René Sterrenburg is technisch architect bij Qurius ETX

Koppeling door ontkoppeling

HET BELANG VAN ASYNCHRONICITEIT

"Sorry, u kunt op dit moment geen bestellingen doen via Internet omdat ons backoffice-systeem uit de lucht is" of "Een moment alstublieft, het systeem is vandaag weer zo traag!" Natuurlijk zijn er veel mogelijke oorzaken van deze problemen. De onderschatte rol van asynchroniciteit is er daar één van. Dit artikel gaat in op de betekenis, de voordelen en de technieken van asynchroniciteit, maar ook de keerzijde van asynchroniciteit komt aan bod.

Sinds de opkomst van computernetwerken - en internet in het bijzonder - is er een onbedwingbare drang tot het koppelen van systemen. Dat is niet zo gek, want de technologie biedt ons daartoe de mogelijkheden. De orders van klanten die op het web worden ingevoerd, willen we automatisch ons ERP-systeem 'inschieten'. Omgekeerd willen we informatie uit oude backoffice-systemen in een nieuw jasje online presenteren. Enterprise Application Integration (EAI) staat hoog op het lijstje van veel ICT-managers. Waarom? We willen de klant de mogelijkheid geven zaken zelf te doen, hem betrekken bij het proces en hem inzage geven in de status. Aan de andere kant willen we zo min mogelijk foutgevoelige, handmatige acties in een bedrijfsproces met kostenreductie als ultieme doel.

De eenvoudigste manier om te koppelen is de directe manier. Vanuit het ene systeem wordt direct een beroep gedaan op het andere systeem, bijvoorbeeld via een webservice/remote procedure call. Hierbij wordt het verzoek meteen verwerkt en krijgen we het resultaat direct terug. Maar wat als er een kink in de kabel komt? Wat als één van de gekoppelde systemen uit de lucht is, of de hoeveelheid verzoeken simpelweg niet aankan? Wat als de bandbreedte opeens wordt dichtgeknepen? Dit is slechts een greep uit een verzameling van bekende problemen die het koppelen van systemen tot een lijdensweg kunnen maken. Het negeren van deze problemen kan leiden tot een totaalsysteem dat een langzame of abrupte dood tegemoet gaat. Er is maar één goede oplossing: ontkoppelen! Ontkoppelen betekent dat we niet direct afhankelijk zijn van een ander systeem. Voor de gebruiker lijkt het echter alsof de twee gewoon direct gekoppeld zijn. Ontkoppeling resulteert in een patroon waarbij het ene systeem (het bronsysteem) niet hoeft te wachten op de verwerking van een verzoek door het andere systeem (het doelsysteem). We spreken in dat geval over een asynchrone koppeling.

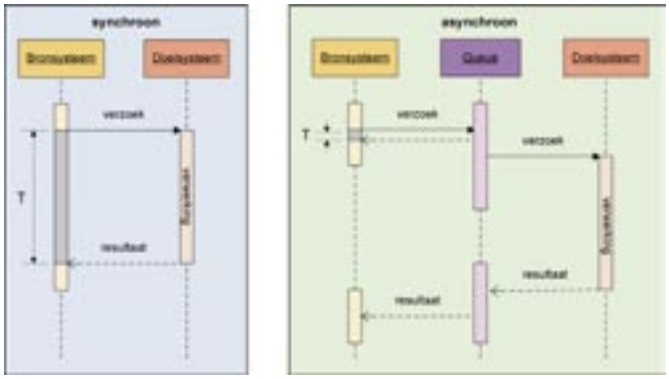
Synchroon versus asynchroon

Het overgrote deel van de software die we bouwen is synchroon. Synchroon betekent letterlijk gelijktijdig. In de ICT staat het voor een techniek waarbij een keten van verzoeken (lees 'method calls') ononderbroken en direct opeenvolgend plaatsheeft. De verwerking stopt zelfs totdat het resultaat is terugontvangen. Asynchroon daarentegen betekent 'niet gelijktijdig'. Bij deze techniek wordt onderscheid gemaakt tussen het verzoek, de uitvoering van het verzoek en het teruggeven van het resultaat. In tegenstelling tot het synchrone scenario hebben deze drie fasen niet gelijktijdig plaats. Als we spreken over het ontkoppelen van systemen, dan bedoelen we dus het koppelen van systemen volgens een asynchroon patroon. In afbeelding 1 wordt het belangrijkste verschil tussen een synchroon en een asynchroon scenario geïllustreerd.

