EVENT DRIVEN SOFTWARE ENGINEERING

VAN ASYNCHRONE COMMUNICATIE TOT BIG DATA ANALYTICS

KOEN VANDERKIMPEN

Smals SECTIE ONDERZOEK 06/2016 • RESEARCH NOTE • 40

Abstract

In computersoftware, en zeker in gedistribueerde systemen, zijn er gebeurtenissen ('Events') in overvloed. Wanneer men deze Events een centrale plaats geeft bij het ontwerpen, dan komt men in het domein van 'Event Driven Software Engineering' (EDSE), een discipline die onder andere de 'Event Driven Architecture' (EDA) omvat.

In deze Research Note wordt getracht om het belang aan te tonen van de rol die Events kunnen spelen bij het opzetten van performante, betrouwbare en onderhoudbare informatiesystemen: applicaties en services kunnen los aan elkaar gekoppeld worden en worden toch in real-time op de hoogte gehouden van nieuwe data. Daarnaast zijn Events op zich ook een erg krachtig paradigma als bouwsteen voor sommige individuele applicaties, of als gegevensbron voor Big Data Analytics en real-time intelligente systemen, zoals deze voor Complex Event Processing (CEP).

Dit alles gaat natuurlijk niet zonder slag of stoot; de technologie om dit te realiseren is weliswaar beschikbaar, maar een goede modellering van de (Business) Events is onontbeerlijk en meteen de grootste uitdaging voor analisten en architecten.

Résumé

En ce qui concerne les logiciels et certainement les systèmes distribués, les événements ('Events') ne manquent pas. Lorsque ces Events sont placés au cœur de la conception, on entre dans le domaine de l'Event Driven Software Engineering (EDSE), une discipline qui englobe entre autres l'Event Driven Architecture (EDA).

La présente Research Note a pour objet de démontrer l'importance du rôle que ces Events peuvent jouer dans la mise en place de systèmes d'information performants, fiables et maintenables. Les applications et services peuvent faire l'objet d'un couplage faible et quand même être informés en temps réel des nouvelles données. D'autre part, les Events sont un très puissant paradigme en tant que composante de certaines applications individuelles ou en tant que source de données pour les Big Data Analytics et les systèmes intelligents en temps réel, comme ceux du Complex Event Processing (CEP).

Bien entendu, tout ceci n'est pas évident. Car si les technologies nécessaires sont disponibles, une bonne modélisation des (Business) Events est indispensable et représente le défi majeur des analystes et architectes.

Inhoud

Deze nota bevat de belangrijkste conclusies van de onderzoeksstudie (2015) 'Event Driven Architecture' (EDA). Daarnaast komen nog een aantal extra zaken aan bod die draaien rond Events binnen IT-systemen en die nog een stapje verder gaan dan EDA. Het geheel aan disciplines rond Events noemen we 'Event Driven (Software) Engineering' (EDSE).

1. INLEIDING

In computersoftware, en zeker in gedistribueerde systemen, zijn er gebeurtenissen in overvloed. Er wordt echter meestal slechts als bijzaak enige aandacht aan besteed. Wanneer men het Event een centralere plaats geeft bij het ontwerpen van software – wat op verschillende manieren kan – komt men in het domein van EDSE. In dit rapport bespreken we een aantal interessante voordelen van deze aanpak.

1.1. Events

Heel veel zaken binnen een softwaresysteem kunnen worden gemodelleerd als een gebeurtenis of Event: het aanklikken van een knop, het indrukken van een toets, het verbinden met een server, enz. Een bijzonder interessante vorm van gebeurtenissen zijn daarbij degene die ook op businessniveau een betekenis hebben: het inloggen van een gebruiker, het versturen van een aanvraagformulier, het aanmaken van een nieuw record met gegevens, of zelfs het beschikbaar worden van een signaal komende uit een Analytics-platform.

Deze gebeurtenissen zijn er altijd, maar we besteden er niet altijd enorm veel aandacht aan: het verbinden van een server wordt simpelweg het oproepen van een hulpdienst vanuit een ander deel van het programma, de gegevens

van een aanvraagformulier worden gewoon gecontroleerd en weggeschreven, het signaal wordt getoond op een dashboard, enz.

We doen dus wel iets met de gegevens die samenhangen met het Event, maar we beschouwen het Event zelf niet als een volwaardig concept. Eens voorbij, verliezen we dan ook het concept van iets dat zich afspeelt op een bepaald moment in de tijd. De betrokken gegevens worden weggeschreven, vaak als deel van een aggregaat, en de historische context en tijdsgebondenheid gaan verloren.

Fig. 1: Wanneer we het in dit rapport en in gerelateerde documenten over Events hebben, zullen we deze steevast voorstellen a.d.h.v. deze achthoekige figuur.

Wanneer we nu het Event wél gaan beschouwen als een object van eerste klasse (in het Engels '*first class citizen*'), dan maken we er als het ware een tastbaar, vastgrijpbaar iets van. Het Event wordt als dusdanig *gereïficeerd* ('écht gemaakt'). Wanneer we dit doen, dan noemen we dit Event Driven Software Engineering. Dit is een brede tak binnen de Software Engineering, die zelf nog uit een aantal verschillende paradigma's bestaat, die we verderop in dit document zullen aanhalen.

