Sinds in 2001 de eerste versie van OMG's Model Driven Architecture visie werd gepubliceerd zijn er fervente vóór en tegenstanders. Voorstanders zien in de MDA kansen om onafhankelijk te worden van (tijdelijke) technologie, en veel (zo niet alle) code te genereren. Bovendien hopen ze software development en onderhoud vijftig tot zeventig procent goedkoper te kunnen maken. Tegenstanders wijzen erop dat UML nog veel te weinig expressie mogelijkheden kent en dat derhalve de code de enige volledige weergave van de programmatuur is. Bovendien is goede UML analyse en design kennis een schaars goed en staan beschikbare tools nog in de kinderschoenen.

thema

Codegeneratie versus offshore ontwikkeling

Model Driven Architecture weer een stap dichterbij

Tegenstanders lijken hiermee vooral naar het heden en het verleden te kijken, voorstanders naar de toekomst. De doeners en de dromers komen ieder jaar echter een stapje dichter bij elkaar, want tools worden beter en UML kennis neemt toe. MDA lijkt dan ook de komende jaren in de Java gemeenschap voor een kleine revolutie te gaan zorgen. Want is het niet zo dat we als Java developers tachtig procent van onze tijd bezig zijn om connectivity en distributie uit te programmeren? Terwijl we daar in de regel vaste patronen (Struts, session façade, value object pattern, et cetera) voor gebruiken? How boring... Met een goede MDA tool zouden we bovendien de RAD-race ook eens kunnen winnen...

Een andere revolutie die gaande is, betreft de India en China development route. In de praktijk blijken gedistribueerde software-engineering projecten in India goed te kunnen worden uitgevoerd voor 55% van de Nederlandse prijs. De China route belooft dat nog eens een stuk goedkoper te kunnen. Het is dan ook zeer opportuun voor de Nederlandse developers om kostprijsverlagende tools op de voet te volgen.

NIEUWE SPELERS Een uitgangspunt in de MDA visie van de Object Management Group is dat de technologie in de meeste gevallen sneller verandert dan de business die met de technologie ondersteund wordt. Er zijn dan ook veel voorbeelden van grote legacy-systemen die niet alleen herbouwd, maar ook opnieuw gespecificeerd moeten worden. Een "Platform Independent Model" van een systeem is derhalve een belangrijke asset voor een organisatie. Als je bovendien in staat zou zijn om op basis van de PIM een Platform Specifiek Model (PSM) te onderhouden voor bijvoorbeeld de J2EE technologie, dan heb je meteen een basis voor grotendeels te genereren systeem (de implementatie).

Voor Java Developers is het volgende scenario nu al haalbaar: stel dat je voor een systeem zou beschikken over een PIM in de vorm van een Rational Rose (UML) model. En je zou over een tool beschikken waarmee je de UML export bestanden zou kunnen inlezen en van de belangrijkste J2EE templates zou kunnen voorzien. En je zou tachtig procent van je code hiermee kunnen genereren, zodat je alleen nog maar de businesslogica zou moeten coderen (in Java of een platformonafhankelijke taal). Dan zou je hiermee tegen kosten die ver-

