FACULTEIT ECONOMISCHE EN
TOEGEPASTE ECONOMISCHE WETENSCHAPPEN
DEPARTEMENT TOEGEPASTE ECONOMISCHE WETENSCHAPPEN

KATHOLIEKE
UNIVERSITEIT
LEUVEN

WEB SERVICE ORCHESTRATIE EN DE BESPREKING VAN HET BPEL4WS-PROTOCOL

Steven Dierick

Verhandeling aangeboden tot
het behalen van de graad van
handelsingenieur in de
beleidsinformatica

Promotor: Prof. Dr. J. Vandenbulcke

- 2004 -

FACULTEIT ECONOMISCHE EN
TOEGEPASTE ECONOMISCHE WETENSCHAPPEN
DEPARTEMENT TOEGEPASTE ECONOMISCHE WETENSCHAPPEN

KATHOLIEKE
UNIVERSITEIT
LEUVEN

Steven Dierick

Web Service Orchestratie en de bespreking van het BPEL4WS-protocol

Korte Inhoud Verhandeling:

Deze verhandeling onderzoekt in welke mate het *Business Process Execution Language for Web Services* (*BPEL4WS*) protocol kan gebruikt worden bij het orchestreren van Web services tot een bedrijfsproces. We bekijken de mogelijkheden van het protocol om bedrijfsprocessen te modelleren en uit te voeren, hoe het protocol samenwerkt met andere Web service-standaarden en hoe het protocol zou kunnen worden uitgebreid en verbeterd. We bespreken ook producten van enkele grote vendors die BPEL4WS-ondersteuning aanbieden.

Promotor: Prof. Dr. J. Vandenbulcke

- 2004 -

Dankwoord

Vooraleer de uiteenzetting te beginnen, wil ik kort een aantal mensen bedanken voor hun bijdrage tot deze verhandeling.

Allereerst zou ik mijn promotor, Professor Dr. Jacques Vandenbulcke willen danken voor de begeleiding en hulp bij het vormgeven van deze tekst. Vervolgens wil ik mijn werkleider, Frank Goethals, bedanken voor zijn bereidwilligheid om teksten na te lezen en de steeds snelle reacties. Hun nuttige raadgevingen en opmerkingen hielpen mij deze eindverhandeling trapsgewijs te verbeteren en uiteindelijk ook te realiseren.

Vervolgens een speciaal woord van dank aan mijn ouders, omdat ze me de kans hebben geven om verder te studeren en me al die jaren hebben gesteund.

Tot slot ook een woord van dank aan mijn zussen, vrienden en collega's studenten, in wiens gezelschap het aangenaam verpozen was tussen het schrijven door.

Steven Dierick

Inhoudsopgave

Algemene Inleiding

Onder het huidige competitieve bedrijfsklimaat is het belangrijk dat ondernemingen snel kunnen reageren op veranderende bedrijfsomstandigheden. Flexibele en aanpasbare IT-systemen zijn de sleutel tot economisch succes. De tijd die nodig is om systemen aan te passen en het budget dat voor handen is, zijn daarbij de twee voornaamste beperkende factoren. Integratie van de verschillende systemen, zowel binnen als buiten de organisatorische grenzen, komt dan ook steeds meer in de belangstelling te staan.

Web services spelen een steeds belangrijker rol in deze integratie. Web services zorgen ervoor dat systemen, applicaties en gegevensbronnen op een flexibele en op standaarden gebaseerde manier met elkaar kunnen interageren via de bestaande infrastructuur van het Internet. Bestaande assets, zoals legacy applicaties, kunnen op deze manier als Web services beschikbaar worden gesteld. Verschillende individuele Web services kunnen vervolgens worden gecombineerd tot geautomatiseerde bedrijfsprocessen. Door deze componentengebaseerde aanpak kunnen services gemakkelijk hergebruikt of hergecombineerd worden.

Om verschillende individuele Web services te orchestreren tot geautomatiseerde bedrijfsprocessen, is er nood aan een standaard om de integratielogica te definiëren. BPEL4WS (Business Process Execution Language for Web Services) is de standaard voor Web service-orchestratie die we in deze verhandeling zullen bespreken.

In het eerste hoofdstuk staan we stil bij de Web service-architectuur. Daarbij leggen we uit wat Web services nu juist zijn, introduceren we het begrip Service Oriented Architecture (SOA) en bespreken we een aantal algemeen aanvaarde Web service-standaarden. Vervolgens onderzoeken we welke rol Web services spelen in de bedrijfsintegratie en hoe Web service-orchestratie daarbij komt kijken. We eindigen het hoofdstuk met een overzicht van enkele belangrijke compositietalen, waaronder BPEL4WS.

In hoofdstuk twee wordt de BPEL4WS-standaard behandeld. We leggen uit waarvoor BPEL4WS kan worden gebruikt en verklaren de twee procesbenaderingen (uitvoerbare en abstracte processen). Het grootste deel van het hoofdstuk is echter gericht op de

bespreking van de volledige BPEL4WS 1.1 specificatie. De belangrijkste structuren en concepten uit deze specificatie worden aangehaald en geïllustreerd met een voorbeeld. We sluiten het hoofdstuk af met de bespreking van twee standaarden die BPEL4WS aanvullen met coördinatievoorzieningen en transactiemogelijkheden.

Het doel van het derde hoofdstuk is de BPEL4WS-standaard te beoordelen. We beginnen met het analyseren van de tien doelstellingen die werden gebruikt om de specificatie op te stellen en kijken of deze doelstellingen zijn gerealiseerd. Vervolgens houden we een analyse op hoog niveau over de bruikbaarheid van beide procesbenaderingen en wordt onderzocht hoe de specificatie verder kan worden uitgebreid om bijkomende functionaliteit te ondersteunen. Tot slot voeren we een analyse op laag niveau om te bepalen in hoever BPEL4WS in staat is om verschillende workflow - en communicatiepatronen te modelleren.

Het vierde en laatste hoofdstuk onderzoekt in welke mate IBM en Microsoft BPEL4WS-ondersteuning bieden in hun producten. Het WebSphere platform van IBM en de BizTalk Server van Microsoft worden in dit hoofdstuk besproken.

Hoofdstuk 1 Web services

Deze verhandeling onderzoekt in hoever de orchestratie van Web services tot stand kan komen met behulp van het BPEL4WS-protocol. Voordat wordt ingegaan op het concept van Web service-orchestratie en de bespreking van het BPEL4WS-protocol, staan we echter stil bij wat Web services zijn en in welke context we deze kunnen plaatsten. We beginnen met een definitie van het concept Web service (paragraaf 1.1) en bespreken vervolgens de belangrijkste aspecten van de Web service-architectuur (paragraaf 1.2). In de laatste paragraaf van dit hoofdstuk introduceren we het begrip Web service-compositie als basis van het volgende hoofdstuk, de bespreking van BPEL4WS.

1.1 Wat zijn Web services?

De Web service-technologie komt steeds meer in de verf te staan. Wat enkele jaren geleden nog werd beschouwd als een hype, wordt nu aanzien als een mogelijke oplossing voor dynamische business-interacties over het Internet.

Vele bedrijven willen werk maken van de integratie van hun systemen. Deze systeemintegratie vindt zowel plaats binnen het bedrijf (EAI of Enterprise Application Integration), als over verschillende bedrijven heen (B2Bi of Business to Business Integration). Deze integratie wordt echter aanzienlijk bemoeilijkt doordat ieder systeem gebouwd is volgens zijn eigen standaarden. Web services bieden nu een eenvoudig, flexibel, op standaarden gebaseerd model, voor het koppelen van applicaties via de bestaande infrastructuur van het Internet.

Er bestaat geen eenduidigheid over wat Web services nu exact zijn. Er circuleren verschillende definities, elk met andere klemtonen en invalshoeken:

Volgens het W3C (World Wide Web Consortium) [W3C WSA '04]:
"A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."

Volgens IBM:
"Web services are self-contained, self-describing, modular applications that can be published, located and invoked across the Web. Web services perform callable functions that can be anything from a simple request to complicated business processes."

In de volgende paragraaf komen de aspecten aan bod die in deze definities vervat zijn.

1.2 Web Service Architecture (WSA)

De Web Service Architecture of WSA, opgesteld door het W3C, heeft tot doel een model en context aan te bieden waarin Web services kunnen worden begrepen samen met de relaties tussen de verschillende specificaties en technologieën [W3C WSA '04].

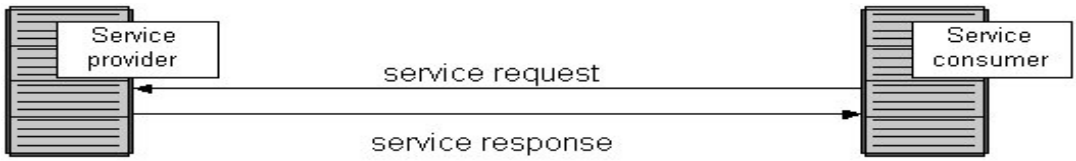
1.2.1 Servicegeoriënteerde architectuur

De WSA is een onderdeel van de Service Oriented Architecture (SOA). De service-georiënteerde aanpak is de (voorlopig) laatste stap in de evolutie van het herbruikbaar maken van softwarecomponenten. De eerste stap was programma's opsplitsen in functies door middel van functionele decompositie. Deze methode evolueerde verder naar een objectgeoriënteerde architectuur en uiteindelijk tot een servicegeoriënteerde architectuur [Leyman, Roller & Schmidt '02].

Het object combineert data en functionaliteit in een unit. Om het object te kunnen gebruiken is er alleen nood aan de beschrijving van de signatuur, want de semantiek van de individuele klassen is verondersteld gekend. Bij services echter wordt de semantiek beschreven en gepubliceerd zodat iedereen, zowel mens als machine, de service kan lokaliseren en aanroepen [Leyman, Roller & Schmidt '02]. De servicegeoriënteerde architectuur is in wezen een verzameling van services die met elkaar communiceren. Deze communicatie kan bestaan uit een simpele gegevensoverdracht of uit meerdere services die een taak coördineren.

De volgende figuur (Figuur 1) toont de basis van een servicegeoriënteerde architectuur. Deze bestaat uit een serviceconsument die een verzoekbericht stuurt voor een bepaalde service naar de serviceaanbieder. De aanbieder stuurt dan op zijn beurt een antwoord

naar de consument. Beide communicatieverbindingen zijn zo gedefinieerd dat ze begrijpbaar zijn voor zowel de consument als voor de aanbieder [Service-Architecture.com '03].



Figuur 1 Service Model (Bron: Service-Architecture.com '03)

1.2.2 Gedistribueerd componentgebaseerd systeem

SOA is een gedistribueerd systeem. Een gedistribueerd systeem bestaat uit afzonderlijke softwarecomponenten die samen moeten werken om tot een vooropgestelde functionaliteit te komen [W3C WSA '04]. Deze samenstelbaarheid is een ontwerp-principe dat er voor zorgt dat het mogelijk is om rijkere functionaliteit te bouwen vanuit eenvoudige componenten. Deze componenten moeten afzonderlijke, onafhankelijke capaciteiten hebben, maar kunnen ook worden samengesteld met andere componenten om complexere oplossingen te creëren. Gedistribueerde componenten bevinden zich vaak in verschillende omgevingen. De componenten moeten daardoor met elkaar communiceren aan de hand van hardware/software protocol stacks, die minder betrouwbaar zijn dan het onmiddellijke aanroepen van code en het gebruik van gedeeld geheugen. Bij het implementeren van een gedistribueerd systeem zal er dus rekening moeten gehouden worden met onvoorspelbare vertragingen, concurrency problemen en onverwachtse onderbrekingen in delen van het systeem [Kendal, Waldo, Wollrath & Wyant '94].

Eén van de voordelen van een componentgebaseerde ontwikkeling is dat hergebruik van services wordt aangespoord. Hierdoor kan de ontwikkelingskost en de tijd nodig om het op de markt te brengen worden gereduceerd.

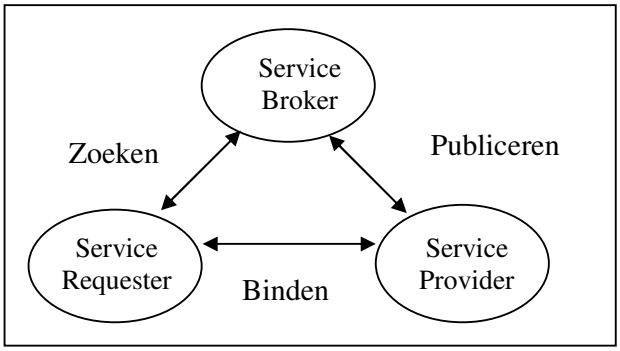
1.2.3 Web service-architectuur

De Web service-architectuur is een gedistribueerde, componentgebaseerde, service-georiënteerde architectuur. Deze aanpak is niet nieuw. Eerdere pogingen zoals CORBA

(Common Object Request Broker Architecture), Java RMI (Remote Method Invocation) en DCOM (Distibuted Common Object Model) hebben systemen opgebracht waar de koppelingen tussen verschillende componenten te strak waren om doeltreffende B2B-interactie te verkrijgen. Deze aanpak vereist te veel afspraken en gedeelde context onder de business-systemen om betrouwbaar en efficiënt te kunnen werken [Bloomberg '01].

De huidige tendens is om te evolueren van sterk gekoppelde monolithische systemen naar losgekoppelde en dynamisch gebonden componenten. In een tijd waar flexibiliteit centraal staat en alles onderhevig is aan veranderingen, zullen systemen die op deze principes zijn gebaseerd, efficiënter zijn in B2B-omgevingen. Alle componenten in het systeem zijn services. De Web services vertegenwoordigen een black-box functie. De manier waarop functionaliteit wordt geleverd wordt verborgen en de Web services tonen enkel een interface met de buitenwereld. Hierdoor hoeven softwareontwikkelaars zich geen zorgen meer te maken over de manier waarop de Web services van anderen zijn geïmplementeerd.

Met de Web service-architectuur kunnen drie rollen worden onderkend: de service requestor, de service provider en de service broker. Er zijn ook drie basisoperaties: publiceren, zoeken en binden (zie Figuur 2).

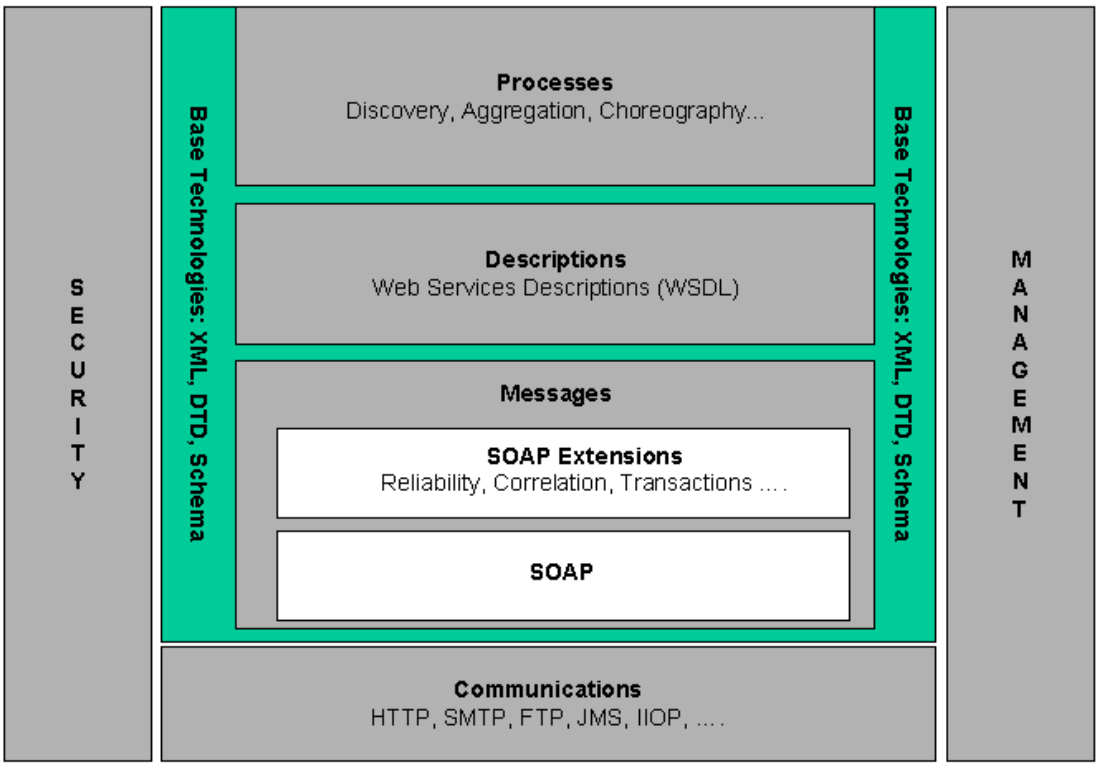


Figuur 2 Web service-architectuur: rollen en operaties (Bron: Gottschalk '00)

De service provider publiceert de beschrijving van zijn Web service bij een service broker ofwel bij de service requestor zelf. Met de zoekoperatie haalt de requestor de beschrijving onmiddellijk op of wordt de service broker aangesproken om de beschrijving van de gewenste service op te halen. Wanneer de requestor de service van de provider wil aanroepen, zal de bindingsinformatie in de servicebeschrijving worden

gebruikt. Op deze manier kan de requestor de service lokaliseren, contacteren en gebruiken [Gottschalk ‘00].

De Web service-architectuur bestaat uit verschillende met elkaar in verband staande standaarden. Algemeen spreekt men van een elkaar aanvullende stack van standaarden of protocollen (zie Figuur 3).



Figuur 3 Web services-architectuur stack (bron: W3C WSA ‘04)

De basis van de Web service-architectuur is XML (eXtensible Markup Language). XML is het algemeen geaccepteerd formaat om gegevens gestructureerd op te slaan in een tekstdocument. Om de structuur te definiëren, wordt er gebruik gemaakt van tags. Deze tags beschrijven de data en voegen er dus semantiek aan toe. XML is vendor, -platform -en taalonafhankelijk. XML is dan ook een fundamenteel onderdeel van bijna elk ander protocol in de stack [W3C WSA ‘04].

De eerste laag in de stack is de transportlaag. Het meest gangbare protocol is HTTP (Hyper Text Transport Protocol) over TCP/IP. Het grote voordeel is dat HTTP kan

gebruikt worden op elke computer, wat niet kon gezegd worden van andere protocollen als DCOM en CORBA. Ook andere protocollen kunnen worden gebruikt, zoals SMTP, FTP en IIOP [W3C WSA ‘04].

De tweede laag die behoort tot de kern van de Web service-stack, staat in voor de informatie-uitwisseling tussen applicaties in een gedistribueerde en gedecentraliseerde omgeving. SOAP (Simple Object Access Model) is het XML-communicatieprotocol voor Web services [W3C SOAP‘03].

De derde laag bevat de beschrijving van de Web services. Het protocol WSDL (Web Service Description Language) beschrijft wat een Web service kan doen, waar ze zich bevindt en hoe ze kan worden aangeroepen. WSDL definieert de interface en mechanismen voor service-interacties [W3C WSDL ‘04].

UDDI (Universal Description Descovery and Integration) omvat een aantal protocollen om Web services op een dynamische manier te zoeken en te gebruiken. Het is ook een publieke verzamelplaats voor de registratie van Web services. UDDI kan gezien worden als een soort Gouden Gids voor Web services [UDDI.org ‘04].

Naast deze protocollen worden er constant nieuwe voorgesteld. Vele aspecten blijken namelijk nog niet volledig behandeld. Beveiliging, transacties, betrouwbaarheid, processen, management en Quality of Service (QoS) zijn enkele domeinen die nog steeds worden onderzocht en waarin nog niet veel overeenstemming is bereikt.

In deze verhandeling zullen we stil staan bij het domein van de processen. We gaan meer bepaald in op een standaard voor beschrijving van de integratielogica en procesautomatisering tussen Web services. In de volgende paragraaf gaan we dieper in op het concept van bedrijfsintegratie en bekijken we welke rol Web services kunnen spelen binnen deze integratie.

1.3 Web service-compositie

1.3.1 Enterprise Integration

De competitieve aard van het huidige bedrijfsklimaat zet ondernemingen onder een immense druk. Om te kunnen overleven, is het noodzakelijk dat bedrijven hun systemen, applicaties en gegevensbronnen integreren in een efficiënt, effectief en snel reagerend systeem. Enterprise Integration (EI) staat dan ook terecht in de belangstelling.

Een eerste aanpak om bedrijfsintegratie te realiseren is ERP (Enterprice Resource Planning). Een ERP-systeem is een totaaloplossing die probeert alle functies van het bedrijf te integreren. Alle onsamenhangende informatiesystemen worden op deze manier ondergebracht in één uniform systeem. Het implementeren van ERP gaat gepaard met hoge uitgaven in zowel tijd, geld als inspanning. Niet elke onderneming is ook geschikt voor een ERP-implementatie, daar er soms ingrijpende veranderingen moeten worden aangebracht aan de bestaande bedrijfsprocessen. Een ERP-systeem is niet alles omvattend zodat er soms nood is aan aanvullende systemen, zoals CRM (Customer Relationship Management) en SCM (Supply-Chain Management). Op deze manier krijgen we eilanden van integratie [Lee, Siau, Hong '03].

Een alternatieve aanpak voor bedrijfsintegratie is EAI (Enterprise Application Integration). EAI maakt geen gebruik van gestandaardiseerde processen, maar doet een beroep op middleware die als brug fungeert tussen de verschillende bestaande applicaties. De kern van EAI-producten bestaat uit een integration broker die als tussenpersoon optreedt tussen de verschillende systemen. Wanneer een systeem met een ander systeem wil communiceren, worden de gegevens verstuurd via een integratieadapter naar de broker en dan via een tweede adapter naar het doelsysteem. Elk systeem heeft dus zijn eigen adapter. Op deze manier kunnen alle applicaties, ook ERP-systemen, communiceren met elkaar door middel van een gemeenschappelijke interface-laag. EAI heeft betrekking op de integratie binnen de grenzen van de organisatie. B2Bi of Business-to-business integration breidt de integratie verder uit naar klanten, leveranciers en andere partners [Lee, Siau, Hong '03].

De Web service-technologie is snel aan het evolueren om te voldoen aan de complexe behoeftes van de bedrijfswereld. Bedrijfsintegratie met behulp van Web services kan nu

dezelfde mogelijkheden bieden als EAI-oplossingen en zal daarbij de kosten en de complexiteit aanzienlijk reduceren. Proprietary EAI-systemen zijn vaak te duur in ontwikkeling, onderhoud en uitbreiding. Deze traditionele EAI oplossingen schieten te kort in flexibiliteit en aanpasbaarheid door de proprietary integratieadapters. EAI met Web services biedt nu echter een platformonafhankelijk en op standaarden gebaseerde oplossing voor bedrijfsintegratie. Ook in een B2B omgeving zal de integratie via Web services efficiënter verlopen [Townsend, Cairns, Schittko '03].

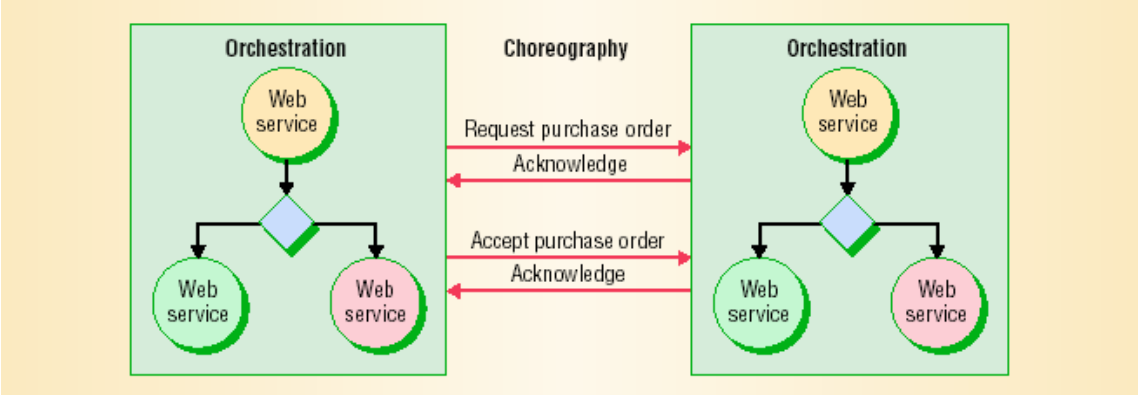
De grenzen tussen de interne (EAI) en externe (B2Bi) integratie worden steeds kleiner. Door outsourcing zijn vele interne bedrijfsfuncties immers niet meer fysiek aanwezig in de onderneming zelf. Bedrijven werken ook steeds meer samen met hun partners. Er wordt dus meer en meer gedacht in termen van bedrijfsprocessen die een virtuele onderneming overspannen. Bedrijven realiseren zich dat economisch succes sterk gebaseerd is op de efficiëntie en de effectiviteit van hun bedrijfsprocessen. BPM (Business Process Management) is de volgende stap in Enterprise Integration. BPM neemt de voordelen uit zowel EAI als workflowsystemen en biedt zo de mogelijkheden om processen te integreren, te automatiseren, te beheren en te optimaliseren.

Web services worden meer en meer aanvaard en gebruikt als de gemeenschappelijke integratietechnologie, zowel binnen als buiten de organisatorische grenzen. De volgende stap in de servicegeoriënteerde architectuur is de compositie van verschillende individuele Web services tot een bedrijfsproces. Er is dus nood aan een uitbreiding op de Web service-protocolstack om integratielogica en automatisering van bedrijfs-processen te standaardiseren. Deze standaarden worden Web service-compostietalen genoemd. Orchestratie en choreografie beschrijven twee aspecten van Web service-compositie. Hoewel ze vaak door elkaar gebruikt worden en grotendeels overlappen, is er toch een nuance verschil.

Orchestratie bestaat uit de coördinatie van interacties tussen de verschillende Web services. Meer bepaald worden de bedrijfslogica en de volgorde van de interacties beschreven in een privaat en uitvoerbaar proces. Orchestratie vertrekt altijd vanuit het perspectief van één van de partijen. Eén enkele partij controleert dus alle procesinteracties [Peltz '03 a, b, c].

Choreografie beschrijft enkel het observeerbare gedrag, namelijk de publieke uitwisseling van berichten tussen verschillende partijen. Bij choreografie is er geen sprake van een uitvoerbaar proces, maar spreekt men van een abstract proces. Choreografie is van natuur meer gericht op samenwerking tussen de verschillende partijen, in tegenstelling tot orchestratie waar één partij alles controleert. In een choreografie beschrijft elke partij van het proces welke rol ze speelt in de interactie [Peltz '03 a, b, c].

Figuur 4 toont de relatie tussen orchestratie en choreografie.



Figuur 4 Orchestratie en choreografie (bron Peltz '03 c),

Voor verdere verduidelijking van de twee begrippen kunnen we gebruik maken van een metafoor van een dans met verschillende dansers. De choreografie is het samenspel van verschillende dansers tot een geheel. De choreografie beschrijft welke rol de danser speelt in het geheel. De orchestratie daarentegen focust niet op de gezamenlijke samenwerking van alle dansers, maar bij orchestratie staat de danser zelf centraal. De danser moet zijn eigen danspassen op elkaar afstemmen en eventueel rekening houden met dansers waarmee hij moet interageren. Bij orchestratie heeft de danser de volledige controle. Bij choreografie is de danser slechts onderdeel van het geheel.

1.3.2 Web service-compositietalen

De eerste generatie Web services waren voornamelijk eenvoudige interface-applicaties bovenop geïsoleerde code en functionaliteit. Enkele voorbeelden zijn het omzetten van munteenheden, creditkaartautorisatie en simpele berekeningen [Fischer '02]. Dit was

voldoende voor de integratie van sommige interne applicaties, maar niet om complete geautomatiseerde bedrijfsprocessen te ondersteunen. Dit vereist namelijk de mogelijkheid om workflows, transactiebeheer, foutenafhandeling, beveiliging en andere kritische informatie gerelateerd aan de context van de bedrijfsprocessen te specificeren. Er is dus nood aan een standaard die nauw samenwerkt met de bestaande Web service-protocollen en het mogelijk maakt om processen te modelleren en uit te voeren.

Bij de ontwikkeling van een standaard voor bedrijfsprocessen, zouden volgende aspecten in acht genomen moeten worden [O'Riordan '02]:

- **Op samenwerking gebaseerde procesmodellen**

  Ervaring in zowel EAI als B2B procesmodellering heeft geleid tot een toenemend gebruik van op samenwerking gebaseerde modellen. In deze modellen worden de verschillende activiteiten voorgesteld samen met hun volgorderelaties en eventuele condities. De processen worden beschreven als een verzameling van samenwerkingsrelaties tussen verschillende deelnemers, zoals organisaties, applicaties, werknemers en andere processen. De algemene beschrijving van de activiteiten en het definiëren van bedrijfsprocessen, kunnen worden voorgesteld door het UML (Unified Modelling Language) Activity Diagram. Een compositietaal zou nu moeten overwegen kenmerken van deze modellen over te nemen.

- **Workflow**

  Workflow is de automatisering van een proces. Het definieert hoe de verschillende partijen in een proces samenwerken om een proces van begin tot einde uit te voeren. De meeste workflowstandaarden ondersteunen subprocessen, zodat activiteiten binnen een workflow kunnen worden geïmplementeerd als een afzonderlijke workflow. Men kan workflow opsplitsen in een controlestroom en een gegevens-stroom. De controlestroom bepaalt de volgorde waarin de verschillende taken worden uitgevoerd. De gegevensstroom beschrijft welke gegevens tussen de verschillende activiteiten worden uitgewisseld.

- **Transactiemanagement**

  Een transactie is een reeks van toestandswijzigingen of activiteiten die als een eenheid moeten worden uitgevoerd [Rollman, Cox '03]. Traditionele

transactiesystemen voldoen aan de ACID-eigenschappen. ACID staat voor Atomicity, Consistency, Isolation en Durability. Een transactie die voldoet aan deze eigenschappen wordt als een ondeelbaar geheel behandeld (=atomicity). Er zijn dan ook slechts twee uitkomsten, het geheel is succesvol en goedgekeurd (committed) of er is iets fout gelopen (aborted) en alle wijzigingen worden ongedaan gemaakt (rolled back). De transactie garandeert ook de consistentie van de gegevens (=consistency). Indien er een fout is opgetreden moeten alle voorgaande wijzigingen ongedaan gemaakt worden (rollback) en de gegevens in hun oorspronkelijke staat terug gebracht worden. Op deze manier wordt ervoor gezorgd dat de gegevens niet vervuild raken. Als meerdere transacties simultaan worden uitgevoerd, moet de uitkomst van elke transactie overeenkomen met de uitkomst indien ze afzonderlijk zouden worden uitgevoerd (=isolation). Verschillende processen mogen dus niet op ongewenste wijze interfereren. De methode om dit te verwezenlijken is 'locking'. Bij het succesvol afhandelen van een transactie moeten alle activiteiten zijn uitgevoerd en moeten alle wijzigingen permanent worden gemaakt (=durability) [Vandenbulcke, Lemahieu '00].

Bij Web services-orchestratie is er nood aan langdurige en gedistribueerde business-transacties. Traditionele transacties zijn daar meestal niet geschikt voor, daar het inefficiënt is om gegevens voor een langere periode te vergrendelen. De isolation levels zullen dus minder strikt moeten worden toegepast bij langdurige transacties. De langdurige transacties worden vaak door middel van scopes opgesplitst in kleinere transacties om zo sneller resources vrij te kunnen geven. In tegenstelling tot de rollback van traditionele transacties worden er nu compensatie-activiteiten uitgevoerd om het systeem terug in een consistente toestand te brengen. Ook omwille van het feit dat Web services losgekoppelde systemen zijn die geen gegevens, plaats en administratie delen, wordt het gebruik van traditionele transacties bemoeilijkt [Bunting, Chapman, Hurley, Little, Mischkinsky, Newcomer, Webber, Swenson '03], [Rollman, Cox '03].

- **Foutafhandeling**

  Als er tijdens het verloop van een bedrijfsproces zich een onverwachte gebeurtenis voordoet, is het belangrijk dat de geschikte herstelacties worden ondernomen. Met compensatie-acties kunnen eerder uitgevoerde activiteiten ongedaan gemaakt

worden. Foutafhandeling wordt gebruikt wanneer er tijdens de uitvoering van een scope een fout wordt opgeworpen. De scope zal de fout dan proberen af te handelen zodat het proces verder kan worden uitgevoerd. Een fout heeft dus niet zoals bij ACID-transacties onmiddellijk een rollback tot gevolg.

- **Service interface beschrijving**

  De interface van een Web service wordt beschreven zodat een gebruiker weet hoe de Web service moet worden aangeroepen. Bij de compositie van meerdere Web services tot een proces moet er opnieuw een mogelijkheid zijn om de interface te beschrijven. De compositie op zich is namelijk ook een Web service. De interface wordt net zoals de beschrijving van enkelvoudige Web services gedefinieerd in een WSDL-document. Op die manier wordt er abstractie gemaakt van het feit of de Web service enkelvoudig dan wel samengesteld is.

- **Beveiliging en betrouwbaarheid van berichten**

  Voor B2B-processen is er betrouwbare en veilige berichtenoverdracht nodig. Wanneer twee partijen berichten uitwisselen moet deze communicatie worden beveiligd zodat een derde partij het bericht niet kan lezen (=interception). Een bericht mag onderweg niet worden aangepast (=alteration) en een bericht wordt ook verwacht aan te komen indien het werd verzonden (=reliability). Tenslotte is er nog het ‘repudiation’ probleem waarbij een verzender ontkent een bericht te hebben gestuurd of een ontvanger ontkent een bericht te hebben ontvangen [Maeda, Nomura, Hara ‘03].

- **Audit trail**

  Het is doorgaans zeer belangrijk voor wettelijke doeleinden dat er een audit trail van bedrijfstransacties wordt bijgehouden. Dit betekent dat een zakenpartner onmogelijk kan beweren dat een transactie niet geaccepteerd is geweest wanneer dit wel het geval was (repudiation probleem). Het bijhouden van de historiek van gegevensuitwisseling kan ook worden gebruikt bij reconstructie van bepaalde gebeurtenissen om zo fouten en onregelmatigheden op het spoor te komen.

- **Uitvoering**

  Publieke processen beschrijven alleen hoe de informatie tussen organisaties zou moeten stromen. Om de volledig automatische uitvoering van een bedrijfsproces binnen een organisatie te doen, moet de complete informatiestroom zowel binnen de organisatie als over de firewall heen gespecificeerd worden. Dat vereist dat het procesmodel zowel de publieke activiteiten als de private volledig beschrijft.

Enkele van de meest gekende compositiestandaarden zijn BPEL, WSCI en BPML [Peltz '03 b].

BPEL4WS of BPEL

Business Process Execution Language for Web Services (BPEL4WS of BPEL) is geïntroduceerd door IBM, BEA Systems en Microsoft. BPEL4WS combineert IBM's Web Services Flow Language en Microsoft's XLANG. BPEL is ondersteund door twee aanvullende specificaties, namelijk WS-Transaction en WS-Coordination.

BPEL is een standaard voor het orchestreren van Web services tot een proces. BPEL heeft als doel de integratielogica en procesautomatisering tussen Web services te standaardiseren.

BPEL is sterk gerelateerd met de WSDL-specificaties en beschrijft hoe WSDL-operaties elkaar moeten opvolgen. BPEL ondersteunt zowel uitvoerbare bedrijfsprocessen als abstracte bedrijfsprocessen. Abstracte processen specificeren het publieke gedrag zonder al te veel details. Uitvoerbare processen bezitten genoeg detail zodat ze kunnen worden uitgevoerd door een BPEL-engine. Uitvoerbare processen modelleren de orchestratie tussen Web services, terwijl de abstracte processen zorgen voor de choreografie [Peltz '03 b, c], [Dumas, ter Hofstede, van der Aalst '03] (Zie '2.2 Wat is BPEL4WS?' voor een meer precieze definitie van de twee procestypes).

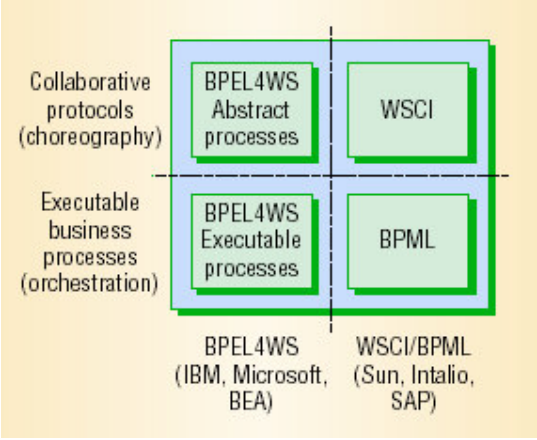
WSCI

De Web Service Choreography Interface (WSCI) is een uitbreiding op de WSDL en beschrijft de samenwerking tussen Web services. WSCI is gezamenlijk ontwikkeld door BEA Systems, Intalio, SAP AG, en Sun. De taal wordt momenteel gestandaardiseerd door het W3C.

WSCI is een choreografietaal die de berichtenuitwisseling beschrijft tussen verschillende Web services. Een belangrijk aspect van WSCI is dat het alleen het observeerbare gedrag tussen Web services definieert. Het behandelt dus niet de uitvoering van interne bedrijfsprocessen zoals BPEL. WSCI beschrijft voor elke deelnemer in de berichtenuitwisseling een WSCI interface en definieert een globaal model dat bepaalt hoe deze interfaces interageren. In WSCI is er geen afzonderlijk controleproces dat de interactie beheert. Waar BPEL voornamelijk de orchestratie behandelt, focust WSCI zich vooral op de choreografie [Peltz '03 b, c].

BPML

BPML (Business Process Modeling Language) is een specificatie van BPMI.org (Business Process Management Initiative Organisation). Het is een metataal voor het modelleren van bedrijfsprocessen. BPML was in de eerste plaats ontworpen om processen te ondersteunen die door een procesmanagementsysteem konden worden uitgevoerd. Nu is er ook WSCI ondersteuning toegevoegd. WSCI wordt gebruikt om de publieke interacties te beschrijven, terwijl BPML instaat voor de private implementaties [Peltz '03 b, c].



Figuur 5 Relatie tussen BPEL, WSCI, BPML (bron: Peltz '03b)

Figuur 5 toont een overzicht van de drie besproken standaarden. BPEL, WSCI en BPML hebben elk een iets anders aanpak voor de compositie van Web services. BPEL en BPML richten zich vooral naar de orchestratie van Web services. Ze beschrijven private uitvoerbare bedrijfsprocessen. WSCI legt zich echter toe op de choreografie, de publieke uitwisseling van berichten. Ook BPEL biedt choreografie aan in de vorm van

abstracte processen. Waar BPEL4WS zelf in staat voor zowel orchestratie en choreografie, werken WSCI en BPML daarvoor samen.

1.4 Conclusie

De Web service technologie ondergaat een continue ontwikkeling. De eerste stappen om eenvoudige Web services te creëren zijn reeds gezet met protocollen als SOAP en WSDL. Om complexere oplossingen te bekomen, is er nood aan compositiestandaarden gebouwd bovenop deze basisprotocollen. Er zijn al verschillende compositietalen in ontwikkeling elk met hun specifieke voor - en nadelen. De volgende hoofdstukken behandelen één van deze compositietalen, namelijk BPEL4WS.

We bespreken BPEL4WS omdat het wordt beschouwd als de *de facto* standaard voor XML-gebaseerde bedrijfsprocesmodellering. Het gewicht van de drie vendors die de specificatie ontwikkelden en de beslissing om de specificatie vrij te geven aan het standaardisatieorgaan OASIS, heeft er immers toe geleid dat BPEL4WS een sterke positie heeft ingenomen tussen de andere compositietalen.

In het volgende hoofdstuk bespreken we de volledige BPEL4WS-specificatie om zicht te krijgen op de mogelijkheden van BPEL4WS. We beschrijven in het derde hoofdstuk de belangrijkste troeven en beperkingen van de BPEL4WS-compositietaal. Ten slotte zullen we in het laatste hoofdstuk enkele producten bespreken die BPEL4WS ondersteunen.

