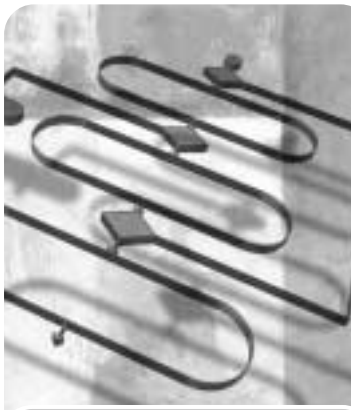
Het begrip 'Enterprise Application Integration' (EAI) kan u nauwelijks ontgaan zijn. Het is een heet hangijzer in hedendaagse IT-problematiek. In de praktijk gaat het negen van de tien keer over problemen die zich afspelen diep in de kerkers van grote bedrijven. Grote, stampende mainframes dienen te worden ontsloten of worden gekoppeld met zojuist verworven andere stampende machines. Het vervelende van EAI scenario's is dat het over het algemeen te maken heeft met zeer gesloten, proprietary systemen in een heterogene omgeving. Dergelijke systemen zijn niet bepaald ontworpen met interoperabiliteit in het vooruitzicht.



thema

J2EE en .NET Interoperability

Een architectuurgedreven aanpak

Hoe vaak is een technologie al niet aangeprezen als de 'silver bullet' voor het oplossen van dergelijke integratievraagstukken? Geen van deze technologieën bleek echter in staat deze belofte zonder meer te kunnen inlossen. De realiteit is dat een succesvolle EAI-aanpak een samenspel is van omzichtig plannen, auditing en een gezonde dosis ontwerp en architectuur 'best practices'. Er is oneindig advies over 'wat' er mogelijk is in EAI situaties, maar zelden weet iemand te vertellen 'hoe' het gedaan moet worden.

Gaandeweg het project worden dan ad-hoc beslissingen genomen, die uiteindelijk resulteren in wat Amerikanen zo mooi weten te verwoorden met: 'Hooking Shit Together'. Dit artikel gaat over de EAI-problemen van morgen: het ontsluiten en koppelen van de twee modernste enterprise platforms die er op dit moment in de markt zijn: J2EE en .NET. Frameworks waarin bij het bepalen van de architectuur wél is nagedacht over connectiviteit.

INVENTARISATIE VAN DE MOGELIJKE OPLOSSINGEN In de tweede helft van 2001 ontstond in de markt de vraag naar een mogelijkheid om J2EE en .NET met elkaar te laten samenwerken. Kennelijk bestaan er bedrijven die besluiten om technologie uit beide werelden in te zetten, of bijvoorbeeld door middel van overnames worden geconfronteerd met de technologiekeuze van een partner. Voor het beantwoorden van de vraag is een korte inventarisatie van de mogelijkheden voor het slaan van een brug tussen beide platforms op zijn plaats. In die inventarisatie zullen zeker aan bod komen: Message Queueing, Wire Level Bridges en natuurlijk de hype van dit moment: Web Services.

MESSAGE QUEUEING Message Queueing, of Message Oriented Middleware (MOM) is een oplossing die het best tot zijn recht komt in asynchrone, disconnected omgevingen. Het geeft de garantie dat een bericht altijd op de plaats van bestemming komt, ook als een verbinding tijdelijk niet beschikbaar is. De verbinding behoeft daarvoor niet synchroon van aard te zijn. Message Queueing wordt wel gezien als de klassieke EAI benadering en is op dat vlak dus 'proven technology'. Beide frameworks bieden deze vorm van connectivity. J2EE heeft de Java Messaging Service (JMS) en .NET biedt MSMQ. In de markt zijn diverse adapters beschikbaar waarmee deze op elkaar kunnen worden aangesloten. Er zijn echter toch enkele nadelen te bedenken waarin deze vorm van interoperability minder goed functioneert. Zo is er een flinke invloed op het programmeermodel. Een applicatie dient wel degelijk rekening te houden met het feit dat berichten asynchroon worden verstuurd. RPC-gebaseerde applicaties, die gewend zijn aan een synchroon request-response

Gaandeweg het project worden dan ad-hoc beslissingen genomen, die uiteindelijk resulteren in 'hooking shit together'

mechanisme kunnen hier niet altijd mee uit de voeten. Message Queueing komt alleen tot zijn recht als de onderliggende infrastructuur goed is. Binnen bedrijven zal dit in vele gevallen wel zo zijn, maar het is onmogelijk om een infrastructuur te bouwen waarop de hele wereld aan kan sluiten. Vaak is dit vanuit beveiligingsoogpunt ook niet gewenst, immers het stampende mainframe in de kelder is op deze manier toegankelijk geworden via internet!