> Er is een revolutie gaande: de India en China development route

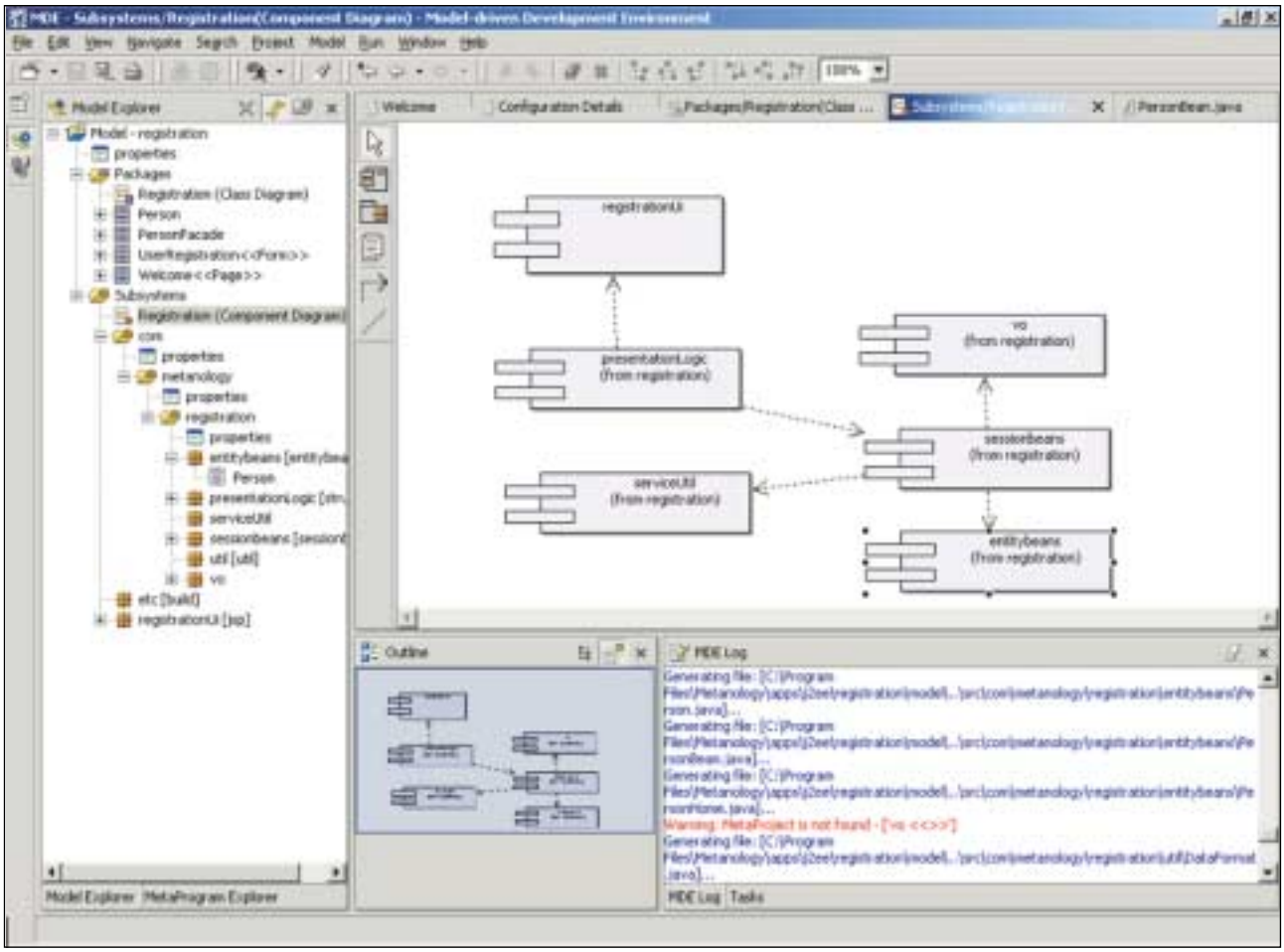
gelijkbaar zijn met de 'India-route' systemen kunnen realiseren. Het zou wel zeer prettig zijn als je bovendien de templates voor de codegeneratie zelf kunt onderhouden. Bij gebrek aan expressiviteit van de UML zou het ook wenselijk zijn - wanneer de toolleverancier zo groot is dat je erop mag vertrouwen - dat de tool net zo'n lange levensduur heeft als het te realiseren systeem.

> Het is zeer opportuun voor de Nederlandse developers om kostprijsverlagende tools op de voet te volgen

Sinds Software Release Magazine op initiatief van Rick van der Lans in oktober 2002 een artikel over dergelijke tools publiceerde zijn er veel nieuwe spelers op de markt bijgekomen. Een interessante speler om bovenstaand scenario mee te demonstreren is de MDE tool van leverancier Metanology.