Het is duidelijk dat interval T, waarin het bronsysteem niets anders kan doen dan wachten op antwoord, aanmerkelijk langer is in het synchrone scenario. Merk tevens op dat in het asynchrone scenario de wachtrij ('queue') een rol speelt (zie verderop in dit artikel het onderdeel 'Technieken voor asynchroniciteit').

Voordelen van asynchroniciteit

Het meest aansprekende voordeel van asynchroniciteit is natuurlijk de geringe 'response'-tijd. Het bronsysteem doet slechts een verzoek en krijgt meteen antwoord, maar dit antwoord is nog niet voorzien van een resultaat. Het verzoek wordt vervolgens op een later moment verwerkt. Dat betekent ook dat het resultaat pas op een later tijdstip beschikbaar komt. Dit leidt tot een nieuwe uitdaging: het resultaat relateren aan het oorspronkelijke verzoek (zie verderop in dit artikel het onderdeel 'De keerzijde van asynchroniciteit'). Een belangrijk ander voordeel is fouttolerantie. Door het verzoek los te koppelen van de uitvoering is het niet noodzakelijk dat het uitvoerende systeem beschikbaar is op het moment van het verzoek. Bovendien is het mogelijk dat een gelijkwaardig ander doelsysteem de taken (tijdelijk) overneemt. Een derde voordeel is schaalbaarheid. Stel dat er veel meer verzoeken komen dan het doelsysteem redelijkerwijs kan verwerken. Op dat moment kan een gelijkwaardig systeem worden ingezet om parallel verzoeken uit te voeren. Zo kunnen in dezelfde tijd meer verzoeken worden verwerkt. Een schaalbaar systeem garandeert dat je mee kunt groeien wanneer de belasting van het systeem toeneemt.
Ten slotte is er nog de invloed die je kunt uitoefenen op de belasting van het doelsysteem. Doordat het verzoek en de verwerking van elkaar gescheiden zijn, heb je invloed op de fre-



Afbeelding 1. Het verschil tussen een synchroon en asynchroon patroon

quentie waarmee verzoeken worden opgepakt en verwerkt. Ook kun je invloed uitoefenen op het aantal verzoeken dat gelijktijdig wordt opgepakt en verwerkt. Dit laatste is vooral belangrijk als je te maken hebt met een oud backoffice-systeem dat niet in staat is om meer dan één verzoek tegelijk te verwerken. Je kunt dus nauwgezet sturen hoeveel verzoeken het doelsysteem tegelijk oppakt voor verwerking, en wanneer het dat doet. De overige verzoeken blijven gewoon wachten in de wachtrij totdat ze aan de beurt zijn.

Technieken voor asynchroniciteit

Gelukkig bestaan er verschillende standaard oplossingen voor het introduceren van asynchroniciteit in een applicatie. Asynchroniciteit is onlosmakelijk verbonden met een ander begrip: de 'queue', oftewel een wachtrij (zie afbeelding 1). De verzoeken moeten immers ergens worden opgespaard totdat het doelsysteem ze kan verwerken. Technieken voor het implementeren van asynchrone koppelingen hebben dan ook met elkaar gemeen dat ze een 'persistent store' hebben die dienst doet als wachtrij.

Technieken voor asynchrone koppelingen

Hieronder volgen drie concrete technieken die kunnen helpen bij het maken van asynchrone koppelingen. Het gaat om MSMQ, Microsoft BizTalk Server en het Microsoft Asynchronous Invocation Application Block.

1) MSMQ

MSMQ staat voor Microsoft Message Queuing en is de oudste techniek van de drie. In .NET is MSMQ toegankelijk via de System.Messaging namespace. Het grote voordeel van MSMQ is de standaard functionaliteit die het biedt, zoals het prioriteren van verzoeken, transaction-awareness en time-out-handling. Codevoorbeeld 1 toont hoe je heel eenvoudig een verzoek (bericht) in een wachtrij plaatst en weer uitleest. Het voorbeeld gaat ervan uit dat er een wachtrij genaamd 'myqueue' bestaat. Deze kun je aanmaken via Computer Management -> Services and Applications -> Message Queueing. Hier zie je ook de berichten verschijnen. Voorwaarde is dat je Message Queueing Services, een standaard onderdeel van Windows NT/2000/XP, hebt geïnstalleerd.