Hoofdstuk 2 BPEL4WS

In dit hoofdstuk bespreken we de compositietaal BPEL4WS. We beginnen met het ontstaan van BPEL4WS in paragraaf 2.1 en bespreken vervolgens in paragraaf 2.2 wat BPEL4WS nu precies is. Daarbij komen vooral de twee verschillende procesbenaderingen aan bod. We vermelden ook kort de verschillen tussen BPEL4WS versie 1.0 en 1.1 en de relatie met WSDL. In paragraaf 2.3 verdiepen we ons in de BPEL4WS 1.1 specificatie en behandelen we de belangrijkste elementen van de processtructuur. Het hoofdstuk eindigt met het aanhalen van twee aanvullende specificaties: WS-Coördination en WS-Transaction. In hoofdstuk 3 zullen we dieper ingaan op de sterktes en zwaktes van BPEL4WS.

2.1 Ontstaan van BPEL4WS

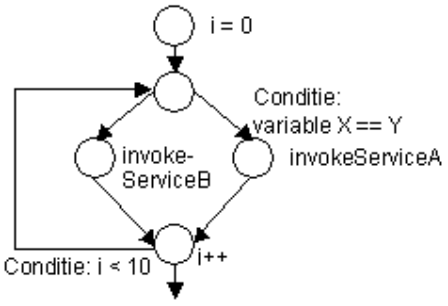
BPEL4WS is één van de meest besproken Web service-compositietalen op het moment. Zoals al is aangehaald in het vorige hoofdstuk, is BPEL4WS ontstaan uit de gezamenlijke inspanningen van IBM, Microsoft en BEA. BPEL4WS combineert de beste eigenschappen van twee voorgangers, IBM's Web Services Flow Language (WSFL) en XLANG van Microsoft.

WSFL onderscheidt twee types van Web services-compositie. Het eerste type definieert de uitvoerbare processen in een stroommodel en het tweede type specificeert de bedrijfssamenwerking in een globaal model. WSFL maakt gebruik van een graaf-gestructureerde controlestroom. De controlestroom duidt de volgorde van activiteiten aan, wanneer ze moeten worden uitgevoerd en onder welke condities. Elke knoop van de graaf stelt een activiteit voor waarbij de relatie met een andere activiteit wordt voorgesteld door 'controlLinks' en 'dataLinks'. De 'controlLinks' zorgen voor de volgorde van uitvoering terwijl de 'dataLinks' de gegevensstroom voorstellen. Er is minstens een 'controlLink' tussen twee activiteiten. Het model ondersteunt enkel vertakkingen (=forks) en parallellisme (=concurrency). Complexe constructies zoals loops, foutenafhandeling, compensatie en aggregatie zijn in het model niet expliciet aanwezig, maar moeten met behulp van verwijzingen en condities geconstrueerd worden [Ebpml.org '04 a], [Snell '02].

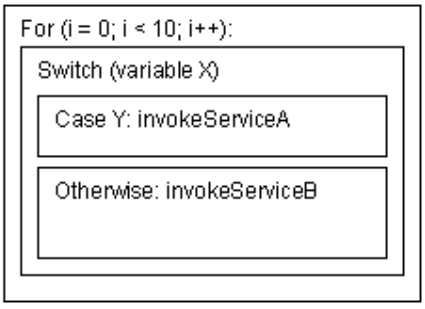
XLANG is net als WSFL een aanvulling op WSDL met als doel verschillende Web services te orchestreren. XLANG is een blokgestructureerde modelleertaal waarbij alle controleconstructies in elkaar genest zijn. XLANG ondersteunt alle belangrijke constructies zoals loops, foutenafhandeling, tijdsbeperkingen, compensaties en aggregatie. XLANG voorziet echter niet in een globaal model [Ebpml.org '04 b].

BPEL4WS is het resultaat van de samensmelting van twee verschillende workflow-benaderingen. BPEL4WS heeft gespecialiseerde controleconstructies overgenomen uit de blokgestructureerde XLANG en graafgeoriënteerde concepten uit WSFL. Het voordeel van een graaf is de grote graad van vrijheid, er zijn namelijk heel weinig beperkingen. Men kan elke activiteit verbinden met bijna elke ander activiteit. Procesontwerpers zijn meestal ook meer vertrouwd met flowdiagrammen en grafische notaties. Het grote voordeel van de blokgestructureerde benadering is dat er bij complexe constructies niet een soort van 'spaghetti' model wordt verkregen, wat bij een graaf wel het geval is. De combinatie van XLANG en WSFL resulteert in een gesofisticeerde taal die gebruik maakt van de kracht van gestructureerd programmeren samen met de directheid en flexibiliteit van een graaf [Weerawarana, Curbera '02].

In respectievelijk figuur 6 en 7 wordt een voorbeeld gegeven van een graaf - en blokgestructureerde opzet van een '`for`'-activiteit met daarbinnen een '`switch`' en een '`case`' [Oorlog '03]. In de graafgestructureerde methode moet bijvoorbeeld de '`for`'-activiteit geconstrueerd worden met links en condities.



Figuur 6 Graafgestructureerd (bron: Oorlog '03)



Figuur 7 Blokgestructureerd (bron: Oorlog '03)

De BPEL4WS 1.0 specificatie werd door IBM, Microsoft en BEA geïntroduceerd in augustus 2002. Later voegden ook Siebel en SAP zich bij de groep. In mei 2003 werd er een nieuwe versie gepubliceerd, BPEL4WS 1.1 en werd het ook aangeboden bij de Organization for the Advancement of Structured Information Standards (OASIS). OASIS richtte daarom een nieuw technisch comité op, het Web Service Business Process Execution Language Technical Committee (WSBPEL TC). Het TC zal werken aan de verdere ontwikkeling van de BPEL4WS-specificatie [OASIS '04].

Het World Wide Web Consortium (W3C) is een ander standaardisatieorgaan dat al eerder een werkgroep had opgericht betreffende orchestratie/choreografie van Web services. De Web Services Choreography Working Group heeft tot doel de verschillende overlappende voorstellen te ordenen en een taal te creëren die de choreografie, de regels van compositie en de interactie tussen Web services beschrijft. Een voorbeeld van een specificatie die bij het W3C is ingediend, is de hierboven besproken Web Services Choreography Interface of WSCI [W3C Choreography '03].

Waar we vroeger alleen concurrentie hadden tussen vendors, blijkt er nu ook concurrentie te bestaan tussen standaardisatieorganen. Het is namelijk de eerste keer dat een OASIS TC (Technical Committee) en W3C WG (Working Group) over hetzelfde onderwerp handelen [Taft '03].

Microsoft verklaart dat OASIS is gekozen als standaardisatieorgaan omdat het werk van OASIS zich oriënteert naar bedrijfsapplicaties en omdat de leden van OASIS bestaan uit technologische bedrijven, experts in bedrijfsproces-automatisatie en klanten die deze geautomatiseerde bedrijfsprocessen gebruiken [WSJ News Desk '03]. OASIS zou ook meer gefocust zijn op de technische kant van het automatiseren van bedrijfsprocessen, waar het W3C meer gericht is op low-level infrastructuur van Web services [Gonsalves '03]. Daarbij is OASIS al aan het werk met andere door hen voorgestelde standaarden zoals WS-Security.

Volgens Steve Bratt, Chief Operating Officer van W3C, is coördinatie van de twee groepen belangrijk om technische oplossingen te kunnen leveren die voldoen aan de noden van de klant.

2.2 Wat is BPEL4WS?

De BPEL4WS-specificatie maakt het mogelijk om een nieuwe Web service te definiëren door bestaande Web services te combineren in een bedrijfsproces. BPEL4WS is een XML gebaseerde standaard om deze bedrijfsprocessen te beschrijven. Het proces wordt zelf, net als een eenvoudige Web service, met een publieke WSDL-interface beschikbaar gesteld. BPEL bestaat uit gespecialiseerde controleconstructies die complexe interactie tussen verschillende partijen mogelijk maken. Op deze manier wordt het Web service-model uitgebreid met de mogelijkheden om applicatie - en bedrijfsintegratie te modelleren.

BPEL4WS gebruikt twee benaderingen om een bedrijfsproces voor te stellen, 1) een bedrijfsprotocol of abstract proces en 2) een uitvoerbaar proces [van Sinderen, Almeida '03].

Een bedrijfsprotocol is de beschrijving van het observeerbare gedrag tussen verschillende partners. Het bedrijfsprotocol bepaalt de berichtuitwisseling en de volgorde waarin de interacties moeten gebeuren. Alle interne details van het bedrijfsproces worden achterwege gelaten. Het proces is daardoor niet uitvoerbaar en wordt een abstract proces genoemd. Het bedrijfsprotocol is publiceerbaar, daar er geen bedrijfsgevoelige informatie in voorkomt. Op deze manier kan een partner, die wil interageren met het proces, aan de hand van het bedrijfsprotocol te weten komen welk gedrag wordt verwacht.

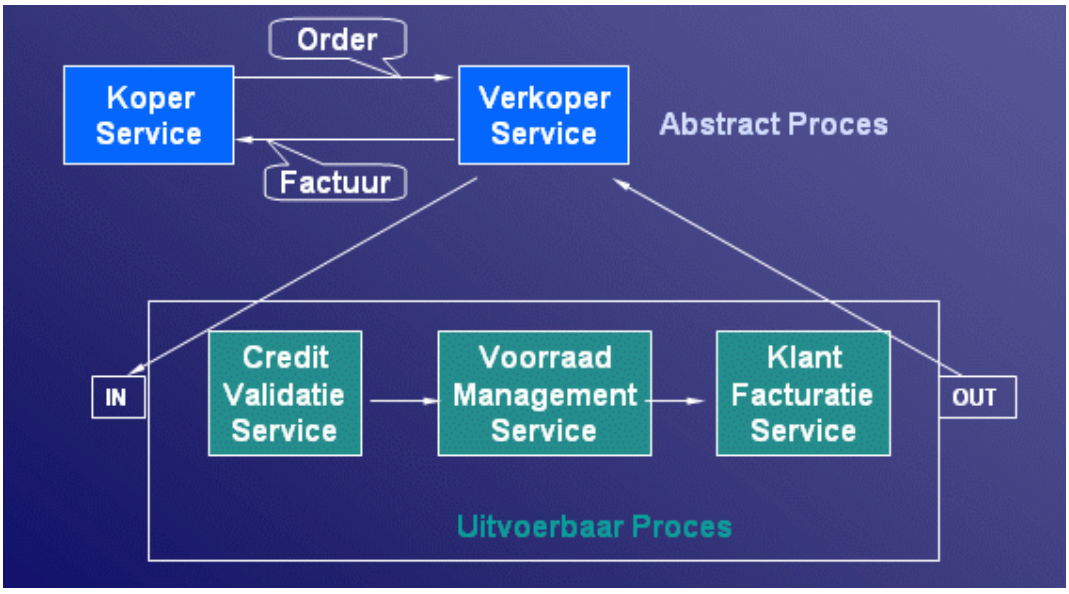
Een uitvoerbaar proces daarentegen beschrijft de interne details van een bedrijfsproces. Zowel de toestand (state) als de logica wordt gedefinieerd. Het uitvoerbare proces bevat gedetailleerde informatie om een proces van begin tot einde te kunnen uitvoeren [BPEL4WS Spec '03], [Masud '03].

In figuur 8 illustreren we hoe abstracte en uitvoerbare processen aan elkaar gerelateerd zijn. Het voorbeeld bestaat uit een eenvoudig aankoopproces waarin twee partijen voorkomen. De koper plaatst een order bij de verkoper door diens service aan te roepen. De verkoper stuurt na enkele controles een factuur terug [Kreger '02].

Het observeerbare gedrag is de uitwisseling van berichten tussen koper en verkoper. Het abstracte proces beschrijft nu die berichtenuitwisseling, zonder dat daarbij informatie wordt vrijgegeven over de interne verwerking van een order. Het abstracte proces van de verkoper definieert dat er eerst een orderbericht ontvangen moet worden en dat daarna de verkoper een bericht zal terugsturen met de factuurgegevens. Aan de hand van het abstracte proces weet de koper dat na het sturen van een orderbericht een factuurbericht verwacht mag worden.

Het uitvoerbare proces van de verkoper bevat ook deze informatie, maar wordt aangevuld met een beschrijving van de interne verwerking van het proces. Vooraleer de factuur kan worden verstuurd, worden een aantal activiteiten uitgevoerd. Bij de ontvangst van een orderbericht wordt eerst een Web service aangeroepen om de creditwaardigheid van de klant te controleren. Vervolgens wordt een service aangesproken om te kijken of de bestelde producten in voorraad zijn. Het proces eindigt met het aanmaken van de factuur zodat deze naar de koper verzonden kan worden.

Waar het abstracte proces enkel de berichtuitwisseling beschrijft, bepaalt het uitvoerbare proces hier de activiteiten tussen de twee opeenvolgende berichten.



Figuur 8 Abstract en uitvoerbaar proces (bron Kreger ’02)

BPEL4WS bouwt op de volgende XML-gebaseerde specificaties: WSDL 1.1, XML Schema 1.0, XPath 1.0 en WS-Addressing.

WSDL heeft de meeste invloed op BPEL4WS. Het BPEL4WS-procesmodel is een laag bovenop het WSDL 1.1 servicemodel. Zowel het bedrijfsproces als zijn partners zijn beschreven in een WSDL-document. Een WSDL-document definieert welke functionaliteit een Web service biedt, hoe deze communiceert en waar de service te vinden is. Een WSDL-document definieert een verzameling van poorten. Elke poort bestaat uit de beschrijving van de operaties die een Web service kan uitvoeren en uit de beschrijving van de berichten die de service kan verwerken [BPEL4WS Spec '03], [Dumas, ter Hofstede, van der Aalst '03].

XML Schema Definition language of XSD is een schematechnologie die gebruikt wordt voor XML. Met XSD definieert men de structuur, inhoud en semantiek van een XML document. Het is een standaard van het W3C en probeert een aantal minpunten op te lossen van een andere technologie, namelijk DTD (Document Type Definition).

XPath is een taal voor het adresseren van onderdelen van een XML document. Met XPath kan er specifieke informatie uit een XML document worden opgevraagd. Het wordt dan ook wel aanzien als een simpele querytaal.

WS-Addressing ten slotte voorziet een transportneutraal mechanisme om berichten en services te adresseren.

In de volgende paragraaf bekijken we de processtructuur van de BPEL4WS 1.1 specificatie. Het grootste verschil met de vorige specificatie (BPEL4WS 1.0) is de opdeling ervan in één gemeenschappelijke kern en in specifieke uitbreidingen voor de twee procesbenaderingen, namelijk uitvoerbare en abstracte processen. BPEL probeert daarbij zoveel mogelijk concepten gemeenschappelijk te gebruiken en dus de concepten eigen aan een specifieke procesbenadering te beperken. Er werden ook enkele terminologische veranderingen aangebracht. In de nieuwe versie zijn er tevens *eventHandlers* bijgekomen voor het opvangen en afhandelen van bepaalde gebeurtenissen. Voor een complete lijst van alle aanpassingen, zie [BPEL4WS Spec '03].

2.3 Processtructuur van BPEL4WS 1.1

In deze paragraaf bespreken we de belangrijkste elementen van een proces gebaseerd op de BPEL4WS 1.1 specificatie [BPEL4WS Spec '03]. De BPEL4WS 1.1 processtructuur is hieronder schematisch weergegeven (Figuur 9).

```
<process>
<!- Definitie & rol van partners die deelnemen in het proces -->
<partnerLinks> … </partnerLinks>
<!- Gegevens die in het proces worden gebruikt -->
<variables> … </variables>
<!- Eigenschappen die asynchrone conversaties mogelijk maken -->
<correlationSets> … </correlationSets>
<!- Procedures voor foutafhandeling -->
<faultHandlers> … </faultHandlers>
<!-  Foutenherstel en undo acties -->
<compensationHandlers> … </compensationHandlers>
<!-  Afhandelen van events -->
<eventHandlers> ... </eventHandlers>
<!- Bedrijfsproces flow -->
(activities)*
</process>
```

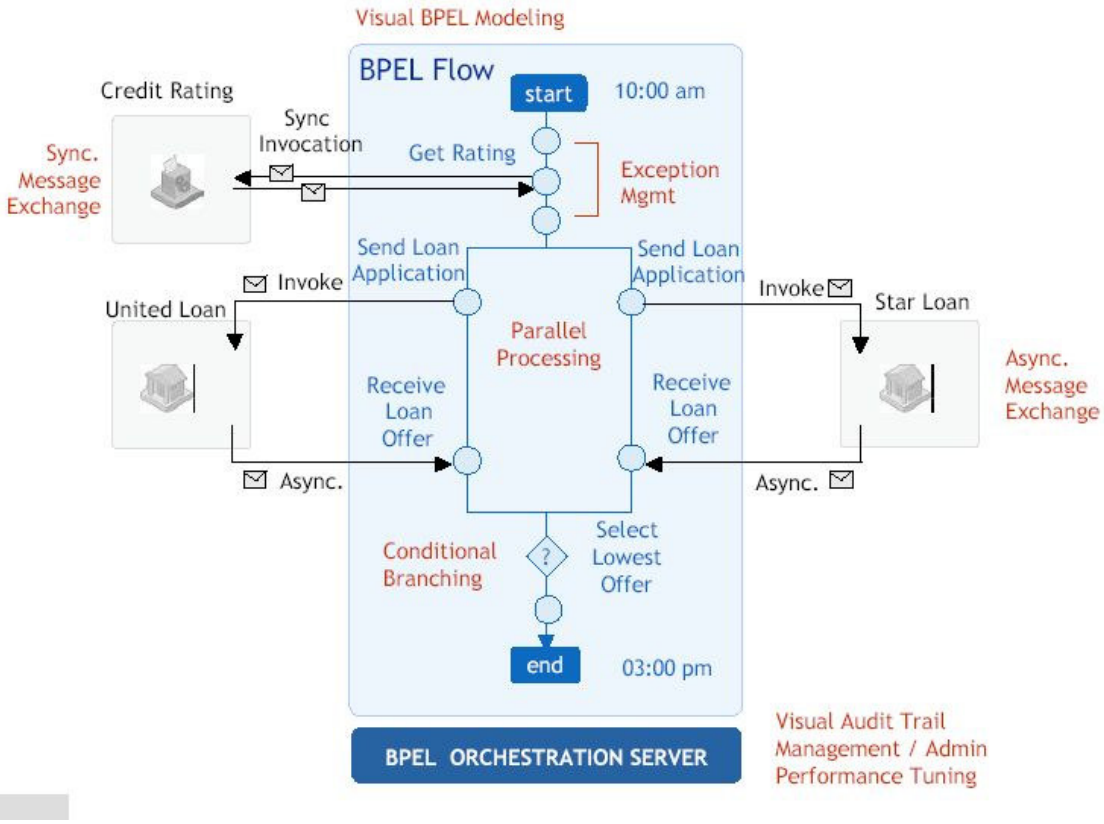
Figuur 9 BPEL Processtructuur

We beginnen met de bespreking van de elementen die gemeenschappelijk zijn voor zowel abstracte als uitvoerbare processen en dus behoren tot de kern van de specificatie. Deze elementen beschrijven de verschillende bedrijfspartners die deelnemen in het proces, de toestandsgegevens (state variabelen) die door het proces worden gebruikt en de uit te voeren activiteiten, zowel enkelvoudige als samengestelde. We bespreken verder ook de specifieke constructies om fouten op te vangen, om de gepaste compensatiehandelingen te verrichten en om op bepaalde gebeurtenissen te reageren. Op het einde van deze paragraaf staan we dan even stil bij enkele typische uitbreidingen eigen aan de twee procesbenaderingen.

Bij de bespreking van de proceselementen zal telkens de syntax worden aangegeven. Een '+' teken in de syntax duidt aan dat het betreffende element minstens één maal moet voorkomen en een '?' betekent dat het element hoogstens één maal mag

voorkomen. Voor de illustratie van de meeste proceselementen maken we gebruik van het *loanFlow* voorbeeld dat te vinden is op de Oracle website [Oracle '04 a]. Het volledige BPEL en WSDL document staan respectievelijk in bijlage 1 en bijlage 2.

Het *loanFlow* proces (zie figuur 10) aanvaardt requests van een client en gaat op zoek naar de gepaste lening. Na het ontvangen van de client's request wordt de *creditRatingService* opgeroepen. Deze interactie gebeurt synchroon. Het proces stuurt het *SNN* (*Social Security Number*) uit het ontvangen bericht van de client naar de *creditRatingService* en krijgt een *creditRating* terug. Na de ontvangst van de *creditRating* worden er parallel en asynchroon twee *loanServices* aangeroepen. Nadat beide services een antwoord terug hebben gestuurd, zal het proces beslissen welk lening de beste is. Het antwoord wordt dan naar de client gestuurd.



Figuur 10 LoanFlow voorbeeld (bron: Collaxa '03)

2.3.1 Samenwerkingsverbanden

Eén van de belangrijkste aspecten dat gemodelleerd moet worden, is de interactie met de verschillende bedrijfspartners van het proces. De verschillende partners moeten gedefinieerd worden alsook de rol die ze spelen in de interactie. Een proces kan daarenboven zelf zowel de rol van provider als van requestor aannemen. Wanneer het proces diensten biedt aan een klant heeft het proces de rol van provider. Het proces kan zelf ook diensten verzoeken aan externe Web services zodat het de rol van requestor bezit.

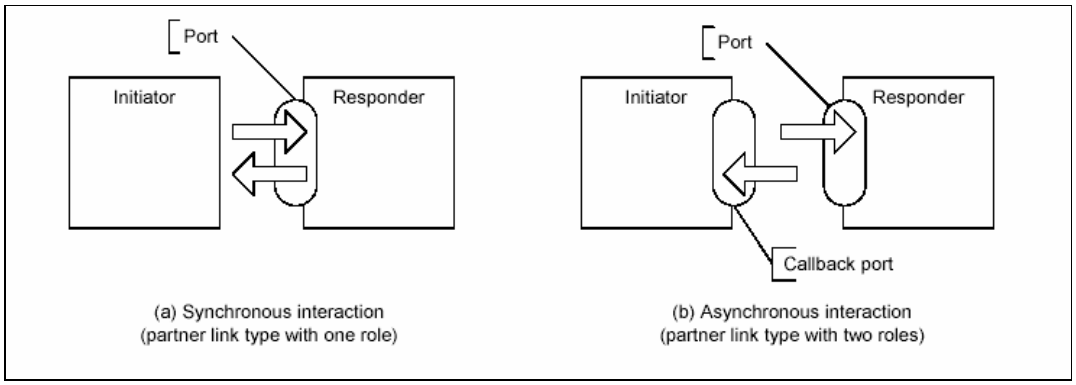
Een partnerlinktype beschrijft de relatie tussen het proces en een bedrijfspartner door te definiëren welke rol elk speelt in de interactie. Elke rol bestaat uit juist één poorttype dat bepaalt welke berichten kunnen worden ontvangen en welke operaties kunnen worden aangesproken. Het partnerlinktype is beschreven ofwel in een afzonderlijk WSDL-document of in het WSDL-document van de poorttypes die in de rollen zijn gedefinieerd.

De syntax van een partnerlinktype is beschreven in volgende figuur (Figuur 11).

```
<definitions name="ncname" targetNamespace="uri"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:plnk="http://schemas.xmlsoap.org/ws/partnerlink/">
   ...
   <plnk:partnerLinkType name="ncname">
      <plnk:role name="ncname">
            <plnk:portType name="qname"/>
      </plnk:role>
      <plnk:role name="ncname">?
            <plnk:portType name="qname"/>
      </plnk:role>
   </plnk:partnerLinkType>
   ...
</definitions>
```

Figuur 11 WSDL syntax: PartnerLinkType (bron: BPEL4WS Spec '03)

Er zijn twee types van interactie tussen het proces en een bedrijfspartner mogelijk, namelijk synchrone en asynchrone interactie [van Sinderen, Almeida '03].



Figuur 12 Interactie alternatieven (bron: van Sinderen, Almeida '03)

Bij de synchrone interactie zal het aanroepende proces een service verzoeken bij het reagerende proces en wacht het aanroepende proces tot het een antwoord krijgt. In deze vorm van interactie moet alleen het poorttype van het reagerende proces worden beschreven. Daar de rol in een partnerlinktype met een poorttype wordt geassocieerd, krijgt dus alleen het reagerende proces een rol in de interactie (Figuur 12a).

Bij asynchrone interactie zendt het aanroepende proces een bericht naar het reagerende proces om een serviceverzoek. Het reagerende proces zendt een bericht terug naar het aanroepende proces via de 'callback' poort van het aanroepende proces. Hier hebben beide processen een poort. Een partnerlinktype zal dus rollen bezitten voor beide processen (Figuur 12b).

```
<partnerLinks>
      <partnerLink name="ncname" partnerLinkType="qname"
            myRole="ncname"? partnerRole="ncname"? />+
      </partnerLinks>
</partnerLinks>
```

Figuur 13 BPEL definitie: PartnerLink (bron: BPEL4WS Spec '03)

De service waarmee een proces interageert wordt in BPEL4WS gemodelleerd als een `<partnerlink>` (zie Figuur 13). Elke partnerlink wordt bepaald door een partnerlink-type. Voor elke interactie met een partner wordt de partnerlink meegegeven, elke partnerlink heeft daarvoor een unieke naam. Om de rol van het proces zelf in de

interactie aan te duiden wordt het attribuut '`myRole`' gebruikt en voor de rol van de partner '`partnerRole`'.

In het *loanFlow* voorbeeld heeft het proces vier verschillende partners, namelijk de *client* die het proces aanroept, de *creditRatingService* en twee *loanServices*. Er worden dus ook vier `<partnerLinks>` aangemaakt (zie Figuur 14). De interactie met de *creditRatingService* is synchroon zodat er in de betreffende partnerlink maar één rol wordt gedefinieerd. De '`myRole`' wordt hier weggelaten, daar de *creditService* geen (callback)poort moet aanroepen. De andere partnerlinks hebben zowel een `myRole` als een `partnerRole`.

```
<partnerLinks>
  <partnerLink name="client"
           partnerLinkType="tns:LoanFlow"
           partnerRole="LoanFlowRequester"
           myRole="LoanFlowProvider"/>

  <partnerLink name="creditRatingService"
           partnerLinkType="services:CreditRatingService"
           partnerRole="CreditRatingServiceProvider"/>

  <partnerLink name="UnitedLoanService"
           partnerLinkType="services:LoanService"
           myRole="LoanServiceRequester"
           partnerRole="LoanServiceProvider"/>

  <partnerLink name="StarLoanService"
           partnerLinkType="services:LoanService"
           myRole="LoanServiceRequester"
           partnerRole="LoanServiceProvider"/>
</partnerLinks>
```

Figuur 14 LoanFlow: Partnerlinks (bron: Oracle '04 b)

Een partnerLinkType kan door meerdere partnerlinks worden gebruikt. In figuur 14 zien we bijvoorbeeld dat elke *loanService* hetzelfde partnerlinktype heeft. Partnerlinktypes worden in een WSDL-document beschreven. De partnerlink met de *client* heeft

bijvoorbeeld het partnerlinktype '*loanFlow*'. Het partnerlinktype bestaat uit twee rollen, namelijk *LoanFlowProvider* en *LoanFlowRequestor* (zie Figuur 15). Elke rol is een poorttype toegewezen.

```
<plnk:partnerLinkType name="LoanFlow">
      <plnk:role name="LoanFlowProvider">
            <plnk:portType name="tns:LoanFlow"/>
      </plnk:role>
      <plnk:role name="LoanFlowRequester">
            <plnk:portType name="tns:LoanFlowCallback"/>
      </plnk:role>
</plnk:partnerLinkType>
```

Figuur 15 LoanFlow: Partnerlinktype (bron: Oracle '04 b)

Partnerlinks beschrijven de relatie tussen slechts twee partners. Een partner heeft echter vaak meerdere relaties. Deze kunnen worden gegroepeerd in een optioneel `<partner>` element.

Alvorens de operaties via een `<partnerLink>` kunnen worden aangeroepen, is er informatie nodig over de eigenlijke netwerklocatie van een service. Daar de fysieke locatie van een service kan wijzigen, heeft WSDL een onderscheidt gemaakt tussen poort en poorttype. Het poorttype beschrijft de abstracte functionaliteit van een service, namelijk de operaties en berichten. Een poort daarentegen bepaalt de eigenlijke toegangsinformatie en definieert een individuele endpoint. Het endpoint bestaat meestal uit het adres waarmee de service kan worden gelokaliseerd. De WSDL-binding zorgt dan voor de verbinding tussen poort en poorttype. Deze binding wordt ook opgenomen in het WSDL-document.

Het is nu echter mogelijk om een partner dynamisch te kiezen door het mechanisme van 'endpoint reference' dat is gedefinieerd in de WS-Addressing specificatie [W3C WSDL '02]. Een BPEL-proces is statisch afhankelijk van een poorttype. De bijhorende poort kan echter dynamisch worden gewijzigd. Endpoint reference maakt het mogelijk om het endpoint van een poort aan te passen at run-time. Het endpoint kan door middel van een bericht worden verkregen of het proces zelf kan dynamisch een endpoint selecteren.

2.3.2 Gegevenshantering

De hierboven besproken partnerlinks worden gebruikt om de verschillende partijen in een interactie te modelleren. De interactie tussen de verschillende partners gaat gepaard met het uitwisselen van berichten. Deze berichten kunnen worden opgeslagen door het proces. Op deze manier is het mogelijk om de toestand (state) van het proces te bewaren. De procestoestand bestaat uit de berichten die zijn uitgewisseld, maar ook uit de gegevens die gebruikt worden voor de bedrijfslogica en voor het opstellen van berichten die naar de partners worden verzonden. De state van een bedrijfsproces wordt opgeslagen aan de hand van variabelen. BPEL gebruikt daarvoor het element `<variable>`. Het type van elke variabele kan ofwel een WSDL berichttype (‘`messageType`’), een XML Schema-type (‘`type`’) of een XML Schema-element (‘`element`’) zijn. De initialisatie van een variabele kan aan de hand van een toewijzings-opdracht of door het ontvangen van een bericht. De syntax van het `<variable>` element is weergeven in figuur 16.

```
<variables>
<variable name=”ncname” messageType=”qname”?
          type=”qname”? element=”qname”? />+
</variables>
```

Figuur 16 BPEL definitie: Variable (bron: BPEL4WS Spec ’03)

Elke variabele behoort tot een scope en moet daarbinnen uniek zijn. Variabelen die tot de globale processcope behoren worden globale variabelen genoemd. Variabelen kunnen ook tot andere scopes behoren en worden dan lokale variabelen genoemd.

In het *LoanFlow* voorbeeld worden de variabelen alleen gebruikt voor het bewaren van ontvangen en verzonden berichten. In figuur 17 bekijken we één BPEL-variabele en de bijkomende WSDL-definities. De variabele slaat het ontvangen bericht van de client op. Het type van de variabele is het WSDL-berichttype ‘*loanFlowRequestMessage’*. Dit messagetype bestaat uit een deel payload dat is beschreven als een XML Schema-element (*loanApplication*). Dit element wordt opnieuw beschreven door een XML Schema-type (*LoanApplicationType*). Het ontvangen bericht van een client bestaat dus uit verschillende velden zoals een SSN (social security number), een emailadres, een naam, het gewenste bedrag voor de lening, …

```
BPEL document:
<variable name="input" messageType="tns:LoanFlowRequestMessage"/>

WSDL document:
<element name="loanApplication" type="s1:LoanApplicationType"/>
<complexType name="LoanApplicationType">
      <sequence>
            <element name="SSN" type="string"/>
            <element name="email" type="string"/>
            <element name="customerName" type="string"/>
            <element name="loanAmount" type="double"/>
            ...
      </sequence>
</complexType>

<message name="LoanFlowRequestMessage">
      <part name="payload" element="s1:loanApplication"/>
</message>
```

Figuur 17 LoanFlow: BPEL-variable en WSLD-definities (bron: Oracle '04 b)

Om het gedrag van het proces te controleren, moet de mogelijkheid bestaan om gegevens op te halen en vervolgens te combineren met andere gegevens. Een BPEL-proces gebruikt daarvoor gegevensuitdrukkingen. De taal gebruikt voor deze uitdrukkingen wordt gespecificeerd in het '`expressionLanguage`' attribuut van het `<process>` element. De standaardwaarde is XPath 1.0. Naast de verschillende ingebouwde XPath-functies heeft BPEL een aantal uitbreidingen op deze functies. Enkele voorbeelden van BPEL-functies zijn: `bpws:getVariableProperty` en `bpws:getLinkStatus`.

BPEL4WS voorziet in vier verschillende types gegevensuitdrukkingen. Ten eerste hebben we de booleaanse uitdrukkingen die een '`true`' of '`false`' waarde uitkomen. Deze uitdrukkingen worden vooral gebruikt in condities, zoals bij een `<while>` of `<case>` activiteit. Een tweede type van uitdrukkingen dwingt een deadline af. Bij evaluatie resulteert deze uitdrukking in een waarde die behoort tot de XML-Schema types '`dateTime`' of '`date`'. Dit wordt bijvoorbeeld gebruikt bij het attribuut '`until`' bij een `<onAlarm>` of `<wait>` element. Uitdrukkingen voor een duur resulteren in een

waarde van het XML-Schema type ‘duration’, meestal gebruikt bij <onAlarm> en <wait> element. Bij het ‘algemene type’ tenslotte horen de relationele operatoren en de wiskundige uitdrukkingen. Het resulteert in een XPath-waarde van het type ‘string’, ‘number’ of ‘boolean’.

In het *LoanFlow* voorbeeld wordt er ook gebruik gemaakt van gegevensuitdrukkingen (zie Figuur 18). De conditie die bepaalt welke van de twee *loanProviders* de beste is, is een voorbeeld van een booleaanse uitdrukking. Er worden namelijk twee waardes met elkaar vergeleken aan de hand van de relationele operator ‘<’ . De uitkomst van deze uitdrukking is ‘true’ of ‘false’. In deze uitdrukking wordt er gebruik gemaakt van de uitbreidingsfunctie bpws:getVariableData. Deze functie kan echter alleen gebruikt worden in uitvoerbare processen en niet in abstracte processen (zie paragraaf 2.3.10). De functie neemt uit het ontvangen bericht van de *loanService* de *APR (Annual Percentage Rate)*. De conditie vergelijkt vervolgens de twee waarden verkregen door de functies. Een andere gegevensuitdrukking behoort tot het algemene type en heeft als uitkomst een getal. Er wordt beroep gedaan op de standaard XPath functie ‘number()’. Deze functie convergeert zijn argument naar een getal.

```
Booleaanse uitdrukking
<case condition = "
      bpws:getVariableData('loanOffer1','payload','/loanOffer/APR') >
      bpws:getVariableData('loanOffer2','payload','/loanOffer/APR')
" />
Algemene uitdrukking
expression="number(-1000)"
```

Figuur 18 LoanFLow: Gegevensuitdrukkingen (bron: Oracle ’04 b)

Een veel gebruikte taak binnen een bedrijfsproces is het kopiëren van de gegevens van de ene variabele naar de andere. Hiervoor wordt de <assign> activiteit gebruikt (zie Figuur 19). Naast het kopiëren van gegevens wordt de <assign> ook gebruikt om nieuwe gegevens te creëren door middel van gegevensuitdrukkingen. Het <assign> element bestaat uit één of meerdere <copy> elementen elk met exact één <from> en één <to> element. Opdat een toewijzing geldig zou zijn moeten de gegevens in het <from> en <to> element van een compatibel type zijn.

```
<assign>
<copy>+
   <from ... />
   <to ... />
</copy>
</assign>
```

Figuur 19 BPEL definitie: Assign (bron: BPEL4WS Spec '03)

De <from> en <to> elementen kunnen op verschillende manier worden gedefinieerd. Een eerste methode en tevens de meest eenvoudige is gewoon de naam van de variabele te vernoemen en de volledige variabele te kopiëren. Iets complexer is binnen de variabele een deel aan te duiden dat moet worden gekopieerd. Men kan ook gebruik maken van een algemene uitdrukking in de <from> om bijvoorbeeld eenvoudige berekeningen te maken. Niet alleen de waarde van variabelen, maar ook de waardes van properties kunnen worden gemanipuleerd. Een laatste manier is om in de <from> een letterlijke waarde op te nemen.

De <assign> activiteit wordt verschillende keren gebruikt in het *loanFlow* voorbeeld. In figuur 20 staan er drie verschillende manieren om gegevens toe te wijzen. In de eerste <from> wordt er een getal aangemaakt met een XPath-functie. Dit getal wordt toegewezen aan de *creditRating* van het *input* bericht. De tweede toewijzing neemt het *SSN* uit het *input* bericht en wijst het toe aan het *SSN* van het *crInput* bericht. Hier wordt dus slechts een deel van het volledige bericht gekopieerd. In het derde voorbeeldje echter wordt een volledige deel rechtstreeks naar een andere variabele gekopieerd.

```
<from expression="number(-1000)"/>
<to variable="input" part="payload"
      query="/loanApplication/creditRating"/>

<from variable="input" part="payload" query="/loanApplication/SSN"/>
<to variable="crInput" part="payload" query="/ssn"/>

<from variable="input" part="payload"/>
<to variable="loanApplication" part="payload"/>
```

Figuur 20 LoanFLow: Assign (bron: Oracle '04 b)

2.3.3 Berichtcorrelatie

Bij het uitvoeren van het procesmodel wordt elke keer een procesinstantie gecreëerd. Wanneer er berichten naar het proces worden gestuurd, is het niet voldoende om ze te sturen naar de correcte WSDL-poort, maar moet ook de correcte instantie van het proces aangesproken worden. Berichtcorrelatie zorgt ervoor dat een bericht tot de juiste conversatie behoort door de juiste procesinstantie te lokaliseren.

Om de correlatie tussen asynchrone berichten te realiseren wordt er gebruik gemaakt van een of meerdere sleutelgegevens die zich in het bericht bevinden, bijvoorbeeld een klantnummer of een ordernummer. Er bestaat dus geen expliciet instantieveld, maar de instantie wordt geïdentificeerd door gegevens die al aanwezig zijn in het bericht. Deze sleutelgegevens worden beschreven aan de hand van properties. Properties zijn een BPEL-uitbreiding op de WSDL-specificatie. Een property heeft een unieke naam in de globale scope en een XLM Schema-type. Een property is een abstract sleutelgegeven. Er is immers nog geen link gemaakt met gegevens uit het bericht. Om die link te maken wordt er beroep gedaan op een propertyAlias. De propertyAlias bepaalt welk veld uit het bericht wordt toegewezen aan een property aan de hand van een XPath-uitdrukking. In een aankoopproces zouden we bijvoorbeeld de property *orderNr* en *InvoiceNr* kunnen hebben (zie Figuur 21). Het veld *order* uit het bericht *POMessage* wordt hier toegewezen aan de property *orderNr*.

```
<bpws:property name="orderNr" type="xsd:int" />
<bpws:property name="invoiceNr" type="xsd:int" />

<bpws:propertyAlias propertyName="orderNr"
      messageType="POMessage" part="PO" query="/PO/order" />
<bpws:propertyAlias propertyName="invoiceNr"
      messageType="InvMessage" part="IVC" query="/IVC/InvNum" />
```

Figuur 21 Property en PropertyAlias (bron: BPEL4WS Spec '03)

De correlatie wordt in een BPEL proces beschreven als een `<correlationSet>`. Elke correlationSet heeft een naam en bestaat uit een lijst van properties. In ons *aankoopproces* voorbeeld kunnen we twee correlatieSets definiëren (zie Figuur 22).

```
<correlationSets>
<correlationSet name="PurchaseOrder" properties="orderNr"/>
<correlationSet name="Invoice" properties="invoiceNr"/>
</correlationSets>
```

Figuur 22 CorrelationSet (bron: BPEL4WS Spec '03)

Berichtcorrelatie zorgt ervoor dat een bericht bij de juiste procesinstantie terecht komt. `CorreslationSets` zullen dus gebruikt worden bij alle binnenkomende en uitgaande berichten. Het toewijzen van een `correlationSet` aan een bericht gebeurt door het element `<correlation>` toe te voegen aan de activiteit die het bericht ontvangt of verstuurt. Figuur 23 toont een voorbeeld van een interactie waar het proces een aankooporder ontvangt en dan asynchroon een bevestiging met factuur terug stuurt. Beide activiteiten (ontvangen en versturen) gebruiken de `<correlationSet>` *PurchaseOrder* zodat de orderbevestiging met het orderverzoek kan worden gecorreleerd. De ontvangstactiviteit (`<receive>`) initialiseert de `correlationSet` door het attribuut initiate op '`yes`' te zetten. Indien het attribuut wordt weggelaten krijgt het de standaardwaarde '`no`' toegewezen.

```
<receive partnerLink="Buyer" portType="PurchasingPT"
         operation="Purchase" variable="PO" />
   <correlations>
      <correlation set="PurchaseOrder" initiate="yes" />
   </correlations>
</receive>

<invoke partnerLink="Buyer" portType=" BuyerPT"
        operation="PurchaseResponse" inputVariable="POResponse">
   <correlations>
      <correlation set="PurchaseOrder" pattern="out">
      <correlation set="Invoice" initiate="yes" pattern="out">
   </correlations>
</invoke>
```

Figuur 23 Correlation (bron: BPEL4WS Spec '03)

2.3.4 Basisactiviteiten

Basisactiviteiten zijn de simpelste vorm van uitvoering. Ze zijn geen opeenvolging van stappen, maar een individuele stap die ofwel interageert met een service ofwel de gegevens manipuleert ofwel zorgt voor foutsignalisatie.

