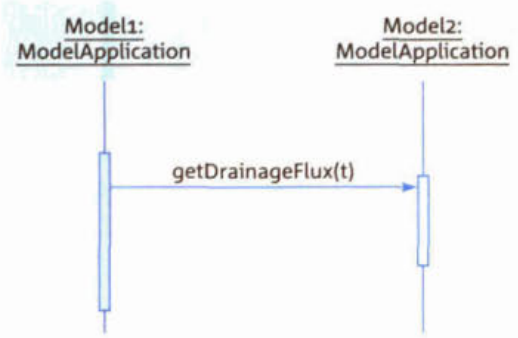
Aggregatie:
Een soilElement bestaat uit nul of meer SoilLayers.

2. Sequence diagram

Een sequence diagram illustreert het berichtenverkeer tussen objecten. De nadruk valt hierbij op de tijd, er wordt dus getoond wanneer een bericht wordt verzonden en ontvangen.



Instantie:
Model1 is een instantie van de klasse ModelApplication.

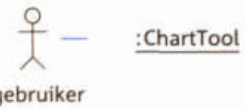


Bericht:
Model1 stuurt een bericht aan Model2, Model1 roept hierbij de methode getDrainageFlux(t) van Model2 aan.

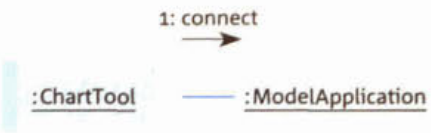
3. Collaboration diagram

Een collaboration diagram beschrijft hoe objecten samenwerken waarbij de nadruk valt op de ruimtelijke relaties tussen objecten. Dit betekent dat de relaties tussen objecten expliciet worden getoond. Er wordt geen tijdsvolgorde in dit diagram opgenomen.

Actor:
Een persoon of extern systeem dat interactie heeft met het te ontwikkelen systeem.



Instantie:
Een gebruiker instantieert een ChartTool (maakt een object van het type ChartTool).

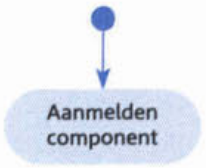


Relatie en bericht:
Een instantie van ChartTool wordt verbonden met een instantie van ModelApplication.

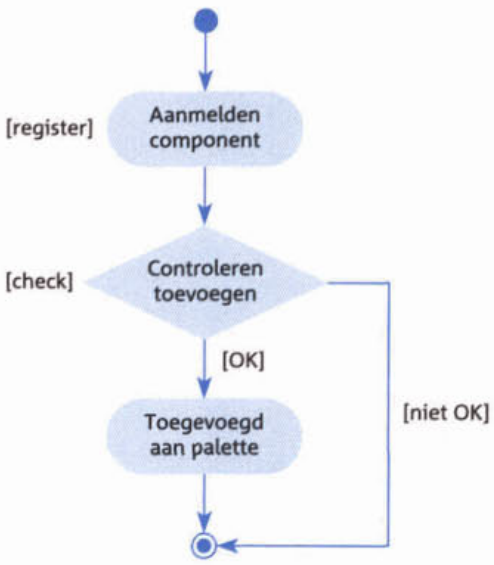
4. Activity diagram

Een activity diagram toont de interacties tussen objecten waarbij de nadruk valt op het uitvoerende werk.

Start toestand:
De initiele toestand van een activiteit.



Actie toestand:
Een voorbeeld van een toestand van een activiteit is het aanmelden van een component aan het raamwerk.



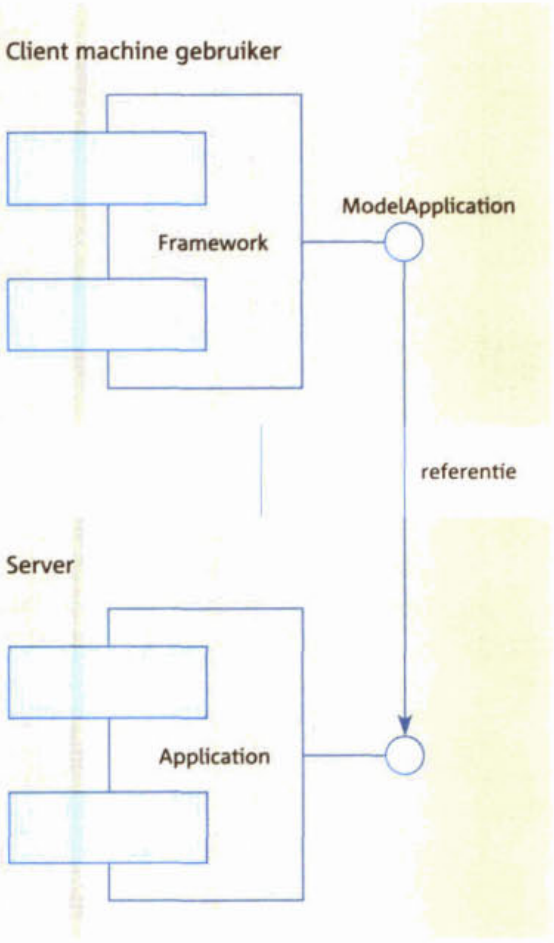
Beslissing:
Bij het registreren van een component binnen het raamwerk dient allereerst gecontroleerd te worden of de component geschikt is (voldoet aan de specificaties van het raamwerk).
Het resultaat van deze controle bepaald de volgende toestand van een activiteit.

Stop toestand:
De eindtoestand van een activiteit.

5. Deployment diagram

Een deployment diagram toont de runtime architectuur van een systeem. Het is een fysieke beschrijving van het systeem in hardware-units met de daarbij behorende software die hierop draait.

Node:
Zowel de cliënt als de server zijn nodes van het systeem.



Associatie:
Een associatie bestaat tussen een cliënt en een server; een cliënt heeft een referentie naar een server.