2) Microsoft Asynchronous Invocation Application Block

Het Microsoft Asynchronous Invocation Application (AIA) Block is de recentste oplossing van de drie. Het is een standaard oplossing die Microsoft gratis aanbiedt aan de ontwikkelgemeenschap. De oplossing is uitvoerig beschreven in een 'patterns en practices'. Alle patterns en practices zijn gebaseerd op 'best practices' en standaard 'design patterns'. Een groot voordeel is dat de volledige broncode beschikbaar is en naar believen kan worden aangepast. Het AIA-block werkt met het concept van 'service agents'. Een service agent bevat de logica voor het verwerken van een verzoek. Het toevoegen van een nieuwe service agent voor een nieuw type verzoek is een kwestie van configuratie. De service agents worden - indien nodig - geactiveerd door een generieke Windows-service, genaamd de 'Remote Object Host'. Deze host is in staat om verschillende verzoeken tegelijk te verwerken door een flexibel te configureren threadpool. Een 'Monitor service' bewaakt de voortgang en grijpt in op het moment dat een verzoek te veel tijd in beslag neemt. Resultaten kunnen na verwerking eenvoudig worden opgehaald. Het AIA-block werkt op basis van Microsoft SQL Server. Het AIA-block is een eenvoudige standaard oplossing met basisfunctionaliteit voor het asynchroon verwerken van verzoeken. Op dit moment ontbreken echter nog zaken als een retry-mechanisme en een elegante foutafhandeling. Codevoorbeeld 2 toont hoe je met het AIA-block een verzoek plaatst en hoe je het resultaat van het verzoek op een later moment weer kunt opvragen. Merk op dat 'MyServiceAgentIdentifier' staat voor de service agent die het verzoek moet verwerken.

3) Microsoft BizTalk Server

Een veel zwaardere oplossing is Microsoft BizTalk Server. BizTalk is een zogenaamde message-broker en als zodanig gespecialiseerd in het centraal uitwisselen van berichten volgens flexibele procesbeschrijvingen (orchestrations). BizTalk Server is met name geschikt voor het flexibel koppelen van verschillende systemen (n-op-m koppelingen) waarbij de structuur van de uit te wisselen berichten sterk uiteen loopt. Het nadeel van BizTalk is de licentieprijs. BizTalk Server is vanaf de 2004-versie goed geïntegreerd in het .NET Framework en in Visual Studio .NET. Berichten kunnen op veel verschillende manieren aan BizTalk worden aangereikt, bijvoorbeeld via eigen C#-code, een bestand, database, FTP, HTTP of MSMQ. Daarnaast zijn vele adapters beschikbaar voor allerlei protocollen en applicaties. Bovendien kun je ook een eigen adapter schrijven voor het aanleveren of versturen van berichten. Voor meer informatie over bovenstaande technieken wordt verwezen naar het onderdeel 'Nuttige internetadressen' aan het einde van dit artikel.

Een praktijkvoorbeeld

Bovenstaande technieken kunnen ook naast elkaar worden ingezet. Als praktijkvoorbeeld nemen we een centraal advertentiedistributiesysteem. Het systeem voorziet in de behoefte van het ontvangen, opmaken en verder distribueren van advertenties via verschillende kanalen. In afbeelding 2 staat een schets van het systeem.