Een eerste activiteit kwam eerder reeds aanbod in bij gegevensafhandeling (paragraaf 2.3.2): de `<assign>` activiteit. Een proces kan interageren met de buitenwereld aan de hand van volgende drie activiteiten, de `<invoke>`, de `<receive>` en de `<reply>`.

Wanneer een proces een operatie van een partner wil aanroepen, zal de `<invoke>` activiteit worden gebruikt (zie Figuur 24). Deze aanroep kan zowel synchroon als asynchroon gebeuren. Asynchrone aanroepen vereisen alleen de input variabele van de operatie, daar er geen antwoord wordt verwacht. Synchrone aanroepen vereisen echter zowel een input als een output variabele.

```
<invoke partnerLink="ncname" portType="qname" operation="ncname"
        inputVariable="ncname"? outputVariable="ncname"?>
    standaard elementen
</invoke>
```

Figuur 24 BPEL definitie: Invoke (bron: BPEL4WS Spec '03)

De 'standaard elementen' vermeld in figuur 24 (en in volgende BPEL-definities) kunnen zowel `<correlation>` activiteiten zijn als `<target>` en `<source>` activiteiten (zie `<link>` activiteit in paragraaf 2.3.5).

De `<receive>` activiteit (zie Figuur 25) specificeert de partnerlink waarvan het proces een bericht verwacht te ontvangen. Een proces wordt gestart en een procesinstantie wordt aangemaakt wanneer een partner het proces aanroept of anders gezegd wanneer het proces een verzoekbericht ontvangt. De `<receive>` activiteit is één van de twee activiteiten (zie ook `<pick>` activiteit paragraaf 2.3.5) die berichten kan ontvangen en dus een procesinstantie kan starten. Indien de `<receive>` het beginpunt van een proces voorstelt, wordt het attribuut '`createInstance`' op '`yes`' gezet.

```
<receive partnerLink="ncname" portType="qname" operation="ncname"
         variable="ncname"? createInstance="yes|no"?>
    standaard elementen
</receive>
```

Figuur 25 BPEL definitie: Receive (bron: BPEL4WS Spec '03)

De `<reply>` activiteit wordt gebruikt om een antwoord te sturen op een voorafgaand verzoek aangenomen door de `<receive>` activiteit. De `<reply>` is alleen van betekenis bij synchrone interacties. Bij asynchrone interacties wordt er immers een antwoord gestuurd door het aanroepen van een operatie van de partner via een `<invoke>` activiteit.

In het *loanFlow* voorbeeld komen verschillende `<receive>` en `<invoke>` activiteiten voor (zie Figuur 26). De eerste `<receive>` activiteit (*receiveInput*) ontvangt het verzoek van de client. Deze activiteit is de start van het proces en maakt dan ook een procesinstantie aan door het attribuut '`createInstance`' de waarde '`yes`' te geven. Het ontvangen bericht wordt in de variabele *input* opgeslagen.

Het proces roept vervolgens de *creditRatingService* aan met de `<invoke>` activiteit (*invokeCr*). Deze aanroep gebeurt synchroon zodat zowel de `inputVariable` als `outputVariable` gedefinieerd zijn. Nadat het antwoord van het synchrone verzoek is ontvangen en is opgeslagen in de `outputVariable`, worden de *loanServices* asynchroon aangeroepen.

Bij het asynchroon aanroepen van een service wordt er geen onmiddellijk antwoord verwacht. Er wordt dus ook geen `outputVariable` gedefinieerd. De *loanService* kan een antwoord sturen door een bericht te sturen met een `<invoke>` activiteit. Aan de proceskant moet dit bericht worden opgevangen door een `<receive>` activiteit. Hier wordt het antwoord van de service opgeslagen in de `variable` van de activiteit *receive_invokeUnitedLoan*.

In het *loanFlow* proces wordt geen gebruik gemaakt van de `<reply>` activiteit. De *creditRatingService* maakt er echter wel gebruik van om te kunnen antwoorden op het synchrone verzoek van het *loanFlow* proces.

```
Creatie procesinstantie
<receive name="receiveInput" partnerLink="client"
         portType="tns:LoanFlow"
         operation="initiate" variable="input"
         createInstance="yes"/>

Synchrone Interactie
<invoke name="invokeCR"
      partnerLink="creditRatingService"
      portType="services:CreditRatingService"
      operation="process"
      inputVariable="crInput"
      outputVariable="crOutput"/>

Asynchrone Interactie
<sequence>
  <invoke name="invokeUnitedLoan"
      partnerLink="UnitedLoanService"
      portType="services:LoanService"
      operation="initiate"
      inputVariable="loanApplication"/>

  <receive name="receive_invokeUnitedLoan"
      partnerLink="UnitedLoanService"
      portType="services:LoanServiceCallback"
      operation="onResult"
      variable="loanOffer1"/>
</sequence>
```

Figuur 26 LoanFlow: Receive & Invoke (bron: Oracle '04 b)

Naast de net vermelde activiteiten bestaan ook nog de `<throw>`, `<wait>` en `<empty>` activiteiten.

Fouten kunnen door een `<throw>` activiteit (zie Figuur 27) worden gesignaleerd. Elke fout moet een globaal unieke naam bezitten. Er kan ook een optionele variabele worden gedefinieerd om verdere informatie te verstrekken over de fout.

Een <wait> activiteit (zie Figuur 27) laat het proces wachten voor een bepaalde duur of totdat er een deadline is bereikt. De <empty> (zie Figuur 27) activiteit doet niets. Deze wordt vaak gebruikt bij het opvangen en onderdrukken van fouten en bij de synchronisatie van meerdere controledraden.

```
<throw faultName="qname" faultVariable="ncname"?>
standaard elementen
</throw>

<wait (for="duration-expr" | until="deadline-expr")>
standaard elementen
</wait>

<empty>
standaard elementen
</empty>
```

Figuur 27 BPEL-definitie: throw, wait, empty (bron: BPEL4WS Spec '03)

2.3.5 Gestructureerde activiteiten

Gestructureerde activiteiten bepalen de volgorde waarin een verzameling activiteiten wordt uitgevoerd.

Een verzameling van activiteiten die na elkaar worden uitgevoerd, kunnen door een <sequence> (zie Figuur 29) worden gemodelleerd. De volgorde waarin elke activiteit wordt uitgevoerd komt overeen met de volgorde van voorkomen. Een activiteit kan pas starten als de vorige is voltooid. Wanneer de laatste activiteit is voltooid, wordt de <sequence> beëindigd. Het *loanFlow* voorbeeld maakt veelvuldig gebruik van de <sequence>. Zo bestaat het volledige proces uit een sequentie van verschillende activiteiten en is de asynchrone aanroep van de *loanService* een sequentie van een <invoke> en een <receive> activiteit.

Wanneer op een gegeven punt in het proces een keuze moet gemaakt worden tussen verschillende alternatieve paden, kan men een <switch> activiteit (zie Figuur 29) gebruiken. De <switch> bestaat uit verschillende vertakkingen en aan de hand van een

conditie kan de juiste vertakking worden gekozen. Elke vertakking wordt voorgesteld door een <case> structuur met een bijhorende conditie. Elke conditie is een booleaanse uitdrukking. Wanneer een <switch> wordt geactiveerd, zullen de condities in volgorde van voorkomen worden geëvalueerd. Bij een positieve evaluatie zal de vertakking worden uitgevoerd en bij een negatieve evaluatie wordt overgegaan naar de volgende conditie. Eenmaal een vertakking is geselecteerd worden de andere vertakkingen onbeschikbaar gemaakt. Er kan dus slechts één vertakking worden uitgevoerd. Men kan een optionele <otherwise> toevoegen aan de <swicth>. De <otherwise> zorgt ervoor dat bijhorende activiteit wordt uitgevoerd als er geen <case> kan worden geselecteerd. Wanneer de <otherwise> wordt weggelaten, wordt er een <otherwise> met een <empty> activiteit verondersteld. De <switch> wordt beëindigd na de uitvoering van de geselecteerde vertakking.

De <switch> speelt een belangrijke rol in het *loanFlow* proces (zie Figuur 28). De switch maakt namelijk de keuze tussen twee verschillende *loanServices*. Er worden dus twee paden gedefinieerd, maar slechts één pad kan worden uitgevoerd. De beslissing welk pad wordt gekozen is afhankelijk van het resultaat van de conditie. Deze conditie bestaat uit een booleaanse uitdrukking die we al hebben besproken in het de paragraaf over gegevensafhandeling (paragraaf 2.3.2).

```
<switch>
   <case condition = "bpws:getVariableData ('loanOffer1',
      'payload','/loanOffer/APR') > bpws:getVariableData
      ('loanOffer2','payload','/loanOffer/APR') ">
      . . . (gebruik loan 2)
   </case>
   <otherwise>
      . . . (gebruik loan 1)
   </otherwise>
</switch>
```

Figuur 28 LoanFlow: Switch (bron: Oracle '04 b)

Om verschillende activiteiten meermaals uit te voeren kan men gebruik maken van de <while> activiteit (zie Figuur 29). De <while> activiteit is een gestructureerde

lusconstructie die herhaaldelijk de onderliggende activiteiten uitvoert totdat een booleaanse conditie niet langer ‘true’ resulteert.

```
<sequence>
standaard elementen/ activiteiten+
</sequence>

<switch>
standaard elementen
<case condition="bool-expr">+
  activiteit
</case>
<otherwise>?
  activiteit
</otherwise>
</switch>

<while condition="bool-expr">
standaard elementen/ activiteiten
</while>
```

Figuur 29 BPEL definitie: sequence, switch, en while (bron: BPEL4WS Spec ’03)

Als in een bepaald punt van het proces verschillende alternatieve paden voorkomen en de keuze van het pad wordt gemaakt door het voorkomen van een bepaalde gebeurtenis (event), kan men dit modelleren aan de hand van een `<pick>` activiteit (zie Figuur 30). Net als de `<switch>` kan het proces één bepaalde vertakking kiezen. De keuze wordt nu echter niet gemaakt door het evalueren van een conditie, maar door het voorkomen van een gebeurtenis. Men onderscheidt het opvangen van alarm gebeurtenissen (`<onAlarm>`) en van berichten (`<onMessage>`). Het `<onMessage>` event wordt uitgevoerd wanneer een bericht met overeenkomend type wordt ontvangen van de partner omschreven in de partnerlink. De `<onMessage>` kan net zoals de `<receive>` activiteit een instantie creëren. Het `<onAlarm>` event wordt uitgevoerd bij het verstrijken van een bepaalde duur of het verlopen van een deadline. Er kan slechts één event worden behandeld, namelijk het eerst voorkomende. Na het uitvoeren van bijhorende activiteiten wordt de `<pick>` activiteit beëindigd. We zullen in paragraaf 2.5.9 andere mogelijkheden zien voor het afhandelen van gebeurtenissen.

```
<pick createInstance="yes|no"? >
    <onMessage partnerLink="ncname" portType="qname"
               operation="ncname" variable="ncname"?>+
      activiteit
    </onMessage>
    <onAlarm (for="duration-expr" | until="deadline-expr")>*
      activiteit
    </onAlarm>
</pick>
```

Figuur 30 BPEL definitie: pick (bron: BPEL4WS Spec '03)

We zouden nu een <pick> activiteit kunnen toevoegen aan het *loanFlow* proces om ervoor te zorgen dat het proces niet oneindig lang moet wachten op een antwoord van de *loanService*. Het proces zoals het nu is gedefinieerd wacht totdat beide *loanServices* een antwoord terug sturen. Als één van de services niet meer in staat is een antwoord te verzenden, betekent dit dat het proces zal blokkeren. Door de <receive> activiteit te vervangen door een <pick> activiteit met een <onMessage> en <onAlarm> event, kan dit probleem worden verholpen (zie Figuur 31). Het <onMessage> event heeft dan dezelfde functie als de <receive> activiteit. Het <onAlarm> event zorgt ervoor dat het proces na een bepaalde periode een actie wordt oderneemt, zoals het opwerpen van een error.

```
<sequence>
      <invoke name="invokeUnitedLoan" ... />
      <pick>
            <onMessage ... variable="loanOffer1" />
            <onAlarm for="3D">
                  onderneem actie na drie dagen
                 (<throw faultName="" ... />)
            </onAlarm>
      </pick>
</sequence>
```

Figuur 31 LoanFlow: Pick

De <flow> constructie (zie Figuur 32) maakt het mogelijk om activiteiten parallel uit te voeren. Een <flow> wordt dus beëindigd wanneer alle activiteiten in de <flow> zijn

beëindigd. Wanneer een <flow> wordt gestart, kunnen alle activiteiten tezelfdertijd beginnen, uitgenomen indien de activiteit inkomende 'links' bevat die nog niet geëvalueerd zijn. Links zorgen voor volgordebeperkingen in de uitvoer van activiteiten. De <link> constructie is overgenomen van de graafgestructureerde WSFL en heeft dezelfde functie als de WSFL-controlLink.

```
<flow>
    standaard elementen
    <links>?
      <link name="ncname">+
    </links>
    activiteit+
</flow>
```

Figuur 32 BPEL definitie: flow (bron: BPEL4WS Spec '03)

De links zijn genaamd en worden afzonderlijk gedefinieerd in de <flow> activiteit. Om twee activiteiten te linken worden de <source> en <target> elementen gebruikt binnen de te linken activiteiten. Het <source> element kan een transitieconditie bezitten. Deze transitieconditie bepaalt de status van een link na de uitvoering van bijhorende activiteit. Indien er geen transitieconditie aanwezig is, is de waarde verondersteld 'true' te zijn. Elke activiteit die een <target> is van een link heeft expliciet of impliciet een joinConditie. De joinconditie bepaalt aan de hand van de status van alle inkomende links of er verder mag gegaan worden. De joinconditie wordt pas geëvalueerd indien alle inkomende link een status hebben. De impliciete joinConditie houdt in dat minstens één van de inkomende links positief moet zijn. Een expliciete joinConditie is een booleaanse expressie. Het bestaat uit booleaanse operatoren en de bpws:getLinkStatus functie.

Alvorens een activiteit met inkomende links (activiteit met een <target> element) kan worden uitgevoerd, moet de linkstatus bepaald worden van alle inkomende links. De linkstatus is positief indien de transitieconditie voldaan is. Als de activiteit klaar is om te starten en de linkstatussen van alle inkomende links zijn bepaald, wordt de joinconditie geëvalueerd. Als de joinconditie waar is, wordt de activiteit uitgevoerd anders wordt er een bpws:joinFailure-fout opgeworpen.

Soms is het aangewezen geen fout op te werpen maar gewoon de activiteit met een negatieve joinconditie over te slaan. Dit kan door het attribuut '`surpressJoinFailure`' op te nemen in de activiteit. Een activiteit die wordt overgeslagen, wordt dus niet uitgevoerd en zal dus nooit een status kunnen geven aan zijn uitgaande links. Een activiteit die wacht op de status van die link zal dus blokkeren. Om dit tegen te gaan worden uitgaande links van niet uitgevoerde activiteiten negatief geëvalueerd. Dit noemt men het *'dead path elimination'* principe en dient om elk gevolgd pad te kunnen evalueren. Op deze manier krijgt elke link een status en zal de transitieconditie altijd geëvalueerd kunnen worden.

```
<flow suppressJoinFailure="yes">
  <links>
    <link name="buyToSettle"/>
    <link name="sellToSettle"/>
    <link name="toBuyConfirm"/>
  </links>

  <receive name="getBuyerInformation">
    <source linkName="buyToSettle"/>
  </receive>
  <receive name="getSellerInformation">
    <source linkName="sellToSettle"/>
  </receive>

  <invoke name="settleTrade"
           joinCondition="bpws:getLinkStatus('buyToSettle') and
                          bpws:getLinkStatus('sellToSettle')">
    <target linkName="getBuyerInformation"/>
    <target linkName="getSellerInformation"/>
    <source linkName="toBuyConfirm"/>
  </invoke>

  <reply name="confirmBuyer">
    <target linkName="toBuyConfirm"/>
  </reply>
</flow>
```

Figuur 33 Flow voorbeeld (bron: BPEL4WS Spec '03)

Figuur 33 toont een voorbeeld van een `<flow>` met vier activiteit en drie links. De activiteiten *getBuyerInformation* en *getSellerInformation* kunnen parallel worden uitgevoerd. Deze twee activiteiten bezitten namelijk geen inkomende links. De `<source>` van beide activiteiten heeft geen transitieconditie en daardoor zal na uitvoering van de activiteit de linkstatus positief zijn.

De activiteit *settleTrade* bezit wel inkomende links en kan pas worden uitgevoerd als de status van alle inkomende links is bepaald en de joinconditie positief is geëvalueerd. Bij impliciete joinconditie hoeft slechts één van de inkomende links positief zijn. In het voorbeeld wordt er echter gebruik gemaakt van een expliciete joinconditie om af te dwingen dat beide inkomende links positief moeten zijn. Indien de joinconditie positief is, zal de *settleTrade* activiteit worden uitgevoerd en zal de status van de uitgaande link *toBuyConfirm* bepaald worden. Nu kan de *confirmBuyer* activiteit worden uitgevoerd.

Indien één van de linkstatussen, gebruikt in de joincondite, negatief wordt geëvalueerd, zal de joincondite een fout opwerpen. De `<flow>` bezit echter het attribuut `surpressJoinFailure`, zodat de fout zal worden onderdrukt. Het proces kan gewoon verder gaan. Door de negative joinconditie kan de activiteit echter niet worden uitgevoerd en zal de linkstatus van de *toBuyConfirm* link niet bepaald worden. Met gevolg dat de *confirmBuyer* activiteit zal blijven wachten op de status van zijn inkomende link. Om dit op te lossen bestaat het `death path elimination` principe dat bij het onderdrukken van een fout de uitgaande linkstatussen negatief zal maken. De *toBuyConfirm* link krijgt dus een negatieve status waardoor de *confirmBuyer* activiteit niet zal worden uitgevoerd. De `<flow>` activiteit kan vervolgens worden beëindigd.

2.3.6 Scope activiteit

Bij het uitvoeren van een proces is het mogelijk dat er fouten optreden zowel in de aangeroepen services als in het proces zelf. BPEL heeft een mechanisme om die fouten op te vangen en af te handelen via een bepaalde subroutine in een `<faultHandler>` element (zie paragraaf 2.3.8). Daarbij worden er ook mogelijkheden voorzien om bepaalde activiteiten later ongedaan te maken. BPEL gebruikt daarvoor een `<compensationHandler>` element (zie paragraaf 2.3.7).

Het afhandelen van dergelijke situaties beïnvloedt gewoonlijk een groep van gerelateerde activiteiten. Deze activiteiten worden gegroepeerd in een `<scope>` structuur. Een scope definieert een context voor de activiteiten die het bevat, en kan naast faultHandlers en compensationHandlers, ook correlationSets en gegevens-variabelen bezitten. Een scope stelt dus een compenseerbare en herstelbare verzameling van activiteiten voor. Door het proces op te delen in verschillende scopes kunnen fouten lokaal worden behandeld.

Het *loanFlow* proces is ook opgedeeld in verschillende scopes. Figuur 34 toont een vereenvoudiging van de processtructuur. Het is een sequentie van verschillende scopes. De *GetCreditRating* scope heeft bijvoorbeeld een eigen faultHandler.

```
<sequence>
      <receive name="receiveInput" ... />
      <scope name="GetCreditRating"> ... (faultHandler) </scope>
      <scope name="GetLoanOffer"> ... </scope>
      <scope name="SelectOffer"> ... </scope>
      <invoke name="replyOutput" ... />
</sequence>
```

Figuur 34 LoanFlow: Scopes (bron: Oracle '04 b)

2.3.7 Compensatie-acties

Bij de uitvoering van een proces is het mogelijk dat een deel van het reeds uitgevoerde gedrag ongedaan gemaakt zal moeten worden. Een dergelijk deel kan worden voorgesteld door een scope met bijhorende `<compensationHandler>` (zie Figuur 35). Elke compensationHandler bezit een activiteit die zal worden uitgevoerd indien het nodig is de scope te compenseren.

Het is belangrijk wanneer een scope succesvol is uitgevoerd dat het compensatiemechanisme de waardes van de variabelen onthoudt. De variabelen kunnen immers door het proces al zijn aangepast eenmaal de compensatieactie moet worden uitgevoerd.

```
<compensationHandler>?
activiteit
</compensationHandler>

<compensate scope="ncname"? >
standaard elementen
</compensate>
```

Figuur 35 BPEL definitie: compensationHandler, compensate (bron:BPEL4WS Spec '03)

Een compensatie kan slechts worden gebruikt, eenmaal de scope succesvol is uitgevoerd. De compensatie kan worden aangeroepen op een expliciete of een impliciete manier. Expliciet door het gebruik van een `<compensate>` activiteit (zie Figuur 35). Deze activiteit kan overal voorkomen en verwijst naar de naam van de scope die moet worden gecompenseerd. Na bijvoorbeeld het succesvol plaatsen van een order, zou de klant kunnen beslissen het order te annuleren. Het annuleren van een order kan dan worden gemodelleerd door de juiste compensatieactiviteit aan te roepen. Impliciete compensatie gebeurt wanneer er een fout wordt opgeworpen. Dan wordt voor alle geneste scopes de compensatieactiviteit uitgevoerd in omgekeerde volgorde van voltooiing.

Figuur 36 toont een voorbeeld van een `<invoke>` activiteit met bijhorende `<compensationHandler>`. De `<invoke>` activiteit roept de *SyncPurchase* operatie aan bij de partner *seller* om een aankooporder te plaatsen. Daar dit een synchrone interactie is zal het antwoord (*getResponse*) worden opgeslagen in de `outputVariable` van de `<invoke>` activiteit. Indien later het order moet worden geannuleerd, kan dit antwoord gebruikt worden als input voor de compensatie. De compensationHandler bestaat uit een synchrone interactie met dezelfde partner, maar ditmaal wordt de operatie *CancelPurchase* aangeroepen. Het plaatsen en annuleren van een order zijn aan elkaar gecorreleerd door middel van een correlationSet. Op deze manier zal bij annulatie de juiste aankoopinstantie bij de partner worden verwijderd.

```
<scope>
    <compensationHandler>
        <invoke partnerLink="Seller" portType="SP:Purchasing"
                operation="CancelPurchase"
                inputVariable="getResponse"
                outputVariable="getConfirmation">
           <correlations>
              <correlation set="PurchaseOrder" pattern="out"/>
           </correlations>
        </invoke>
    </compensationHandler>
    <invoke partnerLink="Seller" portType="SP:Purchasing"
            operation="SyncPurchase"
            inputVariable="sendPO"
            outputVariable="getResponse">
       <correlations>
          <correlation set="PurchaseOrder" initiate="yes"
                       pattern="out"/>
       </correlations>
    </invoke>
</scope>
```

Figuur 36 Compensation (bron: Oracle ’04 b)

2.3.8 Foutafhandeling

Foutafhandeling heeft als doel gedeeltelijk en onsuccesvol werk van een scope ongedaan te maken. Foutafhandeling gebeurt dus tijdens de uitvoering van een scope, terwijl compensatie pas kan gebeuren nadat een scope is beëindigd. Wanneer de fout is afgehandeld, kan de volledige scope nooit als succesvol worden beschouwd zodat er ook geen compensatieacties meer mogelijk zijn op de scope.

Er kunnen verschillende fouten worden opgevangen binnen een scope door gebruik te maken van een `<faultHandlers>` structuur (zie Figuur 37). Elke fout wordt gedefinieerd door een `<catch>` activiteit. Elke `<catch>` heeft een foutnaam en kan een variabele bevatten met relevante foutgegevens. Indien er geen foutnaam aanwezig is, dan zal de `<catch>` alle fouten opvangen met hetzelfde type als de foutvariabele. Met

een optionele `<catchAll>` structuur kunnen alle fouten worden behandeld die nog niet door een `<catch>` zijn opgevangen.

```
<faultHandlers>?
      <catch faultName="qname"? faultVariable="ncname"?>*
            activiteit
      </catch>
      <catchAll>?
            activiteit
      </catchAll>
</faultHandlers>
```

Figuur 37 BPEL definitie: faultHandler (bron: BPEL4WS Spec '03)

Het *loanFlow* voorbeeld definieert één faultHandler (zie Figuur 38). De faultHandler behoort tot de *GetCreditRating* scope en vangt fouten op die worden opgeworpen door de *creditRatingService*. De faultHandler zorgt ervoor dat het veld *creditRating* toch een waarde wordt toegewezen, zodat het proces verder kan werken. Dus in plaats van de waarde te krijgen via de synchrone interactie, wordt de waarde gecreëerd door de faultHandler van de scope.

```
<faultHandlers>
   <catch faultName="services:NegativeCredit"
          faultVariable="crError">
      <assign>
         <copy>
            <from expression="number(-1000)"/>
            <to variable="input" part="payload"
                  query="/loanApplication/creditRating"/>
         </copy>
      </assign>
   </catch>
</faultHandlers>
```

Figuur 38 LoanFlow: faultHandler (bron: Oracle '04 b)

Fouten kunnen op een flexibele manier worden uitgedrukt: enkel door een `faultName` of `faultVariable` ofwel door beide samen. Daardoor is het mogelijk dat een fout bij

verschillende `<catch>` activiteiten kan behoren. De volgende simpele regels maken uitsluitsel:

- Indien er geen `faultVariable` is gedefinieerd dan zal de `<catch>` activiteit met de gepaste `faultName` worden geselecteerd anders wordt de `<catchAll>` uitgevoerd.
- Indien er wel een `faultVariabele` is dan zal die `<catch>` activiteit worden gekozen die de gepaste `faultName` en het juiste type van de variabele bevat. Indien de fout geen naam bezit maar wel een variabele dan zal die `<catch>` worden geselecteerd met het gepaste type. In alle andere gevallen wordt de `<catchAll>` gebruikt.

In de volgende beslissingstabel (Figuur 39) worden alle mogelijkheden voorgesteld.

faultVariable Aanwezig	J					N		
faultName Aanwezig	J			N		J		N
faultName Matching	J		N	-		J	N	-
variable Type Matching	J	N	-	J	N	-	-	-
Catch	X			X		X		-
CatchAll		X	X		X		X	-

Figuur 39 Beslissingstabel voor <catch> selectie

Als er geen `<catch>` of `<catchAll>` wordt geselecteerd kan de fout niet door de scope worden behandeld en wordt de fout doorgegeven aan de bovenliggende scope. Indien een fout in de globale processcope niet behandeld kan worden, dan zal het proces “abnormaal” afsluiten.

Wanneer er zich binnen een scope een fout voordoet, kan de scope de fout zelf behandelen of de fout opwerpen naar de bovenliggende scope. Bij het zelf behandelen van de fout worden de nodige activiteiten ondernomen zodanig dat de bovenliggende scope verder kan worden uitgevoerd. Bij het succesvol afhandelen van een fout worden alle uitgaande links in die scope geëvalueerd zodat de bovenliggende scope zou kunnen verder werken. Dit is het eerder besproken ‘*dead path elimination*’ principe.

Bij de foutafhandeling van een scope worden eerst alle activiteiten die actief zijn binnen die scope beëindigd en daarna wordt het gedrag toegepast dat is gedefinieerd in de <catch>. We bekijken nu voor elke activiteit wat er gebeurt bij het beëindigen.

De <assign> activiteit mag verder worden uitgevoerd daar de levensduur van die activiteit kort is. Evaluatie-uitdrukkingen die al gestart zijn mogen verder worden geëvalueerd. Elke <wait>, <invoke>, <receive>, <reply> activiteit wordt onderbroken en vroegtijdig gestopt. Wanneer er bij het afsluiten van de activiteiten een antwoord komt op een synchrone conversatie, wordt dit antwoord genegeerd. Het beëindigen van <empty>, <terminate> en <throw> activiteiten is niet van toepassing.

Ook alle gestructureerde activiteiten worden onderbroken. Bij een <while> wordt de iteratie onderbroken en worden de activiteiten binnen de <while> beëindigd. Als er een vertakking is geselecteerd bij de <switch> en <pick> dan wordt de activiteit van die vertakking beëindigd. Indien nog geen vertakking was gekozen, moet de <switch> en <pick> onmiddellijk stoppen. Alle geneste activiteiten in een <flow> of <sequence> moeten ook worden afgesloten. Wanneer een scope moet worden afgesloten, dan zal het gedrag van die scope worden onderbroken en wordt de standaardfout bpws:forcedTermination opgeworpen.

2.3.9 Afhandelen van gebeurtenissen

Bij elke scope is het mogelijke om bij bepaalde gebeurtenissen een activiteit uit te voeren. We onderscheiden twee soorten gebeurtenissen. Een eerste is het opvangen van een inkomend bericht (onMessage event). De tweede gebeurtenis is het aflopen van een alarm na een bepaalde tijdsperiode (onAlarm event). Beide worden gegroepeerd in de <eventHandler> structuur (zie Figuur 40).

Wanneer de gebeurtenis zich voordoet wordt de overeenkomende activiteit gelijktijdig met de normale procesactiviteiten uitgevoerd. Dit is het eerste grote verschil met de <pick> activiteit. De <pick> is een onderdeel van het proces en zal dus blokkeren terwijl het wacht op een gebeurtenis. De <eventHandler> wacht ook op een gebeurtenis, maar is niet verbonden met het verloop van het proces. Terwijl een <eventHandler> wacht, loopt het proces gewoon verder.

```
<eventHandlers>?
    <!—er moet minsten één onMessage of onAlarm zijn -->
    <onMessage partnerLink="ncname" portType="qname"
               operation="ncname"
               variable="ncname"?>*
      <correlations>?
          <correlation set="ncname" initiate="yes|no">+
      </correlations>
      activiteit
    </onMessage>

    <onAlarm for="duration-expr"? until="deadline-expr"?>*
      activiteit
    </onAlarm>
</eventHandlers>
```

Figuur 40 BPEL definitie: eventHandlers (bron: BPEL4WS Spec '03)

Het opvangen van een inkomend bericht door een `<onMessage>` vertoont vele gelijkenissen met een `<receive>` of `<pick>` activiteit. In het '`partnerlink`' attribuut wordt de partner aangeduid waarvan het bericht verwacht wordt. Ook het poorttype en de operatie die de partner zal aanroepen om de gebeurtenis te veroorzaken worden gedefinieerd. De variabele bevat dan het eigenlijke ontvangen bericht. Het grote verschil met de `<receive>` of `<pick>` is dat het niet mogelijk is om een instantie aan te maken. Dit komt omdat een `<eventHandler>` slechts beschikbaar is als er al een instantie bestaat. Nadat een `<onMessage>` event is uitgevoerd, blijven de `<eventHandlers>` beschikbaar gedurende het verdere verloop van de scope en kunnen ze dus meerdere keren worden uitgevoerd. De `<pick>` activiteit daarentegen kan maar één gebeurtenis afhandelen.

Het `<onAlarm>` event wordt aangeroepen na het verlopen van een bepaalde tijdsperiode. Met het '`for`' attribuut wordt de duur bepaald waarna het alarm zal worden gesignaleerd. De tijd begint te lopen van zodra de betreffende scope start. Het '`until`' attribuut duidt een specifiek tijdstip aan wanneer het alarm wordt aangeroepen. Juist één van deze attributen (`for` of `until`) moet aanwezig zijn. Het `<onAlarm>` event kan, in tegenstelling met het `<onMessage>` event, hoogstens één maal worden aangeroepen

terwijl de overeenkomende scope actief is. Eenmaal het alarm is uitgevoerd wordt het onbeschikbaar gemaakt voor het verdere verloop van de scope.

Wanneer een scope wordt beëindigd, worden ook alle `<eventHandlers>` onbeschikbaar gemaakt. Events die bezig zijn met hun activiteit mogen worden voltooid en de scope is pas voltooid eenmaal alle events klaar zijn.

2.3.10 Uitbreidingen voor uitvoerbare processen

In deze paragraaf komen enkele elementen aanbod die enkel van toepassing zijn op uitvoerbare processen en dus niet op de bedrijfsprotocollen.

Ten eerste wordt er bij de gegevensuitdrukkingen een nieuwe XPath functie toegevoegd om de waarde van variabelen op te halen: `bpws:getVariableData ('variableName', 'partName'?, 'locationPath'?).`

Vervolgens worden er een aantal standaardfouten gedefinieerd. Wanneer er bij een toewijzingsopdracht gegevens worden gebruikt met niet compatibele types zal de fout `bpws:mismatchedAssignmentFailure` worden opgeworpen. Bij het aanspreken van niet geïnitialiseerde variabelen zal de `bpws:uninitializedVariable`-fout voorkomen. Voor een volledige lijst zie [BPEL4WS Spec '03].

Uitvoerbare processen worden ook uitgebreid met een nog niet besproken activiteit. Om een instantie onmiddellijk te beëindigen, kan er gebruik gemaakt worden van de `<terminate>` activiteit. Alle activiteiten worden dan onmiddellijk stop gezet zonder gebruik te maken van foutafhandeling of compensatieacties.

De `<from>` en `<to>` van een `<assign>` activiteit kan nu ook worden aangevuld met het optionele attribuut '`query`'. De taal waarin deze query staat, wordt bepaald door het attribuut '`queryLanguage`' in het `<process>` element. Indien niet aanwezig wordt de standaard waarde 'XPath 1.0' gekozen. Tot slot moet er nog worden opgemerkt dat het '`inputVariable`' attribuut van een `<invoke>` en het '`variable`' attribuut van een `<receive>` en `<reply>` bij uitvoerbare processen verplicht zijn.

2.3.11 Uitbreidingen voor bedrijfsprotocollen

Ook bij de bedrijfsprotocollen zijn er twee uitbreidingen op de algemene constructies. De eerste uitbreiding betreft het initialiseren van variabelen. Variabelen moeten niet altijd zoals in uitvoerbare processen worden geïnitialiseerd vooraleer te gebruiken, daar berekeningen vaak impliciet gelaten worden. Zo kunnen verwijzingen naar variabelen in sommige activiteiten worden weggelaten. Indien het abstracte proces alleen gebruikt wordt voor het opleggen van een volgorde van berichten, zijn de gegevens van die bericht immers niet belangrijk.

De tweede uitbreiding betreft een nieuw type van toewijzing voor abstracte processen, '*opaque assignment*' genaamd. Hiermee kunnen onbepaalde waardes worden gebruikt. Waardes die eigen zijn aan een privaat proces of komen van externe bronnen kunnen op die manier afgeschermd worden van het publieke proces. Daar de waardes onbepaald zijn, kan dit proces ook niet uitgevoerd worden. In het `<from>` element plaats men dan het attribuut '`opaque`' met de waarde '`yes`'.

2.4 WS-Coordination & WS-Transaction

WS-Coordination (WS-C) en WS-Transaction (WS-T) zijn twee specificaties die samen met BPEL4WS werden vrijgegeven door IBM, Microsoft en BEA. Deze drie specificaties zijn ontwikkeld om bedrijfstransacties over Web services heen mogelijk te maken. Waar BPEL4WS zich toelegt op servicecompositie, zorgt WS-Transaction voor de integriteit van de transacties en zorgt WS-Coordination voor algemene coördinatievoorzieningen. In September 2003 werd WS-Transaction gesplitst in WS-AtomicTransaction en WS-BusinessActivity.

2.4.1 Coördinatie

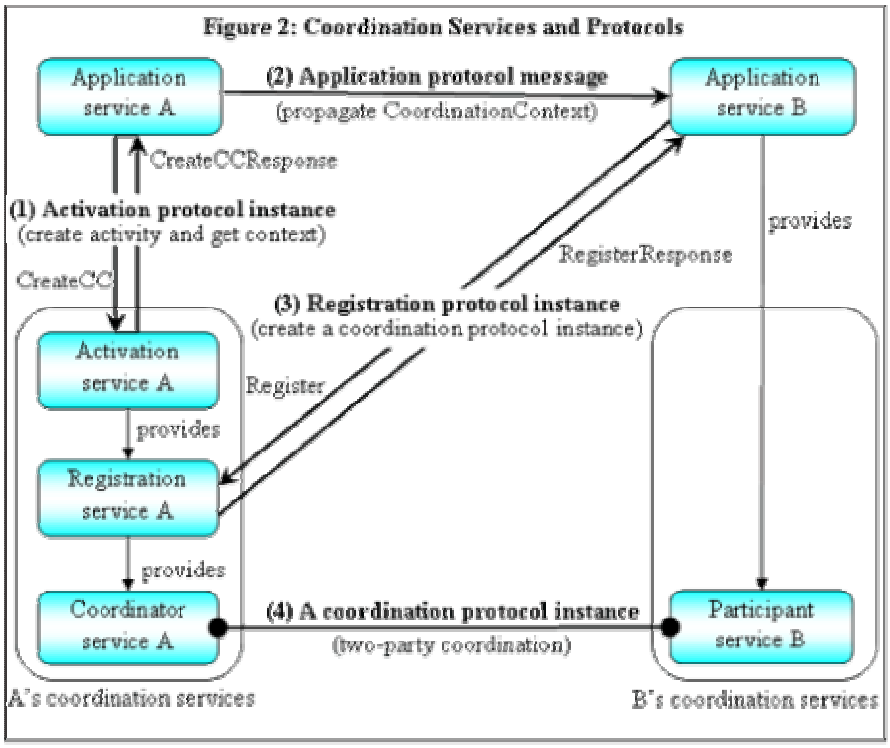
Coördinatie is het verspreiden van informatie door een entiteit aan alle betrokken partijen voor een welbepaalde reden. Deze reden kan het bereiken van een consensus zijn bij een bepaalde beslissing zoals bij transactie protocollen of het waarborgen dat alle partijen een bericht ontvangen zoals bij een multicast [Little & Weber '03a]. WS-C beschrijft nu een algemeen coördinatieraamwerk als basis voor specifieke coördinatie-protocollen.

Het framework bezit drie belangrijke begrippen [Curbera, Khalaf, Mukhi, Tai, Weerawarana '03]:

- De **coördinatiecontext** is de gedeelde informatie die de coördinatie activiteit vertegenwoordigt en wordt verspreid naar alle betrokken partijen. De coördinatiecontext bestaat uit een globale identificatie, het coördinatietype, een vervaldatum, een referentie naar de poort van de registratieservice en kan optioneel nog andere informatie bevatten die relevant kan zijn voor een specifiek protocol.
- De **activatieservice** is een Web service die de coördinatiecontext creëert.
- De **registratieservice** wordt gebruikt door verschillende partijen om zich te registreren voor deelname in de coördinatie.

De activatie - en registratieservice vormen samen de coördinatieservice, ook wel de coördinator genoemd. Wanneer een cliënt een coördinatieactiviteit wil starten (zie Figuur 41), zal er een '`CreateCoordinationContext`' bericht gestuurd worden naar de activatieservice van de cliënts coördinator. De activatieservice maakt de coördinatiecontext aan en stuurt het in een '`CreateCoordinationContextReply`' bericht terug. De activatieservice maakt ook een instantie van de coördinator aan, waardoor de registratieservice beschikbaar wordt.