ONLINE BANKIEREN MDE (Model-driven Development Environment) is een op het open source project Eclipse gebaseerde ontwikkelomgeving voor het genereren van code voor nieuwe technologieën. Met MDE is het mogelijk om op basis van meta-programma's code te genereren voor J2EE-applicatie of ander web-based oplossingen. Deze meta-programma's zijn vele malen sneller te schrijven dan een feitelijke applicatie omdat met allerlei taalspecifieke programmeerdetails geen rekening hoeft te worden gehouden. Van deze meta-programma's zijn voor verschillende platforms (zoals EJB's en Struts) zelfs al standaardversies beschikbaar in MDE, die verder door een architect kunnen worden uitgebreid en aangevuld met commentaar. Aanleiding tot de ontwikkeling van MDE door Metanology was de opkomst van het online bankieren eind jaren negentig en de snel veranderende opvatting over architectuur van systemen voor online bankieren.

Metanology realiseerde zich toen dat er, ten gevolge van deze snelle omschakelingen, veel meer tijd en geld werd geïnvesteerd in architectuur dan in zakelijke functionaliteit. De functionaliteit, die uiteraard centraal staat in een klantgerichte oplossing, werd hierdoor erg duur. Op dat moment is Metanology begonnen met het schrijven van meta-programma's die op een bepaald platform zijn toegespitst. Hiernaast is toen ook begon-



FIGUUR 1. Een model in MDE

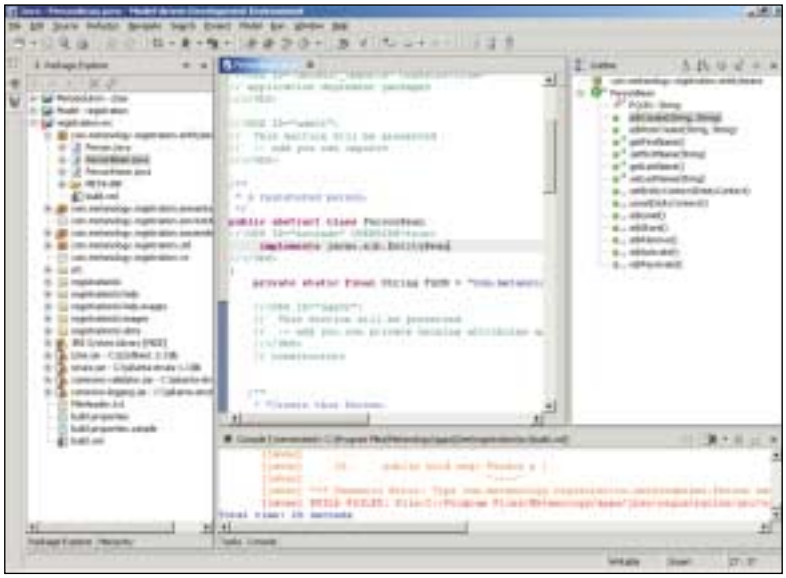
nen met de ontwikkeling van de Model-driven Development Environment omgeving die deze meta-programma's zou kunnen vertalen in code.

MDE-MODELLEN Zoals de naam Model-driven Development Environment al aangeeft, is ontwikkeling met MDE gebaseerd op modellen. MDE is dan ook uitgerust met een grafische editor voor UML-modellen. Bij het opstarten van MDE wordt standaard de Model Explorer weergegeven in het linker-deelvenster. Als je een model opent, wordt de inhoud hiervan rechtsboven weergegeven. Er wordt onderscheid gemaakt tussen classmodellen en componentmodellen. In figuur 1 wordt een component model weergegeven.

