

Esper als EDA-Plattform

Ist Event-driven Architecture (EDA) gefangen im Gartner Hypecycle oder gelingt mit Esper der Durchbruch für echte Businessintegration mit EDA jenseits des Gipfels der inflationären Erwartungen?

von Papick Garcia Taboada und Christian Dedek

Der Durchbruch von EDA in die Mainstream-Adaption wird von Gartner bereits seit Anfang des Jahrtausends für die Zeit jenseits von 2007 prognostiziert [1]. Der Artikel beleuchtet die Frage, ob mithilfe des frei verfügbaren Eventprozessors Esper der Gipfel der inflationären Erwartungen an Event-driven Architecture verlassen und ein Weg der betrieblichen Einführung skizziert werden kann. Dabei wird versucht, die im Rahmen der Implementierung eines Real Time Enterprise auftretende Schlüsselfrage der Integration mit existierenden Infrastrukturen (J(2) EE, ESB, leichtgewichtige Applicationcontainer etc.) zu beleuchten.

Event-driven-Architecture-Treiber

Im globalen ökonomischen Wettbewerb verschafft das frühe Erkennen von Marktbedürfnissen und Kostenpotenzialen sowie die folgerichtige Reaktion darauf entscheidende Vorteile. Beispiele wie Dell oder Google belegen, dass einstmals kleine Mitbewerber durch entschlossenes Ausnutzen dieser Wettbewerbsvorteile binnen weniger Jahre zu Global Playern heranwachsen können. Das von Gartner formulierte resultierende Ideal eines Unternehmens mit maximal automatisierten Geschäftsprozessen und minimalen Prozesslaufzeiten wird als Real Time Enterprise bezeichnet. Es stellt die betrieblichen

IT-Organisationen vor neue Herausforderungen. Nachdem durch SOA-Konzepte innerhalb der IT-Infrastrukturen die Basis für die Automatisierung von Geschäftsprozessen geschaffen wurde, gilt es nun, einen iterativen Regelkreis der Aktivitäten *Plan, Run, Measure* und *Optimize* zu etablieren [2]. Dies führt zur Notwendigkeit der Verarbeitung immer feinkörnigerer Informationen über Zustandsveränderungen in den Geschäftsprozessen, deren Menge und Qualität bisher nicht gekannte Größenordnungen erreichen können. Zusätzlich erhöht sich dabei die Zahl möglicher elektronischer Reaktionsschnittstellen der IT in den Geschäftsprozessen. Die beobachtbaren Änderungen im Zustand der Geschäftsprozesse können auch als Ereignisse modelliert werden. Die Herausforderung des RTE kann dann als Analyse der Korrelation von Ereignissen im so genannten Ereignistornado des Unternehmens beschrieben werden.

Klassische OLTP/OLAP-Architekturen sind den Anforderungen in diesem Ereignistornado nicht gewachsen [3]. Ereignisgesteuerte Architekturen scheinen durch folgende Grundprinzipien eine geeignete Antwort zu bieten:

- Lose Kopplung
- Nachrichtenbasiert
- Ortstransparenz
- Zentraler Nachrichtendienst
- Asynchrone Verarbeitung

EDA und SOA sollten orthogonal zur Implementierung des Real Time Enterprise kombiniert werden. Die SOA liefert mit ihren fachlichen, kombinierbaren, Consumer-unabhängigen Services die Bausteine der Geschäftsprozesse. Die auslösenden und resultierenden Ereignisse dagegen werden durch orthogonale EDA-Erweiterungen abgebildet. Diese Symbiose scheint vielen Herstellern und Visionären bereits Namen wie *SOA 2.0* oder *Next Generation SOA* wert zu sein [4].

EDA-Bausteine

Die Ereignisontologie ist die Basis der losen Kopplung innerhalb einer EDA. Ein Ereignis (Event) beschreibt eine aufgetretene Situation innerhalb eines Systems und ist deren Informationsträger und Dokumentation. EDA-Systeme sind Systeme, in denen aus Beobachtungen von speziellen Situationen entsprechende Ereignisse erzeugt, versendet und von Interessenten entgegengenommen werden. Diese Interessenten analysieren die Ereignisse und leiten daraus Aktionen ab. Aus diesem Modell der EDA lassen sich die folgenden Architekturkomponenten ableiten:

- *Event Sensor*: Beobachtet das System und erkennt die aufgetretene Situation, die zur Ereigniserzeugung führt.
- *Event Producer*: Generiert das Ereignis aus der Aktivierung durch den Sensor und versendet es dann auf den Event Channel.