> Er bestaan al enkele jaren bridges waarin Enterprise JavaBeans zich voor kunnen doen als COM-componenten en andersom

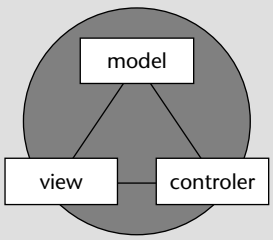
WIRE-LEVEL BRIDGES Een alternatief voor Message Queueing is beschikbaar door gebruik te maken van zogenaamde 'wire-level bridges'. Een bridge is een encapsulatielaag om een bestaande component of applicatie heen, waarin een vertaalslag plaatsvindt van het ene communicatie protocol (wire-level) naar het andere. Communicatie tussen verschillende frameworks is op deze manier synchroon en op maat gesneden. Vaak wordt ondersteuning geboden voor het doorgeven van een transactie en security context en kunnen objecten uit verschillende omgevingen tot een zelfde namespace behoren. Zo bestaan er al enkele jaren bridges waarin Enterprise JavaBeans zich voor kunnen doen als COM-componenten en andersom. Momenteel verschijnen de eerste producten in de markt die dezelfde functionaliteit bieden in een J2EE / .NET scenario. Het product Ja.NET van Intrinsyc is een voorbeeld van zo'n bridge (http://www.intrinsyc.com/products/ja.net). Nadelen die aan een dergelijke oplossing kleven zijn voornamelijk gebaseerd op de grote mate van complexiteit die wordt geïntroduceerd. Ten eerste is er een zeer hoge mate van koppeling tussen de bridge en componenten uit beide frameworks en ten tweede is de geboden oplossing vaak specifiek voor een bepaalde protocolversie, waardoor de kleinste verandering in één van beide situaties de grootste problemen kan opleveren.

WEBSERVICES Een modern alternatief wordt gevormd door webservices. Dit zijn diensten die op het (inter)netwerk worden aangeboden en die middels een implementatie-onafhankelijk protocol (SOAP) communiceren. Ook hiervoor geldt dat zowel J2EE als .NET ondersteuning bieden voor dit fenomeen. In .NET zijn webservices intrinsiek onderdeel van het framework, in J2EE is het voorlopig nog een set van API's en services die bovenop het framework worden aangeboden. In de nabije toekomst zullen deze verder in het framework worden geïntegreerd.

Webservices laten zich prima mappen op een component based architectuur. De Web Services Description Language (WSDL) is de manier om interfaces van componenten implementatie-onafhankelijk te beschrijven. Toolkits zijn in staat om met die beschrijving een proxy te genereren voor zo'n component, zodat het lijkt alsof de webservice een lokaal component is geworden. Demo's en voorbeelden die met de technologie worden meegeleverd laten zien dat er op dit gebied snel vorderingen worden gemaakt. Het ontbrak in het begin echter aan een overkoepelend orgaan, bijvoorbeeld het W3C, zodat al snel verschillende implementaties ontstonden. Nu, ruim een jaar later, hebben de eerste ervaringen geleerd dat er nog de nodige hobbels op de weg zijn naar vlekkeloze integratie via webservices. Een stap in de juiste richting is de oprichting van de Web Services Interoperability Organization (http://www.ws-i.org). De WS-I is een organisatie die zich bezig houdt met het definiëren van platform-, operating system- en pro-

Model-View-Controller

Het MVC Design Pattern verdeelt een software component in drie afzonderlijke delen: een Model, een View en een Controller.