Verder is MDE op Eclipse gebaseerd, waardoor de interface voor gebruikers van Eclipse of WSAD zeer bekend zal voorkomen. MDE is als stand-alone applicatie te installeren, maar ook als plug-in op Eclipse of WSAD. Net als Eclipse werkt ook MDE met perspectives, die de workbench optimaliseren voor een bepaalde taak. In de voorgaande afbeelding is de standaard perspective "MDE" afgebeeld. Een andere perspective is de "Java" perspective (zie figuur 2), die de componenten weergeeft waaruit classes zijn opgebouwd en daarnaast een deelvenster bevat waarin de output van de console wordt gegeven.

PACKAGE STRUCTUUR Je kunt met MDE zelf modellen maken. Hiervoor moet eerst een project worden aangemaakt om deze modellen in op te slaan. Kies hiervoor File, New, Project en selecteer MDE, Model. Voer een naam in voor het project in het Directory tekstvak. Vervolgens kun je een model voor een class aanmaken door in de Model Explorer te rechtsklikken op "Packages" en "Add Class Diagram" te kiezen. De class kan vervolgens met behulp van de knoppen aan de linkerkant van het hoofdvenster worden samengesteld (zie figuur 1).

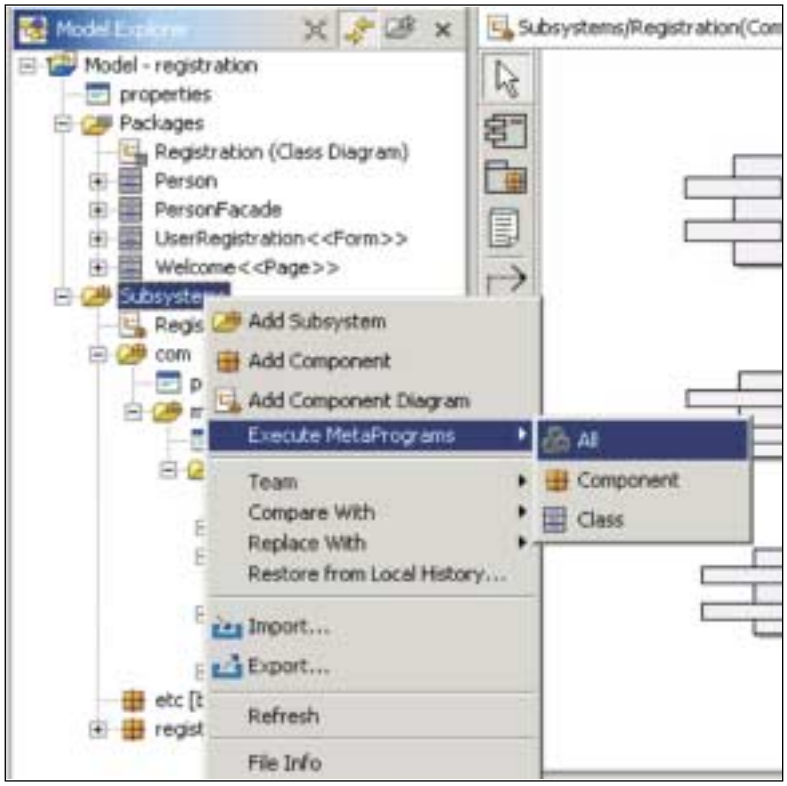
Om een model van een component te maken, dient eerst een package structuur te worden aangemaakt voor de sourcecode. Klik hiertoe in de Model Explorer het project open, rechtsklik op Subsytems en kies "Add Subsystem". Vul de naam van de directory op het hoogste niveau in, bijvoorbeeld "com" en sla het subsysteem op. Rechtsklik vervolgens op de gecreëerde directory "com" en voeg op dezelfde manier het volgende niveau toe. Vervolgens kan een component worden toegevoegd door op subsystems te rechtsklikken of op de package-naam en "Add Components" te kiezen. Naast de mogelijkheden die MDE biedt om zelf een model te maken is het ook mogelijk een bestaand UML-model in MDE te importeren. Zo kan een mdl-bestand dat met Rational Rose is gemaakt, via File, Import in MDE worden geïmporteerd (zie figuur 3).