De systeemschets toont Microsoft BizTalk Server als het centrale advertentiedistributiesysteem, een opmaak-engine ontkoppeld via MSMQ, een bronsysteem ontkoppeld via het Asynchronous Invocation Application Block en een doelsysteem. Via het online advertentiesysteem kunnen advertenties worden opgegeven. Het AIA-block schakelt periodiek een service agent in om de advertenties via een webservice-call bij BizTalk aan te bieden. Vervolgens wordt de advertentie via MSMQ naar de opmaak-engine gestuurd waar deze wordt opgemaakt. De opmaak-engine is niet in staat meer dan één advertentie tegelijk op te maken. De opgemaakte advertentie wordt ten slotte door BizTalk naar het systeem van de drukkerij gestuurd.
Alle voordelen van asynchroniciteit komen in dit systeem bij elkaar. Om twee voorbeelden te noemen: wanneer één van deze systemen faalt heeft dat niet direct gevolgen voor de beschikbaarheid van de andere systemen (fouttolerantie). Ook is het eenvoudig een extra opmaak-engine ernaast te zetten die dezelfde wachtrij gebruikt (schaalbaarheid).

De keerzijde van asynchroniciteit

Waarom zouden we niet alle koppelingen asynchroon maken als het zo veel voordelen biedt? Helaas heeft asynchroniciteit ook een

```
// Plaatsen van een verzoek in de queue door een bronsysteem
MessageQueue queue = new MessageQueue(".\\Private$\\myqueue");
queue.Send("MyMessageBody", "MyMessageLabel");

// Uitlezen van een verzoek in de queue door een doelsysteem
string message = (string)queue.Receive().Body;
```

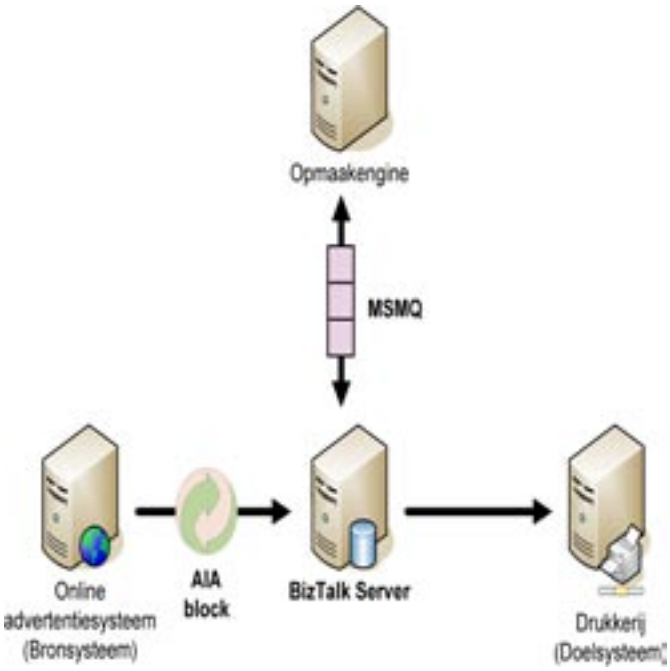
Codevoorbeeld 1.
MSMQ

```
// Plaatsen van een verzoek in de queue door het bronsysteem
AsyncRequestBatch request = new AsyncRequestBatch(10, true);
request.AddRequest("MyServiceAgentIdentifier",
   "MyUniqueMessageIdentifier", new object[]{"MyMessage"});

string requestID = AsyncRequestProcessor.SubmitRequest(request);

// Ophalen van het resultaat, na verwerking, door het bronsysteem
InvocationResult result = ResultsManager.GetResults(requestID);
```

Codevoorbeeld 2.
AIA-block



Afbeelding 2. Praktijkvoorbeeld

keerzijde. Hieronder volgen de belangrijkste uitdagingen waarmee je te maken krijgt als je kiest voor een asynchroon scenario.
De eerste uitdaging betreft het verkrijgen van het resultaat van een verzoek dat pas op een later moment wordt uitgevoerd. Hier zijn twee strategieën voor, te weten: notificatie en polling. De eerste strategie gaat ervan uit dat het doelsysteem op een geschikt moment het bronsysteem notificeert of op de hoogte brengt van het resultaat. De tweede strategie gaat ervan uit dat het bronsysteem zelf periodiek aan het doelsysteem vraagt of deze al een resultaat heeft. Beide strategieën vereisen dat het resultaat te relateren is aan het oorspronkelijke verzoek. Een gangbare manier om dat te doen is het verzoek gepaard te laten gaan van een unieke identifier die eveneens met het resultaat wordt meegestuurd. Op die manier kan het bronsysteem het resultaat matchen met het oorspronkelijke verzoek.