- **Event Channel:** Transportiert Ereignisse zwischen Event Producern und Event Consumern. Die Zusammenfassung aller Event Channels eines Systems wird als Event Bus bezeichnet (Prinzip des zentralen Nachrichtendienstes).
- **Event Consumer:** Empfängt auf dem Event Channel publizierte Ereignisse.
- **Event Processor:** Verarbeitet die empfangenen Ereignisse. Die Formen der Verarbeitung variieren sehr stark, weshalb diese Komponente sehr unterschiedlich realisiert werden kann.

Die Anforderungen an einen solchen Prozessor werden stark durch die möglichen Formen der Ereignisverarbeitung beeinflusst.

Ereignisse können prinzipiell einzeln oder stromartig auftreten. Je nachdem, wie die Eventverarbeitung erfolgt, lassen sich in einer EDA grob drei Gruppen der Verarbeitung innerhalb des Prozessors bilden [5]:

- **SEP – Simple Event Processing:** Ein Ereignis kann als wichtige Zustandsänderung in einer Nachrichtenquelle angesehen werden. Diese Ereignisse lösen durch ihr Auftreten Prozesse in den Systemen aus, die die Nachricht empfangen werden. Das ist Verarbeiten von Nachrichten, wie es in der *Java-Messaging-Service*-Spezifikation beschrieben ist.
- **ESP – Stream Event Processing:** Die flussartige Bearbeitung der eingehenden Nachrichten. Typische Ereignisquellen sind zum Beispiel Sensoren, die eine große Anzahl an Ereignissen abfeuern. Bei der Abarbeitung der Nachrichten ist nicht jede einzelne Nachricht ausschlaggebend, sondern der Fluss als solcher. So wird zum Beispiel erst reagiert, wenn ein Temperatursensor für einen bestimmten Zeitraum durchgehend eine bestimmte Temperatur über- oder unterschritten hat. Ein weiteres Beispiel für *Event Stream Processing* ist die Verfolgung von Börsenkursen: Während eine einzelne Kursschwankung meist nicht aussagekräftig ist, sind Trends dennoch nach einigen Minuten (also nach dem Eintreffen mehrerer Kursereignisse) oder Stunden zu erkennen.

■ **CEP – Complex Event Processing:** Unter CEP versteht man die Erkennung von komplexen Mustern innerhalb eines oder sogar zwischen verschiedenen Ereignisströmen. So haben eventuell Ereignisse wenig Relevanz im Einzelnen, sind aber in bestimmten Zusammenhängen Indiz eines besonderen Ereignisses. Ereignisse können in kausaler, temporaler oder räumlicher Korrelation auftreten. Die einzelnen Ereignisströme können ein sehr hohes Aufkommen von Ereignissen aufweisen. CEP stellt hohe Anforderungen an die verarbeitenden Systeme und an die eingesetzten Beschreibungssprachen sowie letztendlich an die Domänenexperten, welche die Korrelationsregeln aufstellen.

Zur Umsetzung einer EDA stehen unterschiedlichste Implementierungsplattformen bereit. Dazu gehören Application Server, ESB, JMS Provider, Rule Engines, Business Process Engines, ESP/CEP Engines sowie BI- und BAM-Werkeuge. Um innerhalb der konkurrierenden und komplementierenden Plattformen für ein funktionierendes Wettbewerbsmodell zu sorgen, treten unterschiedliche Standardisierungsbemühungen wie das SOA-Reference-Modell, Common Event Infrastructure (CEI), Common Base Event (CBE), WS-Event Notification oder Java Business Integration (JBI) an [6]. Ob es dabei jemals zu einem standardisierten Enterprise Service Bus (ESB) kommt, der die Middleware-Produkte unterschiedlicher Hersteller vereinigt, ist nur schwer abzusehen. Technisch am meisten von der Standardisierungsdebatte getrieben sind im Moment neben der zentralen Ereignisontologie die Komponenten *Producer*, *Channel* und *Consumer*. Der Eventsensor ist nur durch seine Kopplung an einen konkreten Producer betroffen. Für einen Event Processor ist lediglich die Frage des Mappings zwischen standardisierter Ereignisontologie und interner Ereignisrepräsentation von Bedeutung. Es wird also geschlossen, dass die Auswahl eines konkreten Event Processors ohne Berücksichtigung der ungeklärten Standardisierungsdebatte rund um EDA getroffen werden darf. Besonders relevant ist daher die Frage,

welche der drei gesetzten Formen der Eventverarbeitung benötigt werden.