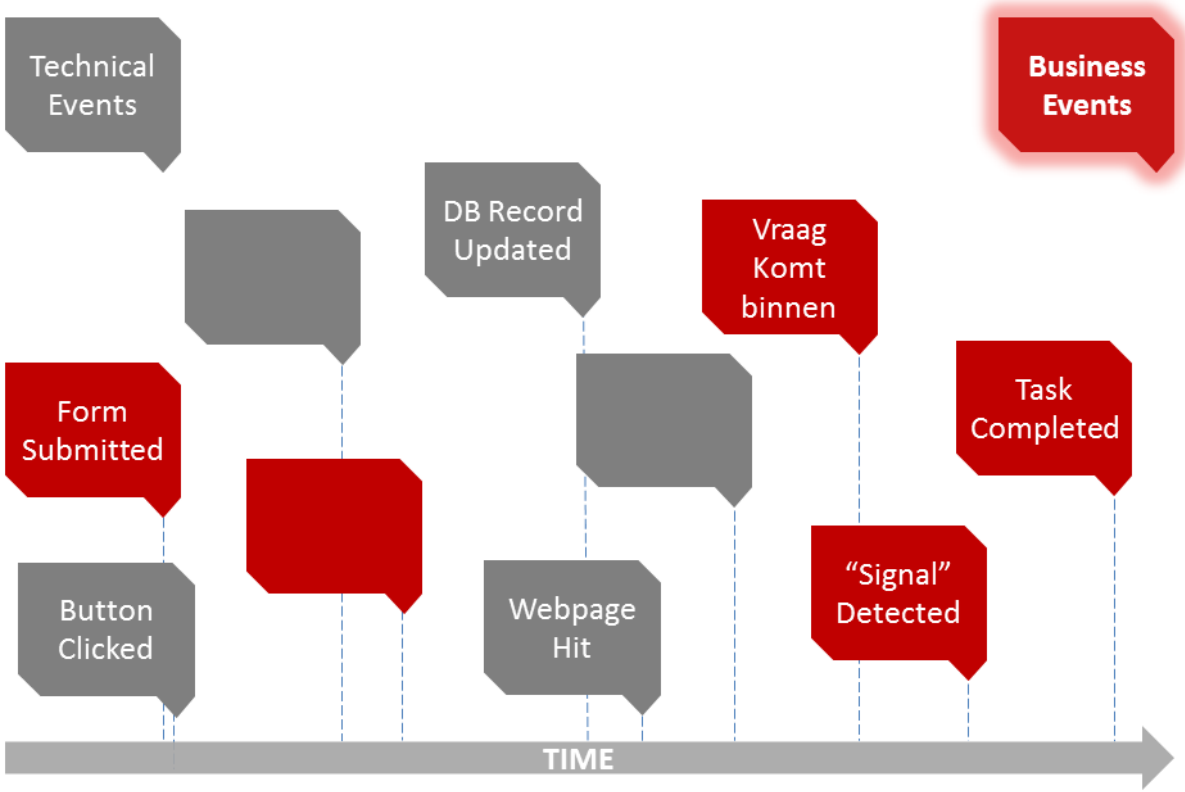


Fig. 2: Bijna alles wat er gebeurt in een softwaresysteem kan worden gemodelleerd als een Event. De categorie van Business Events is daarbij erg belangrijk.

1.2. Context: 'Vortex of Enablers'

Deze studie is de eerste van een reeks lopende onderzoeken naar de bredere architecturale evoluties binnen en rond softwareontwikkeling. Deze ontwikkelingen staan op hun beurt ten dienste van een aantal nog ruimere, strategische, businessgedreven evoluties, waar we momenteel mee worden geconfronteerd: *Cloud, Mobile, Social* en *Information*. Deze grote evoluties staan bij Gartner ook bekend als de *Nexus of Forces*[1].

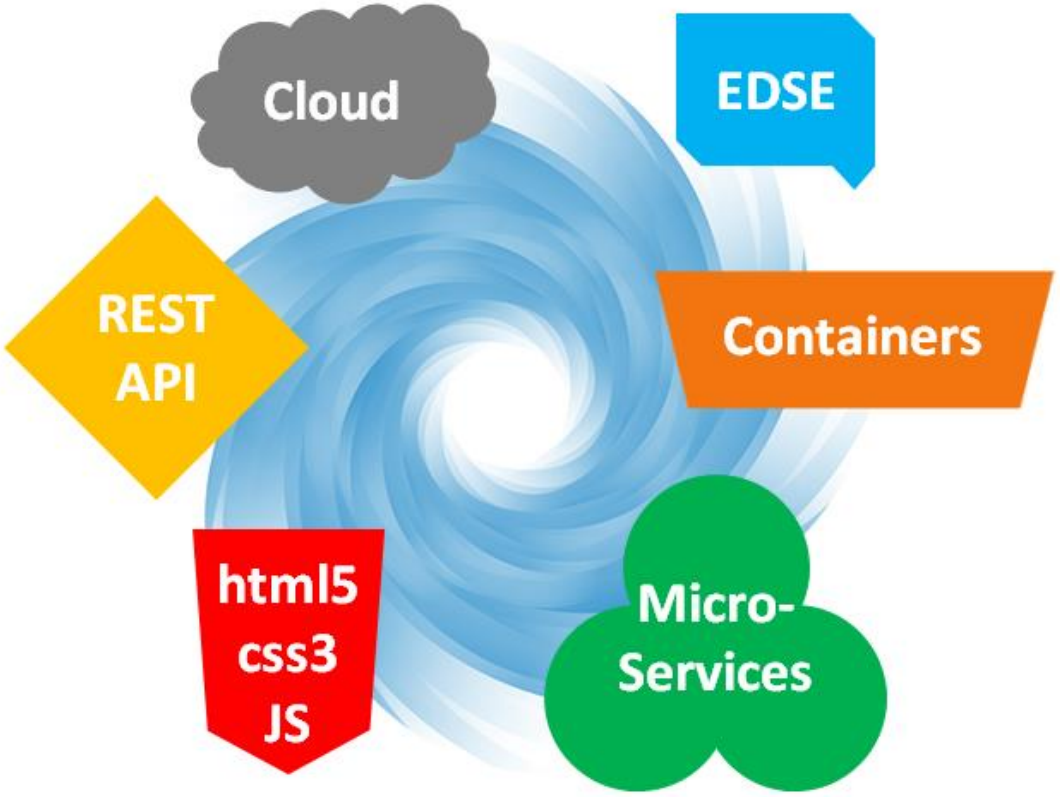


Fig. 3: De « Vortex of Enablers »

Zonder een evolutie binnen softwarearchitectuur en -ontwikkeling zou deze Nexus of Forces technisch echter maar moeilijk haalbaar zijn. Tijdens dit onderzoek identificeerden we zes onderliggende, eerder technische, evoluties, die deze ruimere business(r)evolutie mogelijk maken. Het gaat om Event Driven Software Engineering (soms kortweg 'Events'), de RESTful API economy (ook kortweg 'Rest'), het nieuwe Web (ook wel 'HTML5, CSS3 en Javascript'), Microservices Architectures ('Microservices'), Containers en Cloud. Bij deze laatste gaat het dan vooral om de technische kant van de Cloud, zoals Cloud-Enabled Software, Platform as a Service (PaaS)[2] en Infrastructure as a Service (IaaS), waarbij Cloud binnen de Nexus of Forces

[1] http://www.gartner.com/it-glossary/nexus-of-forces

[2] Zie de nota Application Platform as a Service: https://www.smalsresearch.be/publications/document/?docid=100

zich eerder focust op de businesskant van Cloud, via Software as a Service (SaaS).

Deze zes katalysatoren, die de toenemende complexiteit van de IT-wereld beheersbaar houden, noemen wij de *Vortex of Enablers*. De eerste twee (Events en Rest) richten zich vooral op hoe communicatie tussen systemen evolueert. De drie laatste (Microservices, Containers en Cloud) definiëren meer hoe systemen op zich er gaan uitzien en op welke manier ze kunnen worden uitgerold. De derde, het nieuwe web, verbetert de mogelijkheden die men heeft bij het ontwikkelen van gebruikersinterfaces van computersystemen.