Een tweede uitdaging is de zogenaamde time-out. De uitvoering van het verzoek vindt plaats op een ander tijdstip. Toch wil je de periode begrenzen waarin je het resultaat kunt terug verwachten. Komt dit resultaat niet, of te laat, dan hebben we te maken met een time-out-situatie. Het systeem dat het oorspronkelijke verzoek pleegde, moet met deze situatie kunnen omgaan. Een complicerende factor is dat de status van de uitvoering voor het bronsysteem vaak niet bekend is. Het is dan ook verstandig deze statusinformatie richting het bronsysteem te ontsluiten. Zo kan in het geval van een time-out een afgewogen beslissing worden genomen over een vervolgactie. Zo'n vervolgactie zou bijvoorbeeld de herhaling van het verzoek kunnen zijn. Dit laatste resulteert automatisch in de derde uitdaging: hoe gaan we om met een dubbele ontvangst van hetzelfde verzoek? Je wilt bijvoorbeeld voorkomen dat dezelfde order tweemaal in het backoffice-systeem terechtkomt. Dit kun je bereiken door een unieke identifier mee te sturen met een verzoek. Als het doelsysteem vervolgens een verzoek ontvangt dat reeds is verwerkt, dan wordt dit verzoek genegeerd. Het doelsysteem moet dan wel de reeds verwerkte verzoeken kunnen onthouden op basis van de unieke identifiers.

Ten slotte is er de uitdaging van 'compenserende transacties'. Stel: een klant koopt iets in een webshop en dat resulteert zowel in een asynchrone banktransactie als een asynchrone voorraadafboeking. Nadat de voorraad is afgeboekt faalt de banktransactie. Nu moet de voorraad weer worden bijgeboekt. Zo'n correctie, die de effecten van een bepaald verzoek terugdraait, staat bekend als een compenserende transactie. Een systeem moet vaak extra functionaliteit bieden om compenserende transacties te kunnen ondersteunen. Al deze zaken zijn goed oplosbaar, maar resulteren uiteraard in een complexer systeem. Afhankelijk van de gekozen techniek zal het meer of minder inspanning kosten om de betreffende uitdaging het hoofd te bieden.

Asynchroniciteit in het ontwerp

Het zal duidelijk zijn dat asynchroniciteit niet iets is dat je aan het einde van het ontwikkeltraject nog even introduceert. Je zult moeten bepalen óf en zo ja welke asynchrone techniek(en) je wilt toepassen. Kies je voor asynchroniciteit, dan moet je ook een structurele oplossing bieden voor bovenstaande uitdagingen. Maak daarbij niet de fout alleen over ontkoppeling en asynchroniciteit na te denken bij koppelingen die de systeemgrens overschrijden. Binnen jouw systeemgrens is het minstens zo belangrijk hierover een weloverwogen beslissing te nemen. Laat het daarom een bewuste keuze zijn in ieder architectuurontwerp.

Paradox

Koppeling leidt vaak tot een paradoxale ontkoppeling. Dit resulteert in een asynchroon scenario met voordelen als een geringe responstijd, grote schaalbaarheid en hoge fouttolerantie. Er zijn verschillende .NET-technieken die kunnen helpen bij de implemenntatie van asynchroniciteit. Asynchroniciteit stelt ons echter ook voor een aantal uitdagingen. Zorg er daarom voor dat asynchroniciteit al een rol speelt in het architectuurontwerp!

René Sterrenburg
(renes@quriusetx.nl) is technisch architect bij Qurius ETX (http://www.quriusetx.nl)

Nuttige internetadressen

MSMQ: http://msdn.microsoft.com/library/en-us/dnbda/html/bdadotnetasync1.asp
BizTalk: http://msdn.microsoft.com/bpi/
Biztalk adapters: http://www.microsoft.com/biztalk/evaluation/adapter/default.asp
Aynchronous Invocation Application Block: http://msdn.microsoft.com/library/en-us/dnpag/html/PAIBlock.asp
Pattern & practices: http://www.microsoft.com/resources/practices/default.mspx
Microsoft architectuurcenter: http://msdn.microsoft.com/architecture/
Architecture Webcasts: http://msdn.microsoft.com/architecture/community/webcasts/default.aspx