Esper als Event Processor

Esper ist gemäß den Vorüberlegungen ein Event Processor für CEP- und ESP-Anwendungen, der in Java als Esper und in .NET als Nesper unter dualer Lizenz bereitgestellt wird. Er ist frei im Sinne der GPL-Lizenz, kann aber auch kommerziell lizenziert werden. Die kommerzielle Vermarktung findet über die Firma EsperTech statt. Mit Esper wird die Entwicklung von Anwendungen mit hohem Aufkommen eingehender Ereignisse bzw. Nachrichten vereinfacht und beschleunigt. Dabei können Ereignisse auf verschiedene Arten in Echtzeit analysiert, gefiltert und konsumiert werden.

Am 17. Februar gab das Esper-Projekt die Version 2.0 frei. Neben Performanceoptimierungen wurden auch Änderungen in der API vorgenommen. Für diesen Artikel wurde aus zeittechnischen Gründen Esper 1.x verwendet. Am Beispiel der Engine Esper soll gezeigt werden, wie eine CEP-Eventbearbeitung in einer EDA stattfinden kann.

Aufbau

Als reine CEP Engine muss Esper so in eine Anwendung eingebettet werden, dass die Ereignisse an die Engine weitergegeben und entsprechende Aktionen ausgeführt werden können. Abbildung 1 zeigt die Bestandteile einer Esper-Laufzeitumgebung.

Das folgende einfache Beispiel zeigt, wie man in einem Java-Programm eine Esper Engine konfiguriert und startet.

```
Configuration config = new Configuration();

config.addEventTypeAlias("X", X.class);

EPServiceProvider serviceProvider =
    EPServiceProviderManager
    .getDefaultProvider(config);
```

Zuerst wird eine Instanz eines Konfigurationsobjekts erzeugt. In dem Beispiel sollen Instanzen einer Klasse X als Ereignisquelle dienen. Über die Konfiguration eines globalen Alias kann ein Name (zur späteren Verwendung in den Abfragen) für eingehende Ereignisse einer bestimmten Klasse definiert werden.

Mit der statischen Fabrikmethode `getDefaultProvider` der Klasse `EPServiceProviderManager` wird eine neue Instanz erzeugt und entsprechend konfiguriert. Die Instanz der Klasse `EPServiceProvider` wird im späteren Ablauf für das Empfangen von Ereignissen und für das Anmelden von Abfragen benötigt.

Events

Esper kennt drei verschiedene technische Repräsentationen für Nachrichten:

- POJOs (Plain Old Java Objects)
- Maps (aus der Java Collection API)
- Node (aus der W3C-XML-DOM-Spezifikation)

Gerade die Repräsentation als Java-Objekt vereinfacht die Einbindung in bereits vorhandene Anwendungen.

Im folgenden Beispiel wird gezeigt, wie eine Instanz der Klasse `X` als Ereignis der Esper Engine weitergereicht wird. Die Klasse `X` ist ein POJO und weiterhin nicht wichtig, sie dient lediglich als Stellvertreter für eine beliebige Klasse. Im vorherigen Codebeispiel wurde gezeigt, wie eine Instanz des `EPServiceProvider` erzeugt wurde. Über diesen `EPServiceProvider` bekommt man die Referenz auf die `EPRuntime`: Diese wird benötigt, wenn es darum geht, Ereignisse in die Esper Engine einzuspeisen.

```
EPRuntime runtime = serviceProvider.getEPRuntime();
```

```
for (int loop=0; loop<10; loop++) {
    runtime.sendEvent(new X(loop));
    Thread.sleep(300);
}
```

Dieses Beispiel zeigt die manuelle Einspeisung von Ereignissen. Mit `EsperIO` bietet Esper vordefinierte Adapter für CVS- und JMS-Nachrichtenquellen. Eine definierte Schnittstelle ermöglicht die Implementierung eigener Adapter.