FIGUUR 1: MVC model

Het Model is het onderdeel dat de data en het low-level gedrag van een component voorstelt. Het beheert de data en leidt alle bewerkingen op die data in goede banen. Het Model heeft geen specifieke kennis van zowel zijn Controller als zijn View. Views kunnen zich abonneren op een Model, zodat ze op de hoogte worden gebracht van wijzigingen in het Model. De View is de plek waar de visuele weergave van een component wordt bepaald. Hierin wordt een weergave getoond van de data in het Model. Per Model kunnen ook meerdere Views zijn. De Controller is verantwoordelijk voor de afhandeling van de (gebruikers)interactie. Het verschaft een mechanisme om mee te communiceren richting het Model.

Command Pattern

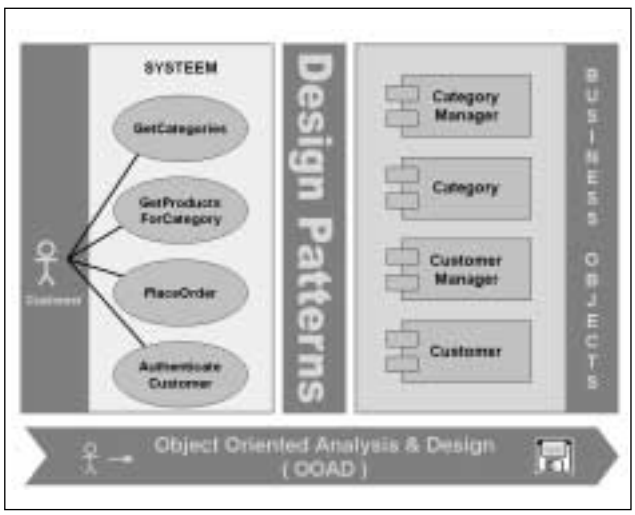
Het Command Pattern wordt in de beschreven architectuur ingezet als een verlengstuk op de Controller van het MVC pattern. Het pattern verpakt een verzoek naar de business logica als een object. Een client van een command object behandelt het command object als een 'black box', door er simpel gezegd de abstracte methode execute() op aan te roepen. Het onderliggende, daadwerkelijk implementerende object weet vervolgens raad met de gevraagde functionaliteit en zal deze uitvoeren. Vertaald naar het gewone leven zou het Command pattern kunnen worden vertaald naar een bestelling die in een restaurant bij een ober gedaan kan worden. In vele gevallen zal de ober deze bestelling slechts doorgeven aan de keuken waar een kok de bestelling klaarmaakt waarna de bestelling vervolgens weer via de ober op uw bord verschijnt. Het voordeel van een dergelijke opzet is dat de onderliggende implementatie losgekoppeld is van de client, zodat er afhankelijk van het soort verzoek een andere implementatie kan worden gekozen. Bijvoorbeeld het aanroepen van een lokale component versus het aanroepen van een web service.

grammeertaalonafhankelijke profielen op basis van webservices. Bij dit initiatief hebben zich al meer dan honderd bedrijven aangesloten, waaronder belangrijke spelers als Microsoft, IBM, BEA en Sun Microsystems. De algemene perceptie is dat er een enorme 'vendor support' bestaat teneinde een algemeen geaccepteerde standaard voor webservices in de markt te zetten.