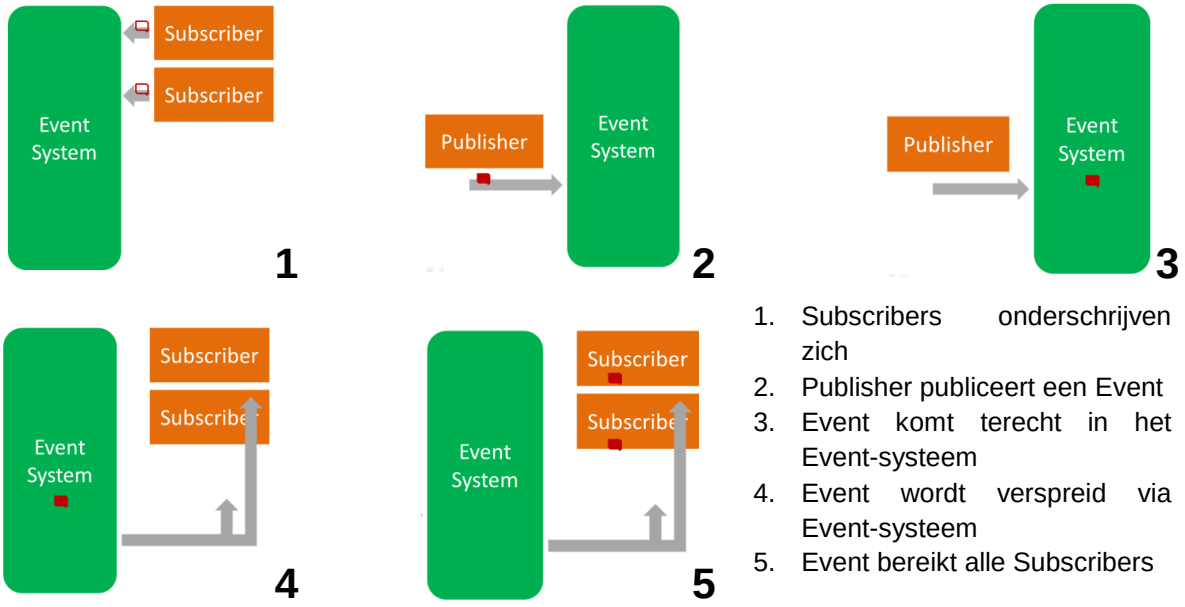
1.3. Publish / Subscribe

Het publish/subscribe mechanisme is al erg lang een gekende techniek binnen software-ontwerp om subsystemen van elkaar los te koppelen. Ook voor Events kan dit mechanisme een belangrijke rol spelen. Het mechanisme speelt zich af tussen drie partijen: de publisher (die een Event kan ‘publiceren’), de subscriber (die zich voor een event kan ‘onderschrijven’), en het Event systeem. Deze partijen kunnen verschillende vormen aannemen (van één welbepaald object binnen een applicatie, over subsystemen heen, tot verschillende en volledig op zichzelf staande applicaties of IT-systemen). Van publishers en subscribers kunnen er meerdere zijn. Van het Event-systeem is er, althans wat de interacties rond minstens één welbepaalde soort Events betreft, slechts één.

Op een bepaald moment kan een subscriber het Event-systeem contacteren om zijn interesse te uiten in een welbepaald Event. Vanaf dan is het zo dat, elke keer dit Event voorkomt, de subscriber hiervan op de hoogte zal worden gebracht. Dit laatste gebeurt wanneer een publisher op zijn beurt het Event-systeem inlicht over een door hem gegenereerd event. Dit laatste wordt ook wel ‘to fire an Event’ genoemd: een Event wordt als het ware afgevuurd. De subscribers hebben dan de verantwoordelijkheid om het Event af te handelen (to ‘handle an Event’). Naast inschrijven, is het voor subscribers ook mogelijk zich opnieuw uit te schrijven voor een Event, een beetje zoals met een nieuwsbrief.

Voor het publish/subscribe mechanisme is het echter niet altijd expliciet vereist om de Events te reïficeren (er een echt object van te maken). Men kan ook gebruikmaken van het principe van callbacks: de subscriber geeft aan wat hij wil dat er gebeurt indien een bepaalde gebeurtenis voorkomt. De publisher kan dan via een centraal systeem aangeven dat iets gebeurd is, zonder verdere details te geven, en de callback wordt door het systeem uitgevoerd.

Daarnaast is het zelfs mogelijk dat er geen centraal systeem is, maar dat subscribers zich rechtstreeks tot publishers richten, zowel met als zonder gereïficeerde Events.



1. Subscribers onderschrijven zich
2. Publisher publiceert een Event
3. Event komt terecht in het Event-systeem
4. Event wordt verspreid via Event-systeem
5. Event bereikt alle Subscribers

Fig. 4: Het publish / subscribe scenario, gebruik makende van Events

Wanneer we de Events echter expliciet modelleren, dan biedt dit mechanisme veel meer opties: publishers kunnen het Event dan verrijken met details over het gebeurde, en deze gegevens kunnen door de subscribers worden gebruikt om het Event beter af te handelen. Tevens is het Event ook onmiddellijk een contract tussen zender en ontvanger, dat onafhankelijk een interface kan bepalen via dewelke de communicatie zal gebeuren. Ten slotte hebben we de optie om het Event voor later hergebruik op te slaan.

Daarnaast krijgen we door het invoeren van een centraal Event-systeem een vorm van indirectie, die toelaat dat publishers en subscribers elkaar niet meer hoeven te kennen.

2. EVENT DRIVEN ARCHITECTURE

Binnen wat we noemen een Event Driven Architecture (EDA), vinden we ook publish/subscribe terug, maar dan op een veel grotere schaal. Meestal gaat het nu om communicatie tussen verschillende applicaties en systemen op een zo robuust mogelijke manier. Het Event-systeem zal nu worden geïmplementeerd a.d.h.v. een *Bus*-systeem, en we zullen het de Event Bus noemen. Technisch kan dit worden geïmplementeerd via een Enterprise Service Bus (ESB) die dit ondersteunt.

EDA is heel nuttig wanneer men een groot aantal applicaties heeft, die dezelfde of gerelateerde Business Events behandelen. Gebeurtenissen binnen één applicatie zullen daarbij vaak relevant zijn voor andere applicaties. De Event Bus zal binnen een dergelijk geheel de verantwoordelijkheid op zich nemen om een Event, door één applicatie gecreëerd, tot bij alle geïnteresseerde applicaties te krijgen. De applicaties hoeven van elkaars bestaan niet af te weten: het is voldoende dat ze van het bestaan van bepaalde Events afweten, en dat ze kunnen communiceren met de Event Bus.