FIGUUR 2. De Java perspective met aan de rechterkant de methoden en velden van de geopende Java class

META-PROGRAMMA'S Nadat modellen zijn gemaakt van de te bouwen classes en componenten van een applicatie kan de sourcecode voor deze applicatie worden gegenereerd met behulp van de bij MDE meegeleverde meta-programma's. Vervolgens kan de gegeneerde sourcecode met behulp van de Java ontwikkelomgeving, die we van Eclipse kennen, in een Java

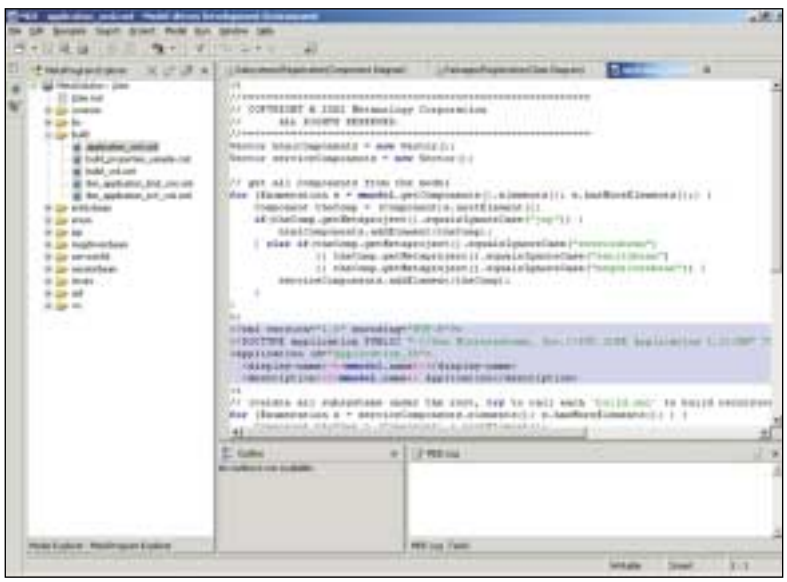
FIGUUR 3. Modellen importeren/exporteren



FIGUUR 4. Source code genereren

project worden opgenomen. Anders dan een model, dat voor een bepaalde oplossing geldig is, maar niet gebonden is aan een specifieke architectuur, wordt een meta-programma geschreven voor een specifieke architectuur, maar niet voor een bepaalde oplossing. Je hebt dus voor elk platform een apart meta-programma.

Om code te genereren uit een meta-programma aan te maken, kies je de optie File, Open MetaSolution. Selecteer vervolgens het geschikte meta-programma. In de huidige versie van MDE is een meta-programma j2ee opgenomen (het bestand j2ee.msl MDE\MetaPrograms\j2ee\j2ee.msl) waarmee J2EE source code kan worden gegenereerd (voor bijvoorbeeld Struts- of EJB-applicaties). MDE weet dan dat te genereren oplossing een J2EE-oplossing moet worden. Om de code te genereren op basis van een UML-model dien je op "Subsystems" te rechtsklikken de opties Execute MetaProgram, All te kiezen (zie figuur 3). De optie "All" houdt in dat zowel van de classes als van de componenten source code wordt gegenereerd.