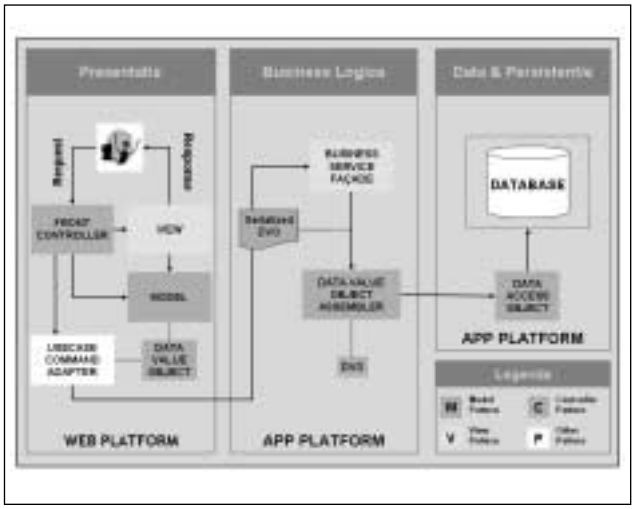
Architectuur en Design Patterns Twee belangrijke, niet-functionele requirements in een robuuste applicatie architectuur zijn uitbreidbaarheid en flexibiliteit. Om uitdrukking te geven aan deze eigenschappen spreekt men over de 'broosheid' of 'brittleness' van een applicatie. Broosheid in een applicatie kan worden voorkomen door te bewerkstelligen dat de mate van koppeling tussen de afzonderlijke componenten in een applicatie altijd laag is en de mate van samenhang juist hoog. Overal in de architectuur waar componenten met elkaar van doen hebben, dient deze regel in overweging genomen te worden. Dergelijke generieke probleemstellingen en hun bijbehorende generieke oplossingsvoorstellen zijn vastgelegd in zogenaamde design patterns. Design patterns zijn onderdeel van het jargon van de architect, ontwerper en ontwikkelaar. Ze zijn framework-, taal- en implementatie-onafhankelijk en worden gebruikt als communicatiemiddel bij het vinden van oplossingen. Dankzij onze IT-voorvaders is er een schat aan patterns als erfgoed achtergelaten. Nog dagelijks verschijnen er nieuwe patterns. Patterns van alle tijden vinden we bijvoorbeeld terug in de klassieker 'Design Patterns' van Erich Gamma et al. Een tweetal van deze patterns vormt de basis voor het schetsen van een generieke oplossing voor de realisatie van interoperability: Model-View-Controller (MVC) in combinatie met het Command Pattern (zie kaders MVC en Command Pattern).

Use Cases Net zoals WSDL een aanvulling is om van een component een implementatie-onafhankelijke beschrijving te geven van de achterliggende services, zo zijn UML Use Cases in de designfase een manier om implementatie-onafhankelijk de interacties met het systeem vast te leggen. Bij het implementeren van de interacties met het systeem zijn use cases het ideale uitgangspunt voor het creëren van de componenten. In figuur 2 ziet u een voorbeeld van interacties met een systeem in een webgebaseerde applicatie. In een moderne meerlaagse architectuur zal deze applicatie typisch worden uitgewerkt in een oplossing waarbij onderscheid zal worden gemaakt tussen presentatielogica, businesslogica en data. De uitwerkingen van de use cases vinden plaats in de componenten van de presentatie en businesslogica laag. Deze componenten zijn op hun beurt weer een prima aanknopingspunt voor interoperability. Door middel van het toepassen van de patterns kunnen we een transparante architectuur bedenken waarin een oplossing gebouwd in het ene framework via webservices naadloos kan communiceren met componenten uit het andere framework.

Applicatie architectuur Het samenvoegen van de genoemde design patterns en de hierboven beschreven manier van interop levert een eenvoudige weergave van een applicatie architectuur op. In figuur 3 ziet u het gebruik van de MVC en Command patterns terugkomen. Tevens wordt er gebruik gemaakt van een ander design pattern, de zogenaamde Data Value Objects (DVO). Zo'n DVO wordt gebruikt voor de uitwisseling



Figuur 2. Use Cases als aanknopingspunt voor Interoperability



FIGUUR 3. Technische Architectuur

van data tussen verschillende componenten of lagen van de architectuur. Best practice hierbij is om het aantal roundtrips tussen verschillende lagen te minimaliseren door het zoveel mogelijk accumuleren van de data die over het netwerk moet. De data wordt vervolgens getypeerd overgezonden, zodat beide kanten dezelfde betekenis kunnen geven aan die data. Een ideale techniek voor het platform-onafhankelijk typeren en structureren van data is natuurlijk het toepassen van XML. Door DVO's in staat te stellen zichzelf te serialiseren naar XML zijn ze gemakkelijk transporteerbaar als de inhoud van een SOAP bericht.