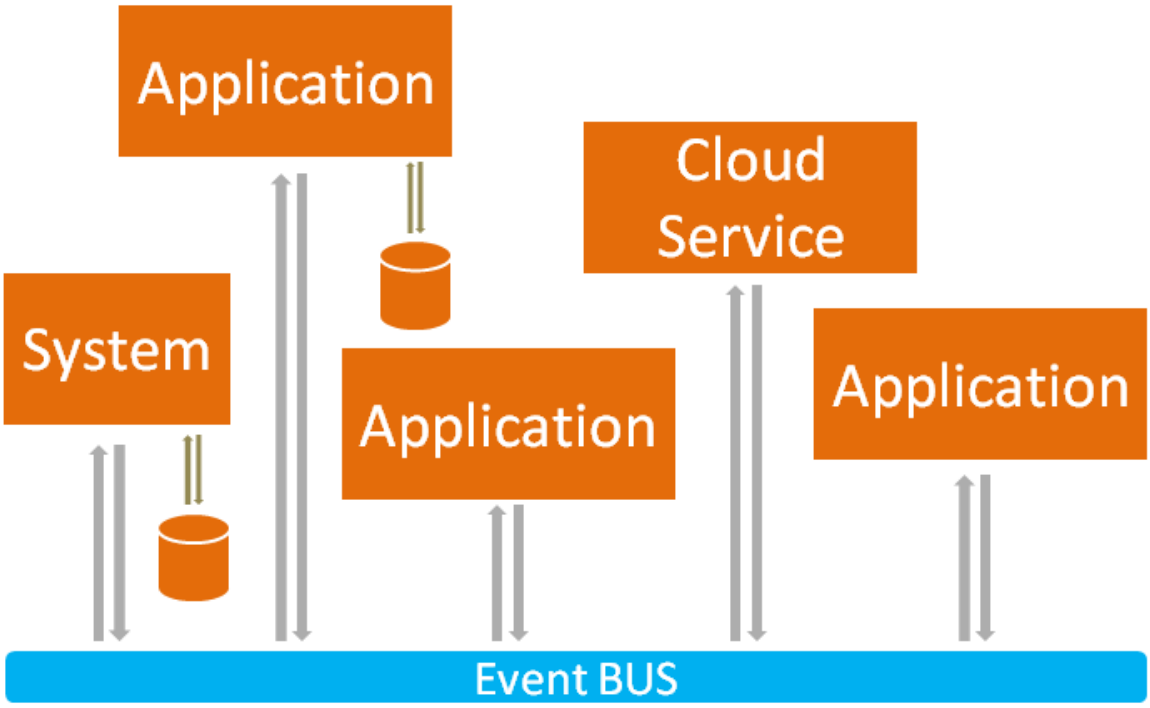


Fig. 5: Event Driven Architecture: de verschillende systemen publiceren en/of onderschrijven zich voor Events, die via de Event Bus naar de juiste bestemming worden gebracht wanneer ze zich voordoen

Deze vorm van architectuur laat een grote loskoppeling toe van de betrokken systemen en applicaties, en zorgt er ook voor dat er snel gereageerd kan worden op business Events, zelfs op plaatsen waar, en op manieren waarop, dit niet initiëel werd voorzien. Men kan zich namelijk inbeelden dat men, eens een bepaald Event bestaat en men weet dat het over de bus passeert, nieuwe ideeën kan krijgen om deze Events aan te wenden en erop te reageren, zowel in bestaande als in nieuwe applicaties.

3. EDA vs REST

Communicatie via Events kan niet altijd de enige communicatie zijn die er tussen verschillende systemen bestaat: soms heeft één systeem ook historische informatie nodig, die reeds lang via een andere applicatie is verwerkt en opgeslagen. Het eventuele Event, indien het al zou bestaan, dat tot die informatie heeft geleid, is op dat moment natuurlijk reeds lang voorbij.

Vaak gaat dit echter om read-only operaties. Hier moeten we dus, binnen het geheel van gedistribueerde applicaties, nog iets voor voorzien. Communicatie die schrijfoperaties veroorzaakt, kan echter – in principe – altijd via Events worden geïmplementeerd (wanneer er een gegeven moet worden aangepast, kan men deze wijziging op zich óók modelleren als een gebeurtenis).

Via *RESTful services* kan men in principe alle mogelijke communicatie tussen applicaties laten plaatsvinden (REST = REpresentational State transfer). Meer over REST-technologie, een vorm van Service Oriented Architecture (SOA), kan men terugvinden in ander onderzoek[3].

Men kan zich dan afvragen of hetzij REST, hetzij EDA, niet redundant is, of welke van de twee architecturen nu de beste oplossing is? Het antwoord is dat beide oplossingen hun plaats hebben in een gedistribueerd ecosysteem. Events werken namelijk typisch asynchroon, terwijl REST zowel asynchroon als synchroon kan worden gebruikt. Dit betekent dat een applicatie onmiddellijk op de hoogte kan worden gebracht, indien er een voor haar interessant Event beschikbaar is. Indien de applicatie echter meer data nodig heeft, die zich niet in een beschikbaar huidig Event bevindt, dan kan het deze gaan opvragen d.m.v. het gebruik van een RESTful service. Het besluit is dus dat we **Events kunnen gebruiken om nieuwe gegevens zo snel mogelijk over het netwerk te verspreiden, naar alle belanghebbenden, en dat we RESTful services kunnen gebruiken om reeds gekende informatie universeel ter beschikking te stellen op het netwerk**, waar alle geïnteresseerden ze kunnen gaan raadplegen. Een mooi complementair geheel dus en het goede nieuws is dat ook REST, en dus de beide benaderingen voor gedistribueerde communicatie, ondersteund kunnen worden door dezelfde onderliggende middlewaretechnologie (de ESB).