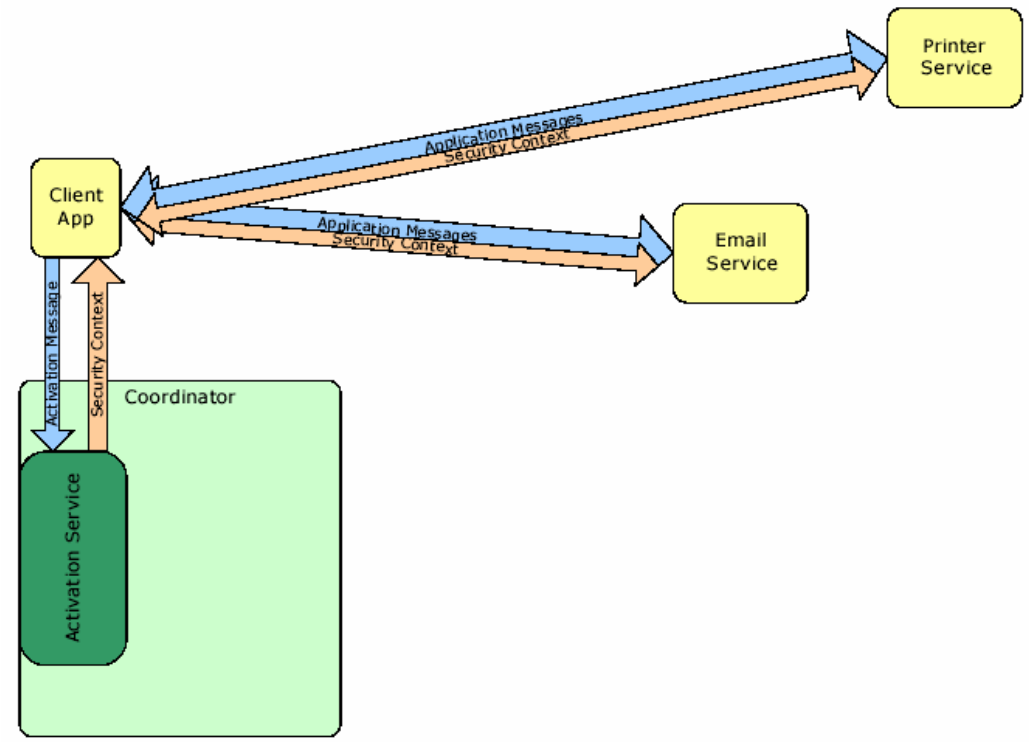
Elke keer de cliënt een service aanroept wordt deze coördinatiecontext opgenomen in het bericht. Zo wordt de aangeroepen partij op de hoogte gebracht van de coördinatieactiviteit en kan ze zich aan de hand van deze informatie registreren bij de registratieservice van de cliënt. Eenmaal de registratieservice een antwoord terug heeft gestuurd, wordt de coördinatieservice van de aangeroepen partij, ook wel participant genoemd, op de hoogte gebracht. Nu kunnen beide coördinatieservices gebruik maken van het coördinatieprotocol om elkaar berichten te sturen [Cabrera, Copeland, Johnson, Langworthy '04].



Figuur 41 Coördinatieservice en protocollen (bron: Cabrera, Copeland, Johnson, Langworthy '04)

Om de werking van WS-Coordination aan te tonen maken we gebruik van volgend voorbeeld [Weber '03]. Een cliënt kan gebruik maken van een printerservice en een emailservice. Voor beide services moet worden ingelogd. Nu willen we gebruik maken van een centrale inlogservice, zodat er slechts eenmaal hoeft te worden ingelogd om de verschillende services te gebruiken.

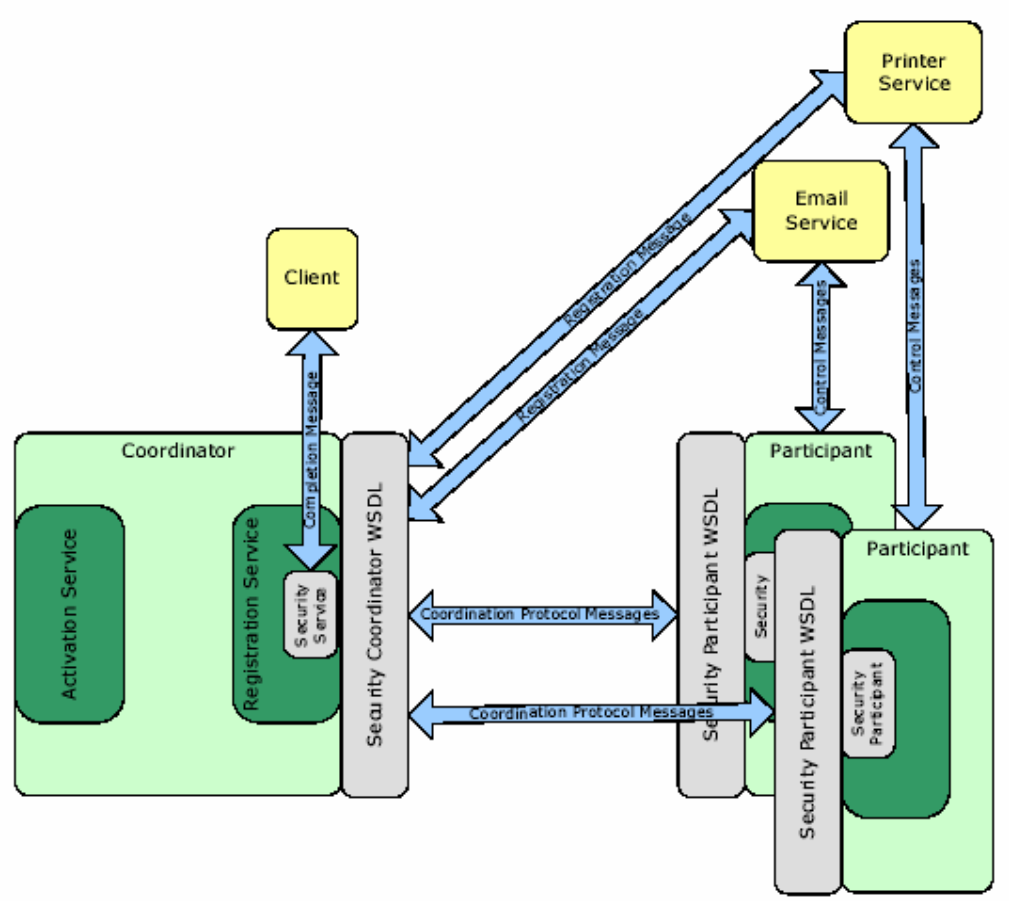
De cliënt zal de coördinatie starten met het sturen van een bericht naar de activatieservice. De activatieservice zal een coördinator en een *SecurityContext* aanmaken. Deze *SecurityContext* zal nu worden verstuurd met de gewone applicatieberichten naar de verschillende services. In de *SecurityContext* wordt het adres van de registratieservice opgenomen en informatie over authenticatie van de cliënt. De activatie en het doorsturen van de *SecurityContext* naar de verschillende services wordt in figuur 42 voorgesteld.



Figuur 42 Inlog voorbeeld: activatie (bron: Weber '03)

Een service die een applicatiebericht ontvangt vergezeld van een *SecurityContext,* zal zijn participant kunnen registreren bij de coördinator. Het adres van de registratieservice kan worden gevonden in de *SecurityContext*. Deze registratie gebeurt elke keer een service een *SecurityContext* ontvangt en zich nog niet eerder heeft geregistreerd. Na de registratie kunnen de coördinator en de participant coördinatieberichten uitwisselen. De authenticatie gegevens van de *SecurityContext* zullen worden gebruikt om de cliënt de juiste privileges te geven.

Wanneer de cliënt wil uitloggen volstaat het de coördinator te contacteren. Hier in het voorbeeld wordt een *completion* bericht gestuurd naar de coördinator. De coördinator zal op zijn beurt alle geregistreerde services verwittigen. De services zullen dan de privileges van de cliënt opheffen. De registratie en het uitloggen worden in figuur 43 voorgesteld.



Figuur 43 Inlog voorbeeld: registratie en coördinatie (bron: Weber '03)

2.4.2 Transacties

Een gedistribueerde omgeving ondervindt betrouwbaarheidsproblemen die niet zo vaak terug te vinden zijn in gecentraliseerde systemen. Decentralisatie maakt het mogelijk dat delen van het systeem falen, terwijl de rest gewoon verder blijft werken. Op deze manier kan de applicatie abnormaal gedrag vertonen. Er wordt verwacht dat een systeem consistent blijft wanneer er zich fouten voordoen. De consistentie van een gecentraliseerd systeem wordt door middel van ACID-transacties gewaarborgd.

De traditionele transacties met ACID-eigenschappen werken heel goed bij transacties van korte duur. Bij langdurige transacties gaat de mogelijkheid om simultaan processen uit te voeren sterk achteruit door gegevens waarop te lang een 'lock' staat. Verder zal bij het afbreken van een transactie veel werk dat reeds is uitgevoerd, moeten worden ongedaan gemaakt. Daar Web services vaak langdurige bedrijfsinteracties hebben, zal

het traditionele transactiesysteem moeten worden aangepast voor een Web service-omgeving.

WS-Transaction is één van de standaarden die is voorgesteld om transacties in een Web service omgeving te gebruiken WS-T definieert twee coördinatietypes: Atomic Transactions (AT) en Business Activity (BA). Waar AT vooral zorgt voor kortlopende transacties, modelleert BA de langdurige transacties.

WS-Transaction is gebouwd bovenop het algemeen coördinatieraamwerk van WS-Coordination. WS-C zorgt alleen voor contextbeheer. Het creëert een context en geeft de mogelijkheid aan services om te registreren bij die context. WS-T vult dit raamwerk aan met het uitbreiden van de coördinatiecontext naar een transactiecontext en er worden naast de registratie - en activatieservice een aantal bijkomende services aangeboden (Completion, CompletionWithAck, PhaseZero, 2PC, Outcome Notification, BusinessAgreement en BusinessAgreementWithComplete) [Litlle, Weber ’03 b].

2.4.3 Atomic Transaction

Atomic transacties (AT) zijn vergelijkbaar met traditionele ACID-transacties en ondersteunen interacties van korte duur. Om een AT te starten zal de cliënt eerst een coördinator lokaliseren die WS-Transaction ondersteunt. De cliënt stuurt vervolgens een verzoek om een context te creëren naar de activatieservice en duidt daarbij aan welk type van coördinatie gevraagd wordt. Nadat de cliënt een transactiecontext heeft gekregen van de activatieservice, gebruikt ze deze context bij elke service aanroep. Op deze manier kunnen de services zich registreren [Litlle, Weber ’03 b].

Eenmaal alle applicatiewerk is voltooid kan de transactie worden beëindigd om alle werk permanent te maken. Daarvoor registreert de cliënt zich voor het Completion of CompletionWithAck protocol. Eenmaal geregistreerd kan de cliënt de coördinator laten weten of ze de transactie wil voltooien (commit) of annuleren (rollback). De coördinator zal dan op zijn beurt de cliënt informeren over het resultaat: transactie voltooid (committed) of afgebroken (aborted). De CompletionWithAck gaat nog een stap verder en moet het resultaat onthouden totdat de cliënt nogmaals een bevestigingsbericht (notified) stuurt naar de coördinator [Litlle, Weber ’03 b].

Alvorens het resultaat van de transactie gekend is, worden er eerst nog andere protocollen gebruikt. Indien een service een waarschuwing wil krijgen van de coördinator juist voordat een 'commit' begint, zal ze zich registreren voor het optionele PhaseZero Protocol. Applicaties die voor zo'n protocol registreren hebben meestal gegevens in een vluchtig geheugen dat ze permanent willen maken vooraleer de 'commit' begint. De coördinator zal een '`PhaseZero`' bericht sturen waarop de service dan antwoordt met een '`phaseZeroCompleted`' of '`error`' bericht. In geval van een '`error`' zal de transactie ongedaan gemaakt worden (rollback) [Litlle, Weber '03 b].

Het volgende protocol dat wordt uitgevoerd is 2PC (Two Phase Commit). Het 'Two Phase Commit' protocol wordt gebruikt om een gemeenschappelijke overeenkomst te bekomen tussen de verschillende services, zoadat de transactie veilig kan worden beëindigd. In de eerste fase start de coördinator door een '`prepare`' bericht te sturen naar elke betrokken service. De service kan dan reageren met een van volgende berichten: `prepared`, `aborted` of `read-only`. Een '`prepared`' bericht betekent dat de service klaar is om de transactie succesvol af te sluiten en dat het dus de nodige toestandswijzigingen heeft opgeslagen om eventueel later te kunnen gebruiken in een 'rollback' of een 'commit'. Door alleen de toestandswijzigingen op te slaan, gaat de originele toestand niet verloren. Bij een '`aborted`' bericht kan de service niet worden afgesloten en zal er dus uiteindelijk een 'rollback' gebeuren. Een '`read-only`' bericht wordt gebruikt indien er kan worden afgesloten, maar wanneer de service niet zal deelnemen in de tweede fase. Dit gebeurt indien er geen wijzigingen zijn aangebracht die permanent gemaakt moeten worden [Verma, Deswal '03], [Litlle, Weber '03 b].

Indien er in deze eerste fase geen fouten worden opgeworpen zal de coördinator overgaan naar de tweede fase door elke betrokken service een '`commit`' bericht te sturen. Elke service kan nu het werk permanent maken. Na het beëindigen van het 2PC protocol kan het Completion of CompletionWithAck protocol verder worden afgewerkt.

Soms toont een service interesse van wanneer en hoe een transactie is afgesloten. Het kan zich daarvoor registreren voor het OutcomeNotification protocol. De coördinator zal deze services inlichten via een '`commited`' of '`aborted`' bericht en zal het resultaat van de transactie onthouden totdat de service reageert met een '`notified`' bericht [Litlle, Weber '03 b].

2.4.4 Business Activity

Business activity is het tweede coördinatietype voor transacties en wordt gebruikt bij langdurige interacties. Business activity is speciaal ontwikkeld voor transacties waar het 'vergrendelen' van bronnen onmogelijk of onpraktisch is. Een BA kan opgedeeld worden in verschillende scopes. Een scope stelt een eenheid van werk voor die gebruik kan maken van verschillende Web services. De scopes kunnen hiërarchisch worden opgedeeld in parent-child relaties. Een parent selecteert welke childs zullen meebeslissen over het resultaat van de transactie. Op deze manier zijn niet-atomische resultaten mogelijk. Een parent kan ook de fouten opvangen van een child, deze fout behandelen, en dan verder werken. Een fout heeft dus niet altijd een rollback tot gevolg. Wanneer een child scope is voltooid kan het de BA verlaten of kan het de parent inlichten dat het gedane werk later kan worden gecompenseerd. Op deze manier kan de parent indien het nodig zou blijken de transactie ongedaan te maken, de compensatie activiteit van een child aanroepen [Weber '03].

Het BA model heeft twee verschillende protocollen: BusinessAgreement en BusinessAgreementWithComplete. Waar de protocollen van een Atomic Transaction gedreven werden van coördinator naar de deelnemers, zijn de business activity protocollen opwaarts gedreven vanuit de deelnemers.

Wanneer bij het BusinessAgreementProtocol een child zijn werk heeft gedaan en niet verder wenst deel te nemen zal het een '`exited`' bericht sturen naar zijn parent. Een child kan beslissen niet verder deel te nemen in de transactie als het toch geen werk heeft verricht dat moet kunnen worden gecompenseerd. Indien het child echter verder wenst deel te nemen aan de BA dan moet het in staat zijn het gedane werk te compenseren. Het zal dan een '`completed`' bericht sturen naar de parent dat het voltooid is en wachten op een bericht van de parent over het uiteindelijke resultaat van de BA. Dit zal een '`close`' bericht zijn indien succesvol en een '`compensate`' bericht indien de parent wenst dat het werk van het child ongedaan wordt gemaakt. Indien het child zijn werk niet kan afmaken zal het een '`faulted`' bericht sturen, waarop de parent zal antwoorden met een '`forget`' bericht. De parent kan ook beslissen een child te annuleren met een '`cancel`' bericht, waarop het child zal reageren met een '`canceled`' bericht [Verma, Deswal '03].

Het BusinessAgreementProtocolWithComplete is identiek met het vorige protocol uitgenomen dat een child afhankelijk is van de parent om te weten of het zijn werk al heeft voltooid. De parent informeert het child wanneer het geen request meer moet verwachten en het zijn taak in de transactie heeft beëindig. Het child verwacht een '`complete`' bericht dat het zal beantwoorden met een '`completed`' bericht [Verma, Deswal '03].

2.4.5 Business Activity en BPEL4WS

WS-AtomicTransaction wordt vaak gebruikt bij interne systemen waar de transacties van korte duur zijn. AtomicTransactions zijn dan ook uitermate geschikt om back-end systemen als databases aan te spreken. Bij het modelleren van processen tussen verschillende bedrijven, komen echter vaak langdurige transacties voor. De reactietijd van andere bedrijven kan van enkele minuten tot zelfs verschillende dagen duren. Door deze langdurige interacties is het inefficiënt om bronnen te vergrendelen met een 'locking' mechanisme zoals in gewone traditionele ACID-transacties. Bij een fout in de transactie moet het gedane werk ongedaan worden gemaakt. Nu is het mogelijk dat er voor de fout werd ontdekt al veel werk wel succesvol is uitgevoerd. Daardoor is het onmogelijk om nog een rollback uit te voeren en moeten er compensatieacties gestart worden op applicatieniveau. Deze compensatie houdt in dat het logische omgekeerde van elke activiteit die behoort tot het afgebroken proces moet worden uitgevoerd, van het meest recente terug naar het eerst uitgevoerde. Dit model is nu het standaard compensatiemodel ondersteund door BPEL4WS (zie: 2.3.7 Compensatie-acties). De BPEL4WS specificatie stelt WS-BusinessActivity dan ook voor als hét protocol om transacties te beheren [Weber '03].

2.5 Conclusie

We zijn dit hoofdstuk begonnen met de historiek van de BPEL4WS-specificatie en met een verduidelijking van de twee procesbenaderingen, namelijk abstracte en uitvoerbare processen. Het grootste deel van dit hoofdstuk is echter gewijd aan de bespreking van de volledige BPEL4WS 1.1 specificatie.

Een BPEL-proces beschrijft interacties met verschillende partners. Om de rollen te definiëren die het proces en zijn partners spelen in de interacties, wordt gebruik gemaakt van partnerlinks. De procestoestand kan worden opgeslagen aan de hand van variabelen. Deze procestoestand bestaat uit de verzonden en ontvangen berichten en uit andere gegevens nodig voor de businesslogica. Om ervoor te zorgen dat een bericht altijd bij de juiste procesinstantie terecht komt, worden correlationsets gebruikt. BPEL voorziet ook in verschillende basisactiviteiten om synchrone en asynchrone interacties te modelleren. De gestructureerde activiteiten dienen om de volgorde van uitvoering te bepalen, zoals de sequentie en de lusconstructie.

De specificatie voorziet ook in mogelijkheden om het proces op te splitsen in logische bedrijfseenheden door middel van een scope. Elke scope kan dan worden vergezeld door een compensatieactie. Een compensatieactie kan worden aangeroepen om een scope ongedaan te maken. FaultHandlers tenslotte zorgen voor het opvangen en behandelen van fouten.

We zijn het hoofdstuk geëindigd met de bespreking van twee protocollen die terzelfder tijd zijn vrijgegeven met de BPEL4WS-specificatie, namelijk WS-Coordination en WS-Transaction. WS-Coordination zorgt voor de coördinatie van acties van verschillende services. WS-Transaction bestaat uit twee toepassing van WS-Coordination. Atomic Transactions worden gebruikt voor transacties van korte duur, terwijl Business Activities instaan voor langdurige transacties.

In het volgende hoofdstuk gaan we de positieve en negatieve aspecten van de BPEL4WS-specificatie belichten. We kijken tevens in welke mate de specificatie kan worden uitgebreid om sommige knelpunten op te lossen.

Hoofdstuk 3 Beoordeling van de BPEL4WS 1.1 specificatie

In dit hoofdstuk bekijken we eerst de verschillende doelstellingen die de BPEL4WS-specificatie voor ogen had en analyseren we of die ook echt zijn behaald. Vervolgens staan we in paragraaf 3.2 even stil bij het gebruiksgemak van beide procesbenaderingen in BPEL. Tenslotte toetsen we BPEL aan een aantal workflow - en communicatie-patronen in paragraaf 3.3.

3.1 Doelstellingen van BPEL4WS

In deze paragraaf bespreken we de doelstellingen die als basis dienden voor de ontwikkeling van de BPEL4WS-specificatie. De auteurs van deze specificatie beschrijven in een nota aan het WSBPEL Technical Comittee tien doelstellingen [Leymann, Roller, Thatte '03]. Deze doelstellingen worden hieronder kort weergegeven. We zullen ook onderzoeken of deze doelstellingen gerealiseerd zijn in de BPEL4WS 1.1 specificatie.

3.1.1 Doelstelling 1, Web services als basis

Doelstelling 1: Het is belangrijk dat BPEL4WS vertrekt van de bestaande Web service-architectuur. BPEL4WS heeft als doel verschillende Web services te orchestreren in een bedrijfsproces. Het proces interageert met Web services via een WSDL-interface. Het BPEL-proces is zelf ook een Web service en stelt dus ook een WSDL-interface ter beschikking. BPEL moet de interacties op abstract niveau definiëren, dit wil zeggen onafhankelijk van de effectieve bindingsinformatie. Op deze manier wordt het procesmodel eenvoudig en herbruikbaar. Door de hechte samenwerking van BPEL met WSDL is het tevens belangrijk dat er rekening gehouden wordt met toekomstige versies van de WSDL-specificatie.

In het vorige hoofdstuk hebben we al besproken dat er een sterke relatie is met het WSDL-protocol. Zo worden er in de partnerlinks verwijzingen gemaakt naar WSDL-poorttypes, een variabele kan een berichttype bezitten dat beschreven is in WSDL en de identificatie van procesinstanties wordt verwezenlijkt door WSDL-properties. De wens om abstracte interacties te beschrijven is verwezenlijkt door onderscheid te maken tussen poort en poorttype. Een poorttype is de abstracte interface van een service. Het

poorttype beschrijft welke operaties kunnen worden aangeroepen en welke berichten kunnen worden verstuurd. Een poort daarentegen bestaat uit effectieve bindingsgegevens, namelijk het adres van de endpoint. Een BPEL-proces wordt nu verbonden met abstracte poorttypes via een partnerlink. De eigenlijke verbinding tussen poort en poorttype gebeurt dan in een WSDL-document [W3C WSDL '02], [BPEL4WS Spec '03].

BPEL4WS is momenteel gebaseerd op WSDL 1.1. De W3C WSDL WG is echter bezig met de volgende versie, namelijk WSDL 1.2. Volgens issue #47 van het WSBPEL TC [OASIS WSBPEL TC '04] is er beslist toch verder te werken met WSDL 1.1, omdat de nieuwe versie nog steeds in ontwikkeling is en er onzekerheid heerst over de beschikbaarheid ervan. Van enkele concepten die zullen verdwijnen (bijvoorbeeld wsdl:message), wordt onderzocht wat de impact daarvan zal zijn op BPEL. Verder zal er een formele samenwerking met de W3C WSDL WG worden opgericht om de wijzigingen op te kunnen volgen.

3.1.2 Doelstelling 2, XML voor de structuur

Doelstelling 2: BPEL4WS beschrijft bedrijfsprocessen in een XML-taal. BPEL4WS bekommert zich niet over grafische presentatiemogelijkheden van het proces noch over een ontwerpmethodologie.

Net als de meeste andere Web service-protocollen is BPEL gebaseerd op XML. Een BPEL-document wordt gevalideerd tegen een XML Schema. De URL naar het schema wordt opgenomen als attribuut in het `<process>` element. Het schema beschrijft de structuur en welke elementen in een BPEL-document mogen voorkomen. Het is een volledige beschrijving van de BPEL-taal. Er komen steeds meer tools op de markt om BPEL-processen te modelleren en grafisch weer te geven. In hoofdstuk 4 van deze verhandeling zullen we enkele van deze tools bespreken.

3.1.3 Doelstelling 3, gemeenschappelijke verzameling van kernconcepten.

Doelstelling 3: BPEL4WS moet in een aantal orchestratieconcepten voorzien die zowel externe (abstract) als interne (uitvoerbare) bedrijfsprocessen kunnen beschrijven. De concepten worden daarbij zoveel mogelijk gemeenschappelijk gebruikt. Men probeert

zo min mogelijk uitbreidingen te definiëren die slechts voor één procesbenadering dienen.

Zoals gezien in het vorige hoofdstuk is het grootste deel van de specificatie van toepassing op zowel uitvoerbare als abstracte processen. Voor elke procesbenadering zijn er slechts enkele uitbreidingen gemaakt. Zo kan een abstract proces gemakkelijk als basis gebruikt worden om een uitvoerbaar proces te maken. Omgekeerd kan van een uitvoerbaar proces een abstract gemaakt worden door de interne details weg te laten.

In issue #24 van het WSBPEL TC [OASIS WSBPEL TC '04] is besloten om verschillende XML Schema's aan te maken voor de uitvoerbare en voor de abstracte procesbenadering van BPEL. Het voordeel van gescheiden schema's is dat abstracte en uitvoerbare processen apart gevalideerd kunnen worden. Met één enkel schema is dit onderscheid niet mogelijk. Er kan dan enkel worden gevalideerd of men te maken heeft met een geldig BPEL-proces. Men is er wel nog niet over eens of er twee schema's, één voor elke procesbenadering, of drie schema's, één voor de gemeenschappelijke concepten en één voor elke uitbreiding, moeten worden aangemaakt. Het issue wordt nu verder onderzocht door het Specification Editor Team.

3.1.4 Doelstelling 4, betreffende het controlegedrag

Doelstelling 4: BPEL4WS zou zowel graafgestructureerde als hiërarchische controlestructuren moeten voorzien. Het gebruik van beide zou naadloos op elkaar moeten aansluiten.

Deze doelstelling volgt rechtstreeks uit de combinatie van de compositietalen WSFL en XLANG. De hiërarchische controleconstructies vinden we terug in de BPEL4WS-specificatie als de gestructureerde activiteiten, namelijk `<sequence>`, `<switch>`, `<while>`, `<pick>` en `<flow>`. De graafgerichte methode om gebruik te maken van controlelinks en condities vinden we ook terug in BPEL4WS, namelijk in de `<link>` activiteit. De link kan verschillende activiteiten met elkaar verbinden aan de hand van transitie - en joincondities. Een link kan dus over de grenzen van een gestructureerde activiteit gebruikt worden. Hierdoor zijn de flexibele eigenschappen van een graaf van

toepassing. BPEL4WS ondersteunt dus beide controlestructuren. In paragraaf 3.3 zullen we de voor - en nadelen bespreken van deze combinatie van controlestructuren.

3.1.5 Doelstelling 5, betreffende gegevensafhandeling

Doelstelling 5: BPEL4WS beperkt de gegevenshantering tot het beschrijven en wijzigen van relevante gegevens voor de opbouw van berichten en voor het sturen van de controlestroom. Meer geavanceerde gegevensbewerkingen moeten in de eigenlijke bedrijfsfuncties plaatsvinden en niet in het bedrijfsproces.

Gegevens komen voor in verzonden en ontvangen berichten, en in de condities die zorgen voor het gedrag van het proces. De enige activiteit om gegevens in variabelen en properties te wijzigen is de `<assign>` activiteit. In combinatie met gegevens-uitdrukkingen en XPath-functies kunnen condities en berichten worden samengesteld.

3.1.6 Doelstelling 6, betreffende properties en correlatie

Doelstelling 6: BPEL4WS moet een identificatiemechanisme ondersteunen voor procesinstanties. De instantie-identificatie moet op applicatieniveau gedefinieerd kunnen worden.

Wanneer berichten naar een proces worden gestuurd, is het belangrijk dat die berichten bij de juiste procesinstantie terecht komen. Elk bericht moet dus de juiste procesinstantie kunnen identificeren. Er wordt daarvoor gebruik gemaakt van één of meerdere sleutelgegevens die zich in het bericht zelf bevinden. Welke gegevens als procesidentificatie worden gebruikt, wordt beschreven via BPEL-correlationsets en WSDL-properties (zie paragraaf 2.3.3). Daardoor is het bepalen van de identificatie heel flexibel en kan deze dus gemakkelijk worden aangepast. De ontwikkelaar kan immers zelf beslissen welke gegevens worden gebruikt voor procesidentificatie.

3.1.7 Doelstelling 7, betreffende de levenscyclus van het proces

Doelstelling 7: BPEL4WS moet in staat zijn om procesinstanties te creëren en te beëindigen. Geavanceerde operaties zoals ‘suspend’ en ‘resume’ moeten in overweging genomen worden voor toekomstige versies.

Door mogelijke langdurige processen en asynchrone interacties is er nood aan ‘statefull’ Web services, daar gewone WSDL services ‘stateless’ zijn. Het is echter niet de bedoeling om twee soorten Web services te onderscheiden. Een klant mag er zich niet van bewust zijn of hij berichten stuurt naar een eenvoudige stateless-service of naar een service met achterliggend statefull-proces. Daardoor wordt er gekozen voor een impliciet beheer van de proceslevenscyclus. Dit wil zeggen dat een instantie automatisch wordt aangemaakt bij het begin van een proces. Om een proces te starten moet er een verzoek naar dit proces gestuurd worden. Het begin van een proces komt dus altijd overeen met het ontvangen van een verzoekbericht. De `<receive>` en `<pick>` activiteit kunnen allebei berichten ontvangen en hebben dus ook de mogelijkheid automatisch een instantie te starten door middel van het attribuut ‘`createInstance`’. Een procesinstantie wordt afgebroken indien het proces ten einde is of kan worden afgebroken door de `<terminate>` activiteit.

Volgens issue #5 van WSBPEL TC is het invoeren van suspend/resume operaties buiten het bereik van een eerste versie van de specificatie. Deze operaties kunnen namelijk ook door de infrastructuur worden behandeld en hoeven dus niet meer in het procesmodel aanwezig te zijn. Het wachten voor een langere periode kan echter al worden gemodelleerd aan de hand van een `<pick>` of `<eventHandlers>` activiteit.

3.1.8 Doelstelling 8, betreffende het model voor langdurige transacties

Doelstelling 8: BPEL4WS moet voorzien in een model voor langdurige transacties met technieken als compensatie en gebruik van scopes. Scopes zorgen ervoor dat bij herstelacties slechts kleine delen worden gecompenseerd in plaats van de globale scope.

BPEL4WS bezit een `<scope>` element waarmee activiteiten kunnen gegroepeerd worden in afzonderlijke bedrijfseenheden. Elke scope kan zijn eigen compensatieacties beschrijven. Compensatieacties kunnen worden aangeroepen in geval van een fout. Doordat compensatie gebeurt op scopeniveau, hoeven niet alle wijzigingen ongedaan gemaakt te worden. Scope en compensatie zorgen ervoor dat de fout zo lokaal mogelijk wordt opgelost. Zoals we zullen zien in paragraaf 3.2.2 worden er van de specificatie nog bijkomende vereisten verwacht betreffende Business Transactie Management (BTM).

3.1.9 Doelstelling 9, betreffende modularisatie

Doelstelling 9: BPEL4WS moet Web services gebruiken als model voor procesdecompositie en procesassemblage.

Een bedrijfsproces is een assemblage van verschillende Web services in een nieuwe Web service. Het BPEL-proces wordt immers zelf als een WSDL-service beschreven. Op deze manier kan een BPEL-proces zelf als subproces in een ander bedrijfsproces worden gebruikt. BPEL4WS voorziet dus in een flexibel en recursief aggregatiemodel.

3.1.10 Doelstelling 10, Compositie met andere Web service functionaliteit

Doelstelling 10: BPEL4WS moet samenwerken met compatibele Web service-standaarden. Alleen als een bepaalde vereiste nog niet aanwezig is, mag deze worden opgenomen in de BPEL4WS-specificatie of moet een andere standaard worden ontwikkeld.

BPEL4WS probeert zich te beperken tot structuren die bijdragen tot het modelleren van bedrijfsprocessen, daarbij zoveel mogelijk gebruik makend van bestaande protocollen. Indien er voor het modelleren van bedrijfsprocessen bijkomende functionaliteit vereist is, wordt er overwogen een nieuw Web service-mechanisme te ontwikkelen. WS-BusinessActivity is een voorbeeld van een dergelijke situatie. Dit protocol beschrijft het gedrag van, op compensatie gebaseerde, herstelacties van BPEL4WS. Ook het vervangen van service referenties door ‘Endpoint referenties’, gedefinieerd in WS-Adressing, is een voorbeeld van hoe men probeert BPEL4WS te herleiden tot alleen de kernconcepten van procesmodellering.

3.2 Bruikbaarheid van BPEL4WS

In deze paragraaf worden de twee procesbenaderingen (abstracte en uitvoerbare processen) van dichtbij bekeken en wordt onderzocht hoe de specificatie kan worden uitgebreid om ontbrekende functionaliteit toe te voegen. We analyseren hier BPEL op hoog niveau. In paragraaf 3.3 zullen we vervolgens een analyse op laag niveau voeren, door verschillende workflow - en communicatiepatronen gedetailleerd te bespreken.

3.2.1 Modelleren van abstracte processen of business protocollen

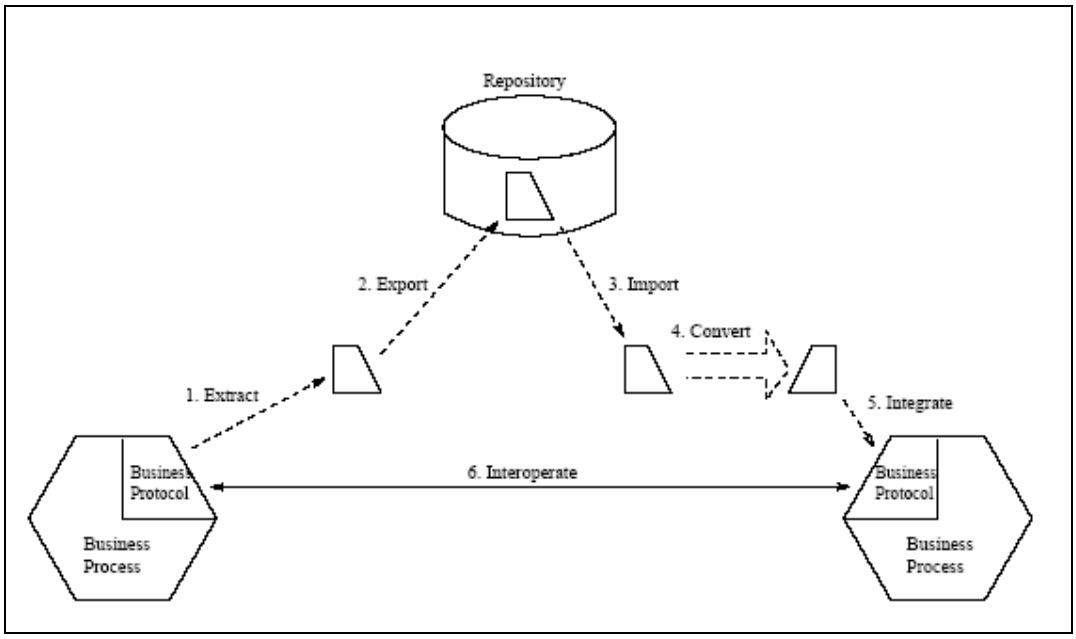
Een abstract proces is volgens de specificatie een beschrijving van de zichtbare (publieke) uitwisseling van berichten tussen de betrokken partijen zonder daarbij interne details vrij te geven.

Het laatste woord over abstracte processen is echter nog niet gezegd. Leden van de Web Services Business Process Execution Language TC zijn nog steeds bezig aan het discussiëren over verschillende aspecten van abstracte processen. Vele issues uit de WS BPEL Issues List betreffende abstracte processen zijn immers nog niet afgesloten. Volgens issue #82 bijvoorbeeld moeten abstracte processen duidelijker in de specificatie worden omschreven. Naar aanleiding van een bijeenkomst van de *Abstract BPEL Clarification Working Group* werd een topic [Rossomando '04] geopend in de *OASIS WSBPEL elist* om abstracte processen te bespreken. In een tekst van Satish Thatte wordt het gebruik van abstracte processen verder verduidelijkt [Thatte '04].

Abstracte processen kunnen worden gebruikt in twee verschillende scenario's. Ten eerste kan een abstract proces dienen om de structurele service-interface uit te breiden met gedragsaspecten. Een WSDL-interface beschrijft enkel welke operaties kunnen worden aangeroepen en welke berichten kunnen worden gestuurd. Daar een Web service kan bestaan uit een proces, is het vaak nodig een bijkomende beschrijving te definiëren, zoals de volgorde van de te versturen berichten. Het abstracte proces bepaalt dus hoe consumenten op een correcte manier met de Web service moeten interageren [Thatte '04].

In figuur 44 wordt grafisch voorgesteld hoe een partner automatisch kan interageren met een bedrijfsproces. In de eerste plaats zou het uitvoerbare proces moeten worden omgezet in een vorm die publiceerbaar is. Dit wil zeggen dat alle interne details moeten worden verwijderd en dat alleen deze delen van het proces worden gebruikt die de choreografie met de partner beschrijven. Eenmaal het abstracte proces is opgemaakt, kan het worden gepubliceerd via een repository (2) of kan het onmiddellijk worden bezorgd aan een partner. De bedrijfspartner zou het abstracte proces dan kunnen gebruiken om het te integreren in een bestaand of nieuw BPEL proces (5). De partner zal zijn eigen uitvoerbare proces aanpassen aan het abstracte proces. Afhankelijk van de complexiteit van de situatie loopt de integratie zeer vlot of is het mogelijk dat er eerst

wat aanpassingen nodig zijn aan het abstracte proces (4). Nadat de integratie is voltooid kunnen beide processen automatisch samenwerken (6). Beide uitvoerbare processen zijn nu immers door middel van het abstracte proces op elkaar afgestemd.



Figuur 44 Bedrijfsintegratie met abstracte processen (bron: Haataja, Tergujeff '03)

Waar we in het eerste scenario een abstract proces ontwikkelden dat een beschrijving was van een reeds bestaand uitvoerbaar proces, wordt in een tweede scenario een uitvoerbaar proces ontwikkeld aan de hand van een abstract proces. Het abstract proces dient als template voor de implementatie van een uitvoerbaar proces.

Stel bijvoorbeeld dat er een overkoepelende organisatie van reisagenten bestaat die een gestandaardiseerde boekingsservice wil ontwikkelen. Een onderdeel van die standaard bestaat uit een abstract proces dat bepaalt op welke manier de service zich moet gedragen. Wanneer een reisagent nu een boekingsservice wil implementeren, zal er rekening gehouden moeten worden met het abstracte proces. De interne details en verwerking kunnen van elke service anders zijn, het publieke gedrag daarentegen is uniform.

Om abstracte processen te modelleren staat de specificatie toe bepaalde elementen weg te laten indien ze niet relevant zijn voor het observeerbare gedrag. Issue #107 van de WSBPEL Issue List stelt een tweede manier voor om in abstracte processen interne details te verbergen, namelijk expliciete verberging. De impliciete manier laat de elementen gewoon weg uit de procesbeschrijving. Bij de expliciete manier wordt de weg te laten activiteit vervangen door een `<opaque>` activiteit en kunnen attributen de waarde '`opaque` ' krijgen.

Er worden verschillende reden aangehaald om de expliciete manier in de specificatie op te nemen. Een eerste reden is voor error-detectie. Bij de impliciete manier is er geen enkele manier om te bepalen of een element is weggelaten of is vergeten. Door `<opaque>` activiteiten te gebruiken maakt men duidelijk dat activiteiten zijn weggelaten. Een tweede reden situeert zich bij het gebruik van abstracte processen als templates. Op deze manier kan door `<opaque>` activiteiten worden aangeven waar elementen ontbreken die in een uitvoerbaar proces van belang zijn. Een laatste reden is het wegnemen van het syntactische verschil tussen uitvoerbare en abstracte processen.

Naar aanleiding van issue #107 is er een resolutie voorgesteld waarin een `<opaque>` activiteit zal worden geïntroduceerd in de BPEL-specificatie. Deze activiteit is enkel beschikbaar voor abstracte processen en bezit dezelfde standaard elementen en attributen als andere BPEL-activiteiten. Vervolgens zal ook de gereserveerde waarde '`##BPELopaque`' kunnen worden gebruikt als attribuutwaarde.

3.2.2 Modelleren van uitvoerbare processen

BPEL4WS is zeer flexibel en bezit een groot uitdrukkingsvermogen om uitvoerbare processen te beschrijven dankzij de samensmelting van twee methodes, de graaf – en de blokgestructureerde methode. In paragraaf 3.3 zullen we dit uitdrukkingsvermogen grondig analyseren en vergelijken met andere compositietalen. Eerst bespreken we echter enkele aanvullingen op uitvoerbare processen, zoals ondersteuning voor Business Transaction Management en asynchrone berichtuitwisseling.