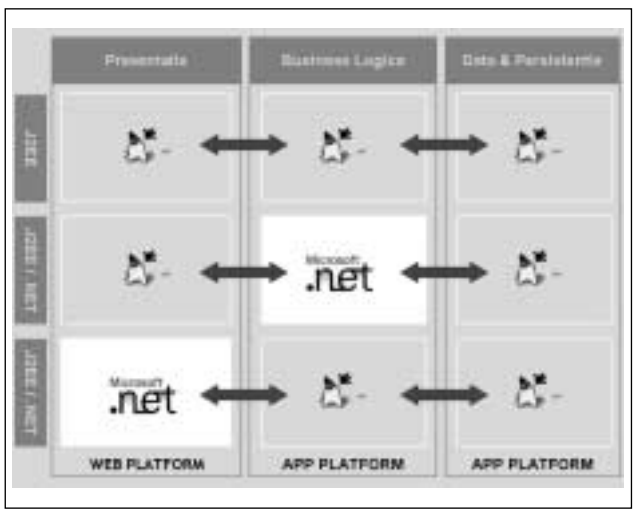
> Dankzij onze IT-voorvaders is er een schat aan patterns als erfgoed achtergelaten; nog dagelijks verschijnen er nieuwe patterns

J2EE IMPLEMENTATIE Implementatie in een J2EE-omgeving van de applicatie-architectuur kan heel gemakkelijk worden gerealiseerd door middel van het toepassen van twee gratis producten die te downloaden zijn van het internet. Het MVC gedeelte wordt gerealiseerd met behulp van Struts, een Open Source framework van de Apache Software Foundation (http://jakarta.apache.org/struts) en geldt in de Java wereld als een de facto standaard.

In de J2EE blueprints (http://java.sun.com/blueprints) van Sun wordt de MVC-architectuur ook wel omschreven als 'Servlet-JSP model-2 architecture' en als best practice aangeraden voor web applicaties. Struts splitst het MVC pattern op in een Servlet (Controller) die uit te breiden is met zogenaamde action objects, Javabeans die (gedeeltes van) het model representeren en jsp pagina's die als de views dienen. Vanuit de action objects worden met behulp van het command pattern (overigens in combinatie met een Factory pattern) van business calls objecten gevormd. Parameters uit de action objecten worden als argumenten aan die objecten aangeboden. Door het toepassen van het Command pattern is er in principe geen verschil meer tussen het aanroepen van een lokaal business component (bijvoorbeeld een EJB) of het aanroepen van een webservice. De daadwerkelijke call zit verstopt in de uitwerking van het onderliggende use case object. Deze keuze zorgt voor de grote mate van flexibiliteit, immers de business logica kan op een ander platform worden geïmplementeerd zonder dat de view of de controller daar last van heeft. Door een simpele aanpassing in de Struts configuratie kan worden gewisseld tussen een lokale aanroep en een .NET webservice.

Het webservices gedeelte in de J2EE implementatie komt voor rekening van GLUE. GLUE is een webservices development kit, waarvan de 'standard edition' een gratis product is, dat ook in commerciële toepassingen gebruikt mag worden (http://www.themindelectric.com/glue). Met behulp van GLUE kunnen WSDL documenten gegenereerd worden voor Java objecten die vervolgens weer in Visual Studio .NET geïmporteerd kunnen worden als een zogenaamde 'web reference'. Visual Studio .NET zal voor deze web references proxy's genereren, waardoor het vanuit de .NET applicatie lijkt of er tegen lokale componenten gesproken wordt. Andersom is uiteraard ook mogelijk, zodat vanuit Java de .NET web services kunnen worden aangesproken.

.NET IMPLEMENTATIE Het is uiteraard ook mogelijk om de voorkant in .NET te implementeren. Ondanks dat er geen Struts voor .NET bestaat is het toch goed mogelijk om MVC te implementeren in een .NET applicatie. ASP.NET, de dynamische webpagina's van



FIGUUR 4. Java en .NET interoperability

.NET, bieden hiervoor een aanknopingspunt. De aspx pagina's vormen de views, de controller kan worden gezien in de zogenaamde code-behind-page en het model is een eigen ADO.NET DataSet. Op een DataSet zit standaard functionaliteit waarmee deze naar XML te serialiseren is. Alle calls naar de businesslaag gaan via webservices. Door middel van het configureren van web references kan worden verwezen naar de J2EE web services. Ook hier is het wijzigen van een configuratie bestand voldoende om te wisselen van lokale aanroep naar een J2EE web service.