Verarbeitung

Die Verarbeitung in einer CEP und ESP Engine entspricht der Umkehrung der Vorgehensweise, wie sie bei Datenbanken üblich ist. Bei relationalen Daten-

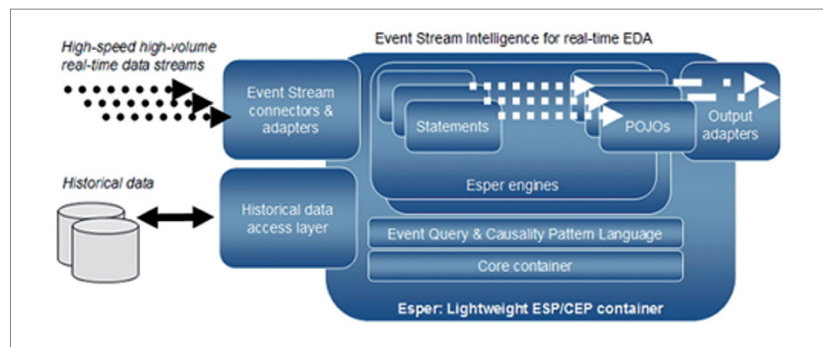


Abb. 1: Esper-Laufzeitumgebung (Quelle: EsperTech)

banken werden Abfragen gegen eine bestimmte Datenmenge geschickt. Bei Esper werden die eintreffenden Nachrichten gegen die Abfragen geschickt.

Durch diese Vorgehensweise entsteht eine kontinuierliche Bearbeitung des Nachrichtenflusses. Die Abfragen werden in einer speziellen Abfragesprache abgelegt. Ändert sich die durch eine Abfrage definierte Treffermenge, so wird die mit der Anfrage angegebene Aktion ausgeführt.

In Listing 1 wird gezeigt, wie eine Abfrage und die entsprechende Aktion in Esper implementiert wird. Mit dem ersten Befehl wird ein Statement in der Esper-Laufzeitumgebung erzeugt bzw. angemeldet. Hier handelt es sich um eine einfache Anfrage: Es wird ein Mengenfenster der Größe 5 definiert.

Mit dem zweiten Befehl wird ein `UpdateListener` erzeugt (in diesem Fall als anonyme innere Klasse) und mit dem Statement verknüpft. Diese Listener werden immer dann benachrichtigt, wenn sich die Treffermenge einer Abfrage geändert hat.

Abfragesprache

Esper führt eine eigene Abfragesprache namens EQL (Event Query Language) ein. Mit der Version 2.0 von Esper wurde diese in Event Processing Language (EPL) umgetauft. Diese Sprache ist sehr stark an SQL angelehnt. Folgende Analogien wurden gewählt:

- Ereignisströme entsprechen Tabellen
- Ein Ereignis entspricht einem Datensatz
- Eigenschaften im Ereignis entsprechen Datenfeldern

Für eine Abfrage in der Esper Engine wird immer eine Trefferliste angelegt. Sobald sich die Anzahl der Ereignisse verändert, sei es, weil neue Ereignisse in die Trefferliste aufgenommen wurden oder ältere entfernt wurden, wird eine Benachrichtigung über die Änderung des Zustands gemeldet. EQL-Abfragen lassen sich grob in zwei Kategorien einteilen: ESP- und CEP-Abfragen.

ESP-Abfragen

Bei ESP-Abfragen werden einzelne Ereignisse bis hin zu Zeit- und Mengenfenstern abgefragt. Bei der einfachen Abfrage

```
select * from Withdrawal
```

werden alle Ereignisse in die Ergebnismenge aufgenommen. An diesem Beispiel werden keine Ereignisse jemals verworfen, da keine Zeit- oder Mengenfenster definiert wurden. Ein Mengenfenster lässt sich durch folgende Abfrage erstellen:

Listing 1:

```
EPStatement statement = epAdministrator.createEQL("select * from
                                                    X.win:length(5)");

statement.addListener(new UpdateListener() {

    public void update(EventBean[] in, EventBean[] out) {

        for (int i=0; i<in.length; i++) {
            EventBean bean = in[i];
            System.out.println("in "+bean.getUnderlying());
        }
        if (out != null)
            for (int i=0; i<out.length; i++) {
                EventBean bean = out[i];
                System.out.println("out "+bean.getUnderlying());
            }
    }
});
```