Het gebrek aan ondersteuning voor Business Transaction Management (BTM) wordt momenteel in het OASIS WS-BPEL TC besproken. Choreology Ltd. heeft namelijk

enkele suggesties voor de BPEL4WS 1.1 specificatie voorgesteld betreffende het invoeren van enkele BTM constructies [Fletcher, Furniss, Green, Haugen, ‘03]. Deze constructies zouden zowel met WS-Transaction, Business Tansaction Protocol (BTP) als Web Service Transaction Management (WS-TXM) moeten kunnen samenwerken. Het integreren van BTM met BPEL heeft tot doel betrouwbare en consistente compositie van Web services te ondersteunen [Green ’03 a].

De BPEL-syntax zou moeten worden uitgebreid om transacties te kunnen starten, meer bepaald door het creëren van een transactiecontext. Bij het aanroepen van een Web service moet het mogelijk zijn de transactiecontext mee te sturen met het bericht aan de hand van de `<invoke>` activiteit. Op dezelfde manier moet een transactiecontext door een `<receive>` activiteit kunnen worden ontvangen zodat de context kan worden opgeslagen in een variabele. Er is ook een manier nodig om aan te duiden dat een proces een transactie wil beëindigen. Het proces moet in staat kunnen zijn om aan te geven of de voltooiing van de transactie negatief (geannuleerd/gecompenseerd) of positief (geconfirmeerd) is. Zoals al aangehaald bij ‘Business Activities’ kan de transactie zelf beslissen welke partijen zullen deelnemen in de transactie, het voltooien van een transactie moet dus op een selectieve manier kunnen gebeuren. Een bepaalde subset van partijen kan op deze manier worden geconfirmeerd of geannuleerd. Ten slotte moet er ook ondersteuning zijn om bedrijfsprocessen te laten registreren als deelnemer aan een transactie [Green ’03 b].

Al deze voorstellen (issue #53 - #59) worden beschreven in de ‘WS BPEL Issues List’ [OASIS WS-BPEL TC ‘04]. Daarbij wordt ook verwezen naar een document van Choreology Ltd. [Fletcher, Furniss, Green, Haugen, ‘03] waarin wordt voorgesteld welke concrete constructies zouden kunnen worden gebruikt. Alle issues inzake BTM zijn momenteel afgesloten zonder veranderingen in de specificatie. De issues komen wel in aanmerking voor herziening. De voornaamste reden om de BTM issues uit te stellen is dat er te weinig tijd is om alles klaar te krijgen voor de eerste versie van de gestandaardiseerde specificatie. Daarbij is er nog onenigheid over de aanpak om BTM features te implementeren. Vele leden van het TC vinden het dan ook geen prioriteit.

Door volledige bedrijfsprocessen als Web service beschikbaar te maken, wordt asynchrone communicatie steeds belangrijker. Processen worden meestal niet

onmiddellijk uitgevoerd, maar over een langere periode. Om asynchrone berichtgeving mogelijk te maken is het belangrijk dat de toestand van het proces wordt opgeslagen. In doelstelling 7 werd beslist om geen apart model te definiëren voor statefull services. De creatie van een procesinstantie zou automatisch gebeuren bij het aanroepen van een startactiviteit (=activiteit met het attribuut `createInstance`). Er wordt dus voor de gebruiker abstractie gemaakt van het feit of een Web service statefull of stateless is. Het creëren van instanties wordt niet alleen door BPEL vereist, maar ook door andere protocollen. Het zou dus handig zijn een algemene en gestandaardiseerde methode te hebben om met asynchrone services om te gaan.

Momenteel wordt er in een OASIS TC werk gemaakt van de Asynchronous Service Access Protocol (ASAP) specificatie [OASIS WS-ASAP TC '03] die als doel heeft algemene voorzieningen aan te bieden om asynchrone services mogelijk te maken zowel in BPEL als in andere protocollen. ASAP is een extensie voor SOAP die workflowtalen (zoals BPEL) kan voorzien van asynchrone faciliteiten. Zo biedt ASAP de mogelijkheid om instanties aan te maken van asynchrone Web services. ASAP zou dus een vervanging kunnen zijn voor doelstelling 7. In plaats dat BPEL zorgt voor de creatie van een procesinstantie, zou ASAP daar nu kunnen voor instaan. ASAP zou er dan ook voor zorgen dat de verschillende asynchrone berichten aan elkaar zijn gecorreleerd. Hoe de ontwikkeling van ASAP de BPEL-specificatie zal beïnvloeden valt af te wachten.

Een laatste opmerking betreft het gebruik van WSDL. De sterke koppeling tussen BPEL en WSDL maakt het eenvoudig om WSDL-services te orchestreren in een BPEL-proces. Het nadeel is dat wanneer een systeem geen WSDL ondersteunt voor de beschrijving van de interface, er een soort omzettingsmechanisme zal ontwikkeld moeten worden om gebruik te kunnen maken van BPEL4WS.

3.3 Patroongebaseerde analyse

In deze paragraaf evalueren we de BPEL4WS-specificatie aan de hand van de patroongebaseerde analyse van van der Aalst, Dumas, Hofstede en Wohed ('03). Deze patronen zijn abstracte vormen van steeds weerkerende situaties in de verschillende

fases van softwareontwikkeling. We bekijken in de volgende analyse een verzameling van workflowpatronen en een verzameling van communicatiepatronen.

De workflowpatronen (WP's) definiëren de behoeftes van een workflow om de controlestroom te modelleren. De controlestroom bepaalt de activiteiten en hun volgorde van uitvoeren. De workflowpatronen zijn verkregen door de analyse van verschillende workflowtalen [van der Aalst, Hofstede, Kiepuszewski, Barros '03] en dienen om een workflowmanagementsysteem te beoordelen naar geschiktheid en uitdrukkingsvermogen. De workflowpatronen kunnen worden gebruikt om Web service-compositietalen te evalueren aangezien de situaties beschreven in workflows ook voorkomen in Web service-composities.

De communicatiepatronen (CP's) beschrijven hoe de interacties tussen verschillende systemen gebeuren bij Enterprise Application Integration. Men maakt onderscheid tussen synchrone en asynchrone interacties en tussen point-to-point en multicast. Men kan deze patronen gebruiken om de communicatiemogelijkheden tussen Web services te modelleren [van der Aalst, Hofstede, Kiepuszewski, Barros '03].

3.3.1 Workflowpatronen in BPEL4WS

Web service-compositie en workflowmanagement staan met elkaar in verband omdat ze beiden betrokken zijn bij het beschrijven van uitvoerbare processen. We bespreken nu de 20 workflowpatronen en in welke mate deze worden toegepast in BPEL4WS. Alle patronen worden ook met een voorbeeld geïllustreerd.

3.3.1.1 WP1: Sequentie

"In een sequentie wordt een activiteit in een proces pas beschikbaar gesteld, na de voltooiing van een voorgaande activiteit in hetzelfde proces."

BPEL4WS kan dit patroon op twee verschillende manieren modelleren, blokgestructureerd en graafgestructureerd. De blokgestructureerde methode gebruikt de `<sequence>` activiteit (zie Lijst 1). Hier worden de activiteiten in volgorde van voorkomen uitgevoerd. De graafgestructureerde methode modelleert de sequentie door gebruik te maken van de `<link>` constructie (zie Lijst 2). Daarbij worden beide activiteiten in een `<flow>` geplaatst en wordt er een link aangemaakt. Vervolgens krijgt

de activiteit die eerst moet worden uitgevoerd een `<source>` element en de activiteit die erna moeten worden uitgevoerd een `<target>` element. Activiteit B kan pas starten als de linkstatus positief is geëvalueerd en de evaluatie van de link gebeurt pas wanneer activiteit A is voltooid.

Lijst 1

```
<sequence>
   activiteit A
   activiteit B
</sequence>
```

Lijst 2

```
<flow>
   <links>
      <link name="L" />
   <links>

   activiteit A
      <source linkName="L" />

   activiteit B
      <target linkName="L" />
</flow>
```

Het *loanFlow* voorbeeld uit hoofdstuk 2 gebruikt verschillende sequenties. De asynchrone interactie met een *loanService* bestaat bijvoorbeeld uit de sequentie van een `<invoke>` en een `<receive>` activiteit (zie figuur 10 en 28 in hoofdstuk 2).

3.3.1.2 WP2: Parallelle splitsing

"Bij een parallelle splitsing wordt de controledraad gesplitst in meerdere controledraden die simultaan of in om het even welke volgorde kunnen worden uitgevoerd."

Het parallel uitvoeren van verschillende activiteiten kan aan de hand van een `<flow>` structuur worden gemodelleerd. Alle activiteiten daarbinnen kunnen terzelfder tijd worden uitgevoerd. In het *loanFlow* voorbeeld wordt een parallelle splitsing gebruikt om simultaan twee verschillende *loanServices* aan te spreken (zie figuur 10 in hoofdstuk 2).

3.3.1.3 WP3: Synchronisatie

"Verschillende parallelle vertakkingen worden samengevoegd in één enkele controle-draad. Een vertakking die is voltooid, kan niet opnieuw worden uitgevoerd zolang men wacht op de voltooiing van de andere vertakkingen. De draden die gesynchroniseerd worden, moeten tot dezelfde procesinstantie behoren."

Een voorbeeld van synchronisatie zou zich kunnen voordoen bij de verwerking van een order. Er zijn bijvoorbeeld twee subprocessen. Enerzijds voor de afhandeling van de betaling van een voorschot en anderzijds voor het beheer van de voorraad. Beide subprocessen zijn onafhankelijk van elkaar en kunnen parallel worden uitgevoerd. De levering van een order (activiteit B) kan pas worden gestart indien er een voorschot is betaald (activiteit A1) en de goederen in voorraad zijn (activiteit A2). Synchronisatie houdt dus in dat alle parallelle activiteiten moeten worden voltooid alvorens verder te gaan. Dit patroon wordt ook wel een AND-join genoemd.

Lijst 3

```
<sequence>
   <flow>
      activiteit A1
      activiteit A2
   </flow>
   activiteit B
</seuence>
```

Lijst 4

```
<flow>
   <links>
      <link name=”L1” />
      <link name=”L2” />
   </links>

   activiteit A1
      <source linkName=”L1” />
   activiteit A2
      <source linkName=”L2” />

   acitiviteit B
      joinconditie “L1 AND L2”
      <target linkName=”L1” />
      <target linkName=”L2” />
</flow>
```

In Lijst 3 kunnen we zien dat activiteit A1 en A2 in parallel worden uitgevoerd en dat activiteit B slechts kan starten eenmaal beide parallelle activiteiten zijn voltooid. Activiteit B staat immers buiten de `<flow>` structuur.

Deze constructie kan eveneens met de graafgestructureerde aanpak gemodelleerd worden (zie Lijst 4). Hier zijn er twee links geconstrueerd (L1 en L2) en krijgen activiteiten A1 en A2 beide een `<source>` element. Bij activiteit B staat er een ‘`joinconditie`’ en voor elke link een `<target>` element. De conditie “`L1 AND L2`” definieert dat beide linkstatussen positief moeten geëvalueerd worden alvorens er gestart kan worden met B.

3.3.1.4 WP4: Exclusieve keuze

“Bij de exclusieve keuze wordt één uit meerdere vertakkingen gekozen aan de hand van een conditie.”

De exclusieve keuze kan worden gemodelleerd door een `<switch>` activiteit waarbij elke vertakking wordt voorgesteld door een `<case>` element met een bepaalde conditie (zie Lijst 5). Indien men wenst gebruik te maken van links (zie Lijst 6) moet voor elke vertakking een link worden aangemaakt (L1 en L2). Er wordt een `<empty>` activiteit gecreëerd waarin alle `<source>` elementen komen te staan. Elke `<source>` heeft een transitieconditie (C1 en C2). Deze condities zullen ervoor zorgen dat de juiste vertakking wordt geselecteerd. Opdat de keuze exclusief zou zijn, moeten de transitiecondities disjunct zijn. Als dit niet het geval is kunnen er meerdere condities voldoen en kunnen er dus meerdere vertakkingen worden uitgevoerd. Het exclusief maken is dan ook geen taak van BPEL zelf, maar van de programmeur. Wanneer een `<switch>` wordt gebruikt is de keuze altijd exclusief. Bij uitvoering van een vertakking worden de andere vertakkingen immers onbeschikbaar gemaakt.

Het *loanFlow* proces gebruikt een `<switch>` om een exclusieve keuze te maken tussen twee *loanServices*. Er worden twee vertakkingen gedefinieerd waarin het bericht wordt aangemaakt om de client op de hoogte te brengen van de beslissing (zie figuur 28 in hoofdstuk 2).

3.3.1.5 WP5: Eenvoudige samensmelting

“Bij een eenvoudige samensmelting komen verschillende vertakkingen bijeen zonder synchronisatie. Synchronisatie is hier zelfs niet mogelijk daar in dit patroon verondersteld wordt dat er nooit meerdere vertakkingen parallel worden uitgevoerd.“

De eenvoudige samensmelting is een punt waar alle vertakkingen van een exclusieve keuze samenkomen. Om de eenvoudige samensmelting te illustreren heeft een klant bijvoorbeeld voor de verzending van zijn order de keuze tussen drie verschillende verpakkingen. Juist vóór het verpakken van het order, zal het proces zich splitsen om de verschillende soorten van verpakking te kunnen behandelen. Er kan slechts een pad gevolgd worden. Nadat de verpakking is voltooid, komen alle vertakkingen samen in de activiteit ‘`verzenden`’. Daar er maar één pad gekozen kan worden, hoeft er niet op de andere te worden gewacht en is er dus ook geen synchronisatie van de verschillende paden.

Lijst 5

```
<switch>
   <case conditie=”C1”>
      activiteit A1
   </case>
   <case conditie=”C2”>
      activiteit A2
   </case>
</switch>
activiteit C
```

Lijst 6

```
<flow>
   <links>
      <link name=”L1” />
      <link name=”L2” />
      <link name=”L1s” />
      <link name=”L2s” />
   </links>

   <empty>
      <source linkName=”L1”
       transitionCondition=”C1” />
      <source linkName=”L2”
       transitionCondition=”C2” />
   </empty>

   activiteit A1
      <target linkName=”L1” />
      <source linkName=”L1s” />
   activiteit A2
      <target linkName=”L2” />
      <source linkName=”L2s” />

   activiteit C
      joinconditie = “L1s OR L2s”
      <target linkName=”L1s” />
      <target linkName=”L2s” />
</flow>
```

We kunnen de eenvoudige samensmelting modelleren door na de `<switch>` een activiteit C op te nemen (zie Lijst 5). Activiteit C zal dan starten nadat één vertakking van de switch is voltooid. Bij de graafgestructureerde methode (zie Lijst 6) moet er voor elke vertakking een bijkomende link worden gedefinieerd (L1s en L2s). De `<source>` van deze links behoren bij de verschillende vertakkingen, terwijl de `<target>` van elke link bij activiteit C staat. Ook hier moet er een joinconditie worden opgenomen, namelijk “`L1s OR L2s`”. We gebruiken hier de operator OR daar er slechts één vertakking moet worden uitgevoerd. Wanneer een transitieconditie niet wordt geselecteerd en de vertakking dus niet wordt geselecteerd, wordt de link negatief geëvalueerd. De bijhorende activiteit zal deze negatieve evaluatie verder doorsturen over zijn uitgaande link (= death path elimination principe).

Bij het *loanFlow* voorbeeld staat de `<invoke>` activiteit buiten de vertakkingen van de `<swith>`. Op die manier komen beide vertakkingen van de `<switch>` terug samen in één controledraad. Daar er slechts één vertakking kan worden geselecteerd, zal na beëindiging van die vertakking onmiddellijk de `<invoke>` worden uitgevoerd. Er is immers geen synchronisatie nodig.

3.3.1.6 WP6: Meerdere keuzes

"In dit patroon kunnen er meerdere vertakkingen gekozen worden aan de hand van enkele condities. De verschillende vertakkingen worden dan parallel uitgevoerd."

Meerdere keuzes kunnen worden gemodelleerd met de graafgestructureerde methode net als in het vorige patroon (Lijst 6). Waar in de exclusieve keuze juist één vertakking werd gekozen, wordt in dit geval ook minstens één vertakking gekozen, maar mogen het er ook meer zijn. De verschillende transitiecondities zijn dus niet meer verplicht disjunct zodanig dat er meerdere links positief geëvalueerd kunnen worden. Dit patroon kan niet met een `<switch>` worden gemodelleerd, daar na uitvoering van een vertakking de andere vertakkingen onbeschikbaar worden.

Het patroon voor meerdere keuzes kan bijvoorbeeld worden gebruikt bij het toewijzen van verschillende kortingen aan een besteld product. Elke vertakking berekent het kortingsbedrag en wordt geselecteerd aan de hand van een conditie. Bij exclusieve keuze zou er bijvoorbeeld de keuze zijn tussen '*korting*' en '*geen korting*' (disjunct). Bij meerdere keuzes kan er nog een vertakking bijkomen voor een '*speciale actie*'. De vertakking '*geen korting*' en de vertakking '*speciale actie*' kunnen nu bijvoorbeeld samen voorkomen.

3.3.1.7 WP7: Gesynchroniseerde samensmelting

"De gesynchroniseerde samensmelting is de samensmelting van verschillende vertakkingen in één enkele draad. Alle actieve vertakkingen moeten worden gesynchroniseerd vooraleer de activiteit na de samensmelting kan worden uitgevoerd."

Een situatie waarin dit patroon zich zou kunnen voordoen is bij de verwerking van een order. Nadat één of beide van de activiteiten *wijziging_order_hoeveelheid* en *wijziging_product_eigenschappen* worden uitgevoerd, moet de activiteit *hercalculeer_prijs* worden gestart. De activiteit *hercalculeer_prijs* wordt slechts eenmaal uitgevoerd nadat één enkele of beide wijzigingen zijn voltooid.

Dit patroon wordt identiek gemodelleerd als de eenvoudige samensmelting. Het verschil zit in de evaluatie van de links. Om te kunnen synchroniseren moeten alle uitgaande links van de vertakkingen worden geëvalueerd. Niet geselecteerde vertakkingen worden

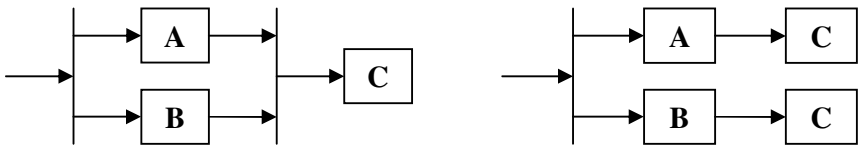
automatisch negatief geëvalueerd (= death path elimination principe). Bij de exclusieve keuze was de enige andere nog te evalueren link die van de actieve vertakking. Na het voltooien van deze activiteit kon onmiddellijk verder worden gegaan. In het geval van meerdere keuzes is het mogelijk dat bij het voltooien van één vertakking er nog andere vertakkingen actief zijn. Links van actieve vertakkingen zijn nog niet geëvalueerd. Daardoor kan er niet verder gegaan worden en moet er dus op alle actieve vertakkingen worden gewacht. Eenmaal alle links positief of negatief geëvalueerd zijn, kan de joinconditie worden gecontroleerd en kan eventueel verder worden gegaan.

3.3.1.8 WP8: Veelvuldige samensmelting

"Bij een veelvuldige samensmelting wordt de activiteit na de samensmelting zoveel keer uitgevoerd als er actieve vertakkingen zijn. Verschillende activiteiten komen dus samen in een punt waarbij geen synchronisatie optreedt."

Een voorbeeld van dergelijke situatie is wanneer twee applicaties parallel worden uitgevoerd (A en B) en beide worden gevolgd door de activiteit '*sluit_applicatie*' (C). Nadat de ene applicatie klaar is met zijn werk, kan de '*sluit_applicatie*' activiteit worden uitgevoerd. Bij het beëindigen van de tweede applicatie moet opnieuw diezelfde activiteit worden aangeroepen.

Dit patroon verschilt van gesynchroniseerde samenstelling in het feit dat de activiteit na samensmelting niet noodzakelijk één keer wordt uitgevoerd, maar zoveel keer als er gekozen vertakkingen zijn. Er moet dus ook niet op andere vertakkingen worden gewacht. BPEL4WS voorziet geen ondersteuning voor deze situatie. Het is namelijk niet mogelijk om verschillende draden over eenzelfde pad te laten lopen zonder de creatie van een nieuwe instantie.



Figuur 45 Gesynchroniseerde en veelvuldige samenstelling

Figuur 45 toont de synchronisatie van de activiteiten A en B. Bij gesynchroniseerde samenstelling wordt activiteit C slechts eenmaal uitgevoerd. Bij veelvuldige samenstelling wordt activiteit C zoveel keer uitgevoerd als er actieve vertakkingen zijn. Daar dit patroon niet in BPEL kan worden gemodelleerd, toont het tweede schema van figuur 45 een eventuele oplossing. De activiteit C wordt hier opgenomen in elke vertakking.

3.3.1.9 WP9: Discriminator

"De discriminator is een punt in het proces dat wacht op de voltooiing van één uit meerdere vertakkingen. Van zodra één vertakking is voltooid, wordt het proces verder gezet en worden de andere vertakkingen als het ware genegeerd. Een discriminator kan pas opnieuw worden gebruikt eenmaal alle vertakkingen zijn uitgevoerd (belangrijk wanneer loops worden gebruikt)."

Dit patroon wordt niet door BPEL4WS ondersteund. Er bestaat geen blok-gestructureerde activiteit noch een manier met links om deze situatie te modelleren. Het probleem met de links is dat een joinconditie pas wordt geëvalueerd, indien de status gekend is van alle links. Het proces moet dus wachten totdat alle vertakkingen zijn uitgevoerd alvorens verder te kunnen gaan. Een discriminator wacht nu echter niet op deze vertakkingen, zodat het niet mogelijk is dit patroon met links te construeren.

Een discriminator kan bijvoorbeeld worden gebruikt om op een snelle manier gegevens op te vragen uit externe gegevensbanken. Er worden verschillende gegevensbanken aangesproken terzelfder tijd. Het resultaat dat eerst wordt verkregen zal het proces verder zetten. Resultaten die later aankomen worden genegeerd.

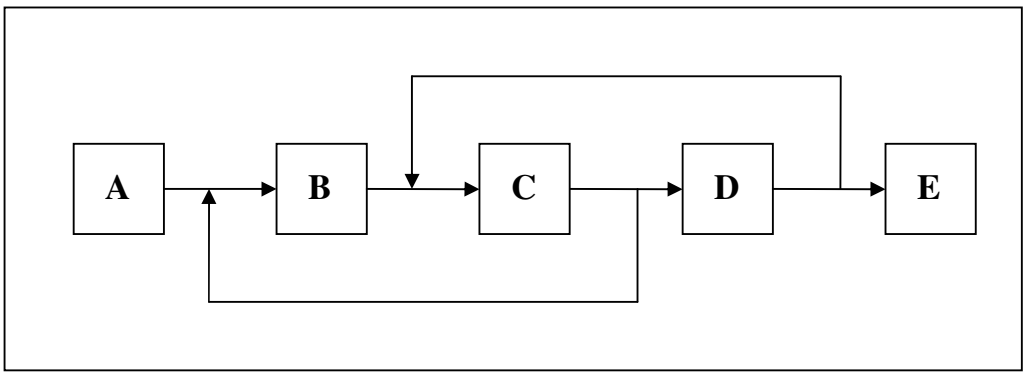
3.3.1.10 WP10: Willekeurige cyclus

"Een willekeurige cyclus is een punt in het proces dat meerdere keren bezocht kan worden zonder beperkingen op te leggen betreffende aantal, plaats en nesting van deze punten."

Dit patroon is niet door BPEL4WS ondersteund. BPEL4WS kan alleen gestructureerde cyclussen modelleren door middel van een `<while>`. Gestructureerde cyclussen bezitten slechts één ingangspunt en één uitgangspunt. Het is dus niet mogelijk om naar

willekeurige delen van het proces te springen. Ook met links kunnen geen willekeurige cyclussen worden geconstrueerd. De links hebben namelijk de beperking dat ze de grens van een lus niet mogen doorbreken en ze mogen ook geen cyclussen vormen.

Figuur 46 toont een voorbeeld van een willekeurige cyclus. Twee lussen worden door elkaar genest. De eerste lus laat de mogelijkheid activiteiten B en C te herhalen. De tweede lus brengt het proces terug naar een positie in de eerste lus. Deze structuur lijkt sterk op de GOTO-constructie in programmeertalen. Men moet dus opletten om op deze manier geen spaghetticode te schrijven.



Figuur 46 Willekeurige cyclus (bron: WorkflowPatterns.com ’03)

Als voorbeeld van een willekeurige cyclus nemen we een e-winkel. Een klant kan een product aan zijn winkelmandje toevoegen (B) en krijgt daarna een tussenresultaat van het totaal te betalen bedrag (C). Deze twee activiteiten worden herhaald zolang de klant producten aan zijn winkelmandje wil toevoegen. Eenmaal de klant beslist om verder te gaan en geen producten meer wil toevoegen, wordt er een overzicht gegeven van de bestelde producten. In dit overzicht is het nu mogelijk een product te wijzigen of te verwijderen (D). Wanneer een aanpassing gebeurt, moet opnieuw het tussentijdse bedrag worden berekend (C) en krijgt de klant weer de kans om een nieuw product toe te voegen. Eenmaal dat er geen wijzigingen meer gemaakt worden, zal het order worden behandeld (E).

3.3.1.11 WP11: Impliciete beëindiging

"Impliciete beëindiging houdt in dat een subproces wordt afgesloten van zodra er niets meer moet gedaan worden. Er is dus geen nood aan een expliciete beëindigingactiviteit."

Een proces eindigt bij één eindpunt. Gestructureerde activiteiten, uitgezonderd de `<flow>` activiteit, eindigen bij het uitvoeren van de laatste activiteit. Men spreekt van expliciete beëindiging. Bij de `<flow>` activiteit kunnen echter meerdere eindpunten worden gemodelleerd. Een subproces kan ten einde lopen, terwijl nog andere subprocessen actief zijn binnen de `<flow>`. Men spreekt hier van impliciete beëindiging. Het subproces eindigt omdat er niets anders meer te doen is. Er is pas sprake van expliciete beëindiging wanneer alle subprocessen in de `<flow>` zijn beëindigd. BPEL ondersteunt dus impliciete beëindiging door middel van de `<flow>`.

3.3.1.12 WP12: Meerdere instanties zonder synchronisatie

"In dit patroon worden meerdere instanties van een activiteit binnen één procesinstantie gecreëerd. Deze instanties kunnen opeenvolgend worden aangemaakt en parallel worden uitgevoerd. De verschillende instanties zijn onafhankelijk van elkaar en moeten in dit patroon niet worden gesynchroniseerd."

Een klant kan bijvoorbeeld meerdere producten bestellen. Voor een marketingstudie worden de voorkeuren van de klanten in kaart gebracht en bij de verwerking van een order wordt voor elk product de activiteit '*verwerk_item*' gestart. Voor elk product zal er een nieuwe instantie van deze activiteit aangemaakt worden. Daar deze instanties niet hoeven te zijn gesynchroniseerd, kan het proces gewoon verder gaan met de verwerking van het order zonder te wachten op de voltooiing van deze instanties.

Deze situatie kan worden gemodelleerd door in een `<while>` lus de `<invoke>` activiteit te gebruiken om een proces B aan te roepen (zie Lijst 7). Het proces B moet dan bij de `<receive>` activiteit het attribuut '`createInstance`' op '`yes`' staan hebben (zie Lijst 8). Bij het doorlopen van de lus worden verschillende instanties van proces B aangemaakt. Al deze instanties zijn zelfstandig en onafhankelijk van elkaar. Na de lus gaat proces A verder en alle instanties van B worden onafhankelijk van elkaar uitgevoerd.

Lijst 7

```
<process A>
   <while cond="C1">
      <invoke process B />
   </while>
</process>
```

Lijst 8

```
<process B>
   <receive process A
        createInstance="yes" />
    . . .
 </process>
```

3.3.1.13 WP13: Meerdere instanties met synchronisatie (aantal gekend 'at design time')

"Binnen één procesinstantie komen meerdere instanties van een activiteit voor. Het aantal instanties dat moet worden gesynchroniseerd is gekend 'at design time'. Eenmaal alle instanties zijn uitgevoerd kan de volgende activiteit beginnen."

Stel dat bij het gebruik van gevaarlijke grondstoffen in het productieproces er toestemming moet zijn van twee verschillende personen. De '`autorisatie`' activiteit zal twee keer moeten worden aangeroepen. Dezelfde activiteit wordt dus tweemaal uitgevoerd. Pas nadat beide instanties van de autorisatieactiviteit zijn voltooid, kan het proces verdergaan. In deze situatie moet de autorisatieactiviteit meerdere keren worden uitgevoerd en is het aantal keer gekend at design time.

Om deze situatie te modelleren wordt de activiteit zoveel keer herhaald als er instanties nodig zijn. Al deze activiteiten kunnen parallel worden uitgevoerd met een `<flow>` structuur. Nadat de `<flow>` is voltooid, kan men verder gaan met de andere activiteiten.

3.3.1.14 WP14: Meerdere instanties met synchronisatie (aantal gekend 'at run time')

"Binnen één procesinstantie komen meerdere instanties van een activiteit voor. Het aantal instanties dat moet worden gesynchroniseerd is veranderlijk en kan afhankelijk zijn van de eigenschappen van het proces of het beschikbaar zijn van bronnen. Het aantal instanties is wel gekend op een gegeven moment in de uitvoering van het proces en vooraleer de instanties worden aangemaakt."

Een voorbeeld van deze situatie kan men vinden in voorraadbeheer. Om een order te kunnen leveren moeten alle items gecontroleerd worden op beschikbaarheid. Eenmaal alle items beschikbaar zijn, kan het order worden geleverd. Het aantal keer dat de voorraadcontrole moet worden uitgevoerd is niet op voorhand gekend, maar is afhankelijk van het aantal verschillende producten in het order. Het aantal items in een

order is gekend op een bepaald moment tijdens de uitvoering van het proces (at run time), namelijk bij het ontvangst van het order.

Lijst 9

```
<process>
   moreInstances:= TRUE
   i:=0

   <while moreInstances OR i>0 >
      <pick>

         <onMessage StartNieuweActiviteitA>
            invoke activiteit A
            i:= i+1
         </onMessage>

         <onMessage EindeActiviteitA>
            i:= i-1
         </onMessage>

         <onMessage GeenInstantiesMeer>
            moreInstances:=FALSE
         </onMessage>

      </pick>
   </while>

</process>
```

BPEL4WS voorziet niet in een structuur die deze logica opvangt. Het is wel mogelijk een oplossing te construeren aan de hand van een `<pick>`, een `<while>`-lus en een teller (zie Lijst 9). De `<pick>` activiteit kan drie berichten opvangen, eentje om een nieuwe instantie te starten, eentje om een bestaande instantie te beëindigen en een bericht dat aangeeft dat er geen instanties meer zullen worden aangemaakt. Bij het creëren van een instantie wordt de teller verhoogd, bij het afsluiten van een instantie wordt de teller verminderd. De lus zal stoppen indien de teller op nul staat en er geen nieuwe instanties meer moeten worden aangemaakt. De boolean variabele '`moreInstances`' zal de waarde '`false`' krijgen van zodra er geen instanties meer moeten worden aangemaakt.

We passen deze oplossing nu toe op ons voorbeeld. Op een gegeven moment in het proces wordt een order ontvangen en is het aantal te controleren items dus gekend. Voor elk item zal er een bericht ('*StartNieuweActiviteitA*') worden gestuurd dat de `<pick>` activiteit zal opvangen. De `<pick>` zal dan een instantie aanmaken en de teller

verhogen. Nadat alle instanties zijn aangemaakt, wordt het bericht '*GeenInstantiesMeer*' opgestuurd. De `<pick>` activiteit zal daardoor de variabele *moreInstances* op '`false`' zetten. Bij het beëindigen van een instantie zal de `<pick>` de teller verlagen. Eenmaal de teller op nul staat zal de `<while>` stoppen en kan het proces verder.

3.3.1.15 WP15: Meerdere instanties met synchronisatie (aantal niet gekend)

"Binnen één procesinstantie komen meerdere instanties van een activiteit voor. Het aantal instanties dat moet worden gesynchroniseerd is veranderlijk en kan afhankelijk zijn van de eigenschappen van het proces. In tegenstelling met vorig patroon is het aantal uit te voeren instanties niet gekend wanneer de eerste instantie wordt uitgevoerd. Tijdens de uitvoering van het proces kunnen er dus steeds nieuwe instanties worden aangemaakt tot op het ogenblik dat er beslist wordt om te synchroniseren."

We kunnen dit patroon illustreren aan de hand van een situatie die zich voordoet in de verzekeringswereld. Bij een verzekeringsclaim moeten verschillende rapporten van ooggetuigen worden behandeld. Bij het verwerken van deze rapporten kunnen er nieuwe getuigen opduiken en zo het aantal instanties van de '*verwerkGetuige*' activiteit veranderen. Op een gegeven moment zal men niet langer wachten op getuigen en gaat de verwerking van het dossier verder.

Net als het vorige patroon wordt dit ook niet direct ondersteund door BPEL4WS-constructies maar kan het worden gemodelleerd aan de hand van dezelfde omweg die we hebben gebruikt in vorig patroon (zie Lijst 9). Het verschil is dat bij het vorig patroon het aantal instanties gekend was alvorens de eerste instantie werd uitgevoerd, in dit patroon is dit niet het geval. Het is dus mogelijk dat er tijdens de uitvoering van het proces voortdurend nieuwe instanties worden bijgemaakt. Eenmaal dat er beslist wordt dat er geen nieuwe instanties aangemaakt moeten worden, kan het proces na synchronisatie verder.

Concreet betekent dit dan de variabele *moreInstances* niet onmiddellijk op '`false`' gezet kan worden. Waar in vorig patroon het bericht *GeenInstantiesMeer* werd gestuurd nadat alle instanties waren aangemaakt, zal dit bericht nu verstuurd worden op een willekeurig ogenblik. Er is namelijk geen kennis over het aantal te maken instanties. Het

is dus goed mogelijk dat nadat alle instanties zijn beëindigd en de teller dus op nul staat, de waarde van *moreInstances* nog steeds '`true`' is. Het proces wacht dan op een bericht om een nieuwe instantie te starten of op het *GeenInstantiesMeer* bericht.

3.3.1.16 WP16: Uitgestelde keuze

"Een punt in het proces waar tussen verschillende vertakkingen kan gekozen worden gebaseerd op informatie die nog niet noodzakelijk aanwezig is op het moment dat het punt wordt bereikt."

Een voorbeeld van de uitgestelde keuze is wanneer er bij het aankomen van een levering twee transportmanieren (A en B) zijn om de goederen naar het juiste departement te brengen. De keuze tussen de twee is afhankelijk van de beschikbaarheid op het moment. De keuze wordt uitgesteld tot er een transportmogelijkheid beschikbaar is. Het proces wacht dus totdat er zich een bepaalde gebeurtenis voordoet waardoor de keuze dan kan gemaakt worden.

Het verschil met de exclusieve keuze is dat de keuze niet onmiddellijk wordt gemaakt, maar uitgesteld is tot er zich een bepaalde gebeurtenis voordoet. BPEL4WS gebruikt de `<pick>` activiteit om deze situatie te modelleren. Wanneer er een vertakking is gekozen worden de andere onbeschikbaar.

3.3.1.17 WP17: Interleaved Parallel routing

"Verschillende activiteiten worden uitgevoerd in willekeurige volgorde. Elke activiteit wordt slechts eenmaal uitgevoerd. De volgorde wordt beslist at run time. Er kunnen echter geen twee activiteiten op hetzelfde ogenblik plaatsvinden."

Door 'mutual exclusion' wordt afgedwongen dat bepaalde parallelle taken niet tegelijk uitgevoerd kunnen worden. Het is mogelijk mutual exclusion te modelleren in BPEL4WS met behulp van het begrip '`serializable scopes`' (zie Lijst 10). Zo'n scope heeft voor het attribuut '`variableAccessSerializable`' de waarde '`yes`', waardoor er voor gezorgd wordt dat er geen gelijktijdige toegang is op de gedeelde variabelen binnen die scope. De serializable scope is gelijkaardig met het standaard isolatieniveau 'serializable' dat wordt gebruikt in databasetransacties. De uit te voeren activiteiten worden elk afzonderlijk in een scope geplaatst die allemaal schrijven naar

één enkele gedeelde variabele (variabele C). Daar de activiteiten in verschillende scopes binnen een `<flow>` zitten, kunnen ze parallel worden uitgevoerd. Maar doordat activiteiten in de serializable scope schrijven naar dezelfde gedeelde variabele, kunnen er geen twee activiteiten op hetzelfde moment worden uitgevoerd. Op die manier worden ze dus na elkaar uitgevoerd.

Lijst 10

```
<flow>

   <scope name=S1 variableAccessSerializable:=”yes”>
      <sequence>
         schrijf naar variabele C
         activiteit A1
         schrijf naar variabele C
      </sequence>
   </scope>

   <scope name=S2 variableAccessSerializable:=”yes”>
      <sequence>
         schrijf naar variabele C
         activiteit A2
         schrijf naar variabele C
      </sequence>
   </scope>

</flow>
```

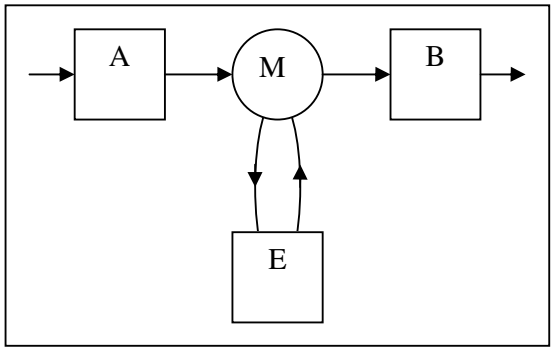
Als voorbeeld voor dit patroon beschouwen we twee controle activiteiten op het einde van een productieproces. Een product wordt onderworpen aan enkele tests die de veiligheid van het product moeten waarborgen. Er worden ook nauwkeurige controles gedaan op de kwaliteit van het product. De kwaliteitscontrole en de veiligheidscontrole kunnen in willekeurige volgorde worden gedaan, maar kunnen natuurlijk niet terzelfder tijd uitgevoerd worden.

Een drietal opmerkingen betreffende deze oplossing worden nu aangehaald. Ten eerste is de semantiek van serializable scopes niet duidelijk gedefinieerd in BPEL4WS. De specificatie spreekt alleen over het gelijkaardige zijn met het isolatieniveau ‘serializable’ van databasetransacties, maar beschrijft niet waar deze overeenkomsten stoppen. Ten tweede is het niet mogelijk om de volgorde waarin de activiteiten worden uitgevoerd at run time te beslissen. De volgorde wordt namelijk bepaald door de transactiemanager. Ten slotte is het niet mogelijk dit patroon te nesten, daar serializable

scopes niet genest mogen worden. BPEL ondersteunt dus dit patroon slecht gedeeltelijk, omdat de volgorde niet at run time kan worden beslist.

3.3.1.18 WP18: Mijlpaal

"Een bepaalde activiteit E kan enkel worden uitgevoerd nadat een bepaalde 'mijlpaal' is bereikt die nog niet verstreken is. Een mijlpaal M is een punt in het proces waar een activiteit A is voltooid en een aansluitende activiteit B nog niet is begonnen (zie figuur 48)."



Figuur 47 Mijlpaal

Een klant die bepaalde aankooporders kan wijzigen of annuleren tot 2 dagen vóór de eigenlijke levering, is een voorbeeld van een situatie waarin een mijlpaal wordt gebruikt. Het *wijzigen van een order* (E) kan pas uitgevoerd worden na het *indienen van een order* (A). Indien de mijlpaal is verstreken (2 dagen voor levering) kan er niets meer worden gewijzigd en wordt de *levering uitgevoerd* (B).