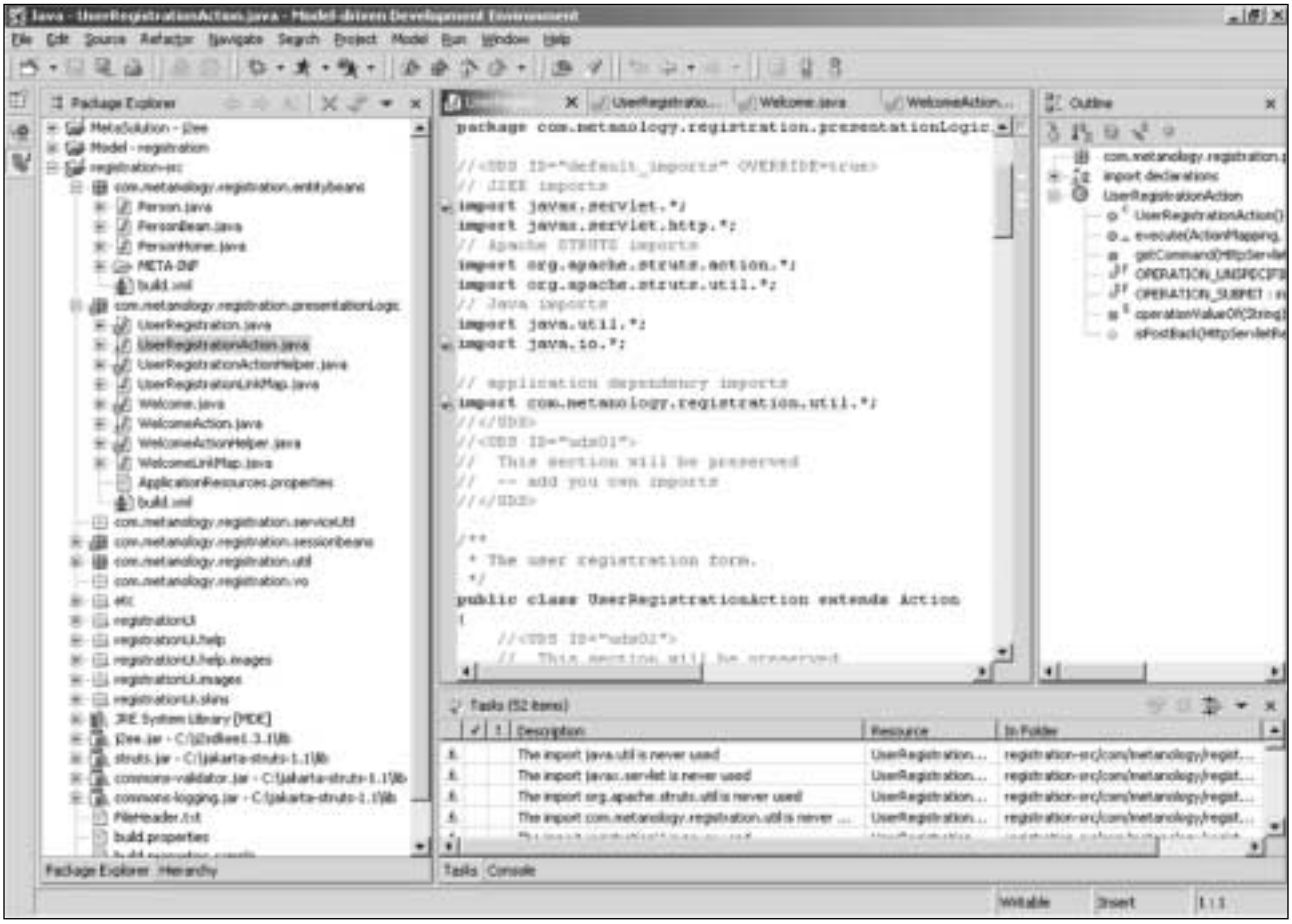
FIGUUR 5. De MetaProgram Explorer

Meta-programma's kunnen ook zelf worden geschreven of aangepast. Een meta-programma heeft een Java- en JSP-achtige syntax en bevat alle technische details van de architectuur. Een voorbeeld van meta-taal is te zien in afbeelding 5. In het linkerdeelvenster is de MetaProgram Explorer geselecteerd en in het hoofdvenster wordt de inhoud van een meta-programma afgebeeld.

LIFE CYCLE TOOLS MDE is uitgerust met life cycle tools. Hiermee kan handmatig ingevoerde code worden opgeslagen en naar nieuwere versies van een applicatie worden gemigreerd. Verder wordt bij het opnieuw genereren van een gewijzigd model de oude door meta-programma's gegenereerde code vervangen door de nieuwe. Je kunt dus op een efficiënte en snelle manier iteratief programmeren.

