ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

PROVOZNĚ EKONOMICKÁ FAKULTA

KATEDRA INFORMAČNÍCH TECHNOLOGIÍ

DIPLOMOVÁ PRÁCE

Událostmi řízená architektura: aspekty tvorby software

Rostislav STŘÍBRNÝ

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Katedra informačních technologií

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Stříbrný Rostislav

Informatika

Název práce

Událostmi řízená architektura: aspekty tvorby software

Anglický název

Event-driven architecture: software development aspects

Cíle práce

Diplomová práce je tématicky zaměřena na problematiku událostmi řízená architektura (event-driven architecture, EDA). Hlavním cílem práce je charakterizovat technologické aspekty tvorby software dle konceptu událostmi řízené architektury. Dílčí cíle diplomové práce jsou:
- analyzovat přístupy k architekturám IT systémů,
- charakterizovat odlišné pohledy na EDA,
- charakterizovat vztah EDA k servisně orientované architektuře (SOA),
- navrhnout a implementovat vlastní řešení softwarové aplikace dle architektury EDA, a to simulací chování v prostředí obchodování na burze cenných papírů.

Metodika

Metodika řešené problematiky diplomové práce je založena na studiu a analýze odborných informačních zdrojů. Vlastní řešení je realizováno formou návrhu a implementace jednoduché softwarové aplikace podle zásad architektury EDA. Na základě syntézy teoretických poznatků a výsledků vlastního řešení budou formulovány závěry diplomové práce.

Harmonogram zpracování

1) Příprava a studium odborných informačních zdrojů, upřesnění dílčích cílů práce a volba postupu řešení: 7/2013
2) Zpracování přehledu řešení problematiky dle informačních zdrojů: 7/2013 - 9/2013
3) Vypracování analytické části práce, diskuze a zhodnocení výsledků: 10/2013 - 11/2013
4) Tvorba finálního dokumentu diplomové práce: 12/2013 - 2/2014
5) Odevzdání diplomové práce a teze: 3/2014

Rozsah textové části

60-80 stran

Klíčová slova

Event-driven architecture (EDA), Service-oriented architecture (SOA), Enterprise Application Integration (EAI), Enterprise Service Bus (ESB), Complex event processing (CEX)

Doporučené zdroje informací

1. CHANDY, K a W SCHULTE. Event processing: designing IT systems for agile companies. New York: McGraw Hill, c2010, xvii, 235 p. ISBN 00-716-3350-2.

2. CUMMINS, Fred. 2009. Building the agile enterprise with SOA, BPM and MBM. Amsterdam Boston : MK/OMG Press/ Elsevier, 2009. str. 306. ISBN: 978-0-12-374445-6

3. ERL, Thomas. SOA: servisně orientovaná architektura : kompletní průvodce. Vyd. 1. Překlad Ondřej Baše, Lukáš Krejčí. Brno: Computer Press, 2009, 671 s. ISBN 978-80-251-1886-3

4. EMMERICH, Wolfgang a Stefan TAI. Engineering distributed objects: Second International Workshop, EDO 2000, Davis, CA, USA, November 2-3, 2000 : revised paper. Vyd. 1. Překlad Ondřej Baše, Lukáš Krejčí. New York: Springer, c2001, viii, 270 p. ISBN 35-404-1792-3.

5. ETZION, Opher a Peter NIBLETT. Event processing in action: Second International Workshop, EDO 2000, Davis, CA, USA, November 2-3, 2000 : revised paper. Vyd. 1. Překlad Ondřej Baše, Lukáš Krejčí. Greenwich, 74° w. long.: Manning, c2011, xxiv, 360 p. ISBN 19-351-8221-8.

6. PULIER, Eric a Hugh TAYLOR. Understanding enterprise SOA: servisně orientovaná architektura : kompletní průvodce. Vyd. 1. Překlad Ondřej Baše, Lukáš Krejčí. London: Pearson Education [distributor], 2006, xxxvii, 242 p. ISBN 19-323-9459-1.

7. TAYLOR, Hugh. Event-driven architecture: how SOA enables the real-time enterprise. Upper Saddle River, NJ: Addison-Wesley, c2009, xviii, 308 p. ISBN 03-213-2211-8.

8. Open SOA Collaboration. Open SOA Collaboration. [Online] http://www.osoa.org

Vedoucí práce

Ulman Miloš, Ing., Ph.D.

Termín odevzdání

březen 2014

doc. Ing. Zdeněk Havlíček, CSc.
Vedoucí katedry

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE
Provozně ekonomická fakulta

prof. Ing. Jan Hron, DrSc., dr. h. c.
Děkan fakulty

V Praze dne 6.11.2013

Čestné prohlášení

Prohlašuji, že svou diplomovou práci „Událostmi řízená architektura: aspekty tvorby software“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 24.3.2014

Poděkování

Rád bych touto cestou poděkoval Ing. Miloši Ulmanovi, Ph.D. za jeho ochotu a rady, které mi poskytoval při tvorbě této práce. Dále pak chci poděkovat své nejbližší rodině, která mě během mého studia podporovala a která mi vždy dodávala nové síly. Především bych pak chtěl poděkovat své manželce Dáše Stříbrné, která mi byla po celou dobu studia neotřesitelnou oporou, a také své mamince Olze Stříbrné, která mě v nelehkých životních podmínkách svědomitě a dobře vychovala.

Událostmi řízená architektura: aspekty tvorby software

Event-driven architecture: software development aspects

Souhrn

Předložená práce studuje oblast informačních technologií, která souvisí s událostmi řízenou architekturou. Metodou studia je analýza a syntéza informací z uvedených pramenů. Práce poskytuje přehled o vzniku EDA včetně charakterizace návazností na vybrané architektonické přístupy. Uvádí definice a charakteristiky EDA, a to včetně užitých základní pojmů. Představuje využitelnost architektury, omezení a rizika spojená s implementací. Charakterizuje základní vnitřní součásti architektury. Věnuje se jednotlivým fázím a způsobům zavádění do praxe, a také obecným přístupům k návrhu IT systémů založených na EDA. Zaměřuje se na synergie se servisně orientovanou architekturou. Uvádí přehled technologií, které tvoří základ implementace EDA, a přehled vybraných produktů vhodných pro její realizaci. Vlastní práce se zaměřuje na návrh a implementaci ukázkové aplikace založené na principech událostmi řízené architektury. Je implementováno jednoduché algoritmické obchodování na akciovém trhu, sestávající z několika systémů a založené na zpracování dat v reálném čase. Výsledkem práce je ucelený náhled na událostmi řízenou architekturu, a potvrzení deklarovaných výhod užití.

Summary:

Following work studies area of information technologies which is related to event-driven architecture. Used study method includes analysis and synthesis of information from mentioned sources. Work provides overview of EDA creation including characterization of related architectonical approaches. It presents definition and characterization of EDA, including definition of used basic terms. It presents architecture usability as well as possible limitations and risks associated with the implementation. It characterizes the basic internal components of the architecture. It addresses various stages and methods of implementation in practice, as well as general approaches to the design of IT systems based on EDA. It focuses on synergies with service-oriented architecture. Provides an overview of the technologies that form the basis for the EDA implementation, and provides an overview of products suitable for its implementation. Respective work focuses on the design and implementation of simple example application based on the principles of event-driven architecture. Simple algorithmic trading in stock market is implemented, consisting of several systems and based on real-time data processing. Work outcome is comprehensive outlook on event-driven architecture and confirmation of declared usage benefits.

Klíčová slova: Event-driven architecture (EDA), Service-Oriented Architecture (SOA), Enterprise Application Integration (EAI), Enterprise Service Bus (ESB), Complex-event processing (CEP)

Keywords: Event-driven architecture (EDA), Service-Oriented Architecture (SOA), Enterprise Application Integration (EAI), Enterprise Service Bus (ESB), Complex-event processing (CEP)

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Při bližším zamyšlení se nad významem technologického pokroku se nejprve části z nás vybaví některá zajímavost z oblasti vědy anebo jednoduše některá z hi-tech novinek, která se kolem nás v poslední době objevila. Teprve až s dalším odstupem si začneme uvědomovat rozsah již dosaženého světového pokroku a také významné milníky lidstva, kterých již bylo dosaženo. Nakonec si vybavíme naše individuální vize a touhy, kam by lidstvo jako celek mohlo v budoucnu směřovat, anebo jednoduše výzvy, kterým čelíme již dnes.

Touha tvořit a objevovat je v nás pevně zakotvena a je univerzální vlastností nejenom v západní společnosti. Aby tato touha mohla být naplněna, je nezbytné přistoupit ke konkrétním krokům, které povedou k dosažení našich osobních cílů a vizí. Těmito kroky zajisté mohou být jak výzkum, vývoj či inovace, ale také oblast výroby a poskytování již existujících produktů či služeb. V dnešní době je nezbytná interdisciplinární spolupráce, jelikož již delší dobu není možné pojmout individuálně všechny dostupné informace ze všech oborů a zpracovávat je nezávisle. Stejně tak není možné uvědomovat si všechny případné souvislosti ať již mezi dostupnými informacemi v jednotlivých oborech anebo v hloubce jednotlivých témat. Všechny tyto životní situace představují události, které vyžadují vzájemnou interakci mezi lidmi. Pro efektivní fungování je tedy bezpochyby nutné, aby tato interdisciplinární spolupráce měla určitý řád. Tento řád může být tvořen oddělením jednotlivých činností - specializací, kategorizací jednotlivých druhů objevů - informací, a v neposlední řadě schopností se ve všech těchto datech orientovat, tedy vyhledávat v nich a tvořit mezi nimi vazby. Takto je například členěna již samotná věda, která se z pohledu oddělení činností dělí na jednotlivé obory. Publikováním a citováním výstupů vědecké činnosti je dosahováno jak cílů kategorizace jednotlivých informací, tak i tvorby příslušných vazeb mezi nimi. Uvedený příklad by šlo chápat jako jistou formu událostního prostředí, kde publikování představuje produkci jistých událostí, a kde citace představuje reakci na ně, tedy jejich konzumaci. Obdobným způsobem by šlo pojmout také mnoho dalších životních situací.

Tento model sdílení informací je jedním z fungujících způsobů jak provádět značně distribuovanou činnost, a to v celku efektivně. Podobné je to v oboru informačních

technologií (IT), kde je také nezbytná interakce mezi oddělenými jedinci, kteří mají omezenou množinu informací. Tentokrát ne však mezi lidmi, ale mezi stroji.

Obor informačních technologií je silně dynamická oblast, kde jen za posledních 10 let došlo k několika technologickým revolucím a zvratům. Navíc s každým rokem přibývá nespočet nových softwarových produktů a vylepšování těch stávajících. Všechny tyto softwarové produkty, či aplikace tvořené na míru, jsou do jisté míry důsledkem obecné snahy o uvedení nových inovativních služeb na trh a tedy odlišení se od konkurence.

Již delší dobu si silní hráči působící na trhu vývoje IT software uvědomují, že pro takto silně heterogenní prostředí je nezbytně nutné nastavit určitá pravidla. Tato pravidla musí umožnit především vzájemnou interakci mezi jednotlivými systémy všech výrobců. Toho bylo částečně dosaženo v některých oblastech IT, především v oblasti vydávání standardizovaných komunikačních specifikací, zavedením tvorby IS dle principů servisně orientované architektury (SOA) a v neposlední řadě širokou akceptací těchto principů ze strany producentů software i jejich zákazníků.

Jedním z dalších významných faktorů působících na schopnost podniku konkurovat je bezpochyby schopnost reakce na události vnějšího světa, a to nejlépe automatizovaným způsobem. Z tohoto důvodu se jednotlivé podniky v nejrůznějších oblastech života snaží vyvinout takové prostředky, které by nejenom usnadnily automatizovanou detekci probíhající události a vyvolání odpovídající reakce na ni, ale které by také umožnily řízení a rychlé nasazování potřebných změn v takto automatizovaných systémech.

Událostmi řízená architektura (EDA) představuje způsob budování softwarových systémů a prostředí, které umožňují detekci vnějších podnětů, jejich vhodnou transformaci a filtraci, a případně také vyvolání automatizované odpovídající reakce. V kombinaci se servisně orientovanou architekturou přináší tento způsob analýzy a návrhu informačních systémů přidanou hodnotu, která umožňuje podniku včasnou reakci v dnešním dynamickém ekonomickém prostředí.

2 Cíl práce a metodika

Cílem diplomové práce je co nejvíce přehledně a uceleně charakterizovat základní aspekty konceptu událostmi řízené architektury, a to především z pohledu tvorby software. Dílčími cíli jsou analýza různých přístupů k souvisejícím architekturám IT systémů, charakterizace odlišných pohledů na uvedený koncept a charakterizace vztahu k servisně orientované architektuře. Cílem praktické části práce je potom navrhnout a implementovat vlastní řešení softwarové aplikace dle architektury EDA, a to simulací chování v prostředí obchodování na burze cenných papírů.

Celkový proces implementace EDA lze pojmout z pohledu manažersko-obchodního, návrhu IT architektury a designu, anebo z pohledu užitých technologií. Existuje mnoho oblastí, ve kterých by se mohla událostmi řízená architektura uplatnit a pro které by mohl být proces implementace odlišný. Jako příklady lze uvést oblast řízení dopravy, zdravotnictví, krizové řízení, telekomunikace, oblasti bankovnictví, pojišťovnictví, a mnohé další. S ohledem na požadované cíle se tato práce věnuje primárně oblasti obecné charakteristiky IT architektury a designu, a vztahu k servisně orientované architektuře. Ostatní oblasti jsou zmíněny okrajově.

Koncept událostmi řízené architektury je v této práci popsán takovou formou, která přináší hrubý přehled, kam EDA ve světě IT zapadá, a základní rysy a požadavky, které musí být splněny při zavádění EDA do praxe. Z tohoto pohledu je tedy nezbytné začít nejprve uvedením základních požadavků na IT architekturu jako celek, a následně pokračovat konkrétněji k samotné událostmi řízené architektuře. Úvodem do tématu je základní charakteristika EDA a uvedení základních principů budování IT systémů dle tohoto přístupu. Dále jsou uvedeny různé možnosti využití a uvedena případná rizika, která je při budování EDA zapotřebí řešit.

Následuje uvedení různých pohledů na implementaci EDA. Nejprve je představena charakteristika základních vnitřních součástí, které tvoří jádro implementace typické událostmi řízené architektury. Poté následuje charakteristika zavádění EDA v podniku, kdy jsou zmíněny typické fáze zavádění, a uveden pohled věnující se oblasti návrhu architektury a designu.

Práce pokračuje uvedením vztahu EDA k servisně orientované architektuře. Pro základní pochopení tohoto vztahu jsou nejprve uvedeny základní definice a pojmy SOA, následované uvedením možných společných užití a synergických efektů obou architektur. Dostatečná pozornost je věnována také oblasti technologické. Především se zde jedná o základní charakteristiku vybraných technologií a produktů, které je možné použít při vývoji software dle principů EDA.

Následuje uvedení vlastní práce, jejímž cílem je implementace jednoduché funkcionality softwarové aplikace dle principů EDA. Vlastní práce zahrnuje návrh vhodné IT architektury, výběr softwarových produktů, návrh a implementaci jednotlivých částí ukázkové aplikace, a následné propojení těchto částí. Cílem je ověření deklarovaných vlastností událostmi řízené architektury.

Ukázková aplikace je realizována v prostředí obchodování na akciovém trhu, a to za užití vybrané strategie automatizovaného algoritmického obchodování. Cílem je vytvořit informační systém, který bude schopný definovaným způsobem reagovat na probíhající události v téměř reálném čase. Jako vstupní údaje budou preferovaně užity aktuální údaje získávané v reálném čase, a očekávaným výstupem bude odpovídající reakce systému na zvolený typ události. Celkově se tedy bude jednat o simulaci automatizovaného chování na základě předem definovaných pravidel.

Metodika řešené problematiky je založena na studiu a analýze odborných informačních zdrojů. Zvolená problematika pokrývá širokou oblast IT, je proto nutné ji nejprve rozčlenit na jednotlivé logické celky, a tyto celky dále pečlivě strukturovat. Posléze bude možné provést detailnější zkoumání jednotlivých odborných zdrojů s ohledem na aktuálně zpracovávanou oblast. V opačném případě by text nebyl dostatečně tematicky souvislý, a tedy by byl i hůře pochopitelný.

Pro prvotní uchopení tématu je možné se inspirovat rozčleněním, které používají autoři zde citovaných odborných publikací. Ve vztahu k servisně orientované architektuře je nutné poznamenat, že autor práce je z teoretického i praktického hlediska obeznámen s oblastmi implementace SOA konceptu v podnikové sféře, a věnoval se tomuto tématu také ve své bakalářské práci. Z tohoto důvodu práce vychází nejenom ze zmiňovaných odborných zdrojů, ale také ze zkušeností autora.

Při tvorbě bude použita analýza konceptu EDA jako celku, návaznost na členění v odborné literatuře a v neposlední řadě syntéza jednotlivých informací plynoucích z těchto pramenů.

Dále je nutné zdůraznit, že téma servisně orientované architektury je v odborné literatuře publikováno převážně v anglickém jazyce. Přitom dostupné české překlady, či přímo autorská díla, často anglické termíny nepoužívají jako slova přejatá, ale zavádí české novotvary. V praxi IT firem však nejsou tyto české překlady příliš používány a není tedy z pohledu autora této práce vhodné se jich striktně držet. Z tohoto důvodu je v textu preferována ta forma daného termínu či obratu, kde její význam je akceptovaný širokou odbornou veřejností. Takto zvolená forma termínu je potom v textu použita výhradně v jednotné podobě. Podpůrným argumentem pro tuto volbu je zároveň fakt, že práce vychází téměř výhradně z anglicky psané literatury.

Pro oblast událostmi řízené architektury je situace taková, že se české termíny užívají v praxi IT firem ještě méně než v případě servisně orientované architektury. Z tohoto důvodu bude práce při překladu do češtiny zmiňovat při prvním užití i originální znění, aby se minimalizovala možnost chybné interpretace.