TOEKOMST VAN DE WEBSERVICES ARCHITECTUUR In dit artikel heb ik een applicatie-architectuur geschetst waarmee het mogelijk is een basis te leggen voor cross-platform interoperability tussen applicaties geschreven in J2EE en .NET. Door gebruik te maken van enkele bekende design patterns is het mogelijk afhankelijkheden te ontkoppelen. Het gebruik van webservices en XML zorgt voor het platform-onafhankelijk maken van component aanroepen en data-overdracht. Echter, interoperability is meer dan alleen maar het doorgeven van data tussen verschillende applicatielagen. Daadwerkelijke interoperability tussen beide platforms zal in de toekomst misschien niet alleen in een peer-to-peer scenario worden toegepast, maar waarschijnlijker zijn intranet/internet implementaties. In dat geval zal er nog veel werk verzet moeten worden. De huidige mechanismen voor het bewaken van security en transacties over platformen heen laten nog veel te wensen over. Om nog maar niet te spreken over de manier waarop transacties worden geïmplementeerd. Trading partners onderling gaan natuurlijk nooit akkoord met scenario's waarbij de ene partij locks legt in de resources van de andere partij om deze vervolgens seconden tot misschien wel weken te laten staan. De WS-I is zich goed bewust van deze problematiek en werkt op initiatief van Microsoft en IBM aan de zogenaamde Global XML Architecture (GXA). GXA maakt gebruik van de uitbreidingsmogelijkheden die SOAP biedt, namelijk uitbreidbare headers. In de SOAP headers zal informatie worden opgenomen die bijvoorbeeld een security of een transactiecontext beschrijven. Infrastructurele software (bijvoorbeeld Microsoft BizTalk) zal dan worden ingezet om lokaal deze header-informatie om te zetten in daadwerkelijke transacties. In plaats van het langdurig locken van resources kan dan bijvoorbeeld worden gekozen voor het direct committen van data en compensational code in het geval dat er iets mis gaat. Op het moment van schrijven staat GXA nog in de kinderschoenen, maar het is zeker de moeite waard dit initiatief te blijven volgen.

> Voor het design pattern DVO is het best practice om het aantal roundtrips tussen verschillende lagen te minimaliseren

Ing. Bert Ertman is werkzaam als IT-architect bij Info Support BV te Veenendaal. E-mail: berte@infosupport.com

PATCHES Patches PATCHES Patches PATCHES Patches PATCHES

Open Source Toolchain voor embedded appliances met Embedded Linux OS

Internet security- en OEM-leverancier SnapGear heeft onlangs een 'doorbraak' aangekondigd in ontwikkelondersteuning voor de Intel XScale(TM) (IXP425) microprocessor. De open source omgeving integreert een Intel-ondersteunde port met SnapGear's Linux expertise en open source tools. Dit kan voor developers, die op dit moment duizenden euro's voor ontwikkeltools moeten neertellen, aanzienlijke kostenbesparingen opleveren.

Een click selectie, via de nieuwe uClinux distributie, configureert de noodzakelijke Linux kernel, library's en applicaties te bouwen voor de IXP425 processor. Met één aanroep van de make utility kan de ontwikkelaar een binary image produceren voor 'load and run' op het IXP425 bord. Conventionele Linux ontwikkelaars zullen bekend zijn met het build process, dat gebaseerd is op standaard Linux kernel configuratie mechanismen.

De Intel XScale (IXP425) processor is een ARM-based RISC core met een kloksnelheid tot 533MHz met 64k cache. Voor embedded productontwikkelaars is dit niet onbelangrijk vanwege de graad van component integratie waaronder een on-chip SDRAM controller, twee high-speed seriële poorten, twee 10/100Mbit Ethernet poorten, UTOPIA interface for ATM /xDSL, PCI bus, host USB, en een encryptie accelerator.

De nieuwe tools zijn geïncorporeerd in een toekomstige release van de uClinux distributie die beschikbaar zal zijn in februari 2003.

De distributie zal ook ondersteuning bevatten voor iAPx86 en Hitachi SuperH(R) cores en zal gebaseerd zijn op de Linux 2.4 kernel met glibc-2.2.5, uClibc en meer dan 150 applicatie packages.