STRUTS Er zijn verschillende installaties mogelijk van MDE. Voor het maken van modellen is de eenvoudigste versie voldoende: MDE for UML. Voor het genereren van sourcecode dient één van de volledige versies te worden geïnstalleerd. Zo zijn op dit moment de volgende installaties beschikbaar: MDE for Struts voor het bouwen van applicaties op basis van het Java framework Struts, en MDE for EJB voor het bouwen van Java-applicaties met Enterprise Java Beans. De meest uitgebreide versie van MDE, MDE for J2EE, bevat mogelijkheden voor het genereren van zowel Struts code als EJB's. Maar laten we eerst eens een kijkje nemen naar MDE for Struts.

Met MDE for Struts maak je eerst een gebruikersinterface model voor een Struts-applicatie worden gemaakt met behulp van standaard UML-diagrammen. Vervolgens genereer je met de Struts meta-programma's van MDE de sourcecode van een volledig Struts-framework. Op basis van één model worden dus alle benodigde classes en bestanden in een Struts-applicatie gegenereerd: HTML-bestanden, JSP-bestanden, Form Beans, Action files, ActionHelper files, alsmede de Struts configuratie- en validatiebestanden en zelfs de Ant build scripts. Als laatste kan dan de businesslogica worden toegevoegd, en de applicatie is compleet. In afbeelding 6 is de code te zien die is gegenereerd voor een Struts-applicatie met een registratiepagina een welkomstpagina.



FIGUUR 6. Struts code

SCENARIO De vraag is echter, wat er nu in de praktijk gebeurt wanneer er wijzigingen in de architectuur of in de applicatie zijn. Er kunnen zich dan verschillende scenario's voordoen. Eén scenario is dat de applicatie moet worden uitgebreid of veranderd. In dit geval wordt het UML-model aangepast, en worden vervolgens opnieuw de MetaPrograms van MDE for Struts uitgevoerd. Met behulp van de MDE Life Cycle tools kan zelfs de business logica naar de nieuwste versie worden meegenomen. Een ander scenario is dat de architectuur verandert. In dat geval dient het meta-programma te worden aangepast. Voor Struts zijn dat de MDE for Struts MetaPrograms. Nadat het meta-programma is aangepast, dient het opnieuw te worden uitgevoerd en kan met MDE Life Cycle de business logica worden bijgewerkt. MDE for Struts is beschikbaar in twee versies. De MDE for Struts Standard Edition (SE) is bedoeld voor ontwikkelaars die niet van plan zijn de Struts-metaprogramma's aan te passen. De uitgebreidere versie MDE for Struts MetaProgramming Edition (ME) bevat ook de broncode van de Struts meta-programma's. Deze versie is voor ontwikkelaars die nieuwe modelleringstechnieken willen toepassen. Zij kunnen dus ook de Struts meta-programma's op basis hiervan uitbreiden.

MDE TEN OPZICHTE VAN ANDERE TOOLS Een belangrijk pluspunt van MDE zijn de uitgebreide mogelijkheden. Zo bevat het een tool voor het maken van UML-modellen, is het toegerust met de uitstekende Java-programmeeromgeving van Eclipse en kunnen meta-programma's ermee worden omgezet in source code. Een vergelijkbare tool is OptimalJ van Compuware. Een groot verschil tussen MDE en OptimalJ is dat MDE op Eclipse is gebaseerd en OptimalJ op NetBeans. Voordelen van MDE zijn de meta-programma's, die niet in OptimalJ aanwezig zijn. Tenslotte heeft MDE een prijsvoordeel. De prijs ligt een stuk lager dan die van OptimalJ. MDE for Struts en MDE for EJB zijn beschikbaar voor $799, MDE for J2EE voor $999.

> Op basis van één model worden dus alle benodigde classes en bestanden in een Struts-applicatie gegenereerd

Carlo Smits