BPEL4WS voorziet geen onmiddellijke ondersteuning voor dit patroon. Het kan wel worden geconstrueerd aan de hand van een `<pick>` activiteit (zie Lijst 11). De variabele '`B_completed`' stelt de mijlpaal voor en zolang die nog niet is bereikt, bezit het de waarde '`false`'. Er is dus een keuze tussen het uitvoeren van activiteit E of B. De while-lus wordt gebruikt om te garanderen dat zolang B nog niet is gekozen, activiteit E een willekeurig aantal keer kan worden uitgevoerd. Van zodra de mijlpaal is verstreken (`B_completed=true`) wordt E onbeschikbaar en kan B worden uitgevoerd.

Lijst 11

```
<process>
   activiteit A
   B_completed:= "False"

   <while B_completed="False">
      <pick>
         <onMessage mE>
            activiteit E
         </onMessage>

         <onMessage mB>
            B_completed:="True"
         </onMessage>
      </pick>
   </while>

   activiteit B
</process>
```

3.3.1.19 WP19: Annulatie van een activiteit

"Het annuleren van een activiteit zal de lopende instantie van die activiteit beëindigen."

Stel dat er op een product twee testen worden gedaan. Er kan echter beslist worden om de tweede test te annuleren wegens tijdsgebrek. Het annuleren van de activiteit kan worden gemodelleerd door het opwerpen van een fout die zal worden behandeld door de faultHandler van de te annuleren activiteit. We zien in figuur 48 twee scopes die elk een test voorstellen. De tweede scope heeft een faulthandler die de fout '*annuleerTest2'* zal opvangen. De fout kan worden opgeworpen door middel van een `<throw>` activiteit. We hebben gezien in hoofdstuk 2 (paragraaf 2.3.8) dat bij het behandelen van de fout alle activiteiten in de scope worden beëindigd. Op deze manier is het dus mogelijk om met behulp van een faulthandler een activiteit te beëindigen.

```
<sequence>
   <scope name="test1">
      // doe test 1
   </scope>

   <scope name="test2">
      <faultHandlers>
         <catch name="annuleerTest2">
            <empty />
         </catch>
      </faultHandlers>
      // doe test 2
   </scope>
</sequence>
```

Figuur 48 Voorbeeld: Activiteit annuleren

3.3.1.20 WP20: Annulatie van een procesinstantie

"Het annuleren van een procesinstantie leidt tot de verwijdering van de volledige workflowinstantie."

De `<terminate>` activiteit wordt gebruikt om de volledige uitvoer van een procesinstantie te stoppen. Alle lopende activiteiten worden onmiddellijk stop gezet zonder enige vorm van foutafhandeling of compensatie.

3.3.2 Communicatiepatronen

In deze paragraaf evalueren we BPEL4WS aan de hand van bepaalde communicatiepatronen. Daar communicatie wordt gerealiseerd door de uitwisseling van berichten tussen verschillende services, gebeurt het modelleren door het verzenden en ontvangen van berichten. Er worden twee grote types van communicatie onderscheiden, namelijk synchrone en asynchrone communicatie. De eerste drie patronen behoren tot de synchrone de laatste drie tot de asynchrone.

3.3.2.1 CP1: Request/Reply

"Request/response communicatie is een vorm van synchrone communicatie waarin de zender een verzoek stuurt naar een ontvanger en op een antwoord wacht alvorens verder te gaan met het proces. Het antwoord kan de verdere verwerking van het proces beïnvloeden."

Lijst 12

```
<process name="processA">
   <sequence>
      . . .
      <invoke partner="processB"
       inputvariable="Request"
       outputvariable="Response"/>
      . . .
   </sequence>
</process>
```

Lijst 13

```
<process name="processB">
   <sequence>
       <receive partner="processA"
        variable="Request" />
       . . .
       <reply partner="processA"
        variable="Response" />
   </sequence>
</process>
```

De synchrone communicatie in BPEL4WS wordt gemodelleerd aan de hand van een `<invoke>` activiteit in het verzoekproces, proces A (zie Lijst 12) en een `<receive>` en

<reply> activiteit in het antwoordende proces, proces B (zie Lijst 13). De <invoke> activiteit heeft twee verschillende variabelen: een inputVariable die de uitgaande gegevens bevat en een outputVariable waarin de inkomende gegevens worden opgeslagen.

3.3.2.2 CP2: One-Way

"De one-way is synchrone communicatie waarbij de zender een verzoek stuurt naar de ontvanger en wacht op een antwoord dat de ontvangst van het verzoek bevestigt."

Dit patroon wordt op identiek dezelfde wijze gemodelleerd als het request/reply patroon. Het verschil is dat proces B onmiddellijk een antwoord terug stuurt van zodra een bericht van proces A wordt ontvangen. Het proces A zal bij ontvangst van de bevestiging verder gaan zonder rekening te houden met de inhoud van het ontvangen bericht.

3.3.2.3 CP3: Synchronous Polling

"Synchronous polling is een vorm van synchrone communicatie waarbij de zender een verzoek stuurt naar de ontvanger, maar in plaats van te blokkeren verdergaat met een deel van het proces. Op door de ontwikkelaar bepaalde tijdsintervallen controleert de zender of er al een antwoord is verstuurd. Wanneer het een antwoord detecteert, wordt dit antwoord verwerkt en stopt het met wachten op een antwoord."

Lijst 14

```
<process name="A">
   <sequence>
      <invoke partner="B" inputVariable="Request" />
      <flow>
         <sequence> . . . </sequence>
         <receive partner="B" variable="Result" />
      </flow>
      gebruik variabele "Result"
   </sequence>
</process>
```

Dit patroon wordt gemodelleerd met behulp van twee parallelle stromen (zie Lijst 14), één voor de sequentie van activiteiten die onafhankelijk zijn van het antwoord en één voor de ontvangst van het antwoord. De communicatie is gestart met een <invoke> activiteit. Omdat het proces na het versturen van een bericht verder moet kunnen

werken, wordt in de `<invoke>` slechts één variabele gedefinieerd, namelijk de `inputVariable`. De `outputVariable` wordt weggelaten. Dit patroon is synchroon omdat een deel van het proces toch nog wordt geblokkeerd en pas verder kan indien het antwoord wordt ontvangen. Waar bij asynchrone interactie het proces als het ware vergeet dat het een bericht heeft gestuurd, wacht het proces hier zoals bij een synchrone interactie op een antwoord.

3.3.2.4 CP4: Message passing

"Message passing is een vorm van asynchrone communicatie waarbij een verzoek door de zender naar de ontvanger wordt verzonden. Wanneer de zender zijn bericht heeft gestuurd, zal de zender als het ware vergeten dat hij iets heeft gestuurd en zal verder gaan met de uitvoering van het proces."

Het modelleren van zo'n situatie is al aangehaald in vorig communicatiepatroon, namelijk een `<invoke>` activiteit met enkel een `inputVariable`.

3.3.2.5 CP5: Publish/Subscribe

"Bij deze vorm van asynchrone communicatie stuurt een zender een bericht naar een ontvanger die zich op voorhand heeft geregistreerd als geïnteresseerd. "

Dit communicatie patroon wordt niet door BPEL4WS ondersteund.

3.3.2.6 CP6: Broadcast

"Broadcast is een vorm van asynchrone communicatie waarbij een zender een bericht stuurt naar alle ontvangers van een netwerk. Elke ontvanger bepaalt of het bericht interessant is door de inhoud te onderzoeken."

Broadcast wordt niet door BPEL4WS ondersteund.

3.3.3 Opmerkingen

We hebben in deze paragraaf de BPEL4WS-specificatie geanalyseerd aan de hand van een raamwerk bestaande uit workflow - en communicatiepatronen. Een samenvatting van de resultaten samen met een vergelijking met XLANG, WSFL, BPML en WSCI is weergegeven in figuur 49 [van der Aalst, Hofstede, Kiepuszewski, Barros '03]. Een '+'

teken in de tabel betekent dat het patroon rechtstreeks wordt ondersteund. Een ‘-‘ teken duidt aan dat er geen rechtstreekse ondersteuning aanwezig is. Dit betekent echter niet dat er geen oplossing voor handen is door bijvoorbeeld andere constructies te combineren. Soms is er een patroon dat slechts gedeeltelijk wordt voorzien en dit wordt aangegeven door een ‘+/-‘ teken.

pattern	*product/standard*				
	BPEL	XLANG	WSFL	BPML	WSCI
Sequence	+	+	+	+	+
Parallel Split	+	+	+	+	+
Synchronization	+	+	+	+	+
Exclusive Choice	+	+	+	+	+
Simple Merge	+	+	+	+	+
Multi Choice	+	–	+	–	–
Synchronizing Merge	+	–	+	–	–
Multi-Merge	–	–	–	+/–	+/–
Discriminator	–	–	–	–	–
Arbitrary Cycles	–	–	–	–	–
Implicit Termination	+	–	+	+	+
MI without Synchronization	+	+	+	+	+
MI with a Priori Design Time Knowledge	+	+	+	+	+
MI with a Priori Runtime Knowledge	–	–	–	–	–
MI without a Priori Runtime Knowledge	–	–	–	–	–
Deferred Choice	+	+	–	+	+
Interleaved Parallel Routing	+/–	–	–	–	–
Milestone	–	–	–	–	–
Cancel Activity	+	+	+	+	+
Cancel Case	+	+	+	+	+
Request/Reply	+	+	+	–	–
One-Way	+	+	+	–	–
Synchronous Polling	+	+	+	–	–
Message Passing	+	+	+	–	–
Publish/Subscribe	–	–	–	–	–
Broadcast	–	–	–	–	–

Figuur 49 Patroon gebaseerde analyse (bron: van der Aalst, Hofstede, Kiepuszewski, Barros ’03)

Uit de tabel blijkt dat de vijf eerste patronen (die met de basisconstructies overeenkomen) in elke taal rechtstreeks worden ondersteund. De patronen van meer geavanceerde constructies zijn niet overal even veel voorzien. We zien ook dat BPEL4WS alle constructies bezit die ondersteund zijn door XLANG en WSFL. Dit komt natuurlijk doordat BPEL4WS een samensmelting is van beide compositietalen en dus zowel de graafgestructureerde als blokgestructureerde methodes toepast. BPEL is in

staat om ‘multi choise’ en ‘synchronizing merge’ te modelleren in tegenstelling tot BPML en WSCI. Dit komt vooral door de ‘death-path’ eliminatie eigenschappen die BPEL heeft overgenomen van WSFL. BPML ondersteunt wel gedeeltelijk ‘Multi-Merge’ terwijl BPEL4WS dit niet doet. De reden hiervoor is dat BPML subprocessen kan aanroepen. De meeste compositietalen ondersteunen ‘deferred choise’ wat hen onderscheid van de gewone workflowtalen. BPEL4WS is de enige van de in de tabel besproken talen die het patroon ‘Interlevead Parallel Routing’ ondersteunt, weliswaar met enkele beperkingen. Dit patroon kan worden gemodelleerd dankzij het concept van ‘serializable’ scopes. Opmerkelijk is dat ook geen enkele taal ‘arbitrary cycles’ ondersteunt.

Voor enkele patronen die niet rechtstreekse door BPEL worden ondersteund, hebben we toch een oplossing kunnen construeren door verschillende activiteiten te combineren. Het ‘milestone’ patroon wordt in de tabel aangeduid als niet rechtstreeks ondersteund, maar we hebben gezien bij de bespreking van het patroon dat er toch een BPEL-oplossing kan gemaakt worden.

BPEL4WS is een expressieve taal in vergelijking met andere talen die aan procesmodellering doen. In bijlage 3 zien we een vergelijkende tabel van verschillende traditionele workflowtalen. De meeste van die talen ondersteunen minder dan de helft van de hierboven genoemde patronen. BPEL4WS is echter niet alomvattend daar sommige patronen moeilijk of niet kunnen worden gemodelleerd. BPEL4WS biedt ook constructies om communicatie te modelleren, in tegenstelling tot traditionele workflowtalen waarin dit niet het geval is.

Het grote uitdrukkingsvermogen van BPEL4WS heeft vooral te maken met de combinatie van blokgestructureerde en graafgestructureerde constructies. Het gebruik van beide benaderingen heeft echter ook zijn nadelen. De taal wordt complex daar er vaak overlappende mogelijkheden zijn om een bepaald patroon te modelleren. Soms wordt de ene benadering gebruikt en soms de andere. Op die manier kunnen twee identieke activiteiten op verschillende manieren worden voorgesteld. De `<sequence>` constructie kan bijvoorbeeld door een `<flow>` en een `<link>` activiteit worden voorgesteld. Er is geen semantisch verschil tussen beide. Men kan zich dus afvragen of

het nodig is om de `<sequence>` te blijven ondersteunen. Deze activiteit is eigenlijk redundant en draagt alleen bij tot de complexiteit van de taal.

Er is na een discussie (issue #23) beslist door het BPEL TC om de `<sequence>` constructie te behouden omwille van terugwerkende compatibiliteit en omdat de sequentie beter door modellering – en analysetools kan worden begrepen.

Een tweede nadeel is dat sommige concepten niet altijd duidelijk zijn omschreven. Dit is vooral het geval bij geavanceerde constructies zoals 'serializable' scopes. Door het gebrek aan een precieze semantiek laat men plaats voor verschillende interpretaties. Er is dus nood aan formalisme.

"There is a need for formalism. It will allow us to not only reason about the current specification and related issues, but also uncover issues that would otherwise go unnoticed. Empirical deduction is not sufficient." – Issue #42, OASIS WSBPEL TC.

Het doel van formalisme is het wegnemen van ambiguïteiten en inconsistenties die meestal verborgen zitten in de omgangstaal. Het moet ook een basis bieden voor het valideren van de belangrijkste taalconstructies en dienen als duidelijke en bondige documentatie van de semantiek van de taal. Op deze manier helpt formalisme bij het proces van ontwerp en standaardisatie.

Het formaliseren van de semantiek van een taal gebaseerd op informele vereisten, is als het ware de omzetting van een spreektaal naar wiskunde. De informele en formele taal zouden elkaar moeten aanvullen om de eisen om te zetten in een specificatie. Het formele model biedt de ultieme referentie aan voor wanneer er verduidelijking met mathematische precisie nodig is die moeilijk kan worden verwoord. Er bestaan al modelleertechnieken die expressief, eenvoudig en formeel zijn. Twee voorbeelden hiervan zijn 'Petri nets' en 'Pi calculus'. Er is dus nood om compositietalen te baseren op formele modellen om duidelijke semantiek en analysemethoden te bieden. [Farahbod, Glässer, Vajihollahi '04].

De WSBPEL TC ziet zeker de voordelen van het formaliseren van BPEL maar is nog niet van plan dit te doen omdat dit momenteel te veel werk zou vragen en de voltooiing

van de specificatie zou vertragen. Ook is een formeel model voor minder mensen toegankelijk en zou dus vooral het informele model worden gebruikt. Daarbij, als er conflicten optreden tussen formeel en informeel model, welk krijgt dan de voorrang? Het onderwerp is dus voorlopig gesloten maar is voor herziening vatbaar [OASIS WSBPEL TC '04].

3.4 Conclusie

De BPEL-doelstellingen zoals beschreven in de eerste paragraaf van dit hoofdstuk zitten grotendeels verwerkt in de BPEL4WS 1.1 specificatie. Enkele puntjes zoals het rekening houden met toekomstige versies van WSDL en het invoeren van de suspend/resume activiteiten worden echter uitgesteld. Er wordt wel al onderzoek gedaan naar de nieuwe versie van WSDL, maar de specificatie zal er zich nog niet op baseren.

Volgens de patroongebaseerde analyse van van der Aalst is BPEL4WS een expressieve taal. BPEL4WS is namelijk in staat om vele patronen te modelleren, weliswaar niet altijd rechtstreeks. De combinatie van de graafgestructureerde en blokgestructureerde methode, zoals beschreven in doelstelling 4, heeft in de eerste plaats gezorgd voor deze expressiviteit. Door de directe en flexibele eigenschappen van de graaf was het mogelijk complexe patronen te modelleren. De gestructureerde activiteiten zorgen op hun beurt voor een eenvoudige en duidelijke weergave van sommige patronen. Aan de andere kant maakt het gebruik van beide methodes de taal complex en niet meer eenduidig. Bepaalde identieke patronen kunnen immers door verschillende methodes worden gemodelleerd zonder dat er enig verschil in betekenis is tussen beide. Er zijn dus eigenlijk redundante constructies in de specificatie opgenomen.

BPEL4WS voorziet in constructies ter ondersteuning van langdurige transacties, namelijk de `<compensatieHandlers>` en `de <scope>` activiteit. Er zouden echter nog meer activiteiten moeten worden toegevoegd om bedrijfstransacties efficiënt te kunnen beheren. Verschillende issues van de WSBPEL Issue List handelen namelijk over het toevoegen van activiteiten voor Business Transaction Management.

We zijn ook tot de vaststelling gekomen dat er nieuwe protocollen in ontwikkeling zijn om BPEL4WS bij te staan. Bijvoorbeeld het Asynchronous Service Access protocol dat

is ontwikkeld om BPEL4WS en andere protocollen aan te vullen met algemene voorzieningen voor asynchrone services.

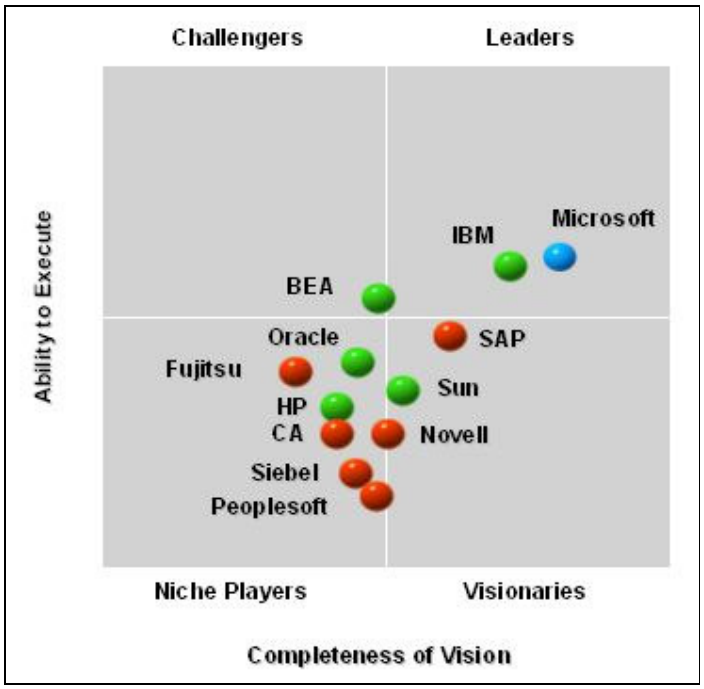
BPEL4WS heeft een groot potentieel om zich tot een goede orchestratietaal te ontwikkelen, maar er moeten hier en daar nog wat aanpassingen gebeuren. Zo worden er nieuwe constructies voorgesteld, wordt de semantiek van bepaalde elementen aangepast en worden sommige aspecten in vraag gesteld. Al deze zaken worden op de voet gevolgd door het OASIS WSBPEL TC. Er moet ook bestudeerd worden in welke mate aanvullende protocollen ontwikkeld en ondersteund kunnen worden.

In het volgende hoofdstuk bespreken we enkele producten die gebruik maken van BPEL4WS. Daarvoor kijken we bij twee grote vendors, namelijk IBM en Microsoft.

Hoofdstuk 4 BPEL4WS vendors en producten

In dit hoofdstuk bespreken we enkele producten die BPEL4WS ondersteunen. We kijken daarvoor bij IBM en Microsoft, die volgens een studie van de Gartner Group tot de leiders behoren bij het aanbieden van Web service-platforms [Andrews, Dias '03]. In figuur 50 zien we dat Web service-platforms van IBM en Microsoft tot de top behoren, zowel op vlak van volledigheid van het platform als op vlak van mogelijkheden tot uitvoeren.

We starten onze bespreking met een niet commercieel product van IBM, vervolgens behandelen we IBM's WebSphere platform en tenslotte staan we stil bij de BizTalk Server van Microsoft. Bij de bespreking maken we onderscheid tussen orchestratie-servers en orchestratie-editors. De servers maken het mogelijk om BPEL-documenten uit te voeren, terwijl de editors dienen om BPEL-documenten aan te maken.



Figuur 50 Vendors van Web service-platforms (bron: Andrews, Dias '03)

4.1 IBM: BPWS4J

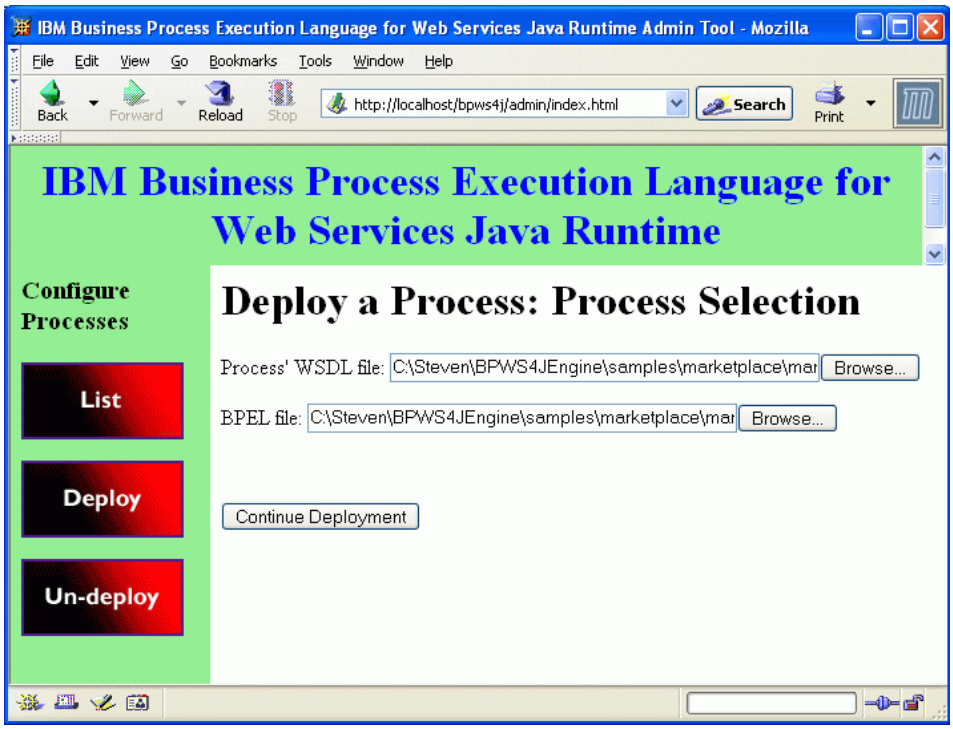
Naar aanleiding van de BPEL4WS 1.0 specificatie, ontwikkelde IBM een BPEL4WS-implementatie, namelijk BPWS4J (Business Process Execution Language for Web Services JavaTM Run Time). BPWS4J werd beschikbaar gesteld op de dag dat de specificatie werd vrijgegeven. Het was dan ook de eerste BPEL-implementatie op de markt [Coverpages '02].

Het BPWS4J-platform bestaat uit twee delen: een engine en een editor. De BPWS4J Engine maakt het mogelijk om bedrijfsprocessen geschreven in BPEL4WS uit te voeren en om die BPEL4WS-documenten te valideren. Wanneer de BPWS4J Engine een proces beschikbaar wil maken, moeten er een aantal documenten worden ingevoerd. Eerst en vooral het BPEL4WS-document dat het uit te voeren proces beschrijft. Daarnaast worden ook de verschillende WSDL-documenten ingegeven, namelijk het WSDL-document dat de interface beschrijft die aan klanten beschikbaar wordt gesteld en de WSDL-documenten die de services beschrijven die door het proces zullen worden aangeroepen. Aan de hand van deze informatie kan het proces worden gepubliceerd en worden uitgevoerd [Alphaworks '02].

Het BPWS4J platform bestaat naast de Engine ook uit een plugin voor de Eclipse editor om BPEL4WS-bestanden te creëren en te wijzigen. Eclipse (eclipse.org) is een open, uitbreidbare IDE (Integrated Development Environment) die door de BPWS4J-plugin een GUI (Graphical User Interface) aanbiedt om BPEL4WS-processen te creëren. Door de BPWS4J-plugin wordt er veel functionaliteit aan de Eclipse editor toegevoegd. Zo kan er tijdens het aanmaken van een proces omgeschakeld worden tussen de XML-broncode en een meer grafisch overzicht in de vorm van een boomstructuur. Er worden eveneens contextgevoelige menu's aangeboden om makkelijker processen te kunnen creëren. Tijdens de procescreatie valideert de plugin voortdurend de syntax tegen de specificatie. De laatste versie van BPWS4J (versie 2.1) biedt ook ondersteuning voor de BPEL4WS 1.1 specificatie [AlphaWorks '02].

We zullen BPWS4J testen aan de hand van het *MarketPlace* voorbeeld dat bij het BPWS4J platform zit. Om BPWS4J te laten werken in een Windows omgeving, moet eerst en vooral een JDK (Java Development Kit) geïnstalleerd zijn en een Servlet 2.2+ en JSP 1.1+ ondersteunende Web applicatie-server. We kiezen hier voor de Apache

Tomcat server. Daarnaast is er ook een XML-parser vereist die JAXP ondersteund, zoals Xerces en moeten ook JavaMail en het JavaBeans Activation Framework worden geïnstalleerd.

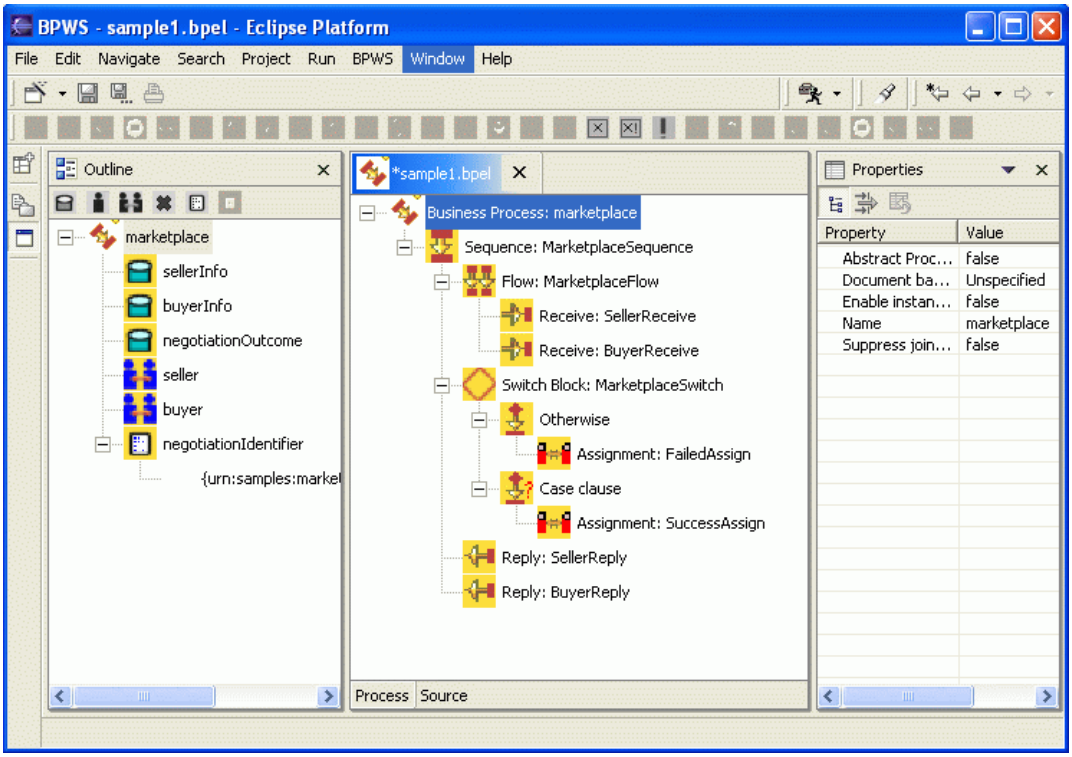


Figuur 51 Beschikbaar maken van een proces

Eenmaal alles is geïnstalleerd, kunnen we de administratiepagina aanroepen van de BPWS4J Engine (zie figuur 51). De administratiepagina biedt de mogelijkheid om processen beschikbaar (deploy) of terug onbeschikbaar (un-deploy) te maken en om de beschikbare processen te bekijken (list). In figuur 51 wordt het *MarketPlace* voorbeeld klaar gemaakt voor gebruik. Daarvoor moet het BPEL en bijhorende WSDL document worden ingegeven. Daarna kan het proces worden aangeroepen.

Om een BPEL-proces te creëren of te wijzigen kan gebruik gemaakt worden van de BPWS4J Editor. Figuur 52 toont een schematische voorstelling van het *MarketPlace* proces dat we zo juist met de engine ter beschikking hebben gesteld. De editor bestaat uit drie vensters en een aantal toolbars. Het middelste venster toont de structuur van het proces. Er kan worden gekozen tusen twee views, de procesview en de bronview. De bronview toont de ruwe XML-code van het proces. In de procesview wordt elke activiteit visueel voorgesteld door een symbool. Het proces is als een boom

gestructureerd, zodat duidelijk is welke activiteiten zijn genest. Processen kunnen in beide views worden aangemaakt en gewijzigd. Aanpassingen in één view worden onmiddellijk doorgevoerd naar de andere view. Beide views zijn ten allen tijde gesynchroniseerd.

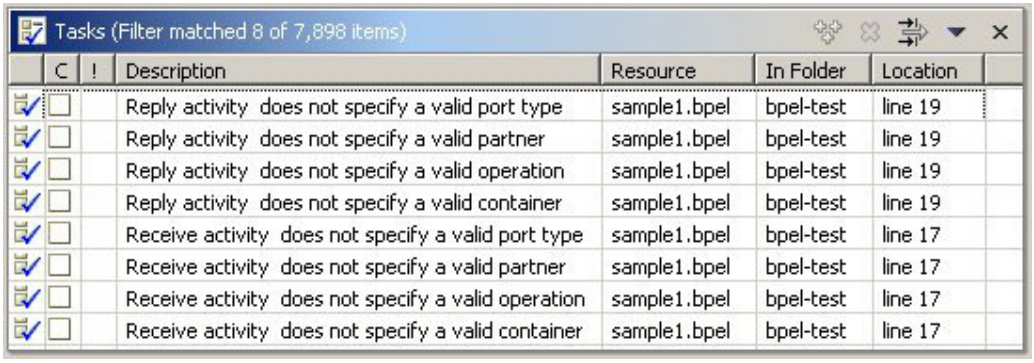


Figuur 52 BPWS4J Editor

Bij het selecteren van een activiteit in de procesview worden in het rechter venster (property view) de eigenschappen van de geselecteerde activiteit weergegeven. In de figuur is het `<process>` element geselecteerd en toont het eigenschappenvenster de verschillende attributen die het `<process>` element kan bezitten. Zo zien we bijvoorbeeld dat de attributen '`Abstract Process`' en '`Suppress Join Failure`' beide '`false`' zijn. In het linkervenster (outline view) worden dan de verschillende variabelen, partnerlinks en correlatiesets gedefinieerd. In de figuur stellen de eerste drie symbolen variabelen voor, vervolgens twee partnerlinks en tenslotte één correlatieset.

De editor controleert tijdens het aanmaken van het proces de geldigheid tegen de BPEL-specificatie. Alle fouten worden in de task view opgesomd en beschreven (zie figuur

53). Daar de editor alleen het BPEL-proces beschouwt, kunnen fouten tegen WSDL-documenten, bijvoorbeeld een verkeerd poorttype, niet worden opgemerkt [Mukhi '02].



C	!	Description	Resource	In Folder	Location
☐		Reply activity does not specify a valid port type	sample1.bpel	bpel-test	line 19
☐		Reply activity does not specify a valid partner	sample1.bpel	bpel-test	line 19
☐		Reply activity does not specify a valid operation	sample1.bpel	bpel-test	line 19
☐		Reply activity does not specify a valid container	sample1.bpel	bpel-test	line 19
☐		Receive activity does not specify a valid port type	sample1.bpel	bpel-test	line 17
☐		Receive activity does not specify a valid partner	sample1.bpel	bpel-test	line 17
☐		Receive activity does not specify a valid operation	sample1.bpel	bpel-test	line 17
☐		Receive activity does not specify a valid container	sample1.bpel	bpel-test	line 17

Figuur 53 BPWSJ Editor: Task view (bron: Mukhi '02)

BPWS4J is ontwikkeld met als doel een experimenteel platform te bieden voor het maken van BPEL-processen. Het is niet geschikt om in een productieomgeving gebruikt te worden wegens verschillende redenen. Ten eerste worden niet alle BPEL4WS-constructies ondersteund. Zo worden bijvoorbeeld niet alle gegevenstypes ondersteund en is er geen ondersteuning voor asynchrone berichtgeving. Sommige constructies zijn ook maar gedeeltelijk geïmplementeerd. Zo kan een `<pick>` activiteit maar één `<onAlarm>` activiteit bezitten en zijn bepaalde standaardfouten niet ondersteund. Tot slot biedt IBM geen support voor het gebruik van BPWS4J [Haataja, Tergujeff '03], [AlphaWorks '02].

BPWS4J is dus eerder gemaakt om initiële ervaring op te doen en eenvoudige projecten uit te werken, maar is niet geschikt om in een commerciële omgeving te werken. Andere orchestratieservers die wel de volledige specificatie implementeren en gericht zijn voor commercieel gebruik, zijn Orchestration Server van Collaxa en ChoreoServer van Openstorm. Deze producten behoren tot de eerste commerciële toepassingen voor BPEL.

Sinds kort (april 2004) bieden nu ook Microsoft en IBM, twee grote vendors in de applicatieservermarkt, BPEL4WS-ondersteuning voor hun platforms. WebSphere van IBM en BizTalk Server van Microsoft worden in de volgende paragrafen besproken.

4.2 IBM: Websphere

IBM's WebSphere platform bestaat uit een verzameling van producten en technologieën voor de ontwikkeling van e-business applicaties. WebSphere richt zich naar drie belangrijke uitdagingen van IT-systemen in de bedrijfswereld. Eerst en vooral moeten IT-systemen en bedrijfsprocessen zich kunnen aanpassen aan de veranderende bedrijfsomstandigheden. Ten tweede moeten nieuwe applicaties gebruik maken van bestaande assets. En tot slot is het belangrijk dat een IT-systeem een stijgende ROI (Return On Investment) genereert [D.H. Brown Associates '04].

WebSphere Business Integration Server Foundation (WBISF) V5.1 en WebSphere Studio Application Developer Integration Edition (WSADIE) V5.1 zijn twee producten die bovenstaande uitdagingen proberen te verwezenlijken. WBISF bouwt verder op de WebSphere Application Server en voegt daar nieuwe functionaliteit aan toe.

De eerste uitdaging om snel op veranderende bedrijfsomstandigheden te kunnen reageren, wordt gerealiseerd door middel van een SOA gebaseerde aanpak. Zoals besproken in hoofdstuk 1, bestaat een Service Oriented Architecture (SOA) uit zichzelf beschrijvende services. Alle bestaande applicaties, systemen en resources kunnen worden voorgesteld als gestandaardiseerde services die men kan hergebruiken en hercombineren om snel in te spelen op veranderende bedrijfsomstandigheden. Deze services zijn immers 'bouwblokken' waarmee flexibele en modulaire applicaties kunnen worden gecreëerd. De Web service-architectuur is een voorbeeld van een SOA. WebSphere Application Server biedt al ondersteuning voor verschillende Web service-standaarden, zoals SOAP en WSDL. WBISF breidt WebSphere Application Server verder uit met de ondersteuning voor Web service-orchestratie/choreografie. Op deze manier kunnen individuele Web services worden gecombineerd tot een nieuw bedrijfsproces [D.H. Brown Associates '04].

Een van de belangrijkste redenen voor het aanpassen van een applicatie is een verandering in de bedrijfslogica. Door een duidelijk onderscheid te maken tussen bedrijfslogica en controle, kan de tijd om applicaties aan te passen aan nieuwe omstandigheden aanzienlijk worden gereduceerd. Business Rules Beans is een extensie voor WBISF die ervoor zorgt dat de business rules kunnen worden gescheiden van de applicatiecode. Business rules zijn uitdrukkingen die het gedrag van bepaalde aspecten

van een business definiëren of beperken. Business rules vertegenwoordigen bedrijfsaspecten die frequent wijzigen door beslissingen van het management, overheid, concurrerende bedrijven,… Een business rule kan bijvoorbeeld een conditie zijn die bepaalt of een klant al dan niet een korting krijgt. Deze conditie kan dan gebruikt worden in een `<switch>` activiteit om de gepaste acties te ondernemen. Indien het totale aankoopbedrag meer dan 100 euro is, krijgt de klant een korting. Door deze conditie op te slaan als een business rule, kan men snel en eenvoudig de rule aanpassen. Stel bijvoorbeeld dat door zware concurrentie beslist wordt om al een korting te geven vanaf 95 euro. Wijzigingen in de rule kunnen nu worden doorgevoerd zonder ingrepen in de applicatie en zonder tussenkomst van de applicatieontwikkelaar [Moghal, Mueller, Boardman, Johnson, Kitabayashi, Dring, Kovari '04].

Integratie van verschillende bedrijfssystemen is een van de belangrijkste trends van de laatste jaren. Ook de integratie met systemen van partners krijgt steeds meer aandacht. Om al deze heterogene systemen te integreren is er nood aan een integratieplatform. WebSphere Business Integration is IBM's integratieplatform dat bestaat uit verschillende engines. WebSphere MQ Workflow ontwikkelt en beheert workflows tussen mensen en systemen. WebSphere InterChange Server dient om bedrijfsprocessen te creëren die bestaande applicaties als ERP en CRM combineren. Tenslotte is er nog de WebSphere Business Integration Message Broker die als middleware dient om berichten tussen heterogene systemen uit te wisselen. WBISF vult nu deze drie engines aan met een bijkomende engine, namelijk de WebSphere Process Choreographer [D.H. Brown Associates '04].

De WebSphere Process Choreographer wordt gebruikt om verschillende bestaande services te combineren tot een bedrijfsproces. De taal die wordt gebruikt om alle bedrijfsprocessen in WBISF te implementeren is BPEL4WS. IBM voorziet 'native support' voor BPEL. Dit wil zeggen dat BPEL niet meer moet worden geconverteerd naar een proprietary formaat. Op deze manier wordt vermeden dat bij conversie informatie verloren gaat. Daar bij conversie er vaak slechts één richting mogelijk is, wordt het moeilijk om processen uit te wisselen. Door de native BPEL support van WBISF is het mogelijk om zowel BPEL te importeren als te exporteren. Op deze manier is men in staat om BPEL-processen te delen tussen verschillende omgevingen en tools [D.H. Brown Associates '04].

Om de programmeur verder te helpen bij het ontwikkelen van efficiënte applicaties, werkt WBISF samen met de WebSphere Studio Application Developer Integration Edition(WSADIE) V5.1. WSADIE voorziet in de tools om J2EE - en Web service-applicaties te creëren, te testen en te beheren. WSADIE bestaat uit editors, wizards, templates en codegenerators om een snelle en efficiënte ontwikkeling te ondersteunen.