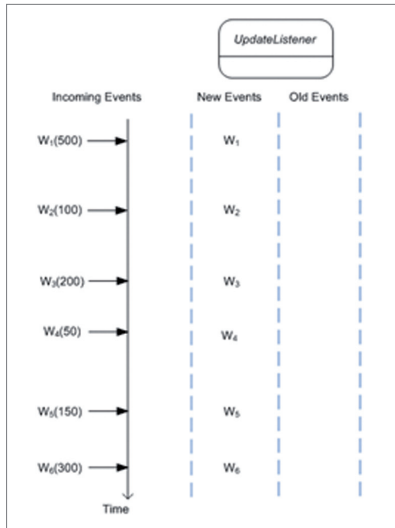


Abb. 2: ESP-Veranschaulichung

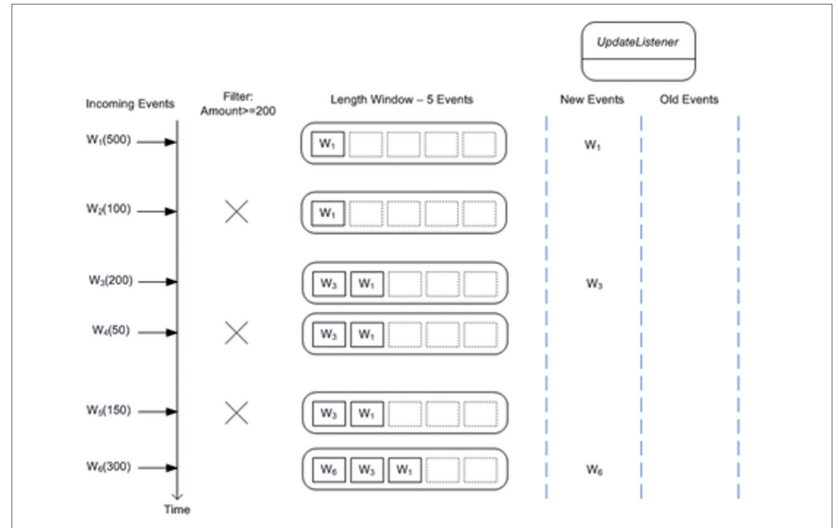


Abb. 3: ESP-Abfragen

```
select * from Withdrawal(amount >= 200).
win:length(5)
```

Mit dieser Abfrage werden nur die Abbuchungen in die Trefferliste aufgenommen, die einen größeren Betrag als 200 aufweisen. Des Weiteren wird ein Mengenfenster der Größe 5 definiert. Es werden folglich nur die letzten 5 Ereignisse in der Trefferliste behalten, sodass das Eintreten mehrerer Ereignisse dazu führt, dass ältere entfernt werden.

CEP-Abfragen

Bei CEP-Abfragen werden ein bis mehrere Ereignisströme untersucht. Über Mustererkennung versucht man besondere Ereignisse zu erkennen. Syntextechnisch führt die Mustererkennung zu einer größeren Abweichung von der SQL-Syntax. Bei EQL werden mustererkennende Abfragen mit dem Schlüsselwort *pattern* eingeleitet. Anhand des folgenden kurzen Ereignisstroms, dem Float-Operator *->* und der Wiederholungsanweisung *every* wird die Verwendung von der Mustererkennung in EQL demonstriert. Gegeben sei folgender Ereignisstrom:

```
A1 B1 C1 B2 A2 D1 A3 B3 E1 A4 F1 B4
```

Je nach Muster werden unterschiedliche Ergebnisse wie folgt erzielt:

```
pattern [ A -> every B ]
{A1,B1}, {A1,B2}, {A1,B3}, {A1,B4}

pattern [ every A -> every B ]
```

```
{A1, B1}, {A1, B2},
{A1, B3}, {A2, B3}, {A3, B3},
{A1, B4}, {A2, B4}, {A3, B4} und {A4, B4}
```

```
pattern [ every A -> B ]
{A1,B1}, {A2,B3}, {A3,B3}, {A4,B4}

pattern [ every ( A -> B ) ]
{A1,B1}, {A2,B3}, {A4,B4}
```