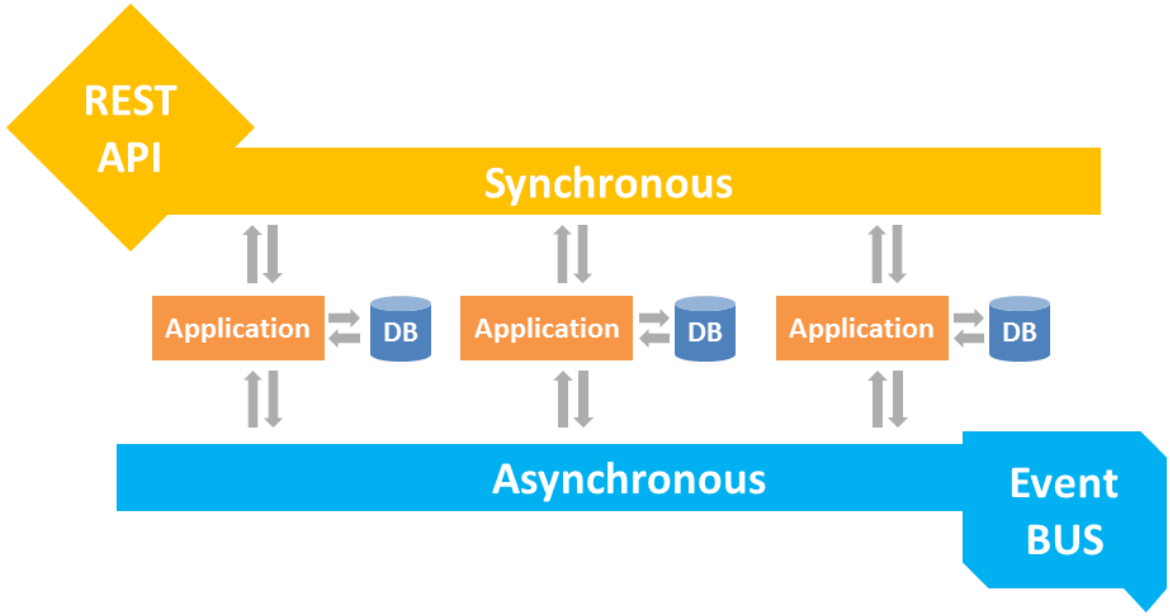


Fig. 6: Communicatie in een moderne gedistribueerde architectuur: een combinatie van een (optioneel) synchroon REST-kanaal en een asynchroon Event-kanaal.

[3] B.v. op de blog van Onderzoek: https://www.smalsresearch.be/data-centric-it-met-rest/

4. EVENT SOURCING

De meeste webapplicaties die we kennen hebben één of andere toestand die evolueert doorheen de tijd en die doorgaans opgeslagen wordt in een achterliggende database. Wanneer we b.v. inloggen op online banking, zien we het huidige saldo van onze rekeningen. Wanneer we morgen geld van onze rekening halen, dan zal deze toestand overmorgen zijn veranderd.

Wanneer we even afstand nemen van deze gang van zaken, dan kunnen we de bedenking maken dat elke huidige toestand in feite het gevolg is van een hele reeks van historische toestandsveranderingen, startende bij één of andere begintoestand, wanneer de applicatie voor het eerst online ging.

Welnu, elke toestandsverandering binnen een systeem kunnen we perfect modelleren als een Event. Wanneer we dit daadwerkelijk doen, dan kunnen we het computersysteem op zo'n manier laten werken, dat de huidige toestand "berekend wordt" op basis van alle in chronologische volgorde gezette toestandsveranderingen, die we als Events in de applicatie laten binnenkomen. Wanneer we daarnaast ook nog deze Events opslaan – i.p.v. de huidige toestand – dan spreken we van "Event Sourcing". Dit wil dus letterlijk zeggen dat de broncode van de applicatie gebaseerd is op Events.

Een voorbeeld: stel dat we een applicatie ontwikkelen voor het spelen van een schaakspel (zie Figuur 7). De toestand van het schaakbord is daarbij in principe het belangrijkste data-element: op basis daarvan kunnen de spelers hun volgende zet bedenken, en kan het spel berekenen of het spel zich in een bepaalde toestand bevindt (b.v. 'schaak' of 'schaakmat'). De toestand van het spel wordt verder nog aangevuld door te bewaren wie er aan beurt is.

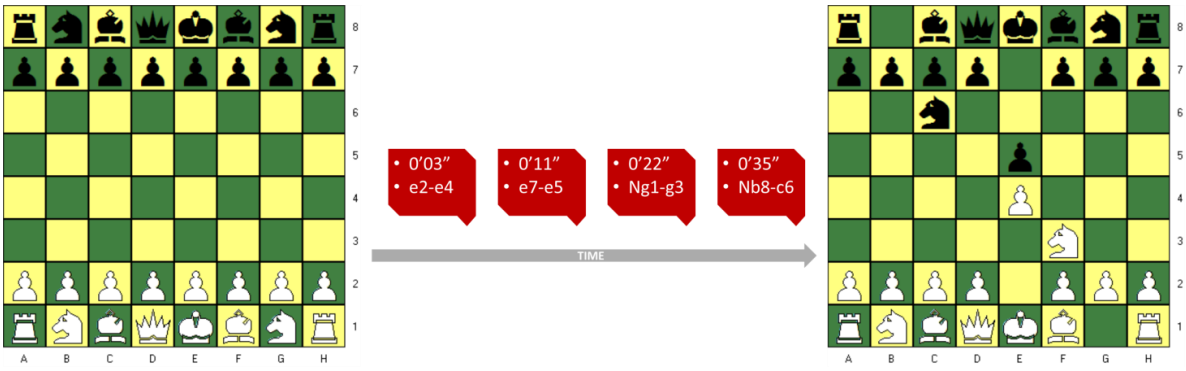


Fig. 7: Een voorbeeld: Event Sourcing in het schaakspel: de huidige toestand van het spel wordt bekomen door de opeenvolgende zetten toe te passen op het bord in de begintoestand. Door de zetten op te slaan, in plaats van enkel het huidige bord, krijgen we een veel rijker inzicht in hoe dit spel is geëvolueerd.

Wanneer we nu Event Sourcing toepassen op deze applicatie, identificeren we elke zet als een belangrijk business Event, en zullen we dus, i.p.v. het bord, elke zet afzonderlijk opslaan. De toestand van het bord kunnen we op elke moment reconstrueren door deze zetten uit te voeren (en we kunnen daarbij ook bepalen wie er aan beurt is), dus we verliezen geen functionaliteit. Wat we winnen is historische informatie over het spel, die we op een efficiënte manier kunnen opslaan. Een bedenking die snel gemaakt kan worden, is natuurlijk dat dit na verloop van tijd zeer onefficiënt zou worden: hoe meer Events er zijn, hoe zwaarder de berekening wordt om de huidige toestand te bekomen. Om

die reden zullen er uiteraard steeds bepaalde optimalisaties aanwezig zijn in de applicatie, zonder wezenlijk de Event-Sourced eigenschappen ervan te veranderen. Zo is het natuurlijk zaak om de huidige toestand in een cache of buffer te houden, zodat enkel nieuwe Events in rekening gebracht moeten worden om de toestand te laten evolueren. Daarnaast is het ook handig om van deze toestand af en toe een back-up weg te schrijven, zodat ook bij het eventuele uitvallen van de toepassing, slechts een deel van de toestand opnieuw berekend dient te worden aan de hand van Events.

Met het bewaren van Events in plaats van enkel de huidige toestand, krijgt men een aantal interessante mogelijkheden. Zoals in de vorige paragraaf gezegd, kunnen we de toestand terug heropbouwen. Sterker nog: we kunnen de toestand van eender welk moment in de geschiedenis van de applicatie terugbrengen (handig om te weten wat er op een bepaald moment aan de hand was). Daarnaast kunnen we nieuwe versies van de applicatie testen door Events opnieuw af te laten spelen voor een testversie: zo krijgt men erg realistische tests. Ten slotte laat de rijkheid aan gegevens binnen de Events ons toe om temporele queries te gaan uitvoeren; dit soort data zouden we bij een standaard aanpak vaak kwijt zijn.

Daarnaast zijn er een aantal speciale vormen van temporele gegevens die automatisch beschikbaar zijn, indien men Event Sourcing goed aanpakt. Zo kunnen Events vrij rechtstreeks worden gelogd en op die manier deel gaan uitmaken van een rijkere applicatielogging. Ook audit logs kunnen vaak (deels) uit de Events worden afgeleid. Ten slotte vormen de Events van een Event-Sourced applicatie automatisch ook een goede back-up van de gegevens, aangezien men de toestand van de applicatie ermee kan heropbouwen.

5. HET MODELLEREN EN BEHEREN VAN EVENTS

Dankzij opslag van historische Events kunnen we vaak na verwerking nog andere nuttige zaken doen (zie sectie 6). Bovendien kunnen we Events soms hergebruiken binnen andere applicaties dan oorspronkelijk voorzien. Maar een Event dat bruikbaar is door meerdere applicaties en systemen, moet ook voor die verschillende partijen interessante informatie bevatten en een goede structuur en interface voorzien. Bijgevolg staat of valt EDSE met het goed gemodelleerd zijn van Events.

Om hergebruik van Events mogelijk te maken, moeten ze op het juiste niveau van abstractie en granulariteit gedefiniëerd zijn. Dit legt een grote verantwoordelijkheid bij de analisten (zowel applicatief als business) om de juiste business Events te identificeren en te modelleren, en de juiste plaats – of plaatsen – binnen het geheel aan services en applicaties te kiezen, waar ze zullen worden gegenereerd.

Bovendien zullen de Events, eens gedefiniëerd, ook vindbaar moeten zijn, en zal men de definitie moeten kunnen raadplegen. Hiervoor is een goed gecentraliseerd beheer nodig, waarbij alle Events staan opgelijst in een glossarium. Hierin moet ook op een gemakkelijke manier kunnen worden gezocht naar Events. De informatie in het Event glossarium zal zowel voor de business mensen en analisten, als voor de architecten en developers nuttig moeten zijn. Er moet in worden beschreven welke informatie precies wordt

opgenomen in het Event wanneer het zal worden afgevuurd, en op welke manier men er toegang toe kan krijgen (zowel technisch als wat autorisatie betreft).

6. COMPLEX EVENT PROCESSING EN ANALYTICS

De Event Driven Architecture is veelbelovend: ze laat (samen met REST) een sterke loskoppeling toe van de betrokken applicaties en systemen, wat het onderhoud van het softwarepark een stuk eenvoudiger maakt. Maar indien we de Events die op de bus binnenkomen permanent gaan bewaren en voor nog allerlei andere zaken gaan (her-)gebruiken, ontketenen we pas echt de kracht van Event Driven Engineering.

Complex Event Processing (CEP) systemen zijn een belangrijke schakel wanneer men applicaties tijdige intelligente beslissingen wil laten nemen op basis van wat er aan het gebeuren is. Dit soort systemen zal naar binnenkomende Events luisteren (net zoals andere applicaties die zich ervoor inschrijven) en zal specifiek letten op het voorkomen van bepaalde *patronen* in Events die zich afspelen, gebaseerd op een aantal voorgedefiniëerde regels. Wanneer het CEP-systeem zo een opeenvolging van Events heeft opgemerkt, zal het reageren. Dit kan het doen door zelf een nieuw, ‘complex’ Event te publiceren, waar weer andere systemen op zullen kunnen reageren. Op die manier transformeren zulke systemen een bepaalde reeks van elkaar opvolgende Events dus in een nieuw, complex Event; dit alles op basis van een set van complexe regels, die door programmeurs, maar vaak ook door een ‘(zelf-)lerend’ systeem, zijn ingegeven.

Op die manier zijn CEP-systemen ook belangrijk in Big Data Analytics, waarbij de Big Data bestaat uit zaken die doorheen de tijd gebeuren (plus uiteraard andere vormen van verwante data). Deze vorm van Big Data is bijvoorbeeld interessant bij het detecteren van fraudepatronen, waarbij fraudeurs gedurende een bepaalde tijd een aantal handelingen uitvoeren die uiteindelijk tot fraude leiden. Het herkennen en vervolgens detecteren van de juiste patronen in een hele reeks van handelingen (waar uiteraard ook niet-verdachte handelingen en zelfs niet-gerelateerde handelingen tussen zitten), is daarbij cruciaal.

CEP laat in deze use case toe de resultaten van Analytics - de herkende patronen van zaken die gebeuren doorheen de tijd - rechtstreeks toe te passen op een systeem dat live is, waardoor ze in real-time kunnen worden gedetecteerd en er dus veel sneller op kan worden gereageerd. Daarnaast genereren de CEP-systemen nieuwe, complexe Events, die de reeds aanwezige Events in de het “Big Data”-systeem kunnen verrijken. Hierdoor vergroot de hoeveelheid data waaruit de analytische applicaties zullen kunnen putten om weer nieuwe Event-patronen aan te leren.

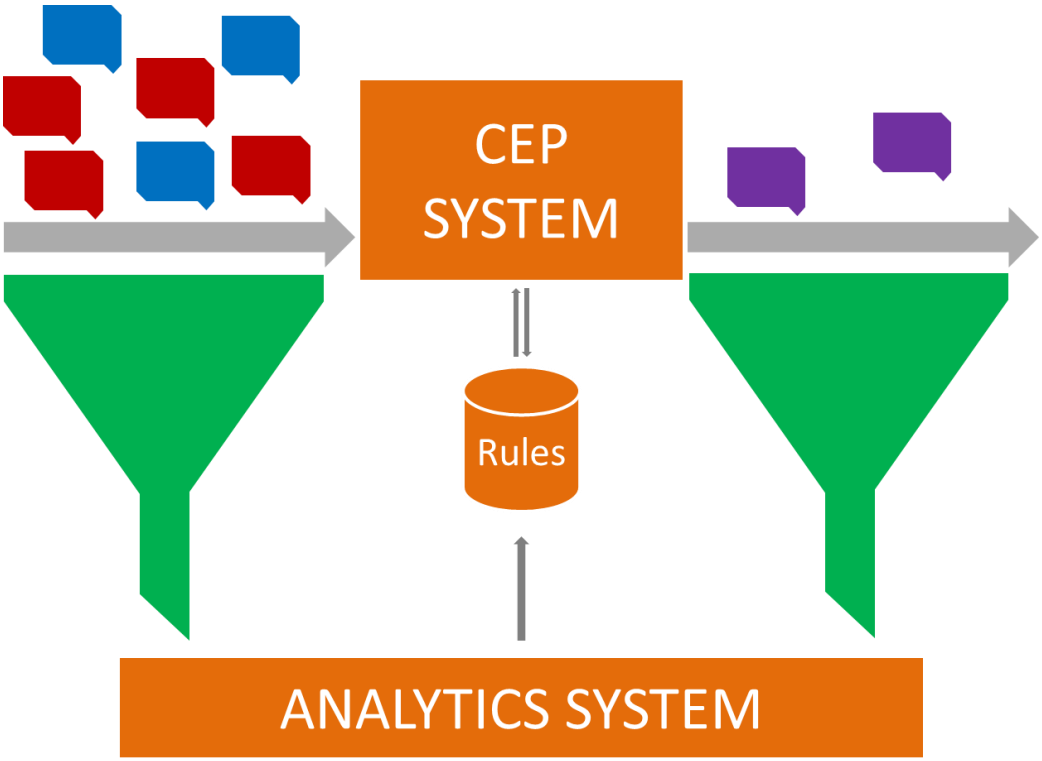


Fig. 8: Complex Event Processing: patronen van eenvoudige Events worden herkend op basis van regels en leiden tot 'complexe Events' (paars in de figuur). Alle Events samen kunnen via Analytics leiden tot nieuwe regels voor patroonherkenning.

7. CASE STUDY: EVENTS IN BUSINESS PROCESS & CASE MANAGEMENT

Binnen de ruimere context van dossierbeheer (Case Management)[4], dat kan gezien worden als een globale oplossing voor het beheer van taken, processen, documenten, en klantgegevens, onderscheiden we twee naar elkaar toegroeiende technologieën om werkprocessen te gaan beheren. Het gaat hier om Business Process Management Suites (BPMS) en (Advanced of Adaptive) Case Management (ACM) Systemen. Het doel van dergelijke systemen is het ondersteunen van bepaalde eenheden van het werk van de businessgebruikers, via allerlei handelingen en transformaties. Dit gebeurt door het behandelen van documenten, het raadplegen van gegevens, en vooral ook door het uitvoeren van zogenaamde 'taken', die eventueel een werkproces of dossier van toestand zullen doen veranderen zodat deze bijvoorbeeld van 'nieuw' op termijn kan evolueren naar 'afgehandeld'.

Het verschil tussen beide oplossingen zit hem in de manier waarop het werk wordt beschouwd: BPMS'en zullen het werk eerder als een min of meer vastgelegd proces gaan beschouwen, waar, stap voor stap, evenwel soms via verschillende mogelijke paden, taken in worden afgehandeld. ACM-systemen zullen het werk eerder als een openstaand dossier beschouwen, waarbij steeds verschillende handelingen mogelijk zijn en de volgorde van die

[4] Zie de blog over Case Management: https://www.smalsresearch.be/case-management-een-beetje-van-alles/

handelingen (het proces) minder streng vast ligt op voorhand. De gebruikers van het systeem zullen daarbij dus meer vrijheid krijgen. Uiteraard zijn er binnen BPMS manieren om de eindgebruikers op een flexibelere manier keuzes te laten, en zijn er ook bij ACM mogelijkheden om structuur te brengen in het werk. Vandaar dat in de huidige state of the art beide types systemen naar elkaar toegroeien, en de belangrijkste aanbieders van dergelijke oplossingen bieden momenteel reeds hybride systemen aan, die beide principes ondersteunen, en vaak zelfs vloeiend in elkaar laten overgaan.

Eén van de belangrijkste doelstellingen van BMPS- en ACM-systemen, is ervoor te zorgen dat taken en processen niet hardgecodeerd worden gebouwd in de applicatie(s). Men zal de businessprocessen en -regels dus eerder gaan modelleren in een taal die kan worden geïnterpreteerd door respectievelijke process en business rules *engines*. Deze aanpak leidt op een natuurlijke manier richting een *modulair* systeem: men zal de motor(en) hebben, een taakbeheer- en een documentbeheersysteem, verschillende applicaties voor eindgebruikers waarin ze hun taken kunnen uitvoeren of de toestand van processen en dossiers kunnen opvolgen, andere applicaties die ondersteunende diensten aanbieden of die niet-menselijke taken zullen uitvoeren, en verder nog applicaties die de onderliggende data van het volledige systeem zullen kunnen consumeren en aan de hand daarvan rapporteren. Een belangrijk deel van de communicatie tussen al deze componenten kan worden opgevangen via Events (zie Figuur 9).

Moderne BPMS- en ACM-systemen hebben EDSE al omarmd: of deze producten nu monolithisch zijn (of slechts zo overkomen door de wijze waarop ze zich tonen aan de gebruiker), of ze profileren zich expliciet als modulair, intern zullen ze meestal van Events gebruikmaken om aan te geven dat b.v. een taak is afgewerkt, of een dossier of proces van toestand is veranderd. De motor van deze systemen kan met deze Events dan rekening houden om nieuwe taken te creëren of de toestand van verschillende entiteiten in het systeem te laten evolueren.

Elke belangrijke gebeurtenis of verandering in toestand (op businessniveau) leidt bij deze manier van werken tot het genereren van een Event. Dit Event wordt op de bus geplaatst en alle geïnteresseerde systemen worden ervan op de hoogte gebracht. Het grote voordeel van Events is dat de publisher van een Event (het systeem dat het event genereert) en de subscribers op een Event (de systemen die op de hoogte worden gebracht) elkaar niet hoeven te kennen. Dit laat een enorme modulariteit en flexibiliteit toe, tot op het business niveau.

Wanneer we dit opentrekken naar integratie en hergebruik worden de mogelijkheden nog ruimer: Events, zowel van het BPMS/ACM-systeem, als van daarbuiten, kunnen worden geïntegreerd in de werkprocessen en dossiers. Events, eveneens zowel van het BPMS/ACM-systeem, als van daarbuiten, kunnen ook door andere applicaties worden opgevangen, en kunnen daarbovenop ook worden opgeslagen, zodat men ze achteraf kan gaan analyseren. Zo kan men b.v. interessante statistieken gaan trekken over de historische afhandeling van werkprocessen op basis van deze Events. En uiteraard kunnen ze ook worden hergebruikt binnen het CEP/Analytics-systeem.

Hoe meer systemen binnen een dergelijk business ecosysteem aangesloten worden op de Event Bus, hoe groter het **voordeel van hergebruik van Events** wordt: meer en meer business Events komen zo in aanmerking voor een rijkere analyse, en bovendien kan het ook zijn dat andere, nieuwe applicaties ontstaan, door gebruik te maken van een reeds bestaande soort Event dat eerder werd ontwikkeld voor een oudere applicatie (dit principe werkt zoals de open api's en open data op het internet, waardoor vele start-ups reeds succesvolle nieuwe toepassingen hebben kunnen lanceren).

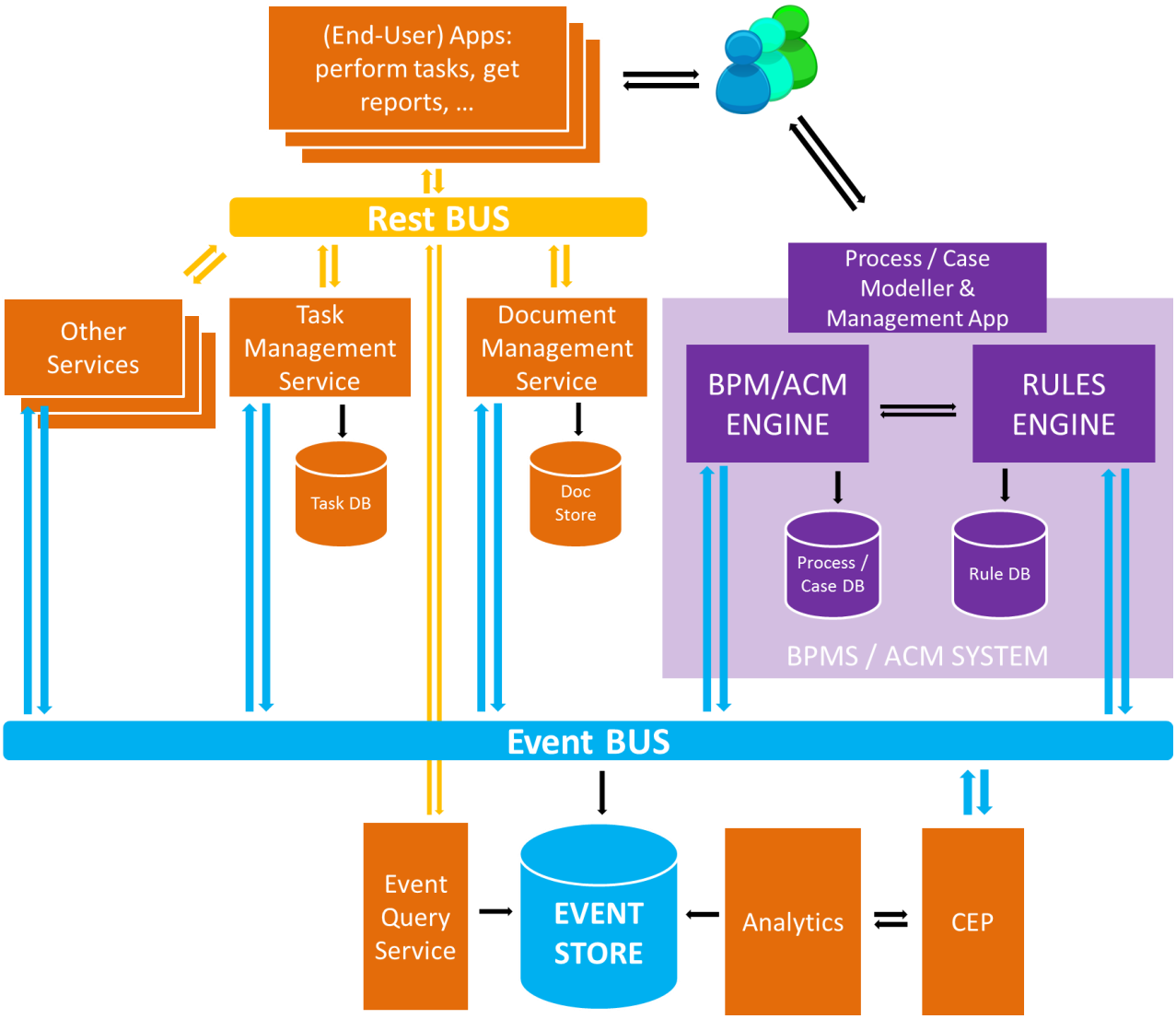


Fig. 9: Een voorbeeld van een modulair stelsel van applicaties en diensten waarbinnen process en case management een centrale en orchestrerende rol spelen. De paarse delen maken typisch deel uit van een BPMS / ACM-systeem, maar ook een aantal van de andere (oranje) zaken zijn vaak in degelijke suites opgenomen.

8. CONCLUSIE

In deze research note werd getracht om het belang van en de mogelijkheden geboden door ‘Event Driven Software Engineering’ (EDSE) en de ‘Event Driven Architecture’ (EDA) aan te tonen en de rol die Events kunnen spelen bij het opzetten van performante, betrouwbare en onderhoudbare informatiesystemen.

Applicaties en services kunnen los aan elkaar gekoppeld worden en worden toch in real-time op de hoogte gehouden van wijzigingen zonder op elkaar te hoeven wachten. In de nota werd ook aangetoond dat EDSE en REST perfect complementair kunnen zijn en elk hun eigen belang hebben in een goed uitgewerkte architectuur.

Daarnaast zijn Events op zich ook een erg krachtig paradigma als bouwsteen voor sommige individuele applicaties, of als gegevensbron voor Big Data Analytics en real-time intelligente systemen, zoals deze voor Complex Event Processing (CEP). Tevens komen Events van pas bij het heropbouwen van een oudere toestand, of voor het uitvoeren van realistische tests of als verrijking van de applicatieloggings.

In de context van Business Process Management of de modernere “Case Management”-systemen bieden Events een natuurlijke match om de verschillende toepassingen die bij deze systemen komen kijken aan elkaar te koppelen en met elkaar te laten communiceren. In combinatie met een Event Store, waar alle Events in worden opgeslagen, wordt dan meteen een solide basis gelegd voor Analytics en andere toepassingen die de opgeslagen Events kunnen hergebruiken.

Dit alles gaat natuurlijk niet zonder slag of stoot; de technologie om dit te realiseren is weliswaar beschikbaar, maar een goede modellering van de (Business) Events is onontbeerlijk en meteen de grootste uitdaging voor analisten en architecten.