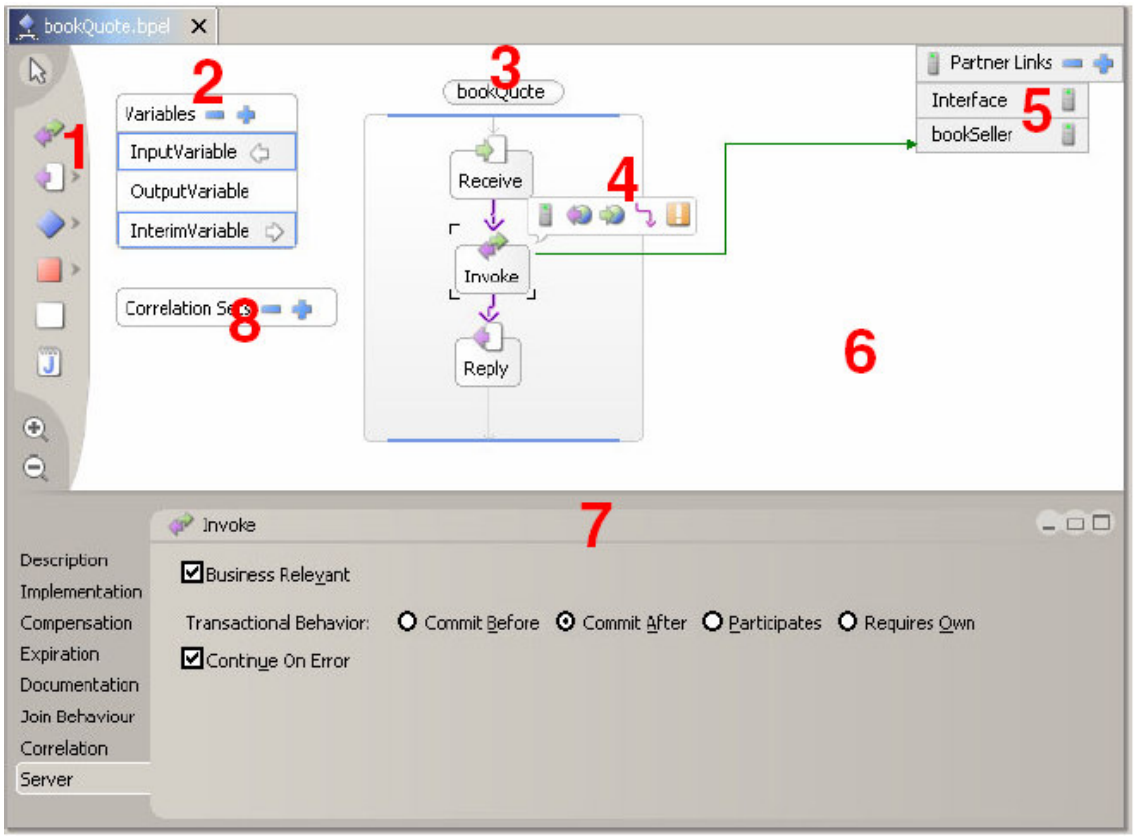
Bij de release van WBISF en WSADIE werd BPEL de standaard procestaal. Vroeger gebruikte men immers Flow Definition Markup Language (FDML), een subset van WSFL. FDML-processen kunnen nog steeds worden ontwikkeld of uitgevoerd, maar het gebruik van FDML wordt afgeraden. WSADIE biedt de mogelijkheid om bestaande FDML-processen om te zetten naar BPEL. WSADIE heeft ook een aantal uitbreiding op de BPEL 1.1 specificatie, namelijk 'human workflow' activiteiten en het opnemen van Java code in het proces [Moghal, Mueller, Boardman, Johnson, Kitabayashi, Dring, Kovari '04].

Het opnemen van Java code in het proces gebeurt aan de hand van Java Snippets. BPEL beschrijft de controlestroom van een bedrijfsproces. De eigenlijke bedrijfsfuncties worden dan door middel van Web services aangeroepen. Java Snippets maken het nu echter mogelijk om bedrijfsfunctionaliteit op te nemen in het proces. Java Snippets worden vooral gebruikt om variabelen te wijzigen in complexe datatypes die niet met een `<assign>` activiteit kunnen worden gedaan en om beperkte functionaliteit aan het proces toe te voegen. Elke Java Snippet bestaat uit een methode van een Java class. De Java code kan de waardes van BPEL-variabelen gebruiken en kan de waardes van BPEL-variabelen wijzigen. Java Snippets zijn geen onderdeel van de BPEL specificatie, het zijn activiteiten die door de WebSphere Process Choreographer worden toegevoegd. De draft van de WBISF handleiding raadt het gebruik van Java Snippets af, omdat Java Snippets de duidelijkheid van het proces aantasten [Moghal, Mueller, Boardman, Johnson, Kitabayashi, Dring, Kovari '04].

De 'human workflow' activiteiten zijn menselijke interacties in een automatisch bedrijfsproces. Een proces dat bijvoorbeeld automatisch schadeclaims behandelt, moet op een gegeven moment worden onderbroken om het risico op fraude door een expert te laten onderzoeken. Eenmaal de expert zijn goedkeuring geeft, kan het proces verder worden uitgevoerd. WebSphere gebruikt de `<staff>` activiteit om menselijke

interacties te modelleren. We kunnen de `<staff>` activiteit vergelijken met de `<pick>` activiteit. De `<pick>` onderbreekt het proces en wacht op het voorkomen van een bepaalde gebeurtenis zoals een inkomend bericht. De `<staff>` activiteit onderbreekt ook het proces en wacht totdat een persoon aangeeft dat het proces verder kan worden uitgevoerd [Moghal, Mueller, Boardman, Johnson, Kitabayashi, Dring, Kovari '04].

Eén van de WSADIE editors is de BPEL Editor. Deze editor maakt het mogelijk om op een visuele manier BPEL-processen aan te maken en te bewerken. Figuur 54 toont de verschillende delen van de BPEL Editor.

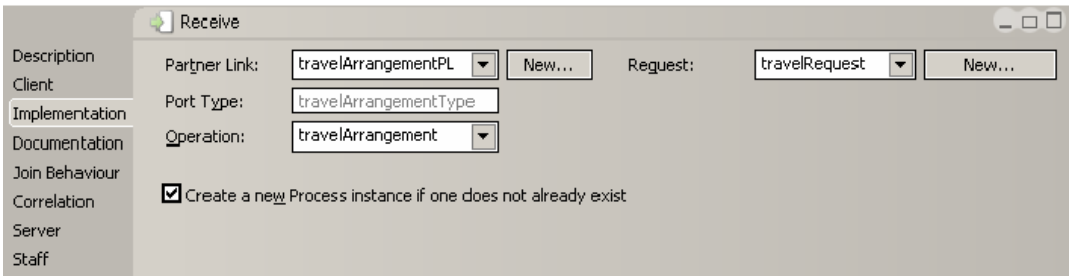


Figuur 54 WebSphere BPEL editor (bron: Moghal, Mueller, Boardman, Johnson, Kitabayashi, Dring, Kovari '04)

De werkruimte (6) wordt opgedeeld in verschillende gebieden. Een proces wordt aangemaakt door activiteiten van de toolbar (1) te verslepen naar het procesgebied (3). In het procesgebied wordt de controlestroom visueel weergegeven. Elke activiteit heeft zijn eigen symbool. Definities van variabelen (2), partnerlinks (5) en correlatiesets (8)

worden in aparte gebieden van het werkblad geplaatst. Bij het selecteren van een activiteit verschijnt er een actiebar (4) met contextgevoelige acties. Zo is het bijvoorbeeld mogelijk om een faulthandler toe te voegen aan een activiteit. Het detailgebied (7) onder het werkblad maakt het mogelijk om het geselecteerde element verder te gaan configureren [Moghal, Mueller, Boardman, Johnson, Kitabayashi, Dring, Kovari '04].

Figuur 55 toont het detailgebied van een `<receive>` activiteit. Hier kan de partnerlink worden gedefinieerd samen met het poorttype en de operatie. Het bericht dat wordt ontvangen wordt opgeslagen in de variabele gedefinieerd in het `request` veld.



Figuur 55 WSADIE: Receive activiteit editor (bron: Moghal, Mueller, Boardman, Johnson, Kitabayashi, Dring, Kovari '04)

De WSADIE heeft ook een testomgeving om processen uit te voeren en te debuggen. De proces debugger biedt een grafische omgeving om breakpoints in het proces te plaatsen. De waardes van variabelen kunnen ook tijdens uitvoering worden opgevraagd en zelfs worden gewijzigd. Verder biedt de debugger de mogelijkheid om het proces stap voor stap uit te voeren [D.H. Brown Associates '04].

WSADIE speelt een grote rol in de ontwikkelingsfase van een applicatie. Alvorens echter aan de ontwikkelingsfase te beginnen, worden twee andere fases van de applicatielevenscyclus doorlopen, namelijk het definiëren van de procesvereisten en het creëren van een visueel model. Deze taken worden niet door de programmeur gedaan, maar door een analist. Om aan de behoeftes van de analist te voldoen wordt een tool beschikbaar gesteld die ook tot de WebSphere familie behoort, namelijk WebSphere Business Integration Modeller. Dit product stelt de bedrijfsanalist in staat om de procesvereisten te definiëren en te documenteren, en om een visueel model aan te

maken van de reeds bestaande processen en van de nieuwe processen gebaseerd op de procesvereisten. Het visuele model kan dan worden geëxporteerd naar BPEL om door de programmeurs te worden gebruikt in WSADIE [D.H. Brown Associates ‘04].

WebSphere Business Integration Monitor (WBIM) is een ander product uit de WebSphere familie ter ondersteuning van WBISF. WBIM is een product voor de business analist om toezicht te houden op de werking en performantie van bedrijfsprocessen. Door de competitieve bedrijfsomgeving wordt het optimaliseren van processen heel belangrijk. Met een grotere zichtbaarheid van de bedrijfsprocessen kunnen potentiële problemen makkelijker worden geïdentificeerd en worden aangepakt. WBIM biedt een realtime inzicht op de performantie van de onderneming door middel van twee tools, het business dashbord en het workflow dashbord [WebSphere Software ‘03].

Het business dashbord levert een high-level view van de performantie van bedrijfsprocessen om zo geïnformeerde management beslissingen te kunnen nemen. De tool maakt het mogelijk om de actuele performantie te vergelijken met de vooropgestelde bedrijfsdoelstellingen. Daarbij is het ook mogelijk om kostanalyses, trendanalyses en tal van andere statistische analyses te maken aan de hand van historische gegevens. Het business dashbord is tevens in staat om te simuleren hoe aanpassingen aan een proces de performantie kunnen verbeteren. Alle informatie geleverd door het dashbord, wordt dan gebruikt om bedrijfsprocessen te optimaliseren [WebSphere Software ‘03].

Het workflow dashboard heeft het operationeel overzicht van de bedrijfsprocessen. Het houdt toezicht op de berichtgeving en audit trail van het proces. Het workflow dashboard controleert of deadlines zullen worden behaald en is in staat om corrigerende acties uit te voeren. Figuur 56 toont hoe het workflow dashboard gegevens over de verschillende procesinstanties weergeeft, zoals de status van een procesinstantie en of de instantie al dan niet vertraagd is [WebSphere Software ‘03].

Item	Activity Instance	Process Diagram	Status	Starting Time	Working Duration	Elapsed Duration	Cost	Is Delayed
1			Ready	Jan 18, 1970 9:40:28 AM	---	---	$0	
2			Running	Sep 5, 2003 11:28:05 PM	---	2 d, 30 m, 51 s	$0	
3			Ready	Jan 18, 1970 9:40:28 AM	---	---	$0	
4			Running	Sep 5, 2003 11:28:08 PM	---	2 d, 30 m, 48 s	$0	
5			Ready	Jan 18, 1970 9:40:28 AM	---	---	$0	
6			Ready	Jan 18, 1970 9:40:28 AM	---	---	$0	
7			Running	Sep 5, 2003 11:28:12 PM	---	2 d, 30 m, 44 s	$0	
8			Running	Sep 5, 2003 11:28:14 PM	---	2 d, 30 m, 42 s	$0	

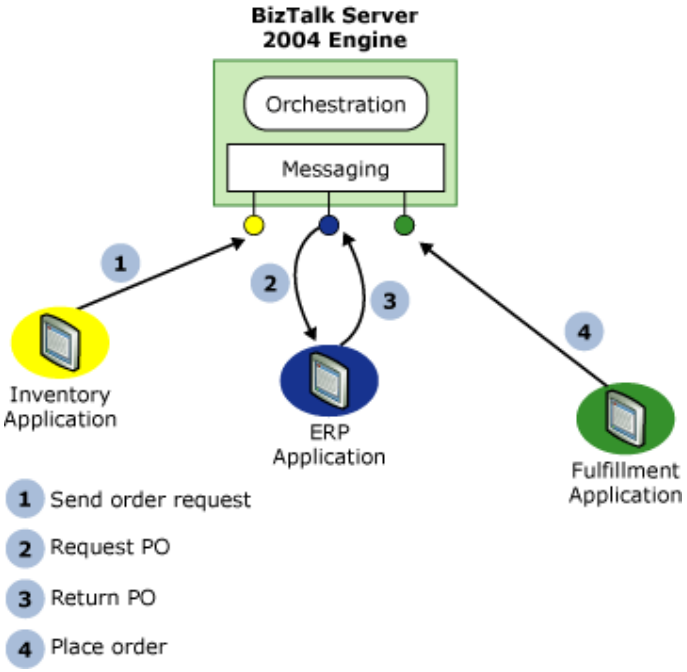
Figuur 56 WebSphere: workflow dashboard (bron: WebSphere Software '03)

Met WebsSphere Studio Application Developper Integration Editor, WebSphere Business Integration Modeller en WebSphere Business Integration Monitor heeft IBM verschillende fases van de levenscyclus van een bedrijfsproces geïntegreerd in één gemeenschappelijk platform. Deze integratie zorgt voor een flexibele en snelle implementatie van nieuwe servicegeoriënteerde oplossingen.

4.3 Microsoft : BizTalk

BizTalk® Server 2004 is de integratie-server van Microsoft die instaat voor het creëren, beschikbaar maken en beheren van bedrijfsprocessen en van op XML gebaseerde webapplicaties. Net als bij IBM staat de SOA-georiënteerde aanpak centraal. BizTalk Server maakt het immers mogelijk om bestaande services te orchestreren tot coherente bedrijfsprocessen.

De kern van BizTalk Server 2004 bestaat uit de engine die twee functies vervult, namelijk orchestratie en berichtgeving. Bij orchestratie moet de engine zorgen voor het correct uitvoeren van de verschillende processtappen, voor het toepassen van de bedrijfslogica en voor het aanroepen van de juiste services. Bij de berichtgeving is de engine verantwoordelijk voor het omzetten en versturen van berichten tussen de verschillende services die deelnemen aan het proces. Figuur 57 toont een eenvoudige voorstelling van de engine in een EAI situatie. De engine verzorgt de berichtgeving tussen de verschillende applicaties en orchestreert alle interacties tot een bedrijfsproces [Microsoft '04].



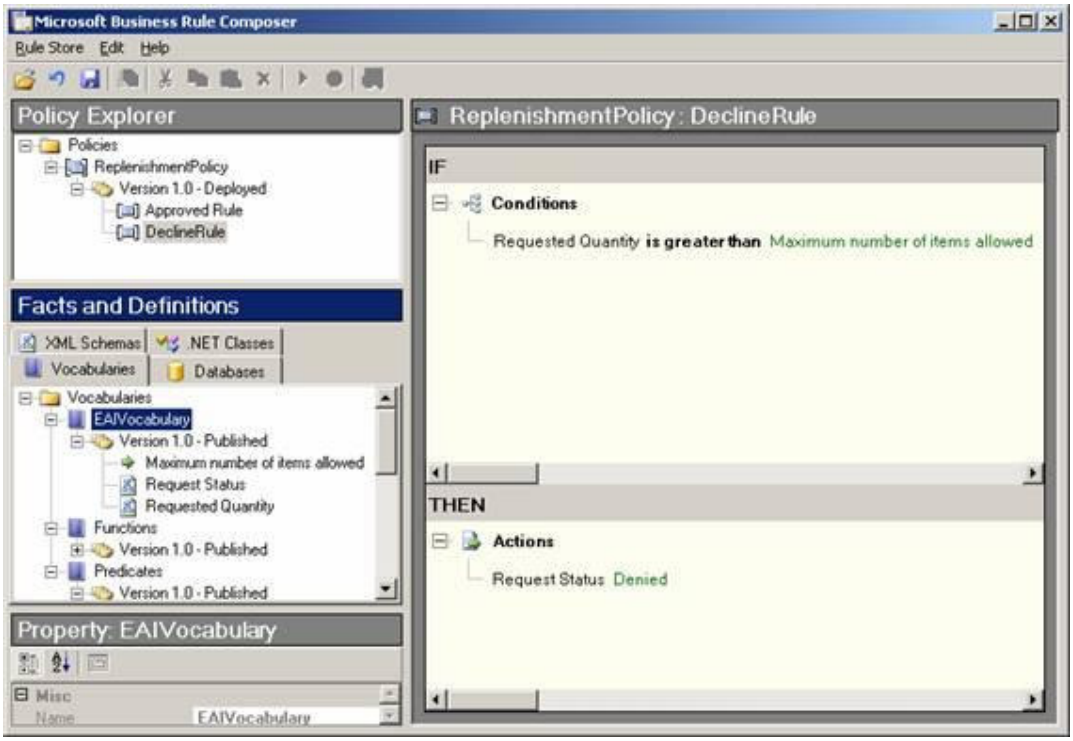
Figuur 57 BizTalk Server Engine (bron: Microsoft '04)

Naast de engine biedt BizTalk ook een mechanisme om business rules te creëren en te beheren. Verder worden er nog bijkomende services toegevoegd als Business Activity Monitoring en Human Workflow Services. We zullen deze drie aanvullingen op de engine hieronder kort bespreken.

Aanpassingen in bedrijfsprocessen zijn vaak het gevolg van wijzigingen in business rules, in tegenstelling tot technologisch gerelateerde veranderingen. Wanneer deze business rules nu ingebed zitten in de programmacode, wordt het moeilijk deze aan te passen. Daarom introduceert Microsoft met de release van BizTalk Server 2004 de Business Rule Composer. Met deze tool is het nu eenvoudig om als analist wijzigingen door te voeren in de business rules.

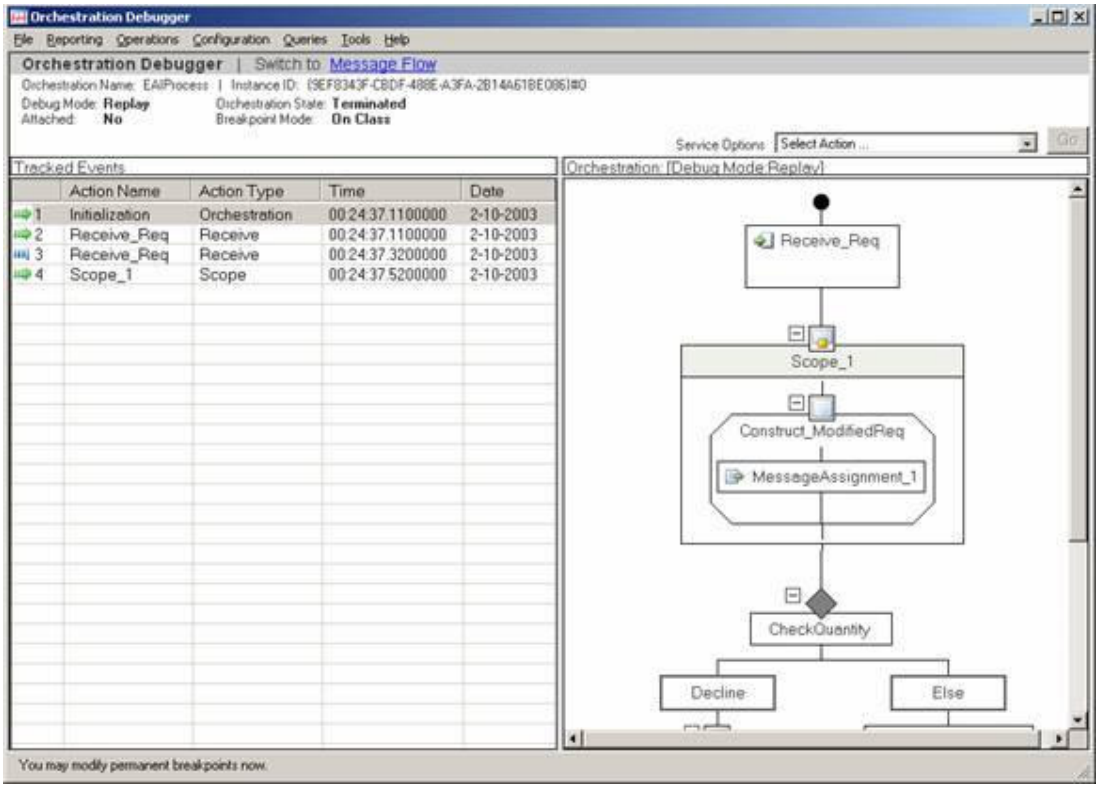
Figuur 58 toont hoe de Business Rule Composer kan worden gebruikt om business rules te creëren. In het geselecteerde venster 'Facts en Definitions' wordt er een woordenschat aangelegd (vocabulary). Elke term in de woordenschat is een gebruiksvriendelijke naam voor een stuk informatie, bijvoorbeeld '*Requested Quantity*'. Elk van deze termen kan een waarde worden toegewezen of kan worden verbonden met een element of attribuut uit een XML Schema. Naast deze termen kunnen ook functies en logische operatoren ('*is greater than*') worden gedefinieerd. Met behulp van deze

woordenschat kunnen we nu business rules aanmaken. Een business rule bestaat uit twee delen, één of meerder condities en één of meerdere acties. Zowel de condities als de acties zijn samengesteld met termen uit de vocabulary. Business Rules kunnen verder worden gegroepeerd in 'policies' [Rijsdijk '04].



Figuur 58 Microsoft Business Rule Composer (bron: Rijsdijk '04)

Business Activity Monitoring geeft de mogelijkheid aan analisten om met behulp van gekende applicaties zoals Microsoft Excel, realtime informatie over lopende processen te bekijken. De Health and Activity Tracking (HAT) tool is in staat om op grafische manier informatie weer te geven over verschillende lopende processen. Daarbij wordt bijgehouden wanneer een proces start en eindigt, welke stappen binnen het proces al zijn uitgevoerd, welke berichten reeds zijn uitgewisseld en op welk tijdstip. Het is ook mogelijk om breakpoints te gebruiken om het proces op welbepaalde plaatsen te onderbreken en te onderzoeken. De HAT tool kan ook gebruikt worden om patronen en trends te zoeken in historische gegevens. Figuur 59 toont de HAT tool met een grafische voorstelling van het te volgen proces en een lijst van de reeds uitgevoerde acties en hun tijdstippen [Rijsdijk '04], [Microsoft '04].



Figuur 59 Health and Activity Tracking (bron: Rijsdijk '04)

Human Workflow Services worden gebruikt om menselijke interacties in het proces op te nemen. De Human Workflow Services infrastructuur laat toe interacties met client applicaties op te nemen in het bedrijfsproces. Enkele belangrijke client applicaties zijn Word, Outlook en InfoPath. Op een gegeven moment in de uitvoering van een proces kan bijvoorbeeld input nodig zijn van een gebruiker. De gebruiker kan dan via een formulier in InfoPath de gewenste gegevens ter beschikking stellen aan het proces [Microsoft '04].

De BizTalk Engine, de Business Rule Composer, de Business Activity Monitoring en Human Workflow Services zijn allemaal onderdelen van de BizTalk Server 2004 en verschillen weinig van IBM's WBISF. Er is echter één groot verschil tussen beide producten. Waar WebSphere native BPEL support biedt, is dit bij BizTalk niet het geval. BizTalk's BPEL ondersteuning bestaat alleen uit import en export van BPEL. Wanneer een BPEL document wordt geïmporteerd zal het worden omgezet in de XLANG/s taal. XLANG/s is een scriptaal en een uitbreiding op de XLANG taal. Door de conversie van BPEL naar XLANG/s moet er bij het importeren rekening gehouden

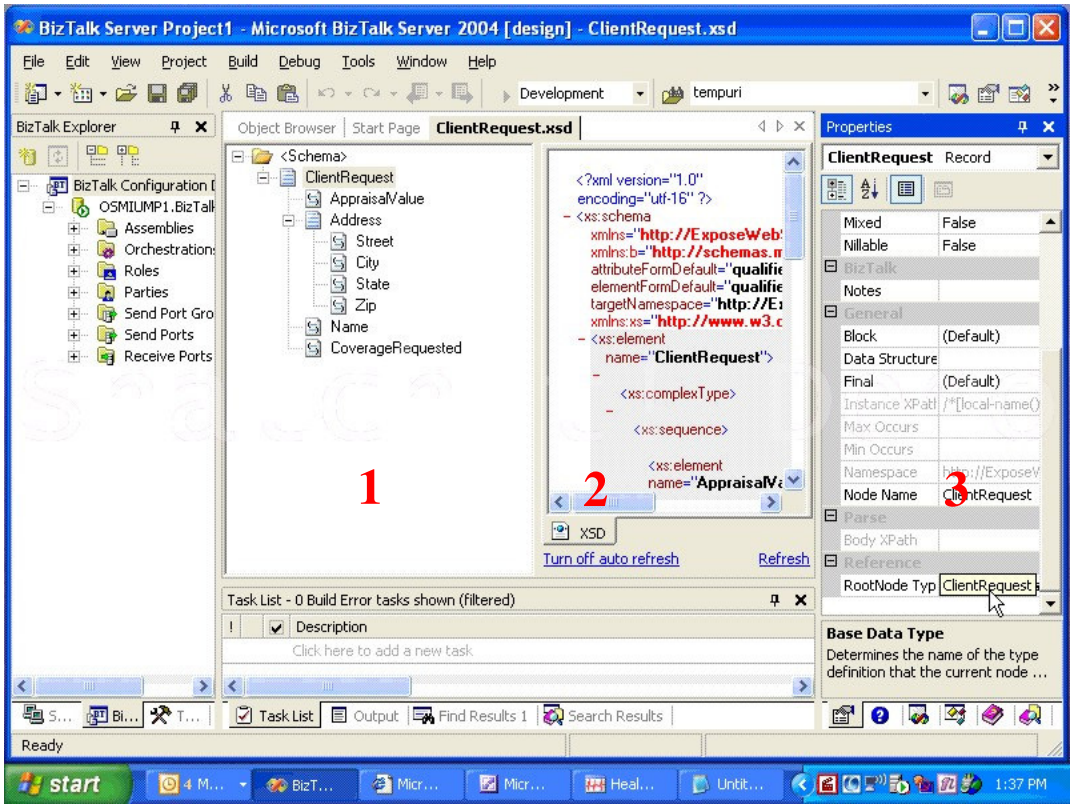
worden met een aantal beperkingen. Zo mag er bijvoorbeeld in het BPEL-document geen gebruik gemaakt worden van gereserveerde XLANG/s woorden. Voor een volledige lijst van de import beperkingen zie [Microsoft BizTalk Server '04 a]. Ook bij het exporteren kunnen er problemen optreden. Sommige componenten zullen immers niet naar BPEL kunnen worden geëxporteerd zoals bijvoorbeeld de 'suspend' activiteit. Voor een volledige lijst van de export beperkingen zie [Microsoft BizTalk Server '04 b].

Net als IBM's WBISF biedt BizTalk de mogelijkheid om op een visuele manier processen te creëren en aan te passen. De Orchestration Designer van de BizTalk Server is een visuele ontwikkelingstool om bedrijfsprocessen te bouwen en is volledig geïntegreerd in Visual Studio. Visual Studio .NET is Microsoft's ontwikkelings-omgeving. De Orchestration Designer is de tool van de ontwikkelaar, daarnaast wordt er ook nog een subset van de Orchestration Designer aangeboden voor de analist, namelijk Orchestration Designer for Business Analysts (ODBA) [Microsoft '04].

Bij het creëren van processen gebruikt de ontwikkelaar drie verschillende tools. De BizTalk Editor om XML Schema's te maken, BizTalk Mapper om de vertalingen tussen verschillende Schema's te definiëren en de Orchestration Designer om de controlestroom van het bedrijfsproces te specificeren. Aan de hand van een toepassing [Woodgate '04] zullen we deze drie tools bespreken.

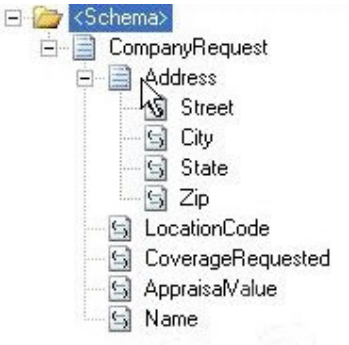
De toepassing is een eenvoudig proces dat bestaat uit het ontvangen van een verzoek van de client met een aanvraag om een bepaald voorwerp te verzekeren (bijvoorbeeld een schilderij). Het verzoekbericht (*CReq*) bestaat uit een aantal gegevens waaronder contactgegevens, de waarde van het voorwerp en de gewenste dekking. Het proces stuurt dan na de verwerking een antwoordbericht (*CResp*) terug naar de client [Woodgate '04].

De eerste stap van de ontwikkelaar is het aanmaken van XML Schema's. De gegevensuitwisseling in BizTalk gebeurt via XML-documenten en elk document moet voldoen aan een XML Schema. Met behulp van de BizTalk Editor kunnen er schema's worden aangemaakt in de XML Schema definitie taal (XSD). Figuur 60 toont het *ClientRequest* schema in de BizTalk Editor.



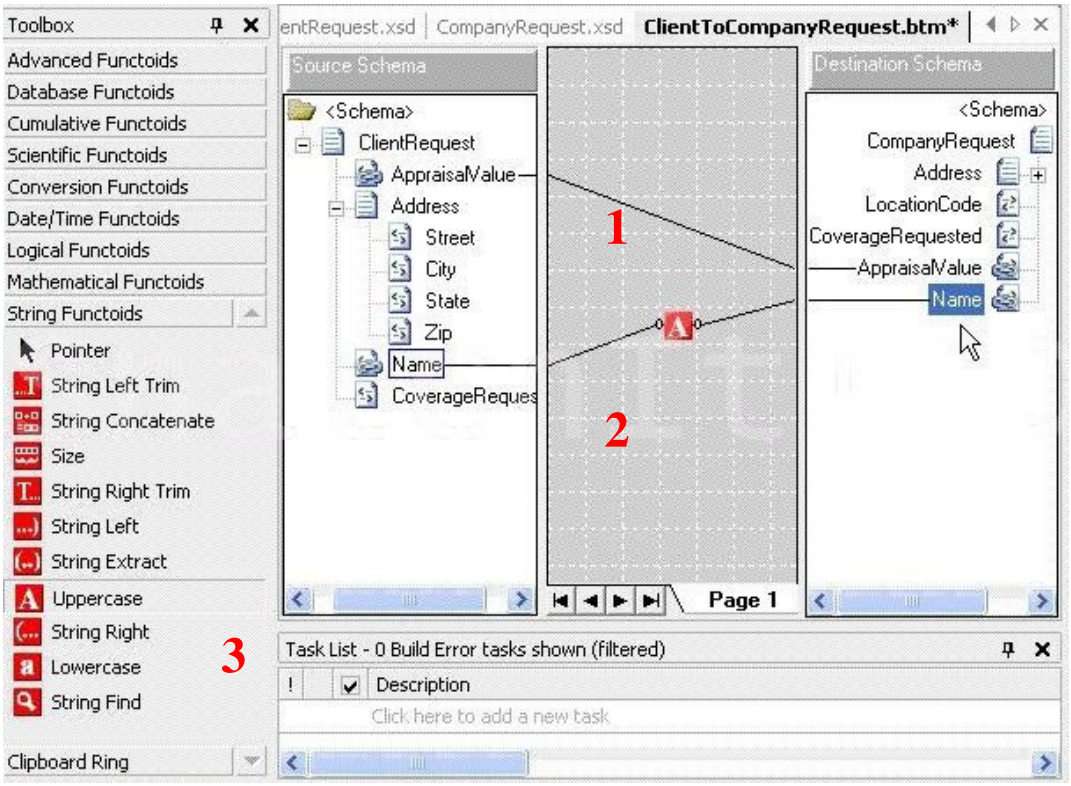
Figuur 60 BizTalk Editor: ClientRequest (bron: Woodgate '04)

Bij het maken van de schema's zijn er twee views voor handen. De boomgestructureerde view (1) heeft een duidelijk overzicht van de verschillende elementen in het document. De bronview heeft de ruwe XML weer van het XSD document (2). Wanneer een element in de boom wordt geselecteerd kunnen de eigenschappen worden gewijzigd door middel van het property venster (3). Alle wijzigingen worden ook onmiddellijk doorgevoerd in de bronview. Op dezelfde manier kan de *CompanyRequest* worden aangemaakt (zie figuur 61).



Figuur 61 XML Schema: CompanyRequest (bron: Woodgate '04)

Een proces bestaat uit het ontvangen en versturen van gegevens. Velden van het ontvangen bericht kunnen rechtstreeks worden gekopieerd naar het te versturen bericht, of kunnen eerst worden bewerkt. De vertaling van een bronspecificatie naar een doelspecificatie, wordt mapping genoemd. Mappings worden aangemaakt met BizTalk Mapper. Het behandelen van de verschillen tussen ons *ClientRequest* en *CompanyRequest* wordt in figuur 62 voorgesteld.

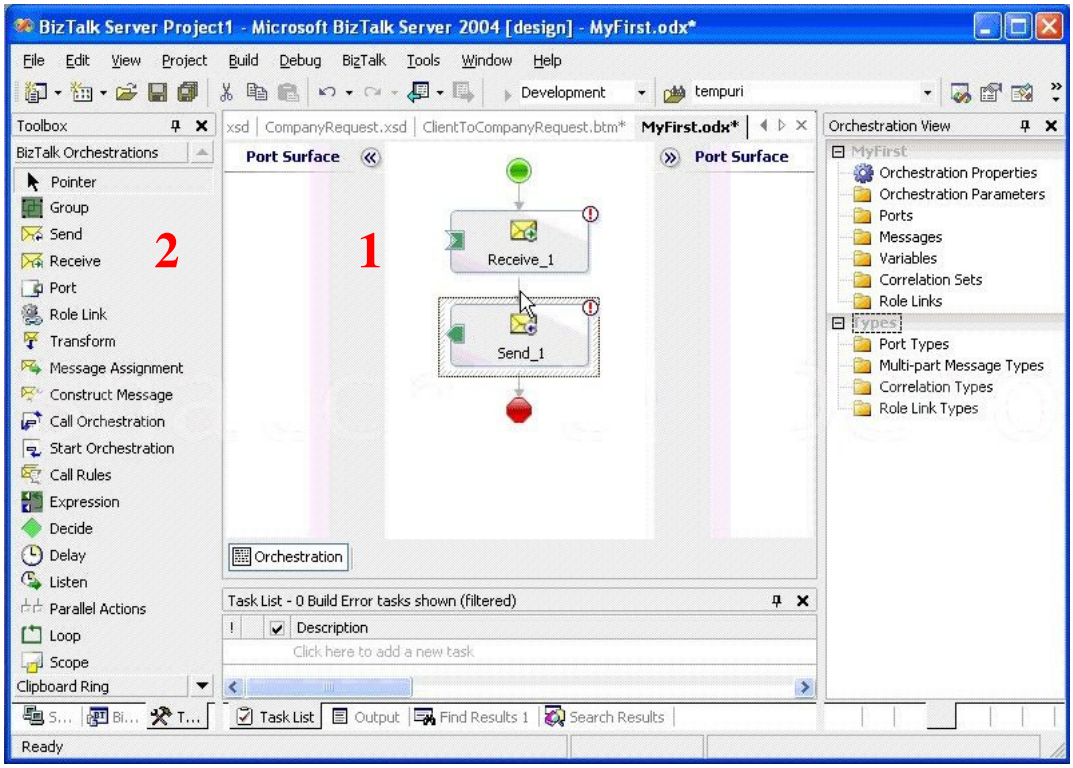


Figuur 62 BizTalk Mapper (bron: Woodgate '04)

Beide schema's kunnen worden ingeladen in de BizTal Mapper en elk bronveld kan dan met een doelveld worden verbonden. Zo zien we dat het *AppraisalValue* veld rechtstreeks wordt gekopieerd naar het doelveld (1). Bij het transformeren van het *Name* veld (2) wordt er gebruik gemaakt van een ingebouwde functie (3) om de waarde van het veld in hoofdletters te plaatsen. Dit is misschien nodig omdat de company service met een legacy systeem is verbonden dat dit veld in hoofdletters verwacht. Het W3C heeft XSLT (Extensible StyleSheet Language Transformation) uitgebracht als de standaard om transformaties tussen schema's te beschrijven. BizTalk Mappings zijn dan ook in XSLT opgeslagen in btm-bestanden [Microsoft '04].

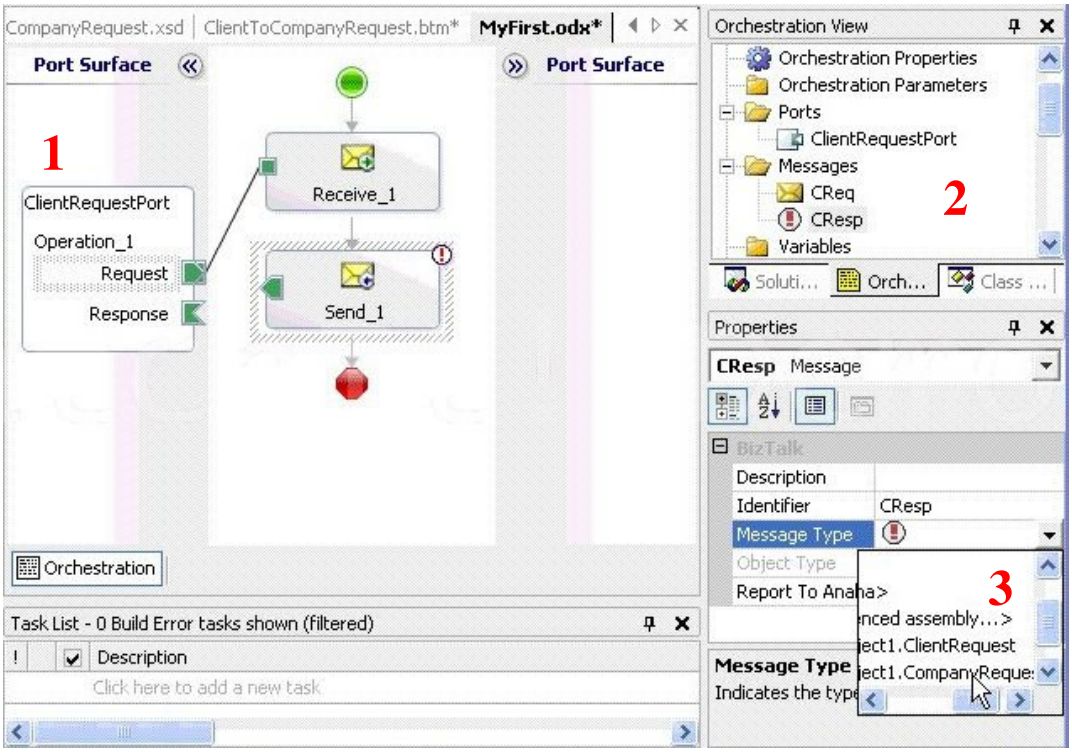
Nadat alle schema's en mappings zijn gemaakt kan er worden gestart met het bouwen van het proces door middel van de Orchestration Designer. Het maken van orchestraties met BizTalk is, door de niet rechtstreekse ondersteuning van BPEL, erg verschillend van WebSphere. In WebSphere worden processen gemaakt door het toepassen van de specificatie. Elk concept uit de specificatie heeft in WebSphere een visuele constructie. BizTalk gebruikt nu echter zijn eigen constructies en benamingen voor de verschillende aspecten van een proces. Zo is bijvoorbeeld een BPEL `<switch>` in BizTalk een decide-constructie, wordt er niet gesproken over partnerlinks en wordt er een onderscheid gemaakt tussen berichten en andere variabelen. Dit alles wordt verder verduidelijkt door het proces te maken dat onze verzekeringsaanvraag behandeld.

In figuur 63 wordt de Orchestration Designer voorgesteld. In het werkgebied (1) zijn er twee activiteiten weergegeven, namelijk een 'receive' activiteit die het verzoek van de client zal ontvangen en een 'send' activiteit die een antwoord zal terug sturen. Alle activiteiten kunnen vanaf de toolbar (2) naar het werkblad worden versleept.



Figuur 63 BizTalk Orchestration Designer: receive & send (bron: Woodgate '04)

BizTalk heeft geen partnerlinks om de berichtuitwisseling tussen verschillende partners te definiëren, maar gebruikt daarvoor de port-constructie. Zoals we hebben besproken in paragraaf 2.3.1 definieert een partnerlink twee rollen bij asynchrone communicatie en één rol bij synchrone communicatie. Elke rol bestaat uit een poorttype dat aangeeft welke operatie kan worden aangeroepen en welk bericht kan worden verzonden. Bij het maken van een BizTalk port moet er ook gedefinieerd worden of de communicatie synchroon of asynchroon is. In ons voorbeeld (zie figuur 64) hebben we een asynchrone communicatie met de client. We zien dan ook op de figuur twee operaties (1), namelijk *request* en *response*, die we kunnen vergelijken met de operaties van de rollen in een BPEL partnerlink. De client zal de *request* operatie aanroepen van het proces en het proces zal antwoorden door de *response* operatie aan te roepen van de client.