Die Angaben in der Tabelle zeigen nur Beispiele für eine Mustererkennung, sind aber keine gültigen EQL-Befehle. Zusätzlich muss noch angegeben werden, welche Information aus dem Treffer herangezogen werden soll. Im folgenden Abfragebeispiel werden beide Ereignisse selektiert, die zu dem Treffer geführt haben:

```
select a,b from pattern [ every a=A ->b=B ]
```

Die EQL-Abfragesprache kennt noch weitere Besonderheiten. In der Referenzdokumentation wird näher darauf eingegangen, hier nur einige Beispiele:

```
( A or B ) where timer:within ( 5 sec )
Ein A oder B in den nächsten 5 Sekunden

( every A ) where timer:within ( 10 sec )
Alle A Ereignisse in den nächsten 10 Sekunden

A -> timer:interval(10 seconds)
Nach A 10 Sekunden warten

every timer:at(5, *, *, *, *)
Alle 5 Minuten
```

Eine kurze Fallstudie

Auf Espers Homepage wird in einer kurzen Fallstudie „raw market data“ ver-

wendet. In dieser Fallstudie wird gezeigt, wie man mit Esper eine Durchsatzanalyse und ein Warnsystem einrichten kann.

Die Ereignisquelle

Der Ereignisstrom besteht aus einer Folge von Markt-Data-Events, die aus zwei verschiedenen Ereignisquellen (Feeds) stammen. Das Beispiel kennt zwei verschiedene Feeds, die als Java-Enumeration abgebildet werden. Jedes Event enthält folgende Informationen:

```
String symbol;
FeedEnum feed;
double bidPrice;
double askPrice;
```

In einem realen Anwendungsfall wäre die durch das FeedEnum implizierte Einschränkung kein angemessener Lösungsansatz. Als Java-Entwickler erkennt man hier die Möglichkeit, alle Sprach-Features von Java 5.0 einsetzen zu können.

Durchsatz ermitteln

Für die Durchsatzermittlung (Ticks per Second) wird eine EQL-Abfrage formuliert, die ein Zeitfenster von einer Sekunde definiert. Somit muss nur noch ermittelt werden, wie viele Ereignisse in dem Zeitfenster vorhanden sind:

```
insert into TicksPerSecond
select feed, count(*) as cnt
from MarketDataEvent.win:time_batch(1 sec)
group by feed
```

Mit einer EPL-Abfrage können die Ereignisse gruppiert (*group-by*-Anweisung analog zur SQL-Abfragesprache) und für einen gegebenen Zeitraum (*win:time_batch()*-Anweisung) gepuffert werden. Somit liefert die oben angegebene Anweisung ein Ereignis pro Feed: Jedes Ereignis wird die Information *feed* und *cnt* enthalten, wobei *cnt* wiederum als Alias für *count(*)* definiert wurde und für die Anzahl der Events im Zeitfenster steht. Damit diese Ereignisse für eine weitere Verarbeitung zur Verfügung stehen, werden sie in einen neuen Ereignisstrom namens *TicksPerSecond* eingespeist (*insert-into*-Anweisung). *MarketDataEvent* stellt die ursprüngliche Ereignisquelle dar.

Warnung bei deutlichem Abfallen des Datendurchsatzes

Nun wird eine Warnung ausgegeben, sobald der Datendurchsatz der letzten 10 Sekunden unter 75% der fortlaufenden Durchschnittsrate fällt. Bei Warnung soll ein Ereignis erzeugt werden, das das Feed, die durchschnittliche Rate und die Anzahl der Events in den letzten 10 Sekunden enthält.

Hierfür werden die im oberen Beispiel erzeugten *TicksPerSecond*-Ereignisse verwendet:

```
select feed, avg(cnt) as avgCnt, cnt as feedCnt
from TicksPerSecond.win:time(10 seconds)
group by feed
having cnt < avg(cnt) * 0.75
```

Dieses Beispiel zeigt, wie die Esper-Syntax bei der Verarbeitung von Ereignisströmen relativ komplizierte Zusammenhänge in einer starken Abfragesprache modellieren kann.

Fazit

Während nachrichtenbasierte Systeme keinen technologischen Paradigmenwechsel darstellen, wird dank EDA die asynchrone Verarbeitung von Nachrichten auf die Architekturperspektive angehoben. Die durch SOA vorangetriebene Verteilung auf der Architekturebene wird somit um die Asynchronität ergänzt, sodass beide Architekturmodelle sich eher ergänzen als ausschließen.

