Huub Ris
is freelance ICT-consultant

J# .NET: gebruik blijven maken van Java datatypes en Java library's

JAVA-APPLICATIES MIGREREN NAAR .NET

J# brengt de Java-taal naar het .NET. Hiermee is bestaande Java broncode zonder enige aanpassing te compileren naar .NET. Microsoft J# ondersteunt alle Java library's (inclusief AWT, JDBC en Beans) en heeft bovendien toegang tot het hele .NET Framework en de library's. Dit betekent dat migratie naar het .NET geen gevolgen heeft voor de bruikbaarheid van de bestaande Java-applicaties, hetgeen een belangrijk bezwaar wegneemt voor bedrijven om over te gaan naar het veel snellere .NET.

Belangrijker is dat het met J# (net zoals alle andere .NET-talen) eenvoudig is webservices te bouwen en te gebruiken; wat in Java een stuk lastiger is. Je kunt met J# een duidelijke meerwaarde geven aan bestaande Java-applicaties. Verder is er voor J# een tool genaamd JBIMP, waarmee binaire Java-classes kunnen worden geconverteerd naar .NET IL-classes. De programmeur heeft niet de broncode nodig om de classes toch als basis te kunnen gebruiken voor verdere ontwikkeling. J# is dus Java, maar dan voor Visual Studio .NET. Voor Java-developers betekent dit dat zij de mogelijkheden van Visual Studio .NET kunnen gebruiken zonder dat zij zich daarvoor eerst hoeven om te scholen in de Microsoft-talen.

Upgrading naar J#

Visual J# .NET bestaat uit een Java-taal-compiler, JDK level 1.1.4 class-library's, en een tool om gecompileerde Java bytecode te upgraden naar MSIL - handig om library's te upgraden naar een form dat ze kunnen gebruiken vanuit .NET-applicaties. Naast JDK level 1.1.4 class-library's, net als andere .NET-compatibele talen, heeft Visual J# .NET volledige toegang tot het .NET Framework, en bevat designers voor Windows Form en WebForm-ontwikkeling.

Upgraden naar Visual J# .NET is de eenvoudigste en snelste manier om Java-applicaties naar het .NET Framework te upgraden. Dankzij een bekende syntax en een set class-library's kunnen ontwikkelaars gebruik blijven maken van hun bestaande kennis van Java, en direct productief zijn op het .NET Framework. Applicaties die zijn geüpgrade naar Visual J# .NET bieden direct voordeel: de upgrade is heel snel, en ontwikkelaars kunnen features toevoegen dankzij het .NET Framework en Microsoft-extensies van de Java-taal. De Java naar Visual J# .NET Upgrade Wizard wordt aangeroepen door een Visual J++ 6.0 project te openen met Visual Studio .NET.

Technologieën zonder automatische migratie

Zowel Visual J# .NET als de Java Language Conversion Assistant richten zich op de Java-taal, JDK level 1.1.4 library's, en de Microsoft-library's die met Visual J++ 6.0 worden geleverd. Sommige applicaties zullen ook gebruik maken van technologieën van latere Java-versies zoals J2EE en J2SE. Deze applicaties zullen vaak een paar additionele modificaties vereisen om ze naar .NET te migreren. Dit is in de meeste gevallen erg eenvoudig: na de migratie zal het grootste deel van de applicatie (businesslogica en JDK level 1.1.4 classes) perfect zijn gemigreerd. De niet-ondersteunde technologieën zullen onveranderd zijn gebleven in de gemigreerde code. Om de migratie te voltooien vervang je de niet-gemigreerde Java-technologie met vergelijkbare .NET-technologie. Doordat .NET een rijkere set classes biedt dan wat beschikbaar is in Java, kun je enkele belangrijke verbeteringen en uitbreidingen maken bij het vervangen van de technologie.

De migratie van *het Swing forms-package* is een goed voorbeeld. Java bevat twee elkaar beconcurrerende forms-packages: de Abstract Windowing Toolkit

Java naar .NET: C# of J#?

Er zijn twee wegen om Java-applicaties naar .NET te migreren: ze upgraden naar Visual J# .NET, of ze converteren naar Visual C# .NET. Elke benadering vereist een verschillende inspanning, en elk heeft verschillende voordelen. Voor beide benaderingen geldt dat de applicatie naar Microsoft Windows wordt gemigreerd. Het besluit welke .NET-compatibele taal wordt gebruikt om de Java-applicaties te migreren is voornamelijk een kwestie van persoonlijke voorkeur. Zoals eerder is besproken compileren zowel Visual J# .NET als Visual C# .NET naar dezelfde MSIL, beide talen hebben toegang tot hetzelfde .NET Framework, en beide talen bieden gelijkwaardige mogelijkheden. Elk migratiepad genereert een verschillend resultaat. Onderstaande tabel toont wat gebeurt met de compositie van Java-projecten als ze naar .NET migreren.

Java-technologie	Upgrade naar J#	Upgrade naar C#
Java-taal	Java-taal	C#-taal
Applet	Niet geconverteerd	Windows Form control
JavaBean	JavaBean	C# class
Abstract Windowing Toolkit (AWT) frame	AWT-frame	Windows Form
WFC Form	WFC Form	Windows Form
Gecompileerde library	Gecompileerde library	Niet geconverteerd
Resource-file	ResX-file	ResX-file

Tabel.

Om je te helpen de keuze te maken tussen upgrading naar Visual J# .NET of conversie naar Visual C# .NET geven we hier drie belangrijke bepalende factoren ter overweging:

· *Migratietijd*

Als een Java-applicatie wordt geüpgrade naar Visual J# .NET blijft Java de taal en 'calls' naar JDK niveau 1.1.4 API's blijven behouden zoals ze zijn. Als dezelfde applicatie wordt geconverteerd naar C# wordt de taal geconverteerd naar C# en Java API-calls worden geconverteerd naar native .NET Framework-calls. De kwaliteit van deze conversie is zeer hoog, maar er is wel sprake van enkele modificaties nadat de *Java Language Conversion Assistant* klaar is. Wanneer de migratietijd belangrijk is zou Visual J# .NET de beste keuze kunnen zijn.

· *Framework*

Applicaties die direct naar C# zijn geconverteerd hebben het voordeel gebruik te kunnen maken van het nieuwe .NET Framework, met verbeterde performance, schaalbaarheid, beveiliging, en versioning over het Java-platform. Applicaties die zijn geüpgrade naar Visual J# .NET blijven gebruik maken van Java datatypes en Java library's, ofschoon je nieuwe functionaliteit kunt toevoegen met het .NET Framework. Als het voor de applicatiearchitectuur belangrijk is gebruik te kunnen maken van het .NET Framework, dan zou C# de beste keuze kunnen zijn.

· *Taal*

De voorkeur voor een taal is in vele gevallen simpelweg de belangrijkste afweging: of je de applicatie nu wilt ontwikkelen in Java of C#. Het is voornamelijk een zaak van persoonlijke voorkeur. Zowel Visual J# .NET als C# stammen af van C++ en hebben een vergelijkbare grammatica en gelijkwaardige mogelijkheden. Visual J# .NET zou de beste keuze kunnen zijn voor personen die in Java hebben geïnvesteerd en gebruik willen blijven maken van Java. C# zou de beste keuze kunnen zijn voor ontwikkelaars die naar een "native" .NET-compatibele taal willen migreren die specifiek is ontworpen om .NET-applicaties te ontwikkelen. Componenten die zijn geschreven in verschillende .NET-compatibele talen kunnen moeiteloos met elkaar samenwerken, wat 'mixed-mode' migraties van Java mogelijk maakt. In een mixed-mode migratie upgrade je een applicatiecomponent naar Visual J# .NET en converteer je een andere naar C#. Praktische voorbeelden omvatten de upgrade van een middle-tier component naar Visual J# .NET en van deze gebruik maken vanaf een client die is geconverteerd naar C#. Je kunt ook van de *Visual J# .NET Binary Converter* gebruik maken om library's beschikbaar te maken voor elke willekeurige .NET-compatibele taal.

(AWT) en Swing. AWT is beschikbaar in JDK level 1.1.4 en wordt ondersteund door Visual J# .NET en geconverteerd naar Windows Forms door de Java Language Conversion Assistant. Een add-on Swing-library was beschikbaar voor JDK 1.1.4 en is een onderdeel van de J2SE en J2EE class-library's. Swing formlay-outs worden niet automatisch gemigreerd naar .NET. Tijdens de migratie blijven de *Javax.Swing* classes onveranderd in de code, en zouden ze moeten worden vervangen door Windows Forms classes. Om dit te doen moet je het volgende stappen uitvoeren voor elke Swing Form in de gemigreerde applicatie:

- voeg een Windows Form toe aan de gemigreerde applicatie;
- voeg controls toe aan het form om dezelfde lay-out te bereiken als het originele Swing form;
- kopieer de event-logic code van het gemigreerde Swing form in event-handlers in de nieuwe Windows Form;
- verwijder de gemigreerde Swing-form van de applicatie.

Windows Forms leveren een rijkere omgeving dan zowel AWT als Swing en bevatten in bijna alle opzichten een superset van de functionaliteit die wordt gevonden op het Java-platform. Windows Forms maken het bijvoorbeeld eenvoudiger de lay-out van forms te controleren. Lay-outmanagers in Java zijn vaak een bron van problemen. Om de gewenste lay-out van een form te krijgen vereist een begrip van lay-outmanagers, en behoorlijk wat code voor zelfs de eenvoudigste lay-outs. Windows Forms vereenvoudigt dit door te zorgen voor 'absolute positionering' van alle controls. Vergroting en verkleining van forms zijn bovendien eenvoudig met Windows Forms dankzij 'docking' en 'anchoring'. Docking specificeert dat een control altijd een bepaald gebied van het scherm inneemt. Anchoring 'prikt' een of meer hoeken van een control aan het form. Als het formaat van het form wordt veranderd worden deze 'punaises' gebruikt om de positie van de control te bepalen. Door gebruik te maken van docking en anchoring kan gecompliceerde code voor form-formaatveranderingen gewoonlijk volledig worden weggelaten. Niet alleen is het eenvoudiger om Swing-forms naar .NET te migreren, je kunt ook een betere applicatie bouwen dankzij de rijkere classes van het .NET Framework.

Microsoft Press

Titel: Microsoft® Visual J#™ .NET (Core Reference)
ISBN: 0-7356-1550-0
Auteur: John Sharp, Andy Longshaw, Peter Roxburgh

Migreren van Java Server Pages

Java Server Pages (JSP) en Servlets zijn beide technologieën voor het creëren van Java-enabled Webpagina's. JSP zorgt voor een script-embedded HTML-architectuur die gelijk is aan ASP. Servlets bieden een mechanisme voor gecompileerde JSP-pagina's zonder HTML-code. Beide items maken gebruik van Java-code, en beide bieden event-based programmering voor Webpagina's. Zowel JSP en Servlets kunnen naar ASP.NET worden gemigreerd, en er zijn aantrekkelijke voordelen om het ook te doen: ASP.NET is een robuustere technologie, met een grotere schaalbaarheid en performance dan JSP, PHP (PHP Hypertext Preprocessor), ASP, en andere script-geïnterpreteerde embedded HTML-architecturen.