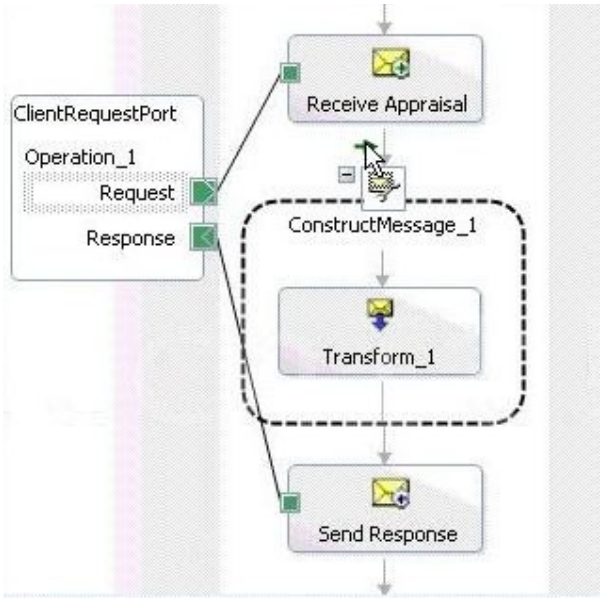


Figuur 64 BizTalk Orchestration Designer: port (bron: Woodgate '04)

Naast de operaties die voorkomen in een interactie moet ook worden bepaald welke berichten zullen worden uitgewisseld. Bij BPEL is dit ook beschreven via een partnerlink, in BizTalk worden de berichten apart gedefinieerd. In het Orchestration view venster (2) worden twee berichten aangemaakt, namelijk de *ClientRequest* (*Creq*) en *ClientResponse* (*CResp*). Bij *CResp* staat een uitroepteken omdat het berichttype nog

niet is bepaald. We wijzen het berichttype *CompanyRequest* toe aan het bericht *CResp*. Dit berichttype is het XML Schema dat we eerder hebben gemaakt met de BizTalk Editor. Als het berichttype is toegewezen moet het bericht bij een send/receive activiteit geplaatst worden. We merken op dat de send activiteit in de figuur nog een uitroepteken heeft staan, omdat het nog geen bericht heeft gekregen dat het kan verzenden. Eenmaal de send activiteit een bericht heeft, kan er een verbinding worden gemaakt met de operatie van de *ClientRequestPort* [Woodgate '04]. Zoals we kunnen zien in de orchestration view (2) worden berichten gescheiden van variabelen. De variabelen worden gebruikt om tussentijdse berekeningen op te slaan en en om berichten aan te maken.

We breiden ons proces nu verder uit met een ConstructMessage-constructie (zie figuur 65). De ConstructMessage maakt het mogelijk om berichten aan te maken. Het creëren van berichten kan via assign activiteiten of voor meer complexe situaties door gebruik te maken van mappings. Daar we reeds een mapping hebben aangemaakt, opteren we voor de tweede mogelijkheid. We kunnen een map gebruiken door middel van de transform-constructie. In de properties van deze constructie duiden we aan wat het input en output bericht is en de locatie van de mapping.



Figuur 65 BizTalk Orchestration Designer: Transform (bron: Woodgate '04)

Eenmaal het proces klaar is, kan het onmiddellijk worden gepubliceerd met de BizTalk Web Services Publishing Wizard. Met Microsoft InfoPath kan er dan automatisch voor de client een formulier worden gecreëerd. Bij verzending van het formulier worden de gepaste SOAP-messages verstuurd naar het proces dat is gepubliceerd en wordt het bericht dat wordt ontvangen ook weergeven.

4.4 Conclusie

In 2002, bij het vrijgeven van BPEL4WS 1.0, was BPWS4J het eerste product dat BPEL ondersteunde. Twee jaar later voegen de twee grootste vendors in het aanbieden van Web service platforms BPEL-ondersteuning toe aan hun producten.

Beide vendors hechten veel belang aan een SOA georieteerde aanpak en bieden een intergratieplatform bestaande uit verschillende producten. Beiden hebben een orchestratie-engine en een orchestratie-editor die worden aangevuld met mogelijkheden zoals het beheren van business rules, het opnemen van menselijk interacties in het proces, het debuggen van de processen en het uitvoeren van verschillende analyses.

Het grootste verschil tussen beide vendors is de manier waarop BPEL wordt ondersteund. IBM opteert voor een volledige ondersteuning, waarbij alle processen worden beschreven met BPEL. Microsoft gebruikt zijn eigen XLANG/s om processen te beschrijven en biedt de mogelijkheid om BPEL te importeren en te exporteren.

Algemeen Besluit

De integratie van IT-systemen komt voor bedrijven steeds meer in de belangstelling te staan. Vele bedrijven kiezen bij deze integratie-uitdaging voor een SOA-gebaseerde aanpak. SOA staat toe om bestaande assets voor te stellen als zichzelf beschrijvende en op standaarden gebaseerde services. Deze services kunnen dan hergebruikt en hergecombineerd worden om op een snelle en flexibele manier te reageren op veranderende bedrijfsomstandigheden. Het combineren van verschillende services tot bedrijfsprocessen en het efficiente beheer van deze processen, spelen een grote rol in het economische succes van een bedrijf.

De Web service-architectuur is een voorbeeld van een SOA en evolueerde over de jaren heen van een hype naar een vaste waarde binnen de IT-wereld. De orchestratie van Web services tot bedrijfsprocessen is echter een domein dat nog steeds wordt onderzocht en waarin nog niet veel overeenstemming is bereikt. Er zijn al verschillende standaarden op de markt gebracht om de integratielogica en de automatisering van op Web service gebaseerde bedrijfsprocessen te beschrijven. In deze verhandeling hebben we één van die standaarden, namelijk BPEL4WS (of BPEL), besproken en onderzocht of ze voldoet aan de vereisten van een compositietaal.

In hoofdstuk 1 (paragraaf 1.3.2) hebben we enkele puntjes aangehaald waar een standaard voor bedrijfsprocessen rekening mee zou moeten houden. Een van die puntjes is de mogelijkheid om de workflow van een proces te beschrijven. De workflow wordt opgesplitst in een gegevensstroom en een controlestroom. De gegevensstroom beschrijft welke gegevens tussen de verschillende services worden uitgewisseld, terwijl de controlestroom de volgorde bepaalt van de uit te voeren activiteiten.

Alvorens het proces gegevens kan uitwisselen met verschillende services, moeten die services gedefinieerd kunnen worden. BPEL gebruikt daarvoor de partnerlink (zie paragraaf 2.3.1). In een partnerlink kan worden aangegeven welke rol het proces en de aangeroepen services spelen in de interacties. Er wordt ook bepaald welke operaties kunnen worden aangeroepen en welke berichten kunnen worden uitgewisseld. Daarnaast moeten er ook activiteiten worden voorzien om de gegevens uit te wisselen. Een BPEL-proces heeft drie verschillende activiteiten ter beschikking die gebruikt kunnen worden in zowel synchrone als asynchrone interacties met Web services (zie

paragraaf 2.3.4). De ‘receive’ activiteit kan berichten ontvangen van een bepaalde partner. De ‘invoke’ activiteit kan een service aanroepen door een bericht te sturen. De ‘reply’ wordt gebruikt om te antwoorden op een synchroon verzoek. Aan de hand van de partnerlinks en de interactieactiviteiten kan de gegevensstroom bepaald worden. Welke gegevens er juist worden verstuurd, wordt beschreven in WSDL-messagetypes.

De controlestroom van het proces wordt beschreven aan de hand van verschillende gestructureerde activiteiten (zie paragraaf 2.3.5). BPEL gebruikt twee workflow-benaderingen om de volgorde van activiteiten voor te stellen, namelijk blok-gestructureerd en graafgestructureerd. Door beide workflowbenaderingen te combineren, kan men gebruik maken van de kracht van gestructureerd programmeren samen met de directheid en flexibiliteit van een graaf. Dit kwam duidelijk naar voor in paragraaf 3.3 bij de bespreking van de verschillende workflowpatronen. Door het gebruik van beide benaderingen, is BPEL immers in staat complexe workflowpatronen te modelleren. In vergelijking met andere talen is BPEL dan ook heel expressief. Deze aanpak heeft echter ook een nadeel. Vele patronen kunnen op verschillende manieren beschreven worden zonder enig semantisch verschil. Dit draagt niet bij tot de eenduidigheid van de taal en kan verwarring scheppen.

Naast de gegevensstroom en de controlestroom voorziet BPEL een aantal constructies voor de afhandeling van fouten die tijdens het proces worden opgeworpen. Eerst en vooral kan het proces worden opgesplitst in aparte eenheden door middel van de scope activiteit. De scope zorgt ervoor dat fouten lokaal worden behandeld. Elke scope kan zijn eigen faultHandlers bezitten. De faultHandlers dienen om fouten op te vangen tijdens de uitvoering van de scope. Op deze manier kunnen de faultHandlers activiteiten uitvoeren om er voor te zorgen dat het proces verder kan werken of op een normale manier kan worden afgesloten. De scope activiteit is ook van belang voor het gebruik van langdurige transacties. In paragraaf 1.3.2 hebben we opgemerkt dat traditionele ACID-transacties niet voldoende zijn in een Web service-omgeving en dat er nood is aan de ondersteuning van langdurige transacties. Langdurige transacties worden vaak opgesplitst door middel van scopes in kleinere transacties om zo sneller resources vrij te geven. In tegenstelling tot de rollback van traditionele transacties worden er nu compensatie-activiteiten uitgevoerd om het systeem terug in een consistente toestand te brengen. BPEL laat toe voor scopes een compensatie te definiëren die kan worden

aangeroepen om de scope ongedaan te maken. De nood aan constructies ter ondersteuning van langdurige transacties is één van de doelstellingen van de BPEL4WS-specificatie (zie paragraaf 3.1.8). Transactiemanagement en foutafhandeling zijn ook twee concepten die we in hoofdstuk 1 hebben aangehaald als vereiste voor een processtandaard.

Het laatste woord over BPEL en transacties is echter nog niet gezegd. De specificatie stelt voor om WS-Business Activity te gebruiken voor langdurige transacties, maar dit is lang niet de enige standaard. Business Transaction Protocol (BTP) en WS-Transaction Management (WS-TXM) zijn twee andere standaarden die langdurige transacties mogelijk maken. In het WSBPEL TC zijn er ook voorstellen gedaan om meer Business Transaction Management (BTM) mogelijkheden toe te voegen aan de BPEL-specificatie. Dit is momenteel echter uitgesteld wegens tijdsgebrek, maar zal zeker nog in de toekomst verder worden behandeld.

We kunnen hieruit afleiden dat de BPEL een vrij volledige taal biedt om bedrijfsprocessen te beschrijven. De interacties met verschillende partners, de gegevens en controlestroom en de constructies voor foutafhandeling en compensatie maken het mogelijk om complexe processen met BPEL te definiëren. Dit wil echter niet zeggen dat de taal af is.

Doorheen de verhandeling is het duidelijk geworden dat er nog verschillende lopende voorstellen zijn om BPEL te verbeteren en uit te breiden, zoals het toevoegen van BTM mogelijkheden. Er is ook een voorstel aan de leden van het BPEL TC voorgelegd om BPEL te formaliseren. Op deze manier kunnen ambiguïteiten en inconsistenties in de taal worden weggewerkt. Het formaliseren helpt ook bij het documenteren van de semantiek van de verschillende constructies om zo onduidelijkheden weg te nemen, zoals de beperkte uitleg over ‘serializable scopes’ en de onduidelijkheid over de betekenis van abstracte processen.

Er worden ook andere protocollen ontwikkeld om BPEL-processen te ondersteunen. We hebben in paragraaf 2.4 besproken hoe WS-Coordination BPEL kan aanvullen met coördinatievoorzieningen en hoe WS-Transaction kan instaan voor zowel atomic als langdurige transacties. Een vrij recente ontwikkeling is die van het Asynchronous

Service Acces Protocol (ASAP) dat BPEL en andere standaarden aanvult met voorzieningen voor asynchrone services.

Enkele andere belangrijk aspecten voor bedrijfsprocessen die we in hoofdstuk 1 hebben besproken zijn beveiliging en betrouwbaarheid van de berichtuitwisseling. Deze aspecten zijn heel belangrijk, maar vallen buiten de scope van deze verhandeling. Het is namelijk niet de taak van BPEL om de beveiliging en betrouwbaarheid van de berichtuitwisseling te garanderen. We hebben immers in doelstelling 10 gezien dat BPEL alleen die taken moet uitvoeren die specifiek met procesmodellering te maken hebben en dat BPEL moet samenwerken met andere protocollen wanneer bijkomende functionaliteit vereist is. Protocollen die hiervoor in aanmerking komen zijn bijvoorbeeld WS-Security en WS-Reliable Messaging. Beide protocollen zijn net als BPEL bij OASIS ondergebracht.

Toekomstig onderzoek zou zich kunnen richten op de samenwerking tussen BPEL en verschillende transactieprotocollen. Het is misschien ook interessant om een merkwaardige ontwikkeling binnen de BPEL-specificatie op te volgen. Eind maart 2004, stelden IBM en BEA in een gezamenlijke paper [BPELJ ‘04] een variant voor van BPEL, namelijk BPELJ. BPELJ is een combinatie tussen BPEL en de Java-taal. De bedoeling van BPELJ zou zijn om proceslogica van BPEL (‘programming in the large”) aan te vullen met mogelijkheden om business functies te implementeren (“programming in the small”). Het merkwaardige van deze ontwikkeling is dat het indruist tegen het principe van platform - en taalonafhankelijkheid. Op deze manier zou bijvoorbeeld ook BPELC# kunnen worden vrijgegeven, met gevolg dat het uitwisselen van bedrijfsprocessen niet meer mogelijk is.

Voor de bedrijven is er nood aan duidelijkheid. Men zal pas Web service-orchestratie echt beginnen toe te passen, als er een keuze is gemaakt tussen het grote aantal processtandaarden. BPEL is op weg, mits een aantal aanpassingen en uitbreidingen, een goede standaard te worden voor het beschrijven van bedrijfsprocessen. Dit jaar nog zou OASIS een gestandaardiseerde versie van BPEL moeten vrijgeven. BPEL gaat dan ook naar alle waarschijnlijkheid dé standaard worden voor XML-gebaseerde bedrijfsproces-modellering.

Bijlagen

Bijlage 1 : BPEL Document van LoanFlow

```
<process name="LoanFlow"
         targetNamespace="http://samples.otn.com"
         suppressJoinFailure="yes"
         xmlns:tns="http://samples.otn.com"
         xmlns:services="http://services.otn.com"
         xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process
/">

<!--
************************************************************************
         This process invokes a synchronous credit rating service
         (which throws NegativeCredit exceptions), then  it invokes
         two asynchronous loan processor services in parallel and
         finally selects the best loan offer received and returns it
         (asynchronously) to its caller.
************************************************************************
-->

  <partnerLinks>
    <partnerLink name="client"
             partnerLinkType="tns:LoanFlow"
             partnerRole="LoanFlowRequester"
             myRole="LoanFlowProvider"/>

    <partnerLink name="creditRatingService"
             partnerLinkType="services:CreditRatingService"
             partnerRole="CreditRatingServiceProvider"/>

    <partnerLink name="UnitedLoanService"
             partnerLinkType="services:LoanService"
             myRole="LoanServiceRequester"
             partnerRole="LoanServiceProvider"/>

    <partnerLink name="StarLoanService"
             partnerLinkType="services:LoanService"
             myRole="LoanServiceRequester"
             partnerRole="LoanServiceProvider"/>
  </partnerLinks>
```

```
<variables>
  <!-- input of this process -->
  <variable name="input"
         messageType="tns:LoanFlowRequestMessage"/>
  <variable name="crInput"
         messageType="services:CreditRatingServiceRequestMessage"/>
  <variable name="crOutput"
         messageType="services:CreditRatingServiceResponseMessage"/>
  <variable name="crError"
         messageType="services:CreditRatingServiceFaultMessage"/>

  <!-- input of united loan and star loan-->
  <variable name="loanApplication"
         messageType="services:LoanServiceRequestMessage"/>
  <!-- output of united loan-->
  <variable name="loanOffer1"
         messageType="services:LoanServiceResultMessage"/>
  <!-- output of star loan-->
  <variable name="loanOffer2"
         messageType="services:LoanServiceResultMessage"/>

  <!-- output of this process -->
  <variable name="selectedLoanOffer"
         messageType="tns:LoanFlowResultMessage"/>
</variables>

<sequence>

  <!-- ************************************************************
       Receive input from requestor - this is what kicks
       off this flow. We get passed a loan application business doc
       ************************************************************
    -->

  <receive name="receiveInput" partnerLink="client"
           portType="tns:LoanFlow"
           operation="initiate" variable="input"
           createInstance="yes"/>

  <!-- ************************************************************
       Invoke the synchronous creditRatingService. Define a scope
```

```
          for handling faults from it and set the credit rating in the
          loan app bus doc if we get a credit rating back. In the case
          of a NegativeCredit exception, set it to -1000.
          ************************************************************
-->
    <scope name="GetCreditRating" variableAccessSerializable="no">

      <!-- Watch for faults (exceptions) being thrown from
           creditRatingService -->
      <faultHandlers>
        <catch faultName="services:NegativeCredit"
               faultVariable="crError">

            <!-- For now, just set creditRating to -1000 for
                 negative credit exceptions -->
            <assign>
                <copy>
                    <from expression="number(-1000)"/>
                    <to variable="input" part="payload"
                        query="/loanApplication/creditRating"/>
                </copy>
            </assign>
        </catch>
      </faultHandlers>

      <sequence>
          <!--
************************************************************
              Copy the SSN from the input LoanApplication variable
              to the creditRating service input variable

************************************************************ -->

          <assign>
            <copy>
                <from variable="input" part="payload"
query="/loanApplication/SSN"/>
                <to variable="crInput" part="payload"
                    query="/ssn"/>
            </copy>
          </assign>
```

```
        <!-- Invoke the CreditRating Service, the URL of this
service's
            WSDL is specified in the deployment descriptor -->

        <invoke name="invokeCR"
                partnerLink="creditRatingService"
                portType="services:CreditRatingService"
                operation="process"
                inputVariable="crInput"
                outputVariable="crOutput"/>

        <!-- Add the credit rating we received to the loan application
            business document -->

        <assign>
            <copy>
                <from variable="crOutput" part="payload"
query="/rating"/>
                <to variable="input" part="payload"
                    query="/loanApplication/creditRating"/>
            </copy>
        </assign>

      </sequence>

    </scope>

    <!-- ************************************************************
         Now we will invoke the 2 loan providers (async web services)
         in parallel. We send each a LoanApplication bus doc and wait
         for a LoanOffer to be returned.
         ************************************************************
-->

      <scope name="GetLoanOffer" variableAccessSerializable="no">
          <sequence>
                <!-- initialize the input of UnitedLoan and StarLoan --
>
                <assign>
                    <copy>
```

```
                        <from variable="input" part="payload"/>
                        <to variable="loanApplication" part="payload"/>
                    </copy>
                </assign>

                <flow>

                <!--
************************************************************
                     Invoke the first loan provider (UnitedLoan)

************************************************************ -->
                  <!-- invoke first loan provider -->
                  <sequence>

                    <!--  initiate the remote service -->
                    <invoke name="invokeUnitedLoan"
                            partnerLink="UnitedLoanService"
                            portType="services:LoanService"
                            operation="initiate"
                            inputVariable="loanApplication"/>

                    <!--  receive the result of the remote service -->
                    <receive name="receive_invokeUnitedLoan"
                            partnerLink="UnitedLoanService"
                            portType="services:LoanServiceCallback"
                            operation="onResult"
                            variable="loanOffer1"/>

                  </sequence>

                  <!--
************************************************************
                        Invoke the second loan provider (StarLoan)

************************************************************ -->
                  <sequence>

                    <!--  initiate the remote service -->
                    <invoke name="invokeStarLoan"
partnerLink="StarLoanService"
```

```
                        portType="services:LoanService"
                        operation="initiate"
                        inputVariable="loanApplication"/>

                <!--  receive the result of the remote service -->
                <receive name="receive_invokeStarLoan"
                        partnerLink="StarLoanService"
                        portType="services:LoanServiceCallback"
                        operation="onResult"
                        variable="loanOffer2"/>

            </sequence>

          </flow>
      </sequence>
    </scope>

    <!-- ************************************************************
         Decide which offer has the lower APR. Copy the offer with the
         lowest APR into the selectedLoanOffer return document.
         ************************************************************
-->
    <scope name="SelectOffer" variableAccessSerializable="no">
         <switch>

            <!-- If loanOffer1 is greater (worse) than loanOffer2 --
>
            <case
condition="bpws:getVariableData('loanOffer1','payload','/loanOffer/APR
') > bpws:getVariableData('loanOffer2','payload','/loanOffer/APR') ">

                <!-- Then take loanOffer2 -->
                <assign>
                    <copy>
                    <from variable="loanOffer2" part="payload"/>
                    <to variable="selectedLoanOffer"
part="payload"/>
                    </copy>
                </assign>

            </case>
```

```
            <!-- Otherwise take loanOffer1 -->
            <otherwise>

                <assign>
                    <copy>
                    <from variable="loanOffer1" part="payload"/>
                    <to variable="selectedLoanOffer"
part="payload"/>
                    </copy>
                </assign>

            </otherwise>

         </switch>
    </scope>

    <!-- ************************************************************
         And, finally, send the return offer back to our client who
         kicked off the whole process.
         ************************************************************
-->
    <invoke name="replyOutput"
            partnerLink="client"
            portType="tns:LoanFlowCallback"
            operation="onResult"
            inputVariable="selectedLoanOffer"/>

  </sequence>

</process>
```

Bijlage 2 : WSDL Document van LoanFlow

```
<?xml version="1.0"?>
<definitions name="LoanFlow"
        targetNamespace="http://samples.otn.com"
        xmlns:tns="http://samples.otn.com"
        xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-
link/"
        xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns:s1="http://www.autoloan.com/ns/autoloan"
        >

  <types>
    <schema attributeFormDefault="qualified"
elementFormDefault="qualified"
targetNamespace="http://www.autoloan.com/ns/autoloan"
xmlns="http://www.w3.org/2001/XMLSchema">
          <element name="loanApplication"
type="s1:LoanApplicationType"/>
          <element name="loanOffer" type="s1:LoanOfferType"/>

          <complexType name="LoanOfferType">
              <sequence>
                <element name="providerName" type="string"/>
                <element name="selected" type="boolean"/>
                <element name="approved" type="boolean"/>
                <element name="APR" type="double"/>
              </sequence>
          </complexType>
          <complexType name="LoanApplicationType">
              <sequence>
                <element name="SSN" type="string"/>
                <element name="email" type="string"/>
                <element name="customerName" type="string"/>
                <element name="loanAmount" type="double"/>
                <element name="carModel" type="string"/>
                <element name="carYear" type="string"/>
                <element name="creditRating" type="int"/>
              </sequence>
          </complexType>
```

```
    </schema>
  </types>

  <message name="LoanFlowRequestMessage">
    <part name="payload" element="s1:loanApplication"/>
  </message>
  <message name="LoanFlowResultMessage">
    <part name="payload" element="s1:loanOffer"/>
  </message>

  <portType name="LoanFlow">
    <operation name="initiate">
      <input message="tns:LoanFlowRequestMessage"/>
    </operation>
  </portType>
  <portType name="LoanFlowCallback">
    <operation name="onResult">
      <input message="tns:LoanFlowResultMessage"/>
    </operation>
  </portType>

  <plnk:partnerLinkType name="LoanFlow">
    <plnk:role name="LoanFlowProvider">
      <plnk:portType name="tns:LoanFlow"/>
    </plnk:role>
    <plnk:role name="LoanFlowRequester">
      <plnk:portType name="tns:LoanFlowCallback"/>
    </plnk:role>
  </plnk:partnerLinkType>

</definitions>
```

Bijlage 3: Patroon gebaseerde analyse van workflowtalen

pattern	product							
	Staffware	COSA	InConcert	Eastman	FLOWer	Domino	Meteor	Mobile[1]
1 (seq)	+	+	+	+	+	+	+	+
2 (par-spl)	+	+	+	+	+	+	+	+
3 (synch)	+	+	+	+	+	+	+	+
4 (ex-ch)	+	+	+/-	+	+	+	+	+
5 (simple-m)	+	+	+/-	+	+	+	+	+
6 (m-choice)	-	+	+/-	+/-	-	+	+	+
7 (sync-m)	-	+/-	+	+	-	+	-	-
8 (multi-m)	-	-	-	+	+/-	+/-	+	-
9 (disc)	-	-	-	+	+/-	-	+/-	+
10 (arb-c)	+	+	-	+	-	+	+	-
11 (impl-t)	+	-	+	+	-	+	-	-
12 (mi-no-s)	-	+/-	-	+	+	+/-	+	-
13 (mi-dt)	+	+	+	+	+	+	+	+
14 (mi-rt)	-	-	-	-	+	-	-	-
15 (mi-no)	-	-	-	-	+	-	-	-
16 (def-c)	-	+	-	-	+/-	-	-	-
17 (int-par)	-	+	-	-	+/-	-	-	+
18 (milest)	-	+	-	-	+/-	-	-	-
19 (can-a)	+	+	-	-	+/-	-	-	-
20 (can-c)	-	-	-	-	+/-	+	-	-

Table 1: The main results for Staffware, COSA, InConcert, Eastman, FLOWer, Lotus Domino Workflow, Meteor, and Mobile.

pattern	product						
	MQSeries	Forté	Verve	Vis. WF	Changeng.	I-Flow	SAP/R3
1 (seq)	+	+	+	+	+	+	+
2 (par-spl)	+	+	+	+	+	+	+
3 (synch)	+	+	+	+	+	+	+
4 (ex-ch)	+	+	+	+	+	+	+
5 (simple-m)	+	+	+	+	+	+	+
6 (m-choice)	+	+	+	+	+	+	+
7 (sync-m)	+	-	-	-	-	-	-
8 (multi-m)	-	+	+	-	-	-	-
9 (disc)	-	+	+	-	+	-	+
10 (arb-c)	-	+	+	+/-	+	+	-
11 (impl-t)	+	-	-	-	-	-	-
12 (mi-no-s)	-	+	+	+	-	+	-
13 (mi-dt)	+	+	+	+	+	+	+
14 (mi-rt)	-	-	-	-	-	-	+/-
15 (mi-no)	-	-	-	-	-	-	-
16 (def-c)	-	-	-	-	-	-	-
17 (int-par)	-	-	-	-	-	-	-
18 (milest)	-	-	-	-	-	-	-
19 (can-a)	-	-	-	-	-	-	+
20 (can-c)	-	+	+	-	+	-	+

Table 2: The main results for MQSeries, Forté Conductor, Verve, Visual WorkFlo, Changengine, I-Flow, and SAP/R3 Workflow.

Lijst van Figuren

Bronnen

AlphaWorks, (2002), *BPWS4J*, IBM AlphaWorks, http://www.alphaworks.ibm.com/tech/bpws4j

Andrews G., Dias R., (2003), *Gartner Application and Integration Conference*, http://blogs.msdn.com/rdias/archive/2003/11/20/56450.aspx

Bloomberg J., (2001), *Web services: a new paradigm for distributed computing*, IBM e-zine The Rational Edge September, http://www-106.ibm.com/developerworks/rational/rationaledge/

BPEL4WS Spec, (2003*), Specification: Business Process Execution Language for Web Services Version 1.1,* IBM DevelopersWorks, http://www-106.ibm.com/developerworks/library/ws-bpel

BPELJ, (2004), *BPELJ: BPEL for Java*, a joint white paper by BEA and IBM, http://ftpna2.bea.com/pub/downloads/ws-bpelj.pdf

Bunting D., Chapman M., Hurley O., Little M., Mischkinsky J., Newcomer E., Webber J., Swenson K., (2003), *Web Services Transaction Management,* Sun Microsystems, http://developers.sun.com/techtopics/webservices/wscaf/wstxm.pdf

Cabrera L., Copeland G., Johnson J., Langworthy D., (2004), *Coordinating Web Services Activities with WS-Coordination, WS-AtomicTransaction, and WS-BusinessActivity,* Microsoft Corporation, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/wsacoord.asp

Collaxa, (2003), *Collaxa's BPEL4WS 101 Tutorial*, Collaxa Inc, http://www.collaxa.com/pdf/collaxa-bpel101.pdf

Coverpages, (2002), *IBM alphaWorks Releases BPWS4J Engine and Editor*, http://xml.coverpages.org/ni2002-08-12-b.html

Curbera F., Khalaf R., Mukhi N., Tai S., Weerawarana S., (2003), *The Next Step in Web Services*, Communications of the ACM, Vol. 46 No. 10

D.H. Brown Associates, (2004), *Building and Deploying Service-Oriented Applications That Extend and Integrate Existing IT Assets,* D.H. Brown Associates, Inc., ftp://ftp.software.ibm.com/software/integration/library/whitepapers/wbisf_dhbrown0414.pdf

Dante Consulting, (2003), *Web services and B2B integration, Dante Consulting,* http://www.dante-consulting.com/docs/web-services/WebServicesB2B.pdf

Duftler M., Khalaf R., (2002), *Business Process with BPEL4WS: Learning BPEL4WS, Part 3*, IBM DeveolpersWorks, http://www-106.ibm.com/developerworks/webservices/library/ws-bpelcol3.html

Dumas M., ter Hofstede A., van der Aalst W., (2003), *Web Service Composition Languages: Old Wine in New Bottles?,* IEEE Computer Society, p. 298-305 http://tmitwww.tm.tue.nl/research/patterns/download/wscl-euromicro.pdf

Ebpml.org, (2004 a), *WSFL*, Ebpml.org, http://www.ebpml.org/wsfl.htm

Ebpml.org, (2004 b), *XLANG*, Ebpml.org, http://www.ebpml.org/xlang.htm

Farahbod R., Glässer U., Vajihollahi M., (2004), *Specification and Validation of the Business Process Execution Language for Web Services*, Simon Fraser University, Cananda, http://www.cs.sfu.ca/~se/bpeltr/TechnicalReport.htm#Sec-Intro

Fletcher T., Furniss P., Green A., Haugen R., (2003) *BPEL and Business Transaction Management: Choreology Submission to OASIS WS-BPEL Technical Committee*, Choreology Ltd, http://www.oasis-open.org/committees/download.php/3263/

Gonsalves A., (2003), *Web Services Titans Submit Spec To OASIS, Bypass W3C*, TechWeb News, InternetWeek, http://www.internetweek.com/shared/printableArticle.jhtml?articleID=8800269

Gottschalk K., (2000), *Web Services architecture overview, IBM,* http://www-106.ibm.com/developerworks/web/library/w-ovr/?dwzone=ws

Green A., (2003 a), *Transacting Business with Web Services, Part 1* , Web Service Journal, Vol. 3, Issue 9, http://www.sys-con.com/webservices/articleprint.cfm?id=647

Green A., (2004 b), *Transacting Business with Web Services, Part 2* , Web Service Journal, Vol. 3, Issue 11, http://www.sys-con.com/story/print.cfm?storyid=39910

Haataja J., Tergujeff R., (2003), *Using BPEL4WS for Supply-Chain Integration*, University of Helsinki, http://www.cs.helsinki.fi/group/web-pil/docs/case_study_II.pdf

Haberl S., (2002), *Business Process Description Languages*, **OFFLINE** http://www.cis.unisa.edu.au/~cissh/research/webflow/bpdl.html

Kendal S.C., Waldo J., Wollrath A., Wyant G., (1994), *A note on distributed Computing*, SUN Tec report TR-94-29, http://research.sun.com/techrep/1994/smli_tr-94-29.pdf

Kreger H., (2002), Web Service Conceptual Architecture, IBM Software Group, www.math.uwaterloo.ca/~bkalali/webService.ppt

Lee J., Siau K., Hong S., (2003), *Enterprise Integration with ERP en EAI*, Communications of the ACM, Vol 46 No. 2, p54

Leyman F., Roller D., Schmidt M.-T., (2002), *Web Services and business process management*, IBM Systems Journals, vol 41, NO 2, 2002, p199

Leymann F., Roller D., Thatte S., (2003), *Goals of the BPEL4WS Specification*, http://xml.coverpages.org/BPEL4WS-DesignGoals.pdf

Little M., Weber J., (2003 a), *Introducing WS-Coordination*, Web Service Journal Vol. 03, Issue 05, http://sys-con.com/story/?storyid=39751

Little M., Weber J., (2003 b), *Introducing WS-Transaction: Part 1*, Web Service Journal Vol. 03, Issue 06, http://www.sys-con.com/story/?storyid=39769

Maeda T., Nomura Y., Hara H., (2003), *Security and Reliability for Web Services*, http://magazine.fujitsu.com/us/vol39-2/paper10.pdf

Masud S., (2003), *Building a Real-World Web Service - Part 2,* XML Journal Vol. 04 Issue 02 http://www.sys-con.com/xml/

McDonald C., (????),*Orchestration, Choreography and Collaboration,* Sun Technology Audiocasts, http://java.sun.com/developer/onlineTraining/webcasts/pdf/35plus/cmcdonald2.pdf

Micorsoft, (2004), Understanding BizTalk Server 2004, Microsoft, http://www.microsoft.com/downloads/details.aspx?FamilyID=8BDB04F6-5974-443E-BDFF-DA79D100BECB&displaylang=en

Microsoft BizTalk Server, (2004 a), *Importing BPEL4WS*, MSDN Microsoft, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sdk/htm/ebiz_prog_orch_bfpe.asp

Microsoft BizTalk Server, (2004 b), *Exporting BPEL4WS*, MSDN Microsoft, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sdk/htm/ebiz_prog_orch_zopr.asp

Moghal S., Mueller R., Boardman L., Johnson R., Kitabayashi H., Dring G., Kovari P., (2004), *WebSphere Business Integration Server Foundation V5.1 Handbook (draft document),* IBM

Mukhi N., (2002), *Creating processes with the BPWS4J editor*, IBM developerWorks, http://www-106.ibm.com/developerworks/webservices/library/ws-bpelcol4/

Myerson J., (2002), *Advancing the Web services stack, IBM devolperWorks,* http://www-106.ibm.com/developerworks/webservices/library/ws-wsa/

OASIS WS-ASAP TC, (2003), *OASIS Asynchronous Service Access Protocol TC*, OASIS, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=asap

OASIS WS-BPEL TC, (2004), *OASIS WS-BPEL TC Issues List*, OASIS, http://www.choreology.com/external/WS_BPEL_issues_list.html

OASIS, (2003), *OASIS Members Form Web Services Business Process Execution Language (WSBPEL) Technical Committee, OASIS News,* http://www.oasis-open.org/news/oasis_news_04_29_03.php

OASIS, (2004), *OASIS Web Services Business Process Execution Language TC,* http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

Oorlog R., (2003), *Bedrijfsprocessen modelleren in een webservices-omgeving*, WSDN.nl, http://wsdn.nl/SiteManager.php?cSite=16&cPage=193&cId=2

Oracle, (2003), *Oracle Backs BPEL4WS and WS-Choreography*, OTN News, http://otn.oracle.com/oramag/webcolumns/2003/techarticles/bpl_news.html

Oracle, (2004 a), *Tutorial 2: Developing a Credit Flow BPEL Process*, http://otn.oracle.com/products/ias/bpel/index.html

Oracle, (2004 b), *BPEL Designer*, http://otn.oracle.com/products/ias/bpel/index.html

O'Riordan D., (2002), *Business Process Standards for Web Services* http://www.webservicesarchitect.com/content/extras/Chapter10.pdf

Peltz C., (2003 a), *Web Service Orchestration and Choreography,* Web Service Journal Vol. 03, Issue 07, http://www.sys-con.com/webservices/

Peltz C., (2003 b), *Web Service Orchestration,* Hewlett-Packard Company, http://devresource.hp.com/drc/technical_white_papers/WSOrch/WSOrchestration.pdf

Peltz C., (2003 c), *Web Services Orchestration and Choreography*, IEEE Computer Society, Computer, October 2003, p46

Peter Fischer, (2002*), Answering the critical Web services questions*, Application Development Trends, July 2002, http://www.adtmag.com/print.asp?id=6467

Rijsdijk R., (2004), *BizTalk 2004 Overzicht - Het eerste product uit het nieuwe Microsoft e-Business platform*, Robert Rijsdijk's BizTalk Server Weblogs, http://sphear.demon.nl/weblogs/robert/articles/363.aspx

Rollman R., Cox W., (2003), *Transactions in Business Processes - A New Model*, Web Service Journal, Vol.03, Issue 10

Rossomando P., (2004), *Next Meeting of the "Informal" Abstract BPEL Clarification Working Group*, OASIS BPEL elist, http://lists.oasis-open.org/archives/wsbpel/200405/msg00067.html

Samtani G., Sadhwani D., (2001), *EAI en Web services : Easier Enterprise Application Integration ?,* Webservicearchitect.com, http://www.webservicesarchitect.com/content/articles/samtani01.asp

Samtani G., Sadhwani D., (2002), *B2Bi en Web services : An Intimidating Task?,* http://www.webservicesarchitect.com/content/articles/samtani02.asp

Service-architecture.com, (2003), Service-oriented architecture, *http://www.service-architecture.com/*

Snell J., (2002), *Getting into the flow: the Web Service Flow Language*, Web Service Journal, Vol. 02, Issue 03 , http://www.sys-con.com/webservices/

Taft D., (2003), *OASIS Forms Committee to Promote BPEL*, Eweek enterprise news & reviews, http://www.eweek.com/article2/0,1759,1047671,00.asp

Thatte S., (2004), Usage of BPEL Abstract Processes, OASIS BPEL elist, http://lists.oasis-open.org/archives/wsbpel/200405/msg00106.html

Townsend D., Cairns R., Schittko C., (2003), *Using BPEL*, Web Service Journal, Vol. 03, Issue 07

UDDI.org, (2004), *About UDDI*, http://www.uddi.org/about.html

van der Aalst W., Dumas M., Hofstede A., Wohed P., (2003), *Pattern Based Analysis of BPEL4WS*, http://tmitwww.tm.tue.nl/research/patterns/download/qut_bpel_rep.pdf

van der Aalst W., Hofstede A., Kiepuszewski B., Barros A., (2003), *Workflow Patterns*, http://www.citi.qut.edu.au/pubs/technical/workflow_patterns.pdf

van Sinderen M., Almeida J.P.A., (2003), *Modelling of services and component interactions,* https://doc.telin.nl/dscgi/ds.py/Get/File-36517/D3.2-draft2.0.pdf

Vandenbulcke J., Lemahieu W., (2000), *Databasesystemen voor de praktijk*, TenHagen Stam, Den Haag

Verma M., Deswal P., (2003), *Approaching Web services transactions*, IBM developerWorks, http://www-106.ibm.com/developerworks/webservices/library/ws-tranart/

W3C Choreography, (2003), *Web Services Choreography Working Group Charter*, http://www.w3.org/2003/01/wscwg-charter

W3C SOAP, (2003), *SOAP Version 1.2*, http://www.w3.org/TR/soap12-part1/

W3C WSA, (2004), *Web Services Architecture (Working Group Note)* http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

W3C WSDL, (2002), *Web Services Description Language,* W3C, http://www.w3.org/TR/2002/WD-wsdl12-20020709/

Weber J., (2003), *Introducing WS-Transaction: Part 2,* Web Service Journal Vol. 03, Issue 07, http://www.sys-con.com/story/?storyid=39795

WebSphere Software, (2003), *IBM WebSphere Business Integration Monitor, Version 4.2.4*, IBM, ftp://ftp.software.ibm.com/software/integration/library/specsheets/wbimonitor_G325-2210-01.pdf

Weerawarana S., Curbera F., (2002), *Business Process with BPEL4WS: Understanding BPEL4WS, Part 1, IBM,* http://www-106.ibm.com/developerworks/webservices/library/ws-bpelcol1/

Woodgate S., (2004), Building your first business process, MSDN TV, http://msdn.microsoft.com/msdntv/episode.aspx?xml=episodes/en/20040520biztalksw/manifest.xml

WorkflowPatterns.com, (2003), *Structural Patterns – Arbitrary Cycles*, http://tmitwww.tm.tue.nl/research/patterns./patterns.htm

WSJ News Desk, (2003), *BPEL4WS Submitted to OASIS: Microsoft Says v.1.1 to Be Royalty-Free,* Web Service Journal, http://www.sys-con.com/WebServices/ article.cfm?id=536