Leider werden bei Überlegungen im Zusammenhang mit EDA die Verarbeitungsmodelle ESP und CEP sehr oft als weniger entscheidend betrachtet. Es sind aber gerade diese Modelle, die im Problembereich des Real Time Enterprise entscheidenden Einfluss haben können. Während Simple Event Processing bereits in vielen Infrastrukturen wie J(2)EE zum Einsatz kommt, sind es die stromartige Ereignisverarbeitung und die Mustererkennung, die noch selten zu finden sind.

Dabei ist in der betrieblichen Praxis die fehlende Standardisierung ein nicht unerheblicher Grund. Der technischen Machbarkeit von ESP/CEP und den damit verbundenen Potenzialen im RTE werden die Kosten einer Implementierung entgegengehalten. Fehlende etablierte Standards führen häufig zu überhöhten Risikokostenbetrachtungen für den Fall einer möglichen Umsetzung basierend auf dem falschen Standard. Die Entscheidung für die Implementierung einer EDA mit ESP/CEP-Modellen wird daher oft in eine Zukunft mit visionären Standards projiziert. Nach dem Gipfel der inflationären Erwartungen an EDA droht nun das Tal der Desillusionierung im Gartner Technology Hypecycle.

Mit dem Projekt Esper steht eine Möglichkeit zur Verfügung, bei geringen

Einstiegskosten frühzeitig Erfahrungen mit ESP/CEP zu sammeln und die eigene EDA-Architekturvision in der Praxis zu überprüfen. Die (Risiko-)Kosten für die Integration des quelloffenen ESP/CEP-Prozessors sind dabei vergleichsweise günstig gegenüber den durch ihn gewonnenen Architekturpotenzialen.

Die von EsperTech demonstrierten Performanzkennzahlen sind nur schwer mit existierenden OLTP/OLAP-Systemen zu vergleichen. Deshalb läßt sich eine Validierung des erwarteten Leistungsumfangs im konkreten Einsatzfall des Unternehmens bisher kaum vermeiden [7].

Der Reifegrad der Dokumentation und die Stabilität der Software selbst erlauben die Realisierung und Untersuchung spezifischer Szenarien mit hoher Kritikalität im Betrieb noch innerhalb des Jahres 2008. Die frühzeitige Produktivsetzung solcher Szenarien kann einer IT-Organisation mit einer Strategie fern des Mainstreams die mit RTE anvisierten Wettbewerbsvorteile verschaffen [8].

Auch wenn sich EDA mit ESP und CEP niemals als die beliebteste Applikationsarchitektur durchsetzen mag, darf angenommen werden, dass sie in der Zukunft eine wichtige Rolle bei der Implementierung von IT-Infrastrukturen im Enterprise-Level einnehmen werden. ■



Papick G. Taboada ist seit 1997 freiberuflicher Softwarearchitekt und Technology Scout. Schwerpunkte seiner Arbeit liegen in der Konzeption und Erstellung von Softwarearchitekturen im Java-EE-Umfeld mit Open-Source-Technologien.



Christian Dedek ist CTO des Competence Centers der Orientation in Objects GmbH in Mannheim. Schwerpunkte seiner Arbeit bilden Qualitätsmanagement und die Softwarearchitektur bei Implementierungsprojekten unternehmenskritischer Anwendungen und Prozesse.

Links & Literatur

- [1] Technology Hypecycle 2006, Gartner (07/2006).
- [2] Zacharias, Roger : Eine perfekte Symbiose, in: Java Magazin 07.2007, S. 60-69.
- [3] Burleson, Don: Measuring Oracle transactions per second, 2007:
www.dba-oracle.com/M_transactions_per_second.htm
- [4] Luckham, David: Complex Event Processing: complexevents.com
- [5] Michelson, B.: Event-driven Architecture Overview, 2006:
www.psgroup.com/detail.aspx?id=681
- [6] soa.omg.org/SOA-docs/EDA-Standards.htm
- [7] Vasseur, A.: Esper Performance, 2007:
docs.codehaus.org/display/ESPER/Esper+performance
- [8] Pezzini/Govekar/Natis/Scott: Extreme Transaction Processing Technologies to Watch: www.gartner.com/DisplayDocument?doc_cd=146107