Slovo podnik je v práci uvedeno v širším významu a vztahuje se na libovolné subjekty, které tvoří samostatnou organizační strukturu a mohou realizovat vlastní řešení v oblasti informačních technologií.

3 Přehled řešené problematiky

Událostmi řízená architektura (EDA) je koncept, který vznikl na základě potřeby inovovat stávající architektonické přístupy v informačních technologiích. Následující kapitoly tento koncept představují v konkrétních souvislostech a poskytují jeho bližší charakteristiku.

3.1 Přístupy k architektuře informačních systémů

Informační technologie slouží pro potřeby reálného světa a z tohoto důvodu se také neustále rozvíjí a přizpůsobují. S postupem času a společně se zdokonalováním dostupného hardwarového a softwarového vybavení dochází ke změnám možností, které jsou schopny informační technologie poskytovat. Současně dochází ke změnám v chápání, jakými způsoby lze softwarové vybavení modelovat, což se projevuje v různorodých architektonických návrzích informačních systémů a interakcích mezi nimi. Následující kapitoly představují vybrané architektonické přístupy k návrhům informačních systémů, které ovlivnily vznik EDA konceptu, či které jej vhodně doplňují. Cílem kapitoly je poskytnout základní kontext pro pochopení vazeb mezi vybranými přístupy a zároveň charakterizovat možnosti za účelem plného využití potenciálu EDA konceptu.

3.1.1 Distributed Computing

Jedná se o základní oblast informačních systémů, která je charakterizována vykonáváním počítačového programu v distribuovaném prostředí. Distribuovaný systém se skládá z autonomních počítačových systémů vzájemně komunikujících pomocí počítačové sítě (Emmerich 2000). Distribuovaný počítačový program je schopen běhu v takovém distribuovaném systému.

3.1.2 Objektově orientovaná analýza a design

Objektově orientovaná analýza a design (OOAD) je způsob tvorby softwarového vybavení, který modeluje počítačové systémy pomocí skupin objektů. Výstupem objektově orientované analýzy je vytvoření datového modelu objektů, který se skládá z dat, metod a vazeb na okolní objekty. Objekty samotné zapouzdřují svá privátní data a umožňují okolním objektům přístup k těmto datům pomocí svých metod.

Objektově orientovaný design aplikuje nefunkční požadavky a omezení podle zvolené vývojové technologie a prostředí. Příkladem takových omezení může být požadavek na podporu distribuovaných transakcí, dobu odezvy při volání metod, vývojová platforma či programovací jazyk.

Objektový přístup je hojně užívaný pro vývoj systémů, jelikož zvyšuje efektivitu a kvalitu především při práci ve větších týmech či při vývoji funkčně podobných aplikací. Mezi přednosti tohoto přístupu se řadí zjednodušení opětovného využívání funkcionality, a také zjednodušení změn a adaptace při zpracování dat (Pulier a Taylor 2006, s. 13).

3.1.3 Klient-server architektura a počátky standardizace

Klient-server komunikace spočívá v takové operaci, kdy si klientský program vyžádá data anebo funkcionalitu po jiném (serverovém) programu, typicky vzdáleně přes počítačovou síť (Pulier a Taylor 2006). Programovací jazyky typicky obsahují předpřipravené knihovny pro síťovou komunikaci a jednotlivé aplikace si tedy mohou zvolit způsob podporované výměny dat po síti. Takové aplikace tedy podporují velmi specifický způsob výměny dat po síti, který není kompatibilní s jinými aplikacemi.

Toto v důsledku vedlo ke snaze standardizovat síťovou komunikaci mezi aplikacemi, což dle Puliera et al. (2006, s. 16) vedlo k vytvoření specifikací CORBA[1] a následně DCOM[2] jako reakce společnosti Microsoft. Pro rozšířený jazyk Java lze uvést technologií RMI[3], která v pozdějších verzích byla rozšířena o podporu CORBA, a následně specifikaci EJB[4], která je součástí serverové specifikace J2EE[5].

---

[1] Component Object Request Broker Architecture: Specifikace standardizační organizace Object Management Group.

[2] Distributed Component Object Model: Proprietární komunikační technologie společnosti Microsoft.

[3] Remote Method Invocation: Objektově orientovaná komunikační technologie jazyka Java.

[4] Enterprise Java Beans: Specifikace vydávaná v rámci Java Community Process Program pro vývoj řízených, serverových komponent umožňujících modulární tvorbu podnikových aplikací.

[5] Java Platform, Enterprise Edition: Sada specifikací vydávaných v rámci Java Community Process Program pro vývoj podnikových aplikací.

Všechny uvedené způsoby se ovšem vyznačují společnou vlastností, a to je přímé volání vzdálených objektů. To v důsledku vede k vytvoření silné vazby mezi jednotlivými počítačovými systémy a jejich návrhy. Problémy s tím související částečně řeší následující IT koncept.

3.1.4 Enterprise Application Integration

EAI je termín používaný pro plánování, metody a nástroje užívané pro modernizaci, konsolidaci a koordinaci různorodých nezávislých aplikací uvnitř podniku. Umožňuje zvýšení efektivity podniku pomocí výměny dat mezi aplikacemi, které byly vyvinuty pro odlišné operační systémy, databáze či za použití proprietárních technologií. Principem je zpřístupnit existující data a business logiku pro další použití, a to ideálně při minimalizaci nutných zásahů do zdrojové či cílové aplikace.

Tento přístup také odstraňuje část problémů, které jsou spojovány s distribuovanými systémy. Jednotlivé aplikace podporují technologie, které nemusí být podporovány v okolních systémech. Produkty EAI tvoří prostředníka mezi těmito systémy a tím tuto nevýhodu potlačují (Pulier a Taylor 2006, s. 43). To snižuje náklady nutné na lidské zdroje pro vývoj a testování změn, ale také na následnou údržbu jednotlivých systémů.

EAI produkty typicky umožňují synchronní i asynchronní výměnu zpráv, transakční zpracování, obsahují podporu směrování či pozdržení zpráv, škálovatelnost a vysokou dostupnost. Nevýhodami jsou naopak vysoké investiční náklady, časová prodleva při zavádění do praxe, nízká flexibilita poskytovaných rozhraní a nevýhody spojené s řízením a kontrolou při větším množství propojených služeb. Pulier et al. (2006, s. XXXVI) mezi další nevýhody tohoto přístupu řadí nutnost udržovat si zaměstnance se specifickými znalostmi konkrétních technologií, které jsou používány v integrovaných aplikacích.

3.1.5 Servisně orientovaná architektura

Nevýhody spojené s předešlým konceptem EAI se snaží řešit servisně orientovaná architektura (Service Oriented Architecture - SOA). Počátky servisně orientované architektury se vztahují k postupné potřebě zavést nové standardy v oblasti počítačové

výměny dat, a to především v období masivního rozšíření a užití Internetu. Mezi klíčové standardy a technologie řadí Pulier et al. (2006, s. 21) především specifikace založené na XML[6] a komunikační protokoly, které souvisí s univerzální interakcí mezi počítači. Tyto technologie umožňují takovou výměnu zpráv, že při jejich použití jsou komunikující aplikace na sobě mnohem více nezávislé – jsou „loose coupled“ (ve volné vazbě). Zároveň jsou výrazně nezávislé i na použité síťové infrastruktuře, jelikož pro přenos se často využívá rozšířený a podporovaný protokol HTTP[7]. Vzdálené užití služeb probíhá voláním publikovaných webových služeb, tedy zasíláním SOAP[8] zpráv a jejich zpracováním.

Servisně orientovaná architektura rozhodně nesouvisí pouze s technologiemi. Cummins (2009, s. 27) ji například definuje jako nové paradigma business designu se stěžejním zaměřením na vytvoření takové společnosti, která bude podporovat agilnost v oblasti rychlosti, nákladů a kvality. Dále zdůrazňuje, že SOA nutně nevyžaduje elektronické technologie, ale automatizace, integrace a modelovaní pomocí těchto technologií je základem pro její optimální implementaci. Plná implementace SOA ovšem také vyžaduje kompletní transformaci podniku.

Hlavním příslibem této architektury je schopnost kontinuálního vývoje a optimalizace podniku na základě potřeb a strategie podniku, a ne na základě omezení IT oddělení (Pulier a Taylor 2006, s. 51). Pulier et al. (2006, s. 52) dále zmiňuje, že jednotlivé prvky SOA mohou být přemístěny, nahrazeny či modifikovány, a to díky volné vazbě mezi jednotlivými službami. Dále je možné tyto jednotlivé služby skládat a spojovat do různých forem a za různým účelem, a tím vytvářet nové procesy či složené aplikace. Hlavním příslibem je tedy flexibilita změn.

Z pohledu vývoje v čase může být SOA vnímána jako přístup, který nepřinesl očekávané výsledky. Většinou se ovšem jedná o názory, které souvisí s chybným nastavením očekávání či přímo se špatnou aplikací SOA, kdy hlavním přínosem SOA není

[6] Extensible Markup Language: Značkovací jazyk. Spravuje organizace W3C.

[7] Hypertext Transfer Protocol: Síťový protokol využívaný především v internetovém prostředí. Vývoj protokolu zajišťují organizace IETF a W3C.

[8] Simple Object Access Protocol: Specifikace komunikačního protokolu. Spravuje organizace W3C.

zakoupený software, ale především aplikování definovaných principů a postupů (Handy 2010) a také zavedení politiky, správy a údržby služeb - tzv. Governance (Gartner 2009).

Vlastní definice principů SOA ovšem přesahuje hranice této architektury a je využitelná i v dalších oblastech návrhu a vývoje software. Gartner (2009) uvádí, že výhoda SOA se mnohem šířeji projevuje v kombinaci s jinými přístupy a technologiemi. Realizací obchodních procesů BPM či komplexních událostí v rámci EDA lze zvýšit agilnost a flexibilitu podniku a jeho procesů. Handy (2010) dále uvádí, že z obchodního pohledu je dnes nutné vnímat cloud jako prostředí, ve kterém je aplikování SOA nanejvýš vhodné a kde lze také těžit z dosažení vyšší flexibility a nezávislosti jednotlivých služeb. Právě z těchto důvodů zůstává i nadále SOA stále platnou a žádanou oblastí, což potvrzuje i výzkum provedený společností Forrester (Heffner 2011).

3.1.6 Business Process Management

Business Process Management je manažerská disciplína zaměřující se na navrhování a průběžné vylepšování rychlosti, nákladů a kvality obchodních operací (Cummins 2009, s. 74). Cummins dále zmiňuje, že tento přístup obnáší analýzu a vylepšování jak manuálních, tak i automatizovaných obchodních procesů, přičemž práce v podniku je typicky vykonávána určitými jednotkami a řízena procesy. Proces je zahájen na základě konkrétní potřeby či cíle, přičemž proces je definován prací, kterou je nutné vykonat, a také kým, kdy a proč má být vykonána (Cummins 2009, s. 77). Jednotlivé kroky procesu přitom mohou být vykonávány manuálně jednotlivými rolemi pracovníků, anebo plně automaticky některým IT systémem.

Průběžná analýza procesů a jejich pružná optimalizace dává možnost managementu podniku provést strategické omezení nákladů či poskytnout příležitost pro generování nových tržeb. Aby tyto změny byly efektivní, je nutné je provést v zamýšleném časovém období (Pulier a Taylor 2006, s. 96). Z pohledu analýzy efektivity procesu je pro management podniku přitom zajímavá možnost spekulativní analýzy návratnosti investovaných prostředků a předpokládaných tržeb hned při úvodním návrhu procesu. Po implementaci a nasazení je poté naopak zajímavá zpětnovazební analýza na základě reálných dat. Některé softwarové produkty obě tyto možnosti dnes již plně podporují.

Užitím servisně orientovaných služeb při automatizaci obchodního procesu lze nejenom minimalizovat nutné náklady na realizaci, ale především snížit dobu uvedení nových či modifikovaných služeb na trh. To je dáno především autonomností, znovupoužitelností a komponovatelností jednotlivých služeb. Zároveň se zvyšuje schopnost managementu podniku se více soustředit na vlastní obchodní proces a zároveň přitom správně předvídat a chápat dopady požadovaných změn na IT infrastrukturu (Pulier a Taylor 2006, s. 96).

Při realizaci vhodné kombinace konceptů EDA, SOA a BPM lze události reálného světa zpracovávat téměř v reálném čase, a na základě vyhodnocených pravidel vyvolávat jednotlivé služby či přímo automatizované procesy.

3.1.7 Cloud Computing

Tento pojem zahrnuje oblast vývoje a používání počítačových technologií, kde je společným prvkem provoz části IT ve virtuálním prostředí na Internetu. Existuje několik obchodních modelů, jako příklad uvádí Svoboda (2009): IaaS - Infrastruktura jako služba, PaaS - Platforma jako služba a SaaS - Software jako služba. Společným znakem je vysoká dostupnost a obrovská škálovatelnost a elasticita, která umožňuje klientům rychle měnit využívané zdroje podle aktuální potřeby. Zároveň se klient nemusí starat o běžnou údržbu spojenou s výměnou a opravami hardware či o aktualizace poskytovaného softwarového vybavení.

Mezi nevýhody je možné zařadit úzkou vazbu na konkrétního poskytovatele služby (je složité a nákladné přejít k jinému poskytovateli), případný odlišný právní řád poskytovatele a klienta, rizika spojená s bezpečností ukládání citlivých dat mimo podnikovou síť a v neposlední řadě závislost na dostupnosti Internetu.

Při porovnání přístupů Cloud Computing a SOA lze konstatovat, že Cloud Computing není následovníkem SOA či naopak. Oba přístupy se vzájemně doplňují a při vhodné kombinaci užití dokonce poskytují vyšší přidanou hodnotu. Lze například kombinovat služby poskytované v rámci SaaS se službami uvnitř podniku, stejně tak lze provozovat již existující služby nově v rámci IaaS.

Specifickým případem je využití Cloud Computing ve spojitosti s konceptem EDA, kdy je pro zpracování komplexních událostí zapotřebí uchovávat a zpracovávat značné

množství dat, a to ideálně v řádech zlomků sekundy. S potřebou zpracování velkého objemu dat se zavádí pojem BigData charakterizovaný v následující kapitole.

3.1.8 BigData

O'Reilly Media (2012, s. 3) charakterizuje BigData jako data, která překročila procesní kapacitu konvenčního databázového systému. Dále uvádí, že tato data jsou příliš objemná, velmi rychle se měnící či nesplňují podmínky pro uložení v typické databázové architektuře.

Přístupy a produkty pro BigData je tedy vhodné užít především v případech, kdy je zapotřebí provádět značné množství méně či více náročných operací nad velkou množinou dat. Podmínkou je zde typicky možnost algoritmicky oddělit prováděnou operaci na menší části tak, aby tyto mohly být vykonávány nezávisle (a distribuovat úlohu mezi více strojů). Za tímto účelem lze využít návrhových vzorů typu MapReduce (Miner 2012, s. 1–3), například na produktu Apache Hadoop či různých NoSQL databázích (O'Reilly Media, Inc. 2012, s. 5).

O'Reilly Media (2012, s. 8) zdůrazňuje, že přestože řešení zpracování "big data" mohou být založena na zpracování v cloudu, tak je nutné brát v potaz také nutnost přenosu těchto dat. To nemusí být při daných objemech možné a je tedy nutné data zpracovat "tam, kde jsou". Toto řešení je nutné především v případech, kdy data vznikají kontinuálně a je přesto zapotřebí je zpracovat v reálném čase (Barlow 2013, s. 21) - příkladem zde je nutnost volby vhodně cílené reklamy při zobrazování webových stránek jednotlivým uživatelům.

Využití kombinace konceptů EDA ve spojitosti s BigData je vnímáno jako velmi vhodné, a to především při využití zpracování pomocí komplexních událostí (Lapkin 2012). Obě technologie jsou zároveň vnímány jako technologie na vzestupu (LeHong a Fenn 2012).

3.2 Definice událostmi řízené architektury

Následující kapitoly uvádí vybrané pohledy na definici podstaty událostmi řízené architektury.

3.2.1 Definice a charakteristika základních pojmů

Tato kapitola poskytuje jednotlivé definice a charakteristiky pojmů užitých v událostmi řízené architektuře. Je nutné poznamenat, že v literatuře není stále ustálena nomenklatura jednotlivých užitých pojmů. Jednotliví autoři tedy používají různé názvy pro charakteristiku pojmu, který má v daném kontextu stejný význam. V následujících kapitolách jsou uvedeny vybrané příklady těchto názvů. V rámci této práce budou nadále preferovány názvy pojmů, jak je definují Luckham a Schulte (2011). V dané práci je snaha o sjednocení názvosloví pojmů užitých ve zkoumané oblasti. Níže charakterizované pojmy a pro ně užité názvy jsou také ve shodě s tím, jak je definuje Etzion a Niblett (2010, s. 42).

Charakteristika jednotlivých pojmů uvádí také jednotlivé názvy v originálním jazyce, u kterých je zapotřebí zajistit přesné pochopení v zamýšleném významu či se na ně později odkazovat.

3.2.1.1 Událost a objekt události

Jednotliví autoři se shodují na tom, že pojem událost je často vnímána různým způsobem (Faison 2006, s. 71; Alves et al. 2013, s. 17; Etzion a Niblett 2010, s. 4).

Faison (2006, s. 71) pojem událost (event) definuje jako detekovatelnou podmínku, která může vyvolat notifikaci (notification). Notifikaci poté definuje jako událostí vyvolaný signál zaslaný do přijímače (receiver), který je určen za běhu. Podmínka (Faison 2006, s. 72) přitom může být založena na libovolném množství různých predikátů. Detekce dané podmínky musí mít potom vazbu na konkrétní notifikaci. Za události zde potom Faison považuje právě ty podmínky, které mohou vyvolat notifikaci.

Alves et al. (2013, s. 17) definují událost jako objekt, který má změnu svého stavu buď okamžitou, anebo v časovém období.

Etzion et al. (2010, s. 4) charakterizují událost dvojím způsobem:

- Událost je výskyt v určitém systému či doméně. Představuje něco, co se již v dané doméně událo, anebo je uvažováno tak, že se událo.
- Událost je programová entita, která reprezentuje výskyt v počítačovém systému.

Význam jednotlivých uvedených charakteristik si uvědomuje Chandy a Schulte (2010, kap. 7 sek. 1) a člení a charakterizují jednotlivá hlediska následovně:

- **Pohled dle změny stavu** (State-change view):
  - Objekt události (event object) je změna stavu čehokoliv.
  - Událost je poté zpráva o změně stavu.
- **Pohled dění** (Happening view):
  - **Událost** je cokoliv, co se děje, anebo je uvažováno tak, že se děje.
  - **Objekt události** reprezentuje, kóduje či zaznamenává událost, obecně za účelem počítačového zpracování.
  - Dle Chandyho et al. (2010, kap. 7 sek. 1) tento pohled vychází z prací Luckhama (2007) a Luckhama et al. (2011).
- **Pohled detekovatelné podmínky** (Detectable-condition view)
  - Tento pohled byl představen na začátku této kapitoly v práci Faisona (2006, s. 71).

Chandy et al. dále uvádějí, že se sice jednotlivé pohledy značně překrývají, ovšem pod pojmem událost je zde chápán trochu odlišný koncept události. **Pohled dle změny stavu** předpokládá existenci věci a změnu stavu této věci ze stavu před do stavu následujícího. **Pohled dění** naproti tomu chápe událost jako aktivitu, která se děje. Když se aktivita děje, mění se stav. Definice ovšem nevyžaduje specifikaci kolik věcí je ovlivněno či nutnost být schopen dokumentovat stav před a stav po aktivitě. Poslední ze jmenovaných pohledů, **pohled detekovatelné podmínky**, považuje za událost pouze to, co může být pozorováno a zároveň o tom může být podána zpráva (notification).

Tabulka 1 - Rozdílné pohledy na událost

Pohled	Změna věci/věcí	Pozorování je hlášeno
Dle změny stavu	Ano	Ano
Dle dění[9]	Ano	Volitelně
Dle detekovatelné podmínky	Volitelně	Ano

Zdroj: (Chandy a Schulte 2010, kap. 7 sek. 1)

3.2.1.2 Doba trvání události

Podle zvoleného pohledu na definici události lze uvažovat také o tom, zda událost probíhá okamžitě, či zda má měřitelnou dobu trvání (Chandy a Schulte 2010, kap. 7 sek. 5).

Chandy et al. dále zmiňují příklady pro jednotlivé typy pohledů:

- **Pohled na události dle dění**: Většina událostí má měřitelnou dobu trvání. Za příklad je zvolen děj koupě domu, který trvá 3 měsíce. Tento děj může mít tři časové značky: záznam o zahájení koupě, záznam o konci koupě a záznam o čase, kdy byla notifikace vytvořena či odeslána producentem události. Událost dle tohoto pohledu tedy může být dlouhotrvající. Některé události přesto samozřejmě mohou být **okamžité**, tedy čas začátku a čas konce této události jsou shodné.
- **Pohled dle změny stavu**: V tomto případě je za dobu trvání události považována doba, po jakou trvala změna stavu (state transition time). Tento pohled umožňuje jak dlouhotrvající, tak i okamžitou událost. V uvedeném příkladu s koupí domu by okamžitou událostí mohl být okamžik uvedení posledního podpisu na smlouvě.
- **Pohled detekovatelné podmínky**: Koncept doby trvání události je již méně aplikovatelný, jelikož událost je snímkem vybraných podmínek. Časová značka tedy obecně odráží přesný bod v čase, kdy bylo učiněno pozorování (a vyhodnoceny dané podmínky).

[9] Tato práce bude nadále preferovat pohled dění při užití pojmů událost a objekt události.

3.2.1.3 Producent události

Faison (2006, s. 77) užívá názvy Event Publisher, Event source či pouze Sender. Definuje je jednoduše jako entitu, která je schopna detekovat události. Samotný akt odeslání notifikačního signálu potom nazývá odesláním notifikace (sending a notification) či vyvoláním události (firing the event). Veškerá data, která jsou součástí této notifikace, jsou nazývána payload.

Etzion et al. (2010, s. 42) užívají název **Event producer**, a definují jej jako entitu na hranici systému zpracování událostí, která zavádí události do systému.

Taylor et al. (2009, s. 53) užívají názvy Event Publisher a **Event producer**. Jako hlavní podmínku pro existenci celého EDA konceptu vnímají právě schopnost vzniku událostí a jejich doručení do EDA komponent. Taylor et al. dále uvádějí, že producenti událostí mohou mít nejrůznější formu. Od softwarových programů až po dedikovaný hardware, který převádí analogová data na digitální formu, kterou následně poskytuje softwaru určenému k detekci událostí. Producenti událostí mohou být kdekoli uvnitř i vně podniku. Data uvnitř událostí nemusí přitom pocházet z daného producenta. Nezřídka je zapotřebí, aby byla tato data uzpůsobena potřebám konkrétního EDA systému (změnou formátu dat).

V textu práce bude nadále užíván název producent události (Event producer) ve významu popsaném v této kapitole.

3.2.1.4 Agent zpracování událostí

Etzion et al. (2010, s. 42) užívají název **Event processing agent**, který definují jako softwarový modul zpracovávající události. Tento modul používá události a může je přeposílat či vydávat nové, tedy může sloužit pro konzumaci a produkci událostí. Etzion et al. ovšem dále zmiňují, že nemůže být tento modul chápán jako konzument či producent událostí (ve výše uvedeném významu), jelikož jsou tyto termíny vyhrazeny pro entity vně systému na zpracování událostí.

Taylor et al. (2009, s. 54) zmiňují názvy Event Listener a Event consumer, ovšem ve významu části funkcionality agenta pro zpracování událostí jako bylo uvedeno v předešlém odstavci. Dále bude tedy ve výkladu Taylorem užitý pojem „konzument

události“ zaměněn za termín „agent pro zpracování událostí“ a pojmy Event Listener a Event consumer budou v této práci užity pod významem uvedeným v kapitole 3.2.1.5 níže. Taylor et al. dále zmiňují, že agenti mohou být umístěni kdekoli. Představují technologii typicky založenou na software, ovšem opět s možností jejich realizace formou specializovaného hardwarového prvku. Znakem těchto komponent je schopnost odlišit příchozí událost od ostatních přijímaných dat. V nejjednodušší formě EDA je schopen agent přijímat pouze specifická data události. Agenti dále musí být schopni přijímat a detekovat konkrétní typ události, být schopni jej rozpoznat a interpretovat. Kritéria interpretace jsou známa jako obchodní pravidla (business rules).

Taylor et al. (2009, s. 54–55) označují jako další součást EDA komponentu pro zpracování událostí, **Event Processor**. Fakticky se zde opět jedná o pojetí agenta dle definice Etziona et al.. Taylor et al. dále zmiňují, že zpracováním události je myšleno zjištění potenciálního dopadu a celkové hodnoty události, a vyhodnocení další případné akce. EDA by přitom měla mít dostatečnou kapacitu k interpretaci přijímaných událostí. Event Processor je potom software, který je schopen posoudit událost, vyhodnotit její důležitost a generovat odpovídající reakci. Taylor (2009, s. 55–56) pod užitým pojmem reakce (Event Reaction) myslí takové zpracování, které je realizováno konzumenty událostí charakterizovanými v následující kapitole.

V textu práce bude nadále užíván název **agent** (Event processing agent) ve významu popsaném v této kapitole.

3.2.1.5 Konzument události

Faison (2006, s. 77) zmiňuje názvy Event Subscriber, Event handler, Notification target, Notification receiver či pouze Receiver. Definuje je jednoduše jako entitu, která přijímá notifikace.

Etzion et al. (2010, s. 42) užívají název **Event consumer**, a definují jej jako entitu na hranici systému zpracování událostí, která přijímá události ze systému.

V textu práce bude nadále užíván název konzument události (Event consumer) ve významu popsaném v této kapitole.

3.2.1.6 Základní událost

Etzion et al. (2010, s. 42) užívají název Raw event a definují jej jako událost, která byla zavedena do systému producentem událostí. Luckham et al. (2011) tento pojem charakterizují jako událost, která se stala v reálném světě.

3.2.1.7 Proud událostí

Do proudů událostí jsou doručovány události generované hardwarovými senzory, distribuované kdekoli z "Internetu věcí" (Internet of Things), z počítačových aplikací či generované ze stovek jiných libovolných zdrojů (Alves et al. 2013, s. 17). Alves et al. dále uvádějí, že do proudu událostí mohou plynule proudit události ve vysokých množstvích či přicházet ve sporadických intervalech, ale proud těchto událostí nikdy nekončí a je vždycky seřazený v čase (stejně jako v reálném světě). Obecně lze proudy událostí rozdělit do 3 kategorií: jednoduché, proudící a s vysokým množstvím událostí.

Tabulka 2 - Charakteristika typů proudů událostí

Typ proudu události	Charakteristika	Příklad
Jednoduchý proud událostí	Nepravidelný tok. Řiditelné množství.	Příjezd zákazníka. Dodání objednávek.
Proudící události	Nepřetržitý tok. Nutnost „držet krok" při zpracování.	Senzor teploty. Aktivita na burze.
Proud s vysokým množstvím událostí	Možná nárazový tok Potřeba řešit nejhorší případ	Vstupy z bojiště. Senzory prostředí.

Zdroj: (Alves et al. 2013, s. 18)

3.2.1.8 Komplexní události

Událostmi řízený systém generuje komplexní události destilací faktů z více příchozích základních událostí do několika komplexních událostí (Chandy a Schulte 2010, kap. 7 sek. 3). Chandy et al. dále charakterizují komplexní událost následovně:

- reprezentuje postřehy na souhrnné úrovni, které jsou více významné a nápomocné pro rozhodování než základní události.

- je abstraktní v tom smyslu, že je jeden nebo více kroků odstraněno ze surových vstupních dat.

Luckham et al. (2011) považují za komplexní událost takovou událost, která shrnuje, představuje, nebo označuje sadu jiných událostí. Tato definice je velmi podobná definici předešlé.

Etzion et al. (2010, s. 42–43) užívají také název Derived event a definují jej jako událost, která byla generována na základě výsledku zpracování událostí uvnitř systému na zpracování událostí. Tato definice tedy umožňuje relativní pohled na komplexní událost v závislosti na tom, ve kterém systému komplexní událost vznikla a ve kterém je finálně zpracována. Při zpracování v dalším (jiném) systému by mohla být považována opět již za základní událost.

3.2.1.9 Kanál událostí a směrovací schéma

Etzion et al. (2010, s. 134) definují **kanál událostí** (event channel) jako prvek (element) pro zpracování, který přijímá událost z jednoho či více zdrojových prvků, provádí rozhodování o směrování (routing) a odesílá vstupní události v nezměněné podobě do jednoho či více cílových zpracovávacích prvků dle provedeného rozhodnutí o směrování.

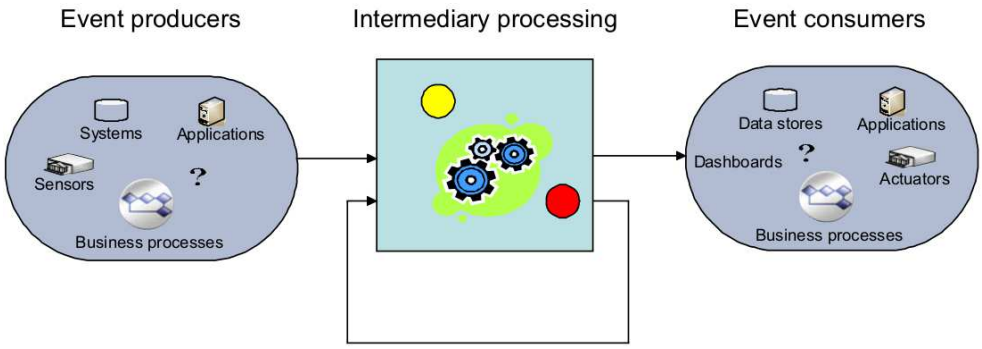
Rozhodování o směrování je dáno **směrovacím schématem** (routing scheme), které může být typu (Etzion a Niblett 2010, s. 136):

- napevno stanovené pravidlo pro všechny události bez rozdílu,
- podle typu zpracovávané události a
- na základě dat obsažených uvnitř události a stanovených rozhodovacích stromů či rozhodovacích tabulek.

3.2.2 Definice a základní charakteristika EDA

Jednoduchou definici událostmi řízené architektury uvádějí Luckham et al. (2011): EDA je architektonický styl, ve kterém jsou komponenty řízeny událostmi a komunikují spolu za užití událostí.

Etzion et al. (2010, s. 40) se při charakteristice EDA zaměřují nejprve na uvedení obecného způsobu konstrukce aplikací podle tohoto principu a základních komponent, které takovouto aplikaci tvoří. Dále uvádějí, že struktura velké většiny systémů na zpracování událostí je podobná konceptu zobrazenému na následujícím obrázku:



Obrázek 1 - Základní koncept zpracování událostí

Zdroj: (Etzion a Niblett 2010, s. 41)

Etzion et al. dále uvádějí, že systém dle EDA obsahuje jeden či více producentů událostí a konzumentů událostí (dle definic uvedených v předešlých kapitolách). Tyto komponenty jsou propojeny distribučním mechanismem událostí, jak je vyobrazeno výše, přičemž v rámci této interakce jsou jednotlivé události také zpracovány agenty. Distribuční mechanismus je často realizován za užití asynchronních zpráv, ale nemusí to být nutně pravidlem. Producent může například zapisovat události do logu událostí, ze kterého konzument napřímo čte. Multiplicita distribuce událostí je typicky v poměru 1:m, tedy událost může být přijata více konzumenty.

Zpracování událostí může nabývat různých forem. Jako příklad lze uvést přesměrování, filtrování a či generování nových událostí (Etzion a Niblett 2010, s. 41). Tyto nové události mohou být zpracovány konzumenty událostí, ale třeba sloužit jako vstupy pro další opětovné zpracování.

Chandy et al. (2010, kap. 3 sek. 1) uvádějí, že EDA je jistým stylem, podmnožinou, možných zpracování událostí reálného světa. EDA pracuje především s objekty událostí, tedy s diskrétními zprávami o události. Dále je EDA založena na principu řízení událostmi (Event-driven), kdy entita koná pouze na základě rozpoznané události. Jedná se o architektonický styl, kde jednotlivé komponenty v softwarovém systému jsou řízeny událostmi, a zároveň jsou minimálně spojeny (minimally coupled). Pojmem spojení je zde

myšlena taková vazba, kdy mezi producentem a konzumentem událostí probíhá jednosměrný přenos objektů událostí.

Business aplikace implementuje EDA, pokud dodržuje následujících 5 principů (Chandy a Schulte 2010, kap. 3 sek. 1):

- **Poskytuje aktuální události**: Notifikace hlásí diskrétní výskyt v době, kdy se děje.
- **"Tlačí" (Pushes) notifikace**: Notifikace jsou tlačeny producentem událostí, a ne taženy (Pull) konzumentem. Producent se rozhoduje, kdy bude notifikace odeslána, jelikož o události ví dříve než konzument.
- **Odpovídá okamžitě**: Konzument reaguje ihned po rozpoznání události.
- **Komunikuje jednosměrně**: Notifikace je jednosměrná. Producent vyšle notifikaci a dále se již danou událostí nezabývá – nečeká na reakci ani nečeká žádnou odpověď od konzumenta.
- **Je bez příkazů**: Notifikace je zpráva a ne specifický požadavek či příkaz. Nepředepisuje akci, kterou by měl konzument vykonat.

Chandy et al. dále vysvětlují uvedené principy tak, že první tři uvedené charakterizují to, co znamená řízeno událostmi (Event-driven), a další dva zase co znamená být minimálně spojeno (minimally coupled). Aplikace by tedy nevyužívala EDA v případě, kdy by producent vysláním notifikace očekával konkrétní reakci od konzumenta, či pokud by očekával jeho odpověď (byl by závislý na konzumentovi).

Taylor et al. (2009, s. 120) uvádějí následující definici EDA, přičemž připouštějí, že ne vždy je možné dosáhnout ideální reálné implementace dle těchto pravidel:

- EDA je ve volné vazbě (loosely coupled) či kompletně oddělena (decoupled).
- EDA užívá asynchronní přenos zpráv, typicky modelem publish-subscribe.
- Granularita EDA je na úrovni události.

- EDA obsahuje[10] producenty událostí (Event producers), agenty (Event listeners, Event processors) a konzumenty událostí (Event reactors). Ideálně založeno na SOAP webových službách a kompatibilních aplikačních komponentách.
- EDA užívá běžně dostupnou infrastrukturu pro přenos zpráv (messaging backbone), jako je ESB stejně tak i adaptéry a zprostředkovatele (intermediaries) pro přenos zpráv.
- EDA není závislá na centrálním řídícím prvku (controller).

Taylor et al. (2009, s. 125) dále uvádějí, že EDA je řízena rozšiřitelností systému a ne jeho ovladatelností, a poháněna obchodními událostmi. EDA je založena na dynamickém determinismu, kde není předem zcela jistě známo, jaký dopad bude mít vyvolání dané události (producentovi nejsou dopředu známi všichni příjemci zprávy). Důsledkem čehož jsou transakce řízeny samotnými událostmi, a ne řídícím mechanismem. Taylor et al. dále uvádějí, že EDA pojímá koncepty usnadňující flexibilitu a rozšiřitelnost, což nakonec zvyšuje schopnost systému vyvíjet se[11]. Negativními důsledky těchto vlastností jsou pak decentralizované řízení a stupeň neurčitosti v systému.

V souhrnu lze uvést, že pro událostmi řízenou architekturu je typické přijetí událostí, jejich odpovídající zpracování a vyvolání odpovídající reakce. Architekturu lze rozdělit na nepřímou oblast a jádro architektury, kde do první jmenované skupiny patří produkce událostí vstupujících do EDA a konzumace událostí, které z EDA vystupují. Do jádra architektury potom patří schopnost přenosu událostí, jejich přijetí a vlastní vhodné zpracování. IT systémy navržené dle EDA využívají asynchronní výměnu zpráv pro přenos objektů událostí, a to s různou kvalitou přenosu. V rámci událostmi řízené architektury je možné realizovat různé komplexní IT systémy, jejichž realizace by za užití jiných přístupů byla velmi obtížná a nákladná.

[10] Český překlad využívá názvy, které byly zvoleny pro dané termíny v předešlých kapitolách.

[11] Toho je dosaženo užitím volné vazby, asynchronnosti a bezstavovosti poskytovatelů služeb.

3.3 EDA z pohledu využitelnosti a rizik

Tato kapitola charakterizuje možnosti užití událostmi řízené architektury v jednotlivých oblastech. Jsou zde uvedeny očekávatelné přínosy při implementaci EDA, ale také různá omezení a rizika.

3.3.1 Oblasti užití a možné přínosy

Událostmi řízenou architekturu je možné využít v mnoha oblastech života. To je dáno především skutečností, že tento architektonický přístup nijak neomezuje možnosti zavedení nových typů událostí (reálného světa) do počítačových systémů, a při vhodném zpracování těchto událostí také iniciaci odpovídajících reakcí. V literatuře se nezřídka zmiňuje výčet možností komerčního užití EDA v podnicích, přičemž aplikace může být i mimo komerční oblast. Principy zpracování událostí jsou natolik obecné, že jsou běžně užívány jako součást prací na nejrůznější téma, například: měření v elektrické síti (Simonov 2013), inkrementálního algoritmu pro vyhledávání chyb (HUANG et al. 2009), či při nutnosti zpracování enormního množství dat (Hoff 2011). Událostmi řízená architektura tedy představuje ucelený pohled na obecnou problematiku zpracování událostí v počítačových systémech, a to včetně postupného zavádění společné terminologie v této oblasti.

Událostní řízení je již delší dobu zmiňováno ve spojitosti s termíny jako Real-time Enterprise (RTE)[12] (Margulius 2002; Botzer 2007) či Real-time Business Intelligence (Agrawal 2009; Linden a Neuhaus 2010). Oba termíny souvisí s různými využitími v mnoha oblastech podnikání. Cílem je typicky zvýšení konkurenceschopnosti podniku, zvýšení efektivity při rozpoznávání dění na trhu, a snížení nákladů.

Taylor et al. (2009, s. 120) uvádějí následující přínosy a oblasti, kde lze EDA užít:

- EDA umožňuje agilnost v operativním řízení změn (change management).
- EDA umožňuje korelaci dat pro analytické a obchodní modelování procesů, řízení a správu (governance).

[12] Základní informace o RTE lze nalézt na adrese: http://en.wikipedia.org/wiki/Real-time_enterprise

- EDA umožňuje dynamický determinismus, kde podnik je schopen reagovat v závislosti na dynamických změnách množiny obchodních pravidel.
- EDA zavádí větší povědomí o událostech v podnikovém "nervovém systému".

Taylor et al. dále zmiňují možnost užití v různých oblastech. V **obchodní sféře** (Taylor et al. 2009, s. 145) především poskytnout vysoce kvalitní data o obchodních podmínkách, která mají dopad na strategická rozhodnutí společnosti. Dále pak potenciál zlepšit jednotlivé typy agilnosti společnosti: **Strategická agilnost** - kapacita dostatečně rapidně měnit strategický plán či obchodní model dle měnících se obchodních podmínek. **Operační agilnost** – schopnost organizace měnit způsob fungování jako odpovědi na měnící se podmínky. Typicky změna strategie přináší nutnost přepracování operativního chodu společnosti, souběžně se změnami v IT. **Funkční agilnost** – schopnost změny aktuálních obchodních procesů v odpovědi na obchodní podmínky.

Mezi možné další příklady užití EDA lze zařadit následující vybrané oblasti:

- **Veřejný sektor** - real-time zpracování a vyhodnocování různorodých dat v oblasti bezpečnosti či regulace (Taylor et al. 2009, s. 147–149).
- **Zdravotnictví** - zvýšení efektivity v oblasti výzkumu díky real-time zpracování více datových proudů; v oblasti nákladů či řízení zdravotnických zařízení (Taylor et al. 2009, s. 151).
- **Dodržování pravidel a zákonů** – Především pro velké nadnárodní korporace. Při vhodné implementaci může EDA poskytovat schopnost vyhodnocování zadaných pravidel, která souvisí s vnitřními předpisy společnosti či zákonnými podmínkami platnými pro jednotlivé pobočky společnosti. Například export osobních údajů do USA zaměstnance pracujícího na pobočce v EU, může být zákonem omezen. Agent pro zpracování události exportu dané informace může danou nevhodnou činnost nahlásit (audit), či ji přímo zablokovat. Zároveň je systém stále dostatečně flexibilní, a připraven na případné změny (Taylor et al. 2009, s. 153).
- dále pak zásobování, energetika, řízení dopravy, pojišťovnictví, bankovnictví a mnohé další.

Chandy et al. (2010, kap. 5 sek. 1) uvádějí následující vlastnosti, které nejvíce iniciují poptávku po událostním zpracování:

- detekce kritických událostí vně podniku,
- rychlé reakce na měnící se situace držením kroku s nově generovanými daty,
- řízení dle výjimek,
- adaptabilita - přizpůsobení chování podnikatelských činností dle měnících se potřeb,
- koordinace obchodních aktivit v podniku za účelem business intelligence, monitoringu aplikací a integrace aplikací.

3.3.2 Možná rizika a omezení

Při rozhodování o zavedení EDA je nutné uvážit hned několik aspektů, které mohou při nevhodné implementaci způsobit závažné škody. Následující kapitoly charakterizují možná rizika a omezení, na které je nutné brát ohled při zavádění EDA v podniku.

3.3.2.1 Obecné

Jednotlivé oblasti možných rizik a problémů lze rozdělit do několika oblastí, které mohou souviset jak s procesem zavádění systému, tak s jeho provozem. Při zavádění systému dochází v rámci fází analýzy a designu k důležitým rozhodnutím, která mají významný vliv na splnění stanovených cílů. V těchto fázích je zapotřebí uvažovat mnohé aspekty, které nemusí být při prvotním uvážení patrné. Mezi ně lze zařadit oblasti jako je dlouhodobá flexibilita řešení, bezpečnost, škálovatelnost, volba způsobů prevence a reakcí na chybné zpracování událostí, a mnohé další.

Fremantle (2008) vnímá jako možnou omezující situaci stav, kdy není možné modifikovat všechny IT systémy, které se integrují do událostmi řízené architektury. Fremantle jako příklad uvádí situaci, kdy není zřejmé, který systém je vlastníkem přenášených dat, tedy zda právě asynchronně zpracovávaná událost je původní či je pouhou reakcí (echem) na již zpracovanou událost.

Mezi další konkrétní příklady problémů lze zařadit složitou synchronizaci dat v distribuovaném modelu po obnově části EDA infrastruktury ze zálohy, či synchronizaci

po provedení ad-hoc korekce dat v některém ze systémů (DeCare Systems Ireland 2013b, 2013a). Clark a Barn (2011) vnímají jako problém také to, že není ujednocen a široce podporován způsob modelování událostmi řízené architektury. Pro ilustraci uvádí, že nejvíce rozšířený jazyk pro modelování, UML[13], neobsahuje žádné specifické vlastnosti, které by podporovaly modelování EDA.

Mezi další typy lze zařadit výkonnostní omezení plynoucí z užité infrastruktury či zvolených technologií. V určitém momentu může dojít k dosažení kritické hranice, kdy se zvyšuje celková doba odezvy při zpracování jednotlivých typů událostí. Příkladem výkonnostních problémů může být jak zvyšující se množství vstupujících událostí, se kterým se v původním návrhu nepočítalo, tak například i samotná zvyšující se náročnost zpracování nově zaváděných či modifikovaných komplexních událostí. Některým problémům lze tedy předejít vhodným návrhem při zavádění systému či vhodně zvoleným kompromisem mezi výkonností a flexibilitou.

Chandy et al. (2010, kap. 4 sek. 1) uvádějí oblasti s **akronymem REACTS**, kde jednotlivé oblasti charakterizují jisté požadavky na událostmi řízenou architekturu a mají tedy vliv na její výslednou podobu: Relevantnost (Relevancy), Úsilí (Effort), Správnost (Accuracy), Úplnost (Completeness), Včasnost (Timeliness) a Bezpečnost (Security). Následující kapitoly uvádějí charakteristiku těchto jednotlivých oblastí, jak je definují Chandy et al. (2010, kap. 4 sek. 2-7).

3.3.2.2 Relevantnost generovaných reakcí

Při správné implementaci EDA je výhodou zvýšení pozornosti uživatelů na významné události v podniku, a potlačení těch nedůležitých či falešně kritických.

Rizikem nevhodné implementace je, že by uživatelé byli zahlceni irelevantními informacemi (Chandy a Schulte 2010, kap. 4 sek. 2). Dobrou praxí je umožnit uživatelům personalizaci systému do té míry, že by sami rozhodovali o relevantnosti jednotlivých typů událostí, které dostávají.

[13] Zkratka Unified Modelling Language: http://www.uml.org/

3.3.2.3 Vynaložené úsilí

Chandy et al. (2010, kap. 4 sek. 3) zmiňují potřebu vyvážit vynaložené náklady na uzpůsobení možností systému pro požadavky jednotlivých uživatelů. Personalizace systému zmíněná v předchozí kapitole zvyšuje náklady a dobu nutnou pro zavedení EDA. Z dlouhodobého hlediska je ale i ekonomickým přínosem, pokud mají uživatelé sami možnost řídit aplikaci a nedostávají nerelevantní události.

Dalším významným okamžikem z pohledu vynaloženého úsilí je fáze analýzy a návrhu EDA, kdy dochází ke konkrétním návrhům jednotlivých typů událostí a způsobu jejich zpracování v zaváděném systému. Ve své složitosti se dá tato činnost přirovnat k návrhu databázového modelu, a vyžaduje tedy podobnou míru znalostí a zkušeností.

Nesprávný návrh vede k dodatečným nákladům a prodlevám v realizaci systému, případně může vést až k provozním problémům, které by měly potenciálně ještě větší negativní dopad. Je zde tedy nutné volit správnou míru kompromisu mezi vynaloženým úsilím v průběhu zavádění, a výše uvedenými riziky. Zde je nutné poznamenat, že jedním z příslibů EDA je vyšší flexibilita změn. Je tedy možné uvažovat i o alternativních metodikách vývoje softwarového řešení, případně lze postupovat za užití běžných postupů jako je užití pilotního projektu.

3.3.2.4 Správnost informace a vyvolané reakce

Chandy et al. (2010, kap. 4 sek. 4) uvádějí, že aplikace dle EDA přináší výhody, pokud generují správné reakce. Cena za nepřesnost v průběhu života aplikace je pak odvozena z frekvence nepřesností a průměrných nákladů spojených s neadekvátní reakcí (na událost). Jednou z takovýchto ztrát je také ztráta důvěry zákazníka v případě nesprávné reakce (nově zavedeného) systému. Existují případy aplikací, kdy je jedna výjimečná ztráta s velkým dopadem významnější než spousta drobných ztrát. Jako prevenci těchto případů je vhodné budovat specifické kontrolní a schvalovací mechanismy, které detekují anomální události s velkým dopadem. Naopak běžné události jsou zpracovány napřímo.

Dvojí kontrola: Dalším typem jsou možné chyby ve vstupních datech, které mohou zapříčinit nežádoucí reakci. Obranou proti těmto jevům je vytvoření mechanismů dvojí kontroly v průběhu zpracování událostí. Výhodou je zvýšení správnosti zvolené reakce, nevýhodou pak zvýšení doby odezvy v provozu a zvýšené náklady při zavádění.

Akceptovatelná míra nepřesnosti: Dalším aspektem je určení míry nepřesnosti, která je z pohledu zamýšlených cílů ještě akceptovatelná. Toto je především zřejmé, pokud má systém detekovat vzácné události. Chandy et al. (2010, kap. 4 sek. 4) v tomto ohledu zmiňují, že množství nesprávných detekcí vzácných událostí bude větší než míra těch správných. S tím obecně souvisí míra falešných hlášení poplachů a náklady s tím spojené. Při příliš velkém množství falešných hlášení začne být aplikace ignorována jejími uživateli. Naopak při příliš malém množství hlášení se relativně prodražuje investice do její realizace (porovnáním s dosaženými přínosy v provozu).

3.3.2.5 Úplnost zpracovávané informace

Chandy et al. (2010, kap. 4 sek. 5) uvádějí jako významný faktor úplnost poskytnuté informace. Je možné rozlišovat falešné negativní hlášení a falešní pozitivní hlášení. V prvním případě nedojde k vytvoření události, a přitom k ní mělo dojít. V druhém případě naopak dojde k vytvoření události, a nemělo k ní dojít. Příkladem je informace o přetížení elektrické sítě. V prvním případě nedojde k vytvoření události, že je síť přetížená, a důsledkem může být výpadek celé sítě. V druhém naopak k vytvoření dojde, a důsledkem může být zbytečné omezení užití uživatelů.

Chandy et al. dále zmiňují, že cena za jedno chybějící hlášení – chybějící kritickou informaci – je typicky vyšší než cena za jedno falešné pozitivní hlášení. Typická je větší frekvence množství falešných pozitivních hlášení než frekvence chybějících (falešných negativních) hlášení. Chandy et al. dále zdůrazňují, že v případě falešných negativních hlášení (chybějící informace) nefunguje implementovaný mechanismus dvojí kontroly. Důvodem je absence informace, kterou by mohlo být možné zpracovat.

3.3.2.6 Včasnost zpracování událostí

Chandy et al. (2010, kap. 4 sek. 6) zmiňují, že efektivita reakce spočívá v její včasnosti. Hodnota reakce téměř vždy klesá se zvyšující se dobou odezvy. Pro různé typy událostí může být průběh této závislosti různý. Někdy je postačující doba odezvy v řádech minut, někdy nepostačují milisekundy. Míra snižování hodnoty v závislosti na době odezvy je dána vývojem nebezpečí či příležitostí, a náklady při nereagování na dané nebezpečí či příležitost.

Kompromis mezi včasností a správností: Je zapotřebí stanovit správnou míru, kdy ještě aplikace reaguje s co nejnižší dobou odezvy, ale zároveň poskytuje akceptovatelný poměr správných a nesprávných hlášení. Jako příklad uvádějí Chandy et al. systém na detekci a hlášení zemětřesení. Pokud by nebyla pozitivní událost hlášena včas z důvodu jejího delšího ověřování, tak by pozbyla svého významu. Pokud by ale docházelo k příliš častému falešnému hlášení o zemětřesení, tak by to mělo také své důsledky. Určení vhodné hranice je tedy složitou záležitostí.

Ve specifických případech je zde možnost řešení za užití prediktivních systémů, které reagují na situaci ještě dříve, než se stane kritickou. Je takto možné získat více dodatečného času na odpovídající reakci. Nevýhodou predikce událostí v budoucnosti je zvýšená míra v generování nesprávných a nekompletních údajů.

Další možností minimalizace vlivů je snaha o snížení času pro příchod a zpracování vstupních událostí. Příkladem je získávání vstupních událostí nejlépe ze systémů, ve kterých přímo vznikají (nevyužívání různých mezičlánků). Dalším příkladem je užití rychlejší infrastruktury pro výměnu zpráv, rychlejší procesory a efektivní užití paralelních systémů. Volbou technologií a návrhem EDA lze tedy pozitivně i negativně ovlivnit výsledný poměr mezi včasností a správností zpracovávaných údajů.

3.3.2.7 Bezpečnost – Různé pohledy

Chandy et al. (2010, kap. 4 sek. 7) zmiňují, že z pohledu bezpečnosti jsou EDA systémy velice důležité, jelikož jsou v nich realizovány systémy kritické pro chod daného podniku. Bezpečnost je jednou z největších překážek pro širší nasazení EDA aplikací.

Jednou z oblastí bezpečnosti je práce s osobními údaji a zachování soukromí zákazníků. Systémy pro zpracování událostí přitom pracují například s údaji o výskytu zákazníka, které získávají z mobilního telefonu. GPS a telemonitorovací zařízení v autech informují provozovatele o aktuální rychlosti vozu, což může mít vliv na výši pojistného. Nejrůznější událostí tedy mohou být analyzovány kýmkoli, kdo k nim má přístup, a to i s odstupem času.

Další oblastí bezpečnosti je obrana proti nejrůznějším útočníkům. Útočník se může vydávat za zákazníka před společností, či za společnost před zákazníkem. Útočník může vysílat falešné signály pro snížení ceny za odběr z elektrické sítě, případně se může pokusit

vyslat signál na odpojení domu od elektrické sítě (vydávat se za provozovatele elektrické sítě). Obranou je znalost vztahů mezi potenciálními útoky a vlastním systémem, a případně vhodné realizace úprav a nastavení systému.

Další formou útoků je útok zevnitř společnosti (insider attacks). Detekce útoku je zde ztížena skutečností, že útočník zná pravidla a algoritmy pro detekci útoků. Zvyšuje se zde důležitost forenzní analýzy, kde pro pochopení důvodů selhání je analyzován moment, kdy byl systém napaden. Aspektem je zde schopnost zjištění původu objektu události s popisem událostí, které byly užity pro jeho vygenerování, a znalostí procesu, který generování provedl.

Mezi významné faktory ovlivňující celkovou bezpečnost je zvyšující se míra (funkčního) propojení jednotlivých systémů uvnitř podniků i mimo ně. Jako důsledek může být řetězení chyb v propojených sítích, což může vést ke katastrofickým důsledkům. Při vhodné implementaci může EDA pomáhat zvyšovat nezávislost jednotlivých částí v takto propojené síti.

Cenou za všechna výše uvedená opatření jsou zvýšené náklady na realizaci a provozování systému založeného na událostmi řízené architektuře.

3.4 Pohled na základní vnitřní součásti EDA

Tato kapitola stručně představuje vybrané součásti užité při typické implementaci událostmi řízené architektury. Jednotlivé kapitoly charakterizují vybrané principy realizace v dané oblasti, a případně i představují konkrétní vhodné užití.

3.4.1 Infrastruktura pro přenos zpráv

Přenos zpráv je základním prvkem událostmi řízené architektury. Přenos může nabývat různých forem, ale typicky se jedná o asynchronní výměnu zpráv mezi jednotlivými komponentami EDA.

Taylor et al. (2009, s. 57) jako jednu ze základních součástí EDA zmiňují infrastrukturu pro přenos zpráv (messaging backbone), která umožňuje komunikaci jednotlivých součástí EDA mezi sebou. Tato infrastruktura je tvořena více částmi, které umožňují přenos za užití různých podporovaných přenosových protokolů a formátů zpráv.

Je založena na standardech či umožňuje přechod mezi vícero standardy pro přenos zpráv. Zároveň by měla být pronikavá (pervasive), tedy co nejvíce umožňovat přístup do jiných systémů, a zároveň být univerzálně dostupná. Infrastruktura pro přenos zpráv by dále měla být spolehlivá; levná na vývoj, údržbu a modifikaci; a zároveň umožňovat co největší oddělení producentů a konzumentů zpráv. Taylor et al. (2009, s. 57) dále zmiňují, že právě z důvodu nákladů na infrastrukturu selhalo mnoho iniciativ pro zavedení událostmi řízené architektury.

3.4.1.1 Charakteristika modelů pro výměnu zpráv

Typickým způsobem realizace přenosů zpráv v událostmi řízené architektuře je **publish-subscribe model** (Taylor et al. 2009, s. 57). Taylor et al. (2009, s. 116) uvádějí i další běžné způsoby výměny zpráv jako je synchronní přenos či asynchronní přenos typu point-to-point. Oba způsoby jsou ovšem nevhodné pro použití v EDA. Synchronní přenosy blokují prostředky na straně odesílatele i příjemce, a příjemce musí být navíc dostupný v době odesílání. To řeší asynchronní zpracování typu point-to-point, které se ale také nehodí pro použití v EDA, jelikož je zapotřebí doručovat stejnou zprávu více příjemcům.

Publish-subscribe model tento problém řeší, jelikož podporuje doručení stejné zprávy více příjemcům. Zpráva je nejprve uložena do repositáře s názvem JMS **topic**. Zpráva zůstává v repositáři i po vyzvednutí až do doby expirace či pročištění (purging). Konzumenti se přihlašují k JMS topic, a specifikují svůj zájem o aktuálně uložené zprávy.

Taylor et al. (2009, s. 120) zdůrazňují, že při užití publish/subscribe modelu je zapotřebí zvážit možné dopady neurčitosti dokončení transakce (transaction complete indeterminism).

3.4.1.2 Charakteristika technologií pro výměnu zpráv

Asynchronní přenos zpráv vyžaduje prostředníka či adaptér, který daný přenos zrealizuje (Taylor et al. 2009, s. 120). Taylor et al. dále uvádějí, že toho může být dosaženo užitím databáze, nativních konstruktů jazyka jako je Java Channels či nejběžnějšího poskytovatele této funkcionality: Message-oriented-middleware (MOM).

MOM je třída aplikací specifických pro správu spolehlivého přenosu zpráv. Seznam MOM produktů je možné nalézt v kapitole 3.8.1.

3.4.2 Charakteristika typů agentů pro zpracování událostí

Základní typy agentů pro zpracování událostí jsou agent pro filtrování a agent pro transformace (Etzion a Niblett 2010, s. 123–127).

3.4.2.1 Filtrovací agent

Úkolem agenta je kategorizovat události z pohledu nastaveného filtru, a nasměrovat je do vhodného výstupního kanálu. Agent může mít tři typy výstupů, na které mohou v EDA navazovat další prvky: událost prošla či neprošla podmínkou pro filtraci, anebo nebylo vůbec možné provést filtrování.

3.4.2.2 Transformační agent

Úkolem agenta je transformace událostí na jiné události. Může nabývat různých forem:

- Agent pro bezstavové zpracování jednotlivých událostí, a to pouze na základě dat v nich obsažených (podle stanovené formule).
- Agent pro obohacování či změnu dat obsažených v samotné události, a to na základě dotazů (typicky vně EDA).
- Agent pro projekci (redukci) dat obsažených ve zpracovávané události.
- Agent pro agregaci více událostí do jedné nové události.

Z výše uvedených typů agentů lze odvodit možné prováděné operace nad událostmi. Charakteristika jednotlivých typů zmíněných operací je součástí následujících kapitol.

3.4.3 Charakteristika kontextu události

Tato kapitola obsahuje stručnou charakteristiku a členění typů kontextů zpracování událostí. Etzion et al. (2010, s. 145) uvádějí následující definici kontextu: Kontext je pojmenovaná specifikace podmínek, které seskupují instance událostí tak, že mohou být zpracovány v určitém vztahu. Každá instance události je přiřazena jednomu či více oddílům kontextu (context partitions) a může nechat vzniknout jeden či více oddílů kontextu.

Kontext je jedním ze základních prvků událostmi řízené architektury. Je možné uvažovat více typů jednotlivých kontextů, a také různé politiky stanovující na základě jakých podmínek může automaticky vzniknout nový kontext.

3.4.3.1 Využití kontextu události

Událostmi řízená aplikace může využívat kontext pro následující tři hlavní účely (Etzion a Niblett 2010, s. 144):

a) Pro zpracování nelimitovaných proudů událostí, kde nelze čekat na doručení všech událostí. Zde je nutné rozdělit proud na sekvence oddílů kontextu (context partitions), které obsahují sadu vstupních událostí z proudu.
b) Proudy událostí mohou obsahovat události, které nejsou navzájem propojeny, ale mohou spolu souviset. Příkladem je snaha o zjištění počtu příchozích událostí konkrétního typu, kdy je zapotřebí vytvořit oddělené kontexty pro jednotlivé typy těchto událostí a nad nimi provádět danou agregační operaci. Pro tyto účely lze využít prostorový či segmentační kontext.
c) Kontext umožňuje jednotlivým agentům stát se citlivými. Vnitřní logika a chování daného agenta je ovlivněno právě zpracovávaným kontextem.

Z uvedeného vyplývá potřeba, aby každý oddíl kontextu byl zpracováván oddělenou instancí agenta. Pokud je agent stavový, každá jeho instance má svůj lokální stav. Události z různých oddílů kontextu jsou tak drženy a zpracovávány odděleně (Etzion a Niblett 2010, s. 145).

3.4.3.2 Základní typy kontextů událostí

Základní typy a podtypy kontextů dle Etziona et al. (Etzion a Niblett 2010, s. 147–175):

Dočasný kontext (temporal context) – Sestává z jednoho či více časových intervalů, které se mohou i překrývat. Každý časový interval odpovídá oddílu kontextu, který obsahuje události z daného časového úseku.

Další členění dočasného kontextu na podtypy dle způsobu určení daného časového intervalu:

- fixní časový interval (jednorázový či periodický),
- interval dle události (vznik a zánik kontextu je řízen predikáty či životností události) a
- kombinace předchozích variant s dodatečnými podmínkami pro velikost a počet zpracovávaných událostí.

Prostorový kontext (spatial context), který je závislý na lokaci výskytu události. Podtypy prostorového kontextu:

- s fixní lokací (závisí na prostorových entitách uvedených v události),
- závislý na vzdálenosti od dané entity (udané v rámci události či v rámci samotného kontextu) a
- závislý na lokaci výskytu dané události.

Segmentační kontext (segmentation-oriented context) je přidělen události na základě jedné či více hodnot údajů obsažených v rámci události. Pro vyhodnocení je možné užít dané hodnoty napřímo, či pomocí vyhodnocení daného predikátového výrazu.

3.4.4 Charakteristika operace filtrování událostí

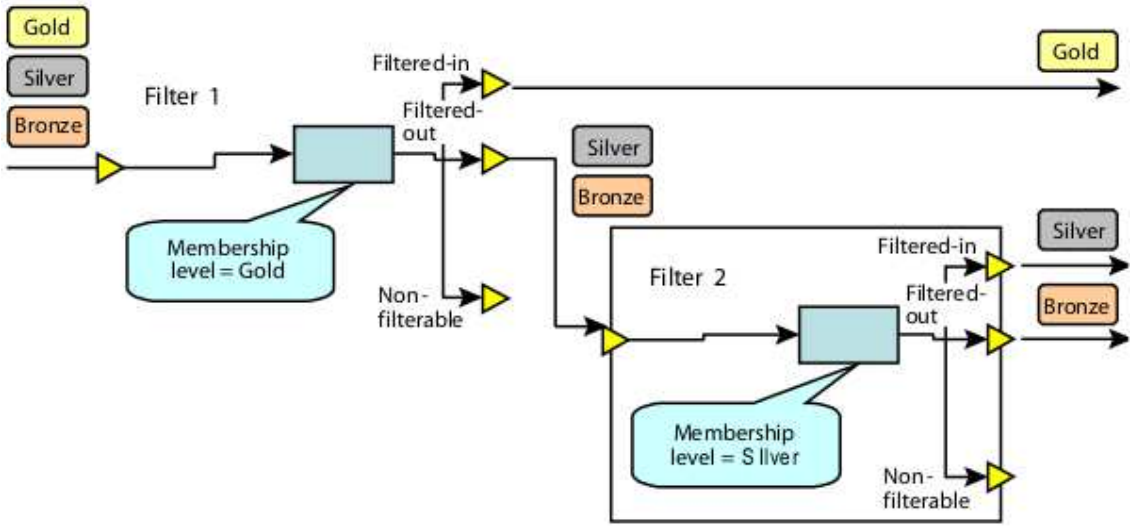
Filtrování událostí je operace, jejímž výsledkem je kategorizace zpracovávaných událostí podle stanovené podmínky či podmínek. Operaci filtrování lze dle Etziona et al. (2010, s. 177) provést:

- na vstupu do libovolného agenta či konzumenta událostí,
- jako součást definice konkrétního typu agenta (specifický typ agenta uvedený výše),
- jako součást definice kontextu událostí.

Výraz pro filtrování má formu predikátu, který je vyhodnocován vůči zpracovávané události. Událost prochází filtrem, pokud je predikát vyhodnocen kladně, a selhává, pokud je predikát vyhodnocen záporně (Etzion a Niblett 2010, s. 178). Etzion et al. dále zmiňují XPath jako příklad jazyka pro vyhodnocování podmínek.

Filtrování na vstupu do agentů a konzumentů událostí předpokládá bezstavovost událostí. Při potřebě zpracovávat události i za užití stavu, je možné užít specifický typ (filtrovacího) agenta určeného pro tento účel (Etzion a Niblett 2010, s. 183).

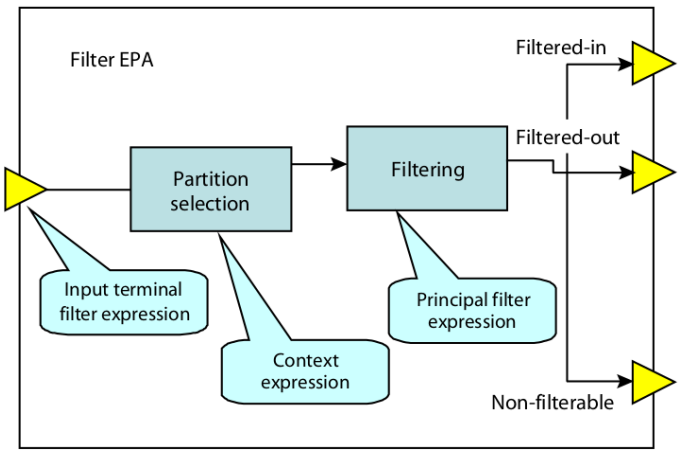
Následující obrázek představuje příklad užití dvou filtrovacích agentů, jejichž cílem je rozdělit jeden proud událostí do více proudů podle údajů obsažených v události (údaj o věrnostní kartě zákazníka).



Obrázek 2 - Příklad dvou filtrovacích agentů pro rozdělení proudu událostí do tří podproudů

Zdroj: (Etzion a Niblett 2010, s. 184)

Poslední jmenovanou možností je filtrování události podle aktuálního kontextu. Filtrování za užití kontextu typicky probíhá tak, že se nejprve zvolí konkrétní instance agenta, který má dané filtrování provést. Agent již poté pracuje v rámci daného kontextu (Etzion a Niblett 2010, s. 185–186). Průběh daného procesu je zobrazen na následujícím obrázku, reprezentujícím vnitřní strukturu filtrovacího agenta s podporou kontextu (výběr kontextu je proveden v rámci kroku “Partition selection”):



Obrázek 3 - Vnitřní struktura filtrovacího agenta s podporou běhu v kontextu

Zdroj: (Etzion a Niblett 2010, s. 186)

Možným užitím filtrovacího agenta s podporou stavů je vzorkování proudu událostí za účelem snížení množství přijímaných událostí (Etzion a Niblett 2010, s. 186).

Jako speciální případ filtrování lze uvažovat i operaci směrování (routing), kdy je událost beze změny přeposlána na požadovaný výstup podle stanovených podmínek.

3.4.5 Charakteristika operace transformace událostí

Operace transformace lze klasifikovat podle toho, zda jsou stavové či bezstavové, a podle počtu vstupů a výstupů (Etzion a Niblett 2010, s. 187). Rozlišujeme následující typy transformací: projekce (project), přeložení (translate), obohacení (enrich), rozdělení (split), agregace (aggregate) a složení (compose).

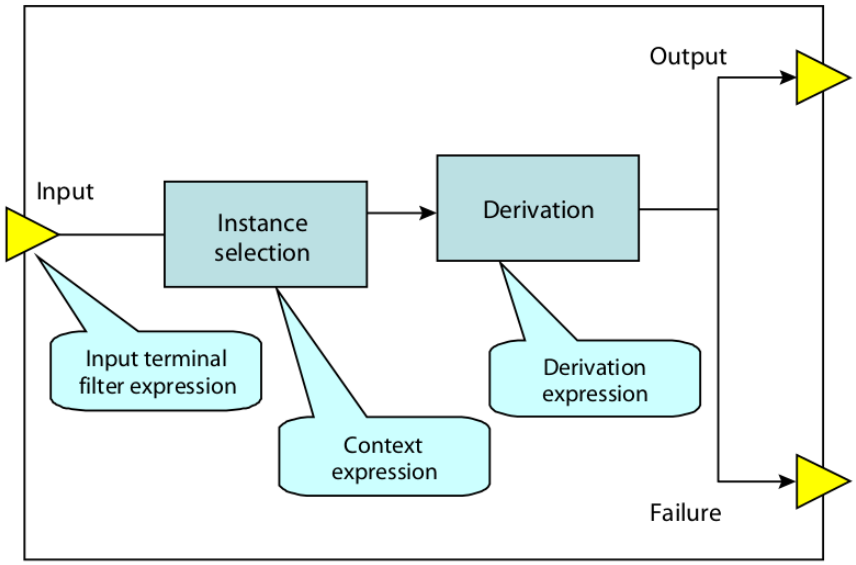
Pro realizaci samotné transformace mohou být užity například následující jazyky (Etzion a Niblett 2010, s. 190):

- Skriptovací jazyky jako jsou JavaScript, PHP a Perl, jejichž silnou stránkou je dobrá podpora pro práci s řetězci,
- Obecně použitelné programovací jazyk jako je Java, či
- XSLT[14].

3.4.5.1 Projekce, přeložení a obohacení

První tři jmenované typy transformací mají shodnou vnitřní struktury, tedy jeden vstup a jeden výstup a krok určující operaci, která je s událostí v transformačním agentovi prováděna. Vnitřní struktura těchto operací je znázorněna na následujícím obrázku:

[14] Extensible Stylesheet Language Transformations: Představuje jazyk pro transformaci XML dokumentů do jiných dokumentů. Spravuje organizace W3C.



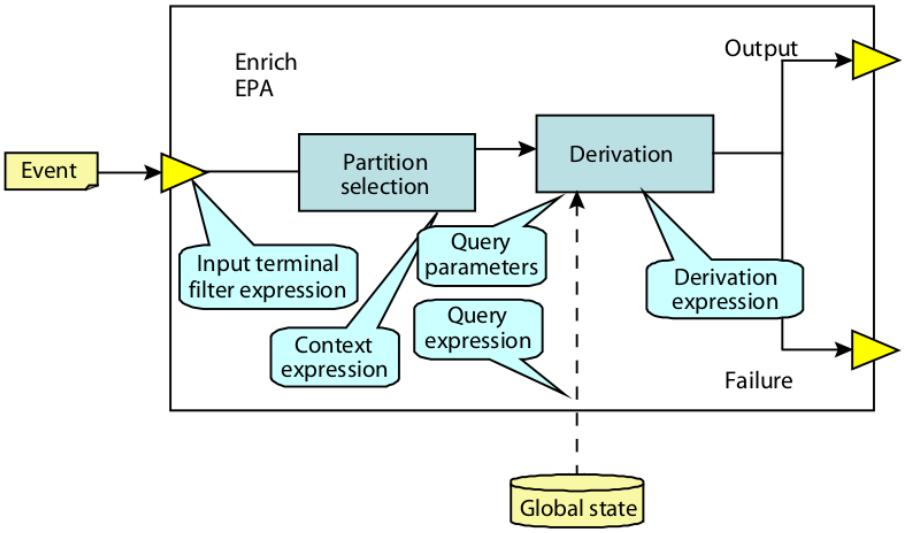
Obrázek 4 - Vnitřní struktura transformačního agenta s podporou běhu v kontextu

Zdroj: (Etzion a Niblett 2010, s. 188)

Transformační logika operace je prováděná v kroku s názvem Derivation (odvození) na základě vyhodnocení výrazu "Derivation expression" (Výraz pro odvození). V rámci vyhodnocení tohoto výrazu jsou přiřazeny hodnoty výstupní události za využití dat ze vstupní události (Etzion a Niblett 2010, s. 188). Etzion et al. dále uvádějí, že v případě operace:

- **projekce** je vybrána podmnožina dat vstupní události, která je použita pro výstupní událost.
- **přeložení (translate)** jsou data kopírována do výstupní události, ale může docházet k jejich modifikacím či vložení nových údajů. Výraz pro odvození specifikuje, jak jsou jednotlivé výstupní údaje vypočteny.
- **obohacení (enrich)** je výstupní událost složena podobným způsobem jako v případě přeložení (translate). Jako vstupní data ovšem jsou využita i data z jiných zdrojů (Global state element).

Následující obrázek znázorňuje kroky posledně uvedené operace:



Obrázek 5 - Vnitřní struktura transformačního agenta pro obohacení událostí s podporou běhu v kontextu

Zdroj: (Etzion a Niblett 2010, s. 192)

3.4.5.2 Rozdělení

Operace rozdělení (split) má na vstupu jednu událost, kterou rozloží do více událostí na výstupu (Etzion a Niblett 2010, s. 194). Typickým použitím je rozdělení složité zprávy události, kterou je zapotřebí zpracovat ve více konzumentech, přičemž každého konzumenta zajímá jiná část této původní zprávy. Případně může být užito pro rozdělení vstupní zprávy, která sloužila pro dávkové zpracování a obsahuje tak dávku jednotlivých událostí.

Operaci lze dále dělit na statické a iterativní rozdělování. Při statickém rozdělování je výstupem vždy stejné množství výstupních událostí, které jsou následně směrovány do jednotlivých výstupních prvků. Počet výstupních událostí je dán počtem výrazů pro odvození, který je ve statické formě rozdělování uveden. Statické rozdělování se odlišuje od přeložení (translate) tím, že právě obsahuje více výrazů pro odvození a tyto jednotlivé výrazy mají přiřazen konkrétní výstupní prvek (Etzion a Niblett 2010, s. 195).

Naproti tomu iterativní rozdělování obsahuje pouze jeden výraz pro odvození. Ten je napsaný v jazyku, který podporuje smyčky či jiný způsob zpracování, který umožňuje iteraci po jednotlivých částech původní vstupní události (Etzion a Niblett 2010, s. 196).

3.4.5.3 Agregace

Předchozí zmíněné operace transformace byly většinou bezstavové, a tedy jednotlivé vstupní události byly zpracovávány nezávisle. Naproti tomu operace agregace a následně charakterizovaná operace složení (compose) jsou stavové a výstupní události obsahují informace odvozené z jedné či více vstupních událostí. Jelikož jsou stavové, tak operují v konkrétním kontextu zpracování (Etzion a Niblett 2010, s. 196).

Etzion et al. dále uvádějí, že operace agregace je často využívána při zpracování proudů událostí. Operace na vstupu příjme události z (jednoho) proudu událostí, seskupí jednotlivé instance vstupujících událostí, a uloží je do příslušného oddílu kontextu (partition context). Jakmile dojde ke spuštění kroku pro odvození výstupní události, jsou tyto instance uložených událostí vhodně zpracovány a následně odstraněny z daného oddílu kontextu. K odvození výstupní události dochází dvojím způsobem, buď na základě signálu o uzavření konkrétního oddílu kontextu, anebo na základě vyhodnocení výrazu pro odvození.

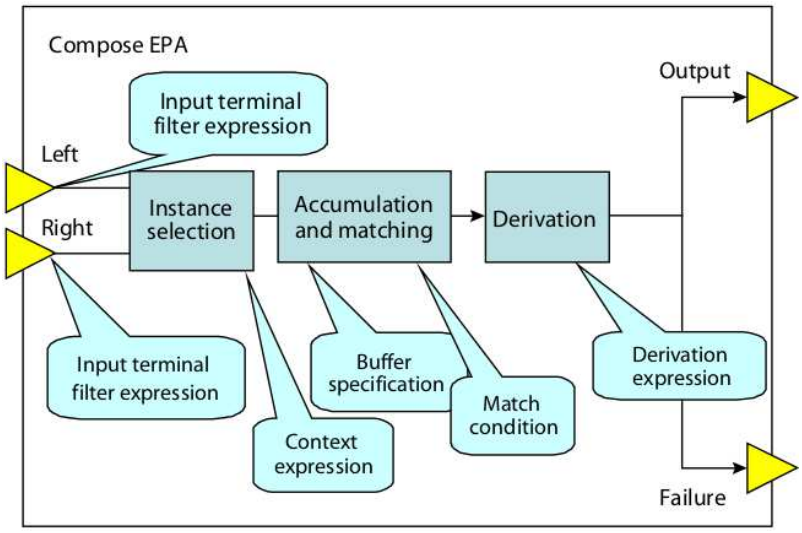
Při zpracování dat uložených ve vstupních událostech je možné užít agregačních funkcí při vyhodnocování dat pro výstupní událost. Příkladem mohou být funkce pro zjištění minimální, maximální či průměrné hodnoty; rozdělení podle unikátních hodnot; zřetězení hodnot a další (Etzion a Niblett 2010, s. 199).

3.4.5.4 Složení

Operace složení je podobná předešlé operaci agregace s tím rozdílem, že na vstupu má dva proudy událostí (Etzion a Niblett 2010, s. 199). Transformační agent realizující operaci složení páruje události pocházející z jednoho proudu s událostmi z druhého proudu, tedy realizuje operaci spojení (join).

Průběh zpracování probíhá tak, že příchozí událost z jednoho proudu událostí je nejprve zaslána správné instanci agenta na základě vyhodnocení vstupní filtrovací podmínky. Ten jednotlivé události akumuluje, a pokouší se je propojit s událostmi pocházejícími z druhého proudu událostí. Tento proces probíhá pokaždé, kdy je přijata nová událost (z libovolného ze dvou proudů událostí). Jakmile dojde k nalezení vhodných událostí vyhovujících stanovené podmínce pro propojení, je pro každý pár událostí spuštěn

krok odvození, a následně vygenerována jedna výstupní událost. Jednotlivé kroky jsou znázorněny na následujícím obrázku:



Obrázek 6 - Vnitřní struktura transformačního agenta pro složení událostí

Zdroj: (Etzion a Niblett 2010, s. 200)

Etzion et al. dále zmiňují, že krok akumulace a spojení je možné kontrolovat stanovením:

- velikosti vyrovnávací paměti pro jednotlivé vstupní proudy událostí (počtem či dobou uchování vstupních událostí),
- politikou pro jednotlivé vstupní proudy událostí v případech odstranění nepropojených událostí,
- podmínkou pro propojení událostí z obou proudů událostí.

Běžnou politikou při nepropojení událostí je pak zahození těchto událostí. Je ovšem možné také zvolit přístup, kdy je nepropojená událost postoupena dál ve zpracování a jako druhá událost je uvedena nulová událost (null event). Druhá událost tak představuje „outer-join" podmínku známou z rozšířené formy relační algebry. Další typickou možností je odeslání takovéto události do kanálu pro selhané události.

Při stanovení nulové velikosti vyrovnávací paměti na jednom proudu událostí se stává propojení událostí jednosměrné (one-way join). Události přicházející v tomto proudu jsou spojovány v akumulované události v druhém proudu, ne však opačně.

3.4.6 Stručná charakteristika detekce vzorů událostí

Jedna ze specifických možností při implementaci EDA je realizace procesů pro detekci vzorů událostí. To umožňuje celou řadu využití, která nejsou běžná v jiných typech architektur, alespoň ne při podmínce zpracování v reálném čase.

Vzor událostí je možné definovat jako předpis, specifikující jednu či více kombinací událostí. Pro libovolnou skupinu událostí je možné definovat jednu či více podmnožin událostí, které splňují daný vzor (Etzion a Niblett 2010, s. 216–217).

Detekce vzorů událostí je realizována specifickým typem agenta, který slouží právě pro daný konkrétní účel a je na něj optimalizován. Z tohoto důvodu jsou jednotlivé vzory kategorizovány, a jsou případně uvedena i jejich další dělení. Etzion et al. (2010, s. 214–236) uvádějí následující kategorie vzorů a jejich dalšího dělení:

- **Základní vzory** – Jedná se o jednoduché vzory vztahující se k základním operacím nad typy událostí či kolekcemi typů událostí. Dále je lze dělit na vzory:
  - s logickými operátory (příklady: all, any, absence),
  - s prahem (threshold; příklad: count, value max/min/average),
  - se selekcí podmnožin (příklad: relative n highest/lowest values), a
  - modální vzory (příklad: always, the sometimes).
- **Dimenzionální vzory** – Vzory vztahující se k času, místu či kombinaci času a místa. Lze je dělit na vzory:
  - dočasné (příklady: sekvence; prvních N; posledních N)
  - trendové (příklady: zvyšující se vzor; snižující se vzor; stabilní vzor)
  - prostorové (příklady: minimální/maximální/průměrná vzdálenost), a
  - prostorově-dočasné (příklady: pohybující se konstantně v jednom směru; pohybující se směrem k objektu; stacionární vzor).

Detekci vzorů událostí je možné vyhodnocovat **okamžitě** při přidání události do zkoumané množiny, či **s časovým odložením** až do doby kdy je dočasný oddíl kontextu agenta uzavřen (Etzion a Niblett 2010, s. 237). Existují však i další politiky pro vyhodnocování vzorů událostí než pouze na základě času zpracování. Například z pohledu omezení množství vytvořených zkoumaných množin (cardinality policies) či například

z pohledu toho, kolikrát může být jedna událost vyhodnocována v rámci dané zkoumané množiny událostí (consumption policies).

3.5 Pohled na implementaci EDA v podniku

Tato kapitola představuje různé pohledy na zavádění událostmi řízené architektury a její následné provozování v rámci podniku.

3.5.1 Kategorizace

Zavedení a rutinní užití událostmi řízené architektury je postupný proces, který vyžaduje postupné zapojování jednotlivých rolí v podniku a který přináší celou řadu problémů a omezení, na které je nutné brát zřetel. Úvodní fáze tohoto procesu souvisí s rozhodováním a určením způsobu, jakým bude EDA v dlouhodobém časovém období implementována. Často se zde prolínají vlivy konkrétních potřeb a očekávání managementu, stejně tak i architektonické představy o cílovém řešení. Role jednotlivých pracovníků účastnících se tohoto procesu tedy předurčuje, jakým pohledem budou nahlížet na celý proces implementace. Následující kapitoly analyzují jednotlivé aspekty ovlivňující výslednou podobu událostmi řízené architektury podniku.

3.5.2 Fáze zavádění EDA

Každé změně v oblasti informačních technologií v rámci podniku předchází rozhodnutí, zda danou změnu provést či nikoli. V případě prohlubování architektury integrace systémů se navíc jedná o takovou změnu, kde není očekáván okamžitý přímý obchodní přínos, ale naopak jsou zde očekávány snížené náklady z dlouhodobého hlediska.

Rozdělení uvedené v následujících odstavcích volně navazuje na postup uváděný Chandym et al. (2010, kap. 11) pro oblast EDA, a také Pulierem et al. (2006) pro oblast SOA.

3.5.2.1 Rozhodnutí o zavedení EDA

Zavedení událostmi řízení architektury v podniku je dlouhodobým strategickým rozhodnutím managementu, na které se váže i řada negativních dopadů. Impulzem pro takové rozhodnutí může být snaha docílit větší flexibility podniku (viz. kapitola 3.3.1 - Oblasti užití a možné přínosy). Mezi negativní dopady lze zařadit různá rizika spojená

s nutností komplexních změn v pracovních procesech podniku a dále pak riziko případného nenaplnění očekávání, či přímo riziko neúspěchu celého projektu.

Výstupem této fáze je strategické rozhodnutí managementu o zavedení EDA včetně stanovení hrubého plánu. Plán rozdělí implementaci do jednotlivých etap a fází, kde hlavním cílem první (pilotní) etapy je snaha identifikovat a eliminovat negativní dopady rozhodnutí, ověřit životaschopnost projektu, a následně upřesnit návratnost dále investovaných prostředků.

3.5.2.2 Základní analýza potřeb a stanovení cílů

Pro úspěšnou implementaci je zapotřebí nejprve provést analýzu potřeb, problémů a cílů podniku jak v oblasti IT, tak i v oblasti zaběhlých pracovních procesů a postupů. Při návrhu EDA je nejprve nutné identifikovat požadavky na EDA jako celek a ujasnit si jednotlivá očekávání, a následně stanovit požadované cíle.

3.5.2.3 Určení základní architektury a principů realizace

Ze stanovených cílů pro realizaci EDA by měly být patrné základní obchodní požadavky na EDA, z nichž lze odvodit požadavky technické (frekvence událostí a jejich velikost a komplexností, požadovanou kvalitou služeb, doba odezvy, apod.).

Na základě těchto technických údajů lze učinit základní rozhodnutí o architektuře, která by měla být schopna stanovené (technické) cíle splňovat. V některých případech bude vhodnější volit přímou transformaci vstupních dat (například z důvodu nutnosti rychlejšího zpracování), v jiném případě může zase být užit ESB pro mediaci zpráv a sjednocení vstupních i výstupních formátů. Další variantou je napojení na SOA s ohledem na vizi ještě větší flexibility podniku.

V této fázi je zapotřebí brát zřetel na možná rizika, jak je uváděno v kapitole 3.3.2 (Možná rizika a omezení), a učinit základní stručná rozhodnutí o způsobu jejich řešení.

3.5.2.4 Volba produktů a technologií pro implementaci

Lze zvolit různé technologie a produkty pro implementaci EDA. Takové rozhodnutí však musí brát v úvahu situaci v konkrétním podniku, jelikož je nutné jak minimalizovat ekonomické a organizační dopady, tak i naplnit deklarovanou vizi managementu.

Jednotlivá rozhodnutí lze učinit na úrovni řízení projektu. Rozhodnutí s významnými dopady by ovšem měla být odsouhlasena také managementem.

Mezi významná kritéria rozhodování lze zařadit omezení daná investičním záměrem, stávajícím IT prostředím podniku, a v neposlední řadě vědomostmi a schopnostmi zaměstnanců.

Chandy et al. (2010, kap. 9 sek. 2) uvádějí, že pro úspěšné zavedení událostního zpracování je vhodné získat či vyškolit zaměstnance na zvolené produkty a technologie, případně najmout externí specialisty v dané oblasti. Dále pak jako vhodné zmiňují zvolení takových produktů, které je možné snadno integrovat s událostmi řízenou architekturou.

3.5.2.5 Pilotní projekt

Kritickou fází je realizace pilotního projektu, který má za cíl ověření procesních a technologických předpokladů, a zároveň identifikaci různorodých rizik a negativních dopadů.

Z tohoto důvodu je nutné nejprve vybrat vhodnou oblast podniku, na které bude možné uvedené předpoklady a dopady ověřit. Je vhodné volit z takových oblastí, kde nedochází k přílišným vnějším vlivům a které nejsou příliš komplexní. To usnadní identifikaci problémů souvisejících pouze s pilotním projektem, a zároveň usnadní průběžnou interpretaci výsledků. Nevhodné jsou tedy ty části podniku, které jsou zatíženy akutními problémy či častou potřebou změny. Následujícím krokem je vytvoření architektury a designu nového systému pro zvolenou oblast a vlastní implementace.

Závěrem pilotního projektu je jeho vyhodnocení, které může vést i k rozhodnutí o zastavení zavádění EDA v podniku.

3.5.2.6 Postupná implementace

Implementace událostmi řízené architektury není jednorázová operace, ale dlouhodobý proces, který v prvních fázích postupně převádí jednotlivé části IT prostředí podniku na paradigma událostního řízení. Především se zde jedná o postupné napojování jednotlivých existujících IT systémů jako producentů událostí, a následně i konzumentů událostí produkovaných v rámci EDA.

Je sice vhodné postupovat po jednotlivých krocích přesně dle stanoveného plánu, ale zároveň je nezbytné reagovat na nové situace a změny v okolním prostředí. Řízení pomocí událostí poskytuje dodatečnou úroveň flexibility, je tedy možné dlouhodobý plán postupně rozšiřovat či měnit.

3.5.3 Identifikace vhodných oblastí pro realizaci

Chandy et al. (2010, kap. 11 sek. 1) uvádějí následující postup při stanovování cílů, který by měl maximalizovat přínosy při budování EDA v podniku:

- **Identifikace aplikací vhodných pro událostní zpracování**: Je vhodné zvolit systémy, které mohou ze zavedení EDA nejvíce profitovat - s ohledem na přínosy a rizika zmíněná v kapitole 3.3 (EDA z pohledu využitelnosti a rizik). Je nejprve vhodné zvolit aplikaci, kde dochází k průběžným a rutinním reakcím na události.
- **Identifikace uživatelských komunit**: Různé role pracovníků podniku jsou krátkodobě či dlouhodobě zasaženy zavedením EDA. Vhodné je zaměřit se na skupinu, kde automatizace zpracování událostí bude mít největší obchodní přínos. Vhodné je zapojit nejprve interní pracovníky. Lze tak získat zpětnou vazbu, a demonstrovat managementu návratnost učiněné investice (ROI).
- **Identifikace scénářů a reakcí**: Je vhodné identifikovat a zvolit scénáře a následné reakce, které jsou pro vybranou skupinu uživatelů nejvíce cenné z obchodního hlediska.
- **Identifikace zdrojů dat**: Identifikace potřebných zdrojů dat pro realizaci vhodné reakce ve zvolené aplikaci, a to pro zvolenou uživatelskou komunitu a scénář.
- **Identifikace událostí a transformací dat**: Při snaze o sestavení konkrétní potřebné výstupní reakce může být identifikována potřeba dalšího potřebného vstupního údaje (události), který doposud nebyl k dispozici.
- **Odhad nákladů a přínosů, a vytvoření plánu do budoucnosti**: Odhadnout přínosy a náklady EDA v podniku z dlouhodobého hlediska je složité, jelikož postupně dochází k propojování různých divizí podniku. Je proto vhodné demonstrovat obchodní přínosy EDA postupně v jednotlivých etapách.

3.5.4 Možnosti návrhu událostmi řízené architektury

Následující kapitoly představují různé kombinace možností, které je možné zvolit při návrhu událostmi řízené architektury. Volba těchto možností musí především vycházet ze zvolených cílů pro implementaci EDA v podniku. Při implementaci komplexních aplikací s nutností větší flexibility je z dlouhodobého hlediska například vhodnější volit spíše systémy s užitím volné vazby, jak je naznačeno v následující kapitole.

3.5.4.1 Kategorizace podle užité vazby mezi komponentami

Etzion et al. (2010, s. 34–35) uvádějí, že koncept EDA je založen na principu, kdy producenti a konzumenti událostí jsou navzájem co nejvíce odděleni (s volnou vazbou). Producent událostí neočekává konkrétní reakci od konzumenta událostí, a zároveň nevyžaduje žádnou odpověď či potvrzení, že k nějaké reakci vůbec došlo. Konzument událostí naopak není závislý na zpracování na straně producenta událostí, pouze na samotné vyvolané události. Etzion et al. obě tyto vlastnosti pojmenovávají principem oddělení (Principle of decoupling).

Taylor et al. (2009, s. 59) dělí EDA na následující typy:

- **Implicitní typ EDA**: Implicitní typ je podobný modelu jak jej uvádějí Etzion et al.. Je více flexibilní a dynamický. Implementace tohoto typu byla dříve příliš složitá, což se mění se zaváděním otevřených standardů. Jedná se o agilnější přístup, ovšem je složitější predikovat a ověřovat přínos (dopředu není zřejmě, jak bude událost zpracována; není možné ověřit původce události).
- **Explicitní typ EDA**: Je reprezentován stylem, kdy jsou producenti a agenti úzce propojeni. Producenti událostí zasílají zprávy předem známým agentům (event listener), nezřídka s napevno určenou cílovou adresou. V současnosti je tento přístup alespoň částečně využíván (Taylor et al. 2009, s. 59).

Volba typu realizace EDA podle uvedeného členění má velký vliv na dlouhodobou strategii a rozvoj EDA v podniku, a to především z pohledu očekávané flexibility či naopak výkonnosti v provozu. Při zahájení realizace EDA by tedy mělo dojít k volbě jednoho z uvedených typů i s ohledem na dlouhodobé cíle podniku.

3.5.4.2 Kategorizace podle způsobu zpracování událostí

Taylor et al. (2009, s. 59) dále představují a charakterizují níže uvedené tři základní vzory pro zpracování událostí. Jednotlivé uvedené vzory je možné užít nezávisle či kombinovaně podle stanovených cílů realizace.

Zpracování jednoduchých událostí - Nejjednodušší forma EDA, kdy smyslem producenta událostí je generovat zprávy a zasílat je agentům. Agenti provedou následně jejich zpracování vhodným způsobem.

Zpracování proudů událostí (Event stream processing) - Způsob podobný předešlému. Agent přijímá zprávy od producentů událostí, reaguje ale pouze pokud je splněna daná podmínka pro zpracování. Při nesplnění podmínky jsou zprávy agentem ignorovány.

Zpracování komplexních událostí (Complex event processing – CEP) - Jedná se o nejsložitější způsob implementace EDA, kdy je zapotřebí reagovat na více událostí za užití více podmínek. CEP zpracovává více proudů událostí zároveň, navzájem je propojuje dle stanovené logiky daných podmínek, a ty následně vyhodnocuje. CEP tedy může průběžně sbírat data z nesouvisejících zdrojů, rozlišovat mezi užitečnými a zbytečnými informacemi, a reagovat odpovídajícím způsobem.

CEP ovšem může být využit i dalším způsobem. Agent může určovat události, které způsobily daný přínos. Agent monitoruje výskyt konkrétního přínosu, a poté odvozuje kauzalitu události či událostí, které jej zapříčinily. Toto je vysoce cenné při prokazování hypotetických situací prostřednictvím simulace[15].

3.5.4.3 Kategorizace podle významu jednotlivých událostí

Mezi další faktory tvorby IT systémů dle EDA lze zařadit níže uvedené členění zpráv (notifikací, objektů událostí), jak je uvádějí Chandy et al. (2010, kap. 8 sek. 1-3). Opět zde platí, že je možné při realizaci zvolit vhodnou kombinaci uvedených typů zpráv.

[15] Taylor et al. (2009, s. 59) zde uvádějí přirozenou vazbu na systému umělé inteligence, která rozpoznává vzory informací a navrhuje reakce na vzory událostí. Taylor navrhuje tuto synergii využít při stanovování obchodních pravidel a pro automatizaci rozhodování za užití CEP.

Transakční notifikace (Transactional notifications) – Typicky generované obchodními aplikacemi uvnitř společnosti, dodavateli, zákazníky či jinými zadavateli. Tyto zprávy informují o události, která mění stav v okolních systémech a případně generuje další změny. Jedná se o zprávu, která je důležitá a jedinečná, a je nutné zajistit její bezpečné doručení. Jako příklad Chandy et al. zde zmiňují zahájení a postupné zpracování koupě akcií na burze, kdy každá akce musí vyvolat v cílovém systému konkrétní operaci dle ve zprávě zaslaných údajů.

Pozorovací notifikace (Observational notifications) – Typicky používané při modelování komplexních událostí, které pouze sledují různé zdroje příchozích událostí. Jednotlivé události nemusí být v takovém případě významné. Pokud dojde k nedoručení jedné sledované události, tak nemusí být provedeno její opětovné znovu-doručení později. K ovlivnění sledované komplexní události by mohlo ovšem dojít v případě, že by došlo ke ztrátě mnoha příchozích zpráv. Většinou jsou tyto notifikace řádově menší než v případě transakčních notifikací. Není zde nutné přenášet veškeré detailní údaje o kontextu události (například přesné údaje o kupujícím), ale postačí informace obecnějšího charakteru (př.: došlo ke koupi akcii pro daný titul a v daném objemu).

3.5.4.4 Kategorizace podle způsobu užití komplexních událostí při zpracování

Níže je uvedeno členění dle Chandyho et al. (2010, kap. 8 sek. 4) z pohledu využití komplexního zpracování událostí v obchodních aplikacích. Pro jednotlivé způsoby realizace jsou zde uvedeny také vybrané typové příklady možného užití.

Subsystémy pro předzpracování proudů událostí – Jsou užity při zpracování proudů událostí s velkým objemem, kde dochází k filtrování dle stanovených podmínek. Takto získané události jsou následně určeny pro periodické analytické aplikace či pro průběžné (ne real-time) kontrolní aplikace. Jedním z možných užití tohoto přístupu je zmenšení objemu dále zpracovávaných dat, a to odstraněním redundantních či jinak nerelevantních dat. Dalším možným užitím je naopak obohacení zpracovávaných dat z dalších dostupných zdrojů, aby nemusely být obohacovány později v procesu zpracování.

Příkladem je zařízení pro čtení RFID dat, které odstraňuje duplicitní načtené údaje před jejich odesláním k dalšímu zpracování.

Jednoúčelové (pure-play) sledovací systémy - Sbírají pozorovací notifikace ze senzorů a jsou málo či vůbec integrovány na okolní systémy. Samotné sledování je výslednou aplikací.

Příkladem je systém pro varování před tsunami, kde systém jednoúčelově zpracovává příchozí událostí z bójí umístěných na různých lokacích, a vysílá varování s predikcí o čase, místě a rozsahu předpokládané vlny tsunami.

Sledování homogenních systémů – Zařízení na bázi CEP pro usnadnění správy aplikačního systému, sítě, IT subsystému, výroby či jiného systému.

Příkladem je sledování konkrétního IT systému z nejrůznějších hledisek od zdraví, propustnosti, výkonu, dodržení SLA či z pohledu dalších jiných metrik. Další příklady jsou sledování výkonu výrobních linek továrny, elektráren či dalších podobných zařízení. Jsou především detekovány anomálie či změny stavů, které mohou být významné.

Sledování heterogenních systémů – Aplikace na bázi CEP, které sledují dva či více autonomních systémů. Lze užít v případech sledování komplexních aktivit či end-to-end procesů, které mají více heterogenních zdrojů událostí a které je relativně těžké implementovat.

Příkladem je řízení dodavatelského řetězce (supply chain management - SCM), kde existuje mnoho autonomních účastníků daného procesu. Každý z těchto účastníků má vlastní aplikační systém a rozhoduje se relativně nezávisle. Proces nemůže být predikován především s ohledem na nenadálé události, které mohou nastat z nejrůznějších příčin (výpadky výroby u dodavatele, počasí, zpoždění s dopravou, apod.). Žádný procesní model či BPM proces není schopen pokrýt všechny možné eventuality.

Z pohledu návrhu EDA je SCM systém složen především z pozorovacích notifikací, ze kterých jsou skládány komplexní události umožňující lepší náhled na přesuny materiálu a zboží. Nejtěžší částí je domluva, implementace a údržba zachytávání událostí pocházejících z různorodých IT systémů od různých obchodních partnerů.

Aplikace s podporou CEP – Primární aplikace na zpracování transakčních notifikací, které jsou vyvolány komplexními událostmi či užívají komplexní události jako své vstupy (k vyvolání dalšího zpracování).

Příkladem jsou obchodní aplikace pro řízení provozu letecké společnosti, kde je zapotřebí zpracovávat tisíce příchozích událostí za sekundu a na jejich základě efektivně vyhodnocovat a realizovat další akce. Tento systém je užit pro řízení časů letů, cateringu, posádek letů, doplňování paliva, údržby letadla či jednotlivých informací souvisejících s konkrétními lety.

3.5.4.5 Kategorizace typů producentů a konzumentů událostí

Etzion et al. (2010, s. 91–93,104–108) uvádějí, že je možné jednotlivé producenty a konzumenty kategorizovat do následujících základních skupin:

- Hardwaroví producenti či konzumenti událostí,
- Systémy s lidskou interakcí a
- Softwaroví producenti či konzumenti události.

Pro jednotlivé kategorie lze následně očekávat různé chování při práci s událostmi, či je možné předem očekávat lepší či horší možnosti z pohledu integrace s EDA systémem.

3.5.4.6 Kategorizace podle užití stavů při zpracování

Další variantou při návrhu EDA je možnost užití či neužití stavů při zpracování jednotlivých událostí, a to v kombinaci s užitím vhodných operací nad danou množinou událostí. Pro bezstavové zpracování je například typické užití operací typu filtrování, směrování a logování událostí. Pro stavové naopak různé formy transformace událostí, včetně rozdělení události na více událostí, agregace událostí do jedné, či slučování více proudů událostí do jednoho výsledného (Etzion a Niblett 2010, s. 44–45).

3.5.4.7 Nefunkční požadavky

V rámci přípravy implementace je vhodné vyhodnotit, jaké jsou nefunkční požadavky na realizaci. Mezi tyto lze zařadit škálovatelnost, dostupnost a bezpečnost (Etzion a Niblett 2010, s. 265). Existuje však celá řada dalších, které by bylo vhodné při rozhodování uvažovat, například spolehlivost, použitelnost, udržovatelnost, snadnost testování a mnohé další.

Etzion et al. (2010, s. 265–266) uvádějí různé dimenze z pohledu škálovatelnosti. Mezi ně řadí objem zpracovávaných událostí, množství agentů, producentů, konzumentů a

kontextů událostí. Dále pak složitost a náročnost výpočtů a v neposlední řadě prostředí, které události zpracovává. V jednotlivých oblastech je nutné se především zamyslet nad možnými okolnostmi, které by mohly mít vliv na škálovatelnost.

Z pohledu dostupnosti systému je nutné uvažovat nejenom procentuální dobu, kdy je platforma dostupná, ale také další důsledky případných výpadků či odstávek systému (Etzion a Niblett 2010, s. 267). Etzion et al. zmiňují jako možné řešení zavedení podpory přesné obnovy systému do stavu před výpadkem. To lze realizovat například průběžným logováním při zpracování jednotlivých událostí, ovšem za cenu prodloužení doby zpracování a celkové průchodnosti událostí. Pro některé aplikace nemusí ovšem být zavedení této možnosti nákladově efektivní či vůbec smysluplné, například pro systémy založené na statistickém zpracování.

Jako další nefunkční požadavek zmiňují Etzion et al. (2010, s. 267) bezpečnost. Kromě již uvedených rizik v kapitole 3.3.2.7 lze zmínit například i potřebu řešit autorizaci pro produkci či konzumaci událostí, vhodnou filtraci nevalidních událostí na vstupu i výstupu, auditování doručených a zpracovaných událostí či auditování aktivit prováděných samotným systémem. V neposlední řadě může být nutné zabezpečit také všechny databáze a komunikační trasy užívané systémem. Opět zde ale platí, že zavedení uvedených opatření má za důsledek zvýšení doby odezvy systému při zpracování jednotlivých událostí.

3.5.5 Návrh událostmi řízené architektury

Tato kapitola představuje možný postup tvorby návrhu událostmi řízené architektury.

3.5.5.1 Obecný postup

Pro zavádění EDA v podniku je zapotřebí provést řadu kroků. Především je nutné identifikovat cíle pro danou etapu zavádění, jak je uvedeno v předešlých kapitolách. Na základě těchto cílů je vhodné následně provést základní návrh samotné architektury. Je nejprve zapotřebí provést napojení jednotlivých producentů událostí na infrastrukturu pro přenos zpráv, a zároveň i vytvořit podmínky, které zajistí, že bude EDA schopna vyvolávat reakce v jednotlivých systémech pro konzumaci událostí. Jedna z vhodných možností jak

výše uvedené realizovat je uvedena v kapitole 3.6 (Vztah k servisně orientované architektuře).

Dále je zapotřebí stanovit vhodné kombinace možností, které budou při návrhu architektury využity. Volba vhodných kombinací uvedených možností závisí především na stanovených cílech pro danou etapu zavádění. Přesto je vhodné již dopředu uvažovat o dlouhodobém využití jednotlivých možností. Je proto vhodné stanovit návrhové standardy a zvolit užité technologie, které se s postupem času nebudou příliš měnit.

Jednotlivé možnosti pro návrh architektury a volbu technologií je možné nalézt v kapitolách:

- 3.4 - Pohled na základní vnitřní součásti EDA,
- 3.5.4 - Možnosti návrhu událostmi řízené architektury a
- 3.7 - Technologie.

V neposlední řadě je vhodné se zaměřit na zvolení jednotlivých produktů, kde lze vhodnou volbou dosáhnout celkového usnadnění zavádění událostmi řízené architektury. Seznam vybraných produktů pro relevantní oblasti je uveden v kapitole 3.8 (Produkty).

Následující kapitoly představují vybrané pohledy na způsob návrhu událostmi řízení architektury.

3.5.5.2 Návrh architektury sítě pro zpracování událostí

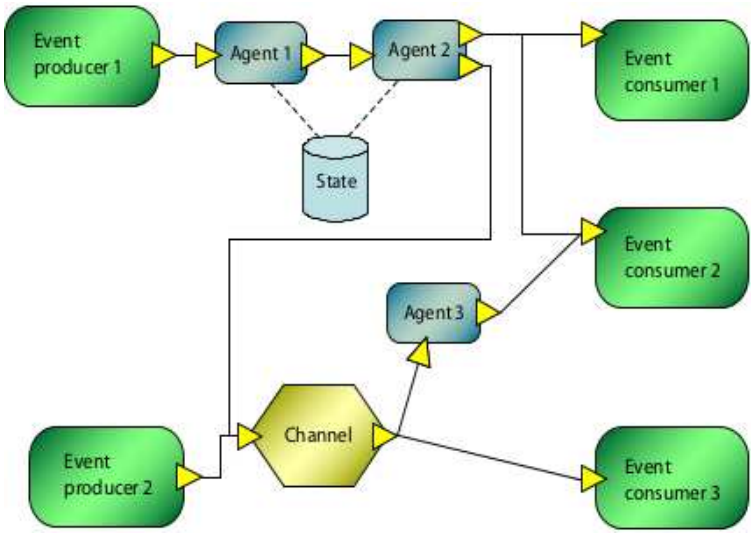
Základním možným pohledem při návrhu EDA je představa budoucí architektury pomocí **sítě pro zpracování událostí** (Event processing network - EPN). Etzion et al. (2010, s. 116–119) uvádějí, že EPN je tvořena producenty a konzumenty událostí, agenty, kanály událostí, a v neposlední řadě také prvky udržujícími globální stav událostí (Global state elements).

Poslední jmenovaná část slouží při potřebě udržení stavu či kontextu zpracování událostí, a může nabývat různých forem (Etzion a Niblett 2010, s. 138–140):

- Historické události držené pro zpracování v pozdější fázi.
- Referenční údaje, ke kterým mohou agenti přistupovat, například z důvodu nutnosti obohacení či očištění údajů přenášených ve zpracovávané události. Samotné referenční údaje nejsou drženy v systému na zpracování údajů.

- Stavy externích entit, které také nejsou drženy v systému na zpracování událostí, ale které mohou být použity při zpracování událostí. Například stav zpracování konkrétního obchodního procesu, vyhlášený stav pohotovosti či například aktuální stav počasí. Tento stav se může měnit jako důsledek zpracování události konzumentem událostí.
- Stav zpracování události (uložený na disku či ve sdílené paměti), který je dostupný mezi agenty a který může být aktualizován aplikací na zpracování událostí.

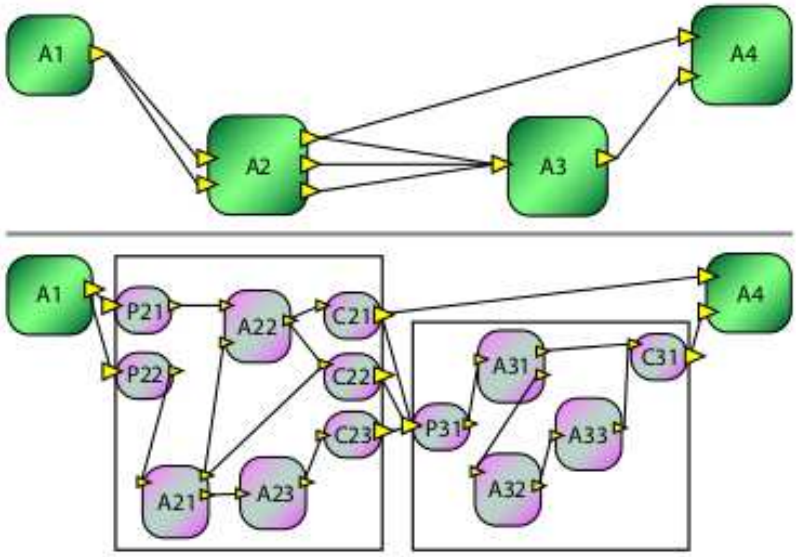
Následující obrázek představuje příklad **sítě pro zpracování událostí** s užitím jednotlivých hlavních typů prvků (komponent):



Obrázek 7 - Příklad sítě pro zpracování událostí

Zdroj: (Etzion a Niblett 2010, s. 116)

Dále je možné modelovat také složitější struktury s **vnořenými sítěmi pro zpracování událostí** (Nested event processing network). Následující obrázek představuje příklad takového uspořádání, kde v horní části je uvedena EPN se čtyřmi uzly, přičemž uzly A2 a A3 obsahují vnořené EPN. Spodní část potom zobrazuje plnou síť s rozšířeným zobrazením těchto vnořených EPN uzlů.



Obrázek 8 - Příklad užití vnořených sítí pro zpracovní událostí

Zdroj: (Etzion a Niblett 2010, s. 119)

Modelování EPN využívá pouze prvků (komponent), které jsou nezávislé na zvolené platformě, a které jsou abstrakcí charakterizující funkční chování systému pro zpracování událostí (Etzion a Niblett 2010, s. 119).

Užití modelování EPN není nutnou podmínkou při implementaci EDA systémů. Výhodou jejich užití je ale větší přehlednost a schopnost pochopení modelu, možnost validace sítě užitím různých analytických technik (detekce problémů typu ukončení, nedostupných uzlů či nedeterministického chování) a také možnost zkoumání různých výkonnostních optimalizací (Etzion a Niblett 2010, s. 121).

3.5.5.3 Návrh komplexních událostí

Předpokladem této kapitoly je již identifikovaná potřeba vytvoření konkrétní komplexní události. Je tedy dopředu znám konzument dané komplexní události, a jsou také identifikovány konkrétní údaje, které daný konzument na svém vstupu vyžaduje.

Nejprve je zapotřebí identifikovat podmínky, za kterých může daná komplexní událost nastat. To vyžaduje detailní znalost procesů podniku a možností potenciálních producentů vstupních událostí. Pro zjednodušení bude v dalším textu předpokládáno, že jsou potřebné vstupní události již identifikovány a dostupné pro zpracování v EDA systému.

Při tvorbě komplexních událostí je nutné vždy stanovit způsob, jakým bude daná komplexní událost detekována. Za tímto účelem je možné zvolit různé návrhové vzory pro detekci (komplexních) událostí. Základní výčet těchto vzorů je uveden v kapitole 3.4.6 (Stručná charakteristika detekce vzorů událostí) výše. Nutnost užití konkrétních vzorů pro detekci událostí implikuje nutnost užití konkrétních kombinací možností EDA[16].

Následuje návrh vnitřní struktury výstupní události (reakce na komplexní událost), kde jednotlivé údaje události jsou složeny ze vstupních událostí, či získány jiným způsobem (například obohacením z jiných zdrojů dat).

3.5.5.4 Modelování hierarchie komplexních událostí

Chandy et al. (Chandy a Schulte 2010, kap. 7 sek. 3) uvádějí, že jednotlivé vztahy mezi komplexními událostmi a základními událostmi mohou být reprezentovány pomocí **hierarchie událostí**. Komplexní událost je na vyšší úrovni této hierarchie než události, které ji generovaly. Luckham et al. (2011) uvádění následující definici hierarchie událostí: Model, který představuje vztah mezi událostmi, které jsou na různých úrovních abstrakce k sobě navzájem.

Při modelování hierarchie událostí je tedy zapotřebí vycházet z událostí, které danou komplexní událost generují. To implikuje, že není vhodné cyklicky využívat události ze stejné či vyšší úrovně abstrakce, jelikož by mohlo dojít (v dlouhodobém časovém horizontu) k cyklickým závislostem a z toho plynoucím konfliktům.

3.6 Vztah k servisně orientované architektuře

Tato kapitola představuje pohled na vztah mezi událostmi řízenou a servisně orientovanou architekturou. Pro pochopení vztahů je nejprve zapotřebí uvést základní charakteristiku a definici servisně orientované architektury. Následně je uvedena specifická forma kombinující oba přístupy: Událostmi řízená SOA.

Přístupy EDA a SOA jsou vzájemně komplementární a jejich vhodnou kombinací lze dosáhnout vyšší celkové flexibility podniku. Cummins (2009, s. 208) mezi důležité

[16] Možnosti EDA jsou uvedeny v kapitolách 3.4 (Pohled na základní vnitřní součásti EDA) a 3.5.4 (Možnosti návrhu událostmi řízené architektury).

důvody užití obou přístupů řadí, že SOA samotná není schopna identifikovat a reagovat na nezvyklé události. Pomocí EDA je naopak možné skládat běžné události do komplexních, ty následně analyzovat a vyhodnocovat. Tato metoda se nazývá Complex-Event Processing (CEP) a skládá se z průběžného monitorování, reportování, zaznamenávání a filtrování jednotlivých událostí. Kombinace určitých zpracovávaných událostí může představovat událost, která je z pohledu dalšího zpracování zajímavá. Následně lze uplatnit flexibilitu SOA pro rychlé upravení či doplnění existujících business procesů v podniku tak, aby byly schopny standardně řešit i nově detekované situace.

3.6.1 Servisně orientovaná architektura

Tato kapitola navazuje na základní představení SOA v kapitole 3.1.5, a představuje vybrané pohledy na definici její podstaty.

3.6.1.1 Základní vlastnosti

Erl (2009, s. 42) rozděluje SOA historicky na dvě období: Prvotní SOA a Současná SOA. Prvotní SOA se opírá o standardy a specifikace založené na XML a základních webových službách. Současná SOA staví na prvotním modelu, a působením průmyslu a technologického pokroku rozšiřuje své původní ideje (Erl 2009, s. 43). Podle Erla (Erl 2009, s. 45) je základním znakem SOA představa architektury, která podporuje servisní orientaci právě za použití webových služeb.

Tabulka 3 - Základní vlastnosti Současné SOA

Základní vlastnost Současné SOA	Charakteristika vlastnosti
Součást jádra servisně orientované výpočetní platformy.	SOA jde za hranice běžné IT architektury. Představuje servisně orientovaný princip fungování celého podniku.
Zlepšuje kvalitu služby.	Zavádění standardů souvisejících s prováděním služeb bezpečně, spolehlivě, dle požadavků na výkon a datovou integritu.
Zásadní autonomnost služeb.	Nezávislost a soběstačnost služeb formou samosprávy na úrovni výměny zpráv. Zprávy jsou dostatečně inteligentní, aby

	kontrolovaly způsob, jak je budou zpracovávat přijímající služby.
Je založena na otevřených standardech.	Přenos i samotná zpráva jsou založeny na obecně přijatých standardech, které navíc nevyžadují, aby jednotlivé služby sdílely typový systém. Přemosťuje většinu nestejnorodostí uvnitř podniku, a lze tedy vždy zvolit možnost komunikace mezi službami.
Podporuje rozmanitost výrobců SW.	Standardizovaný systém komunikace umožňuje širší výběr produktů nezávisle na stávajícím softwarovém vybavení podniku.
Stará se o vnitřní spolupráci.	Při tvorbě aplikací lze aplikovat takové principy návrhu, které vybaví služby vlastnostmi, které přirozeně podporují spolupráci. Služby tedy bude možné začlenit do budoucích požadavků na integraci aplikací.
Podporuje zjistitelnost.	Podporuje a doporučuje používání takové formy registru služeb, která podporuje oznamování a zjišťování služeb v podniku i mimo podnik.
Podporuje federace.	Umožňuje zapouzdřit zastaralou a nezastaralou aplikační logiku a vystavit ji pomocí standardizovaného systému komunikace. Umožňuje zavést jednotnost do dříve nesdružených podniků a propojit je.
Podporuje princip architektonické kompozice.	Služby jsou nezávislými jednotkami logiky, ze kterých lze skládat automatizované řídící procesy. To podporuje i zavádění otevřených standardů pro podporu flexibilní kompozice (rozšíření WS-*).
Stará se o základní znovupoužitelnost.	Klade důraz na návrh služeb dle principu znovupoužitelnosti, přestože takovýto přímý požadavek neexistuje. Služby musí být neutrální k řídícím procesům i řešením automatizace, které je upotřebí.
Zdůrazňuje rozšiřitelnost.	Klade důraz na takový návrh rozhraní služeb, aby úroveň granularity rozhraní nelimitovala rozšiřování funkcionality. Službu poté lze rozšířit o dodatečnou logiku při minimalizaci negativních

	dopadů.
Podporuje servisně orientovaný modelový vzor řízení.	Služby lze navrhnout tak, aby vyjadřovaly řídící logiku. Modely BPM a modely entit lze přesněji reprezentovat pomocí uspořádaného sjednocení služeb pro řízení.
Implementuje vrstvy abstrakce.	Umožňují zavádění vrstev abstrakce nastavením služeb jako výhradních přístupových bodů k různým prostředkům a procesní logice. Správným návrhem tedy lze oddělit řídící a aplikační domény podniku. Každý koncový bod poté musí vědět pouze o existenci dalšího, což dovoluje každé doméně vyvíjet se více nezávisle. Toto podporuje schopnost přizpůsobovat se změnám v řízení a technologii.
Podporuje organizační flexibilitu.	Působením na řídící reprezentaci služeb, abstrakci služeb a především volnou vazbou mezi řídící logikou a aplikační logikou nabízí potenciál ke zvýšení organizační agilnosti. Jednotlivé automatizované části podniku jsou poté mnohem více na sobě nezávislé, což se nejvíce projevuje během vnitřní reorganizace, sloučení společností, změně v obchodním poli působnosti či při náhradě zavedené technologické platformy v podniku.
Je stavebním blokem.	Často bude SOA reprezentovat pouze jednu z mnoha užitých architektur v podniku a je tedy stavebním blokem poskytujícím služby. Servisně orientovaný podnik naopak považuje SOA za podnikový standard.
Je vývojem.	Využívá úspěšně vlastnosti předchozích distribuovaných architektur a přebírá jejich úspěšně vlastnosti. Liší se od nich návrhem a principy servisní orientace a webových služeb, a také užitím nových návrhových vzorů a technologií. Jmenovitě znovupoužitelnost, zapouzdření, komponovatelnost a abstrakce aplikační a řídící logiky.

Stále dozrává.	Stále existují omezení v rámci funkcionality webových služeb, která limitují nasazení v rámci podniku. Tato omezení jsou ovšem dočasná, jelikož standardizační organizace postupně vytvářejí dodatečné specifikace, které zahrnují jejich řešení.
Je dosažitelným ideálem.	Při zavádění servisně orientované architektury v podniku dojde k hybridnímu přechodnému prostředí, kdy bude společně existovat distribuované prostředí složené z původního a servisně orientovaného řešení. Technologie umožňující implementaci jsou dostupné již dnes a ani výše zmíněný přístup není tedy limitem.

Zdroj: (Erl 2009, s. 44–56)

Chandy et al. (2010, kap. 9 sek. 1) uvádějí, že aplikace implementuje SOA při splnění následujících pěti principů:

- Aplikace musí být modulární. Je tedy možné individuálně přidávat, nahrazovat či modifikovat softwarové komponenty (agenty).
- Komponenty musí být distribuovatelné. Musí být schopny běhu na různých počítačích a za běhu komunikovat mezi sebou odesíláním zpráv po síti.
- Rozhraní komponent musí být možné nalézt jiným vývojářem aplikace. Rozhraní a externě dostupná charakteristika komponenty musí být jasně definovány a zdokumentovány v metadatech. SOA metadata jsou téměř vždy v softwarové formě jako je soubor, webová stránka, zpráva, registr či repositář.
- Softwarová komponenta poskytující SOA službu může být nahrazena další komponentou nabízející stejnou službu za předpokladu, že poskytují stejné rozhraní. Návrh rozhraní je oddělen od interní implementace služby.
- Komponenty poskytující služby musí být sdílené a znovupoužitelné. Musí tedy být umožněno, aby je využívalo větší množství nezávislých aplikací.

Chandy et al. dále zmiňují, že implementací prvních čtyř výše uvedených principů je komponenta ve volné vazbě, což vede k větší flexibilitě.

3.6.1.2 Definice SOA

Vlastnosti uvedené v předchozí kapitole shrnuje Erl (2009, s. 55) ve své Formální definici Současné SOA:

- Představuje otevřenou, rozšiřitelnou, federační a komponovatelnou architekturu, která podporuje servisní orientaci a skládá se ze služeb, které jsou autonomní, schopné kvality (QoS), podporují rozmanitost výrobců software, spolupráci, zjistitelnost a jsou implementovány jako webové služby.
- Může zavést abstrakci řídící logiky a technologie, což vede k volné vazbě mezi těmito doménami.
- Vyvinula se ze starších platforem, zachovává užitečné vlastnosti tradičních architektur a přichází s různými principy, které se starají o servisní orientaci na podporu servisně orientovaného podniku.
- Je ideálním univerzálním standardem v rámci celého podniku, ale dosažení tohoto stavu vyžaduje přechod a podporu na stále se vyvíjející sady technologií.

3.6.1.3 Principy servisní orientace služeb

Servisní orientace služeb je základním pojmem, který vychází z potřeby oddělit jednotlivé zájmy na dílčí části, které spolu souvisí (Erl 2009, s. 242). Tím lze dosáhnout rozdělení komplexních problémů na sadu menších problémů, které již lze izolovaně řešit.

Jedná se vlastně o jinou implementaci již použitého konceptu, na kterém staví objektově orientované programování a programování založené na komponentách (Erl 2009, s. 242). Implementace samotná se přitom řídí již uvedenými vlastnostmi Současné SOA.

Jednotlivé následně uváděné principy spolu souvisí a vzájemně se podporují (Erl 2009, s. 267). Některé principy servisní orientace jsou odvozeny z principů objektové orientace (Erl 2009, s. 268) a některé jsou přirozenými vlastnostmi webových služeb. Mezi takové lze zařadit abstrakci služeb, komponovatelnost, volnou vazbu a dohodu služeb (Erl 2009, s. 270). Ostatní vlastnosti nevyplývají přímo jako výhody užití webových služeb, a je proto nutné se na ně zaměřit v rámci návrhu a modelovaní služeb.

Tabulka 4 – Běžné principy servisní orientace služeb

Princip služby	Zaměření a cíle
Služby jsou opětovně použitelné.	Služby jsou zkoumány a navrhovány z pohledu znovupoužitelnosti, a to bez ohledu na okamžité příležitosti užití.
Služby sdílejí formální dohodu.	Za účelem interakce služeb nemusí služby sdílet nic jiného než formální dohodu, která definuje všechny služby a podmínky pro výměnu informací.
Služby jsou volně vázané.	Služby musí být navrženy pro interakci, aniž by musely existovat úzké vazby mezi službami.
Služby abstrahují logiku v pozadí.	Jediná část služby, která je viditelná okolnímu prostředí, je vystavena prostřednictvím dohody služeb. Logika v pozadí, která netvoří část dohody služeb, je pro žadatele neviditelná.
Služby jsou komponovatelné.	Služby se mohou slučovat s dalšími službami. To umožňuje reprezentovat logiku na různých úrovních granularity a podporuje znovupoužitelnost a tvorbu vrstev abstrakce.
Služby jsou autonomní.	Logika řízení službou je omezena explicitní hranicí. Služba má kontrolu nad vším uvnitř své hranice a dokončení její úlohy není závislé na jiných službách.
Služby jsou bezstavové.	Služby by neměly být nuceny uchovávat informaci o stavu, jelikož to může ovlivnit jejich schopnost zůstat volně spojené. Návrh služeb proto musí udržovat služby maximálně bezstavové a případně přesunout správu informací o stavu do jiné úrovně.
Služby jsou zjistitelné.	Cílem je zpřístupnit definici služeb, aby je mohli objevit žadatelé služeb, čímž se zvýší jejich znovupoužitelnost.

Zdroj: (Erl 2009, s. 242–243)

3.6.2 Charakterizace vztahu k SOA

EDA je možné oproti SOA považovat za více realistickou v sofistikovaném, dynamickém a moderním obchodním prostředí (Clark a Barn 2011). EDA může být dále vnímána jako jistá specializace SOA, kde pro komunikaci mezi rozhraními komponent je užito jedno generické rozhraní, komponenty mají mezi sebou ještě větší volnou vazbu a mohou být navíc vyvolávány potenciálně vícero událostmi. Událostmi řízená architektura se tak stává více flexibilní než SOA.

Taylor et al. (2009, s. 156) uvádějí, že existuje silné propojení mezi současnými způsoby implementace EDA a způsobem návrhu architektur dle principu servisní orientace. Dále zmiňují, že v jistém smyslu je možné vnímat zavádění EDA jako specializovanou a vyspělou formu SOA, jelikož sdílí mnoho stavebních prvků a konceptů.

Dle Chandyho et al. (2010, kap. 9 sek. 1) jsou zase SOA a EDA komplementární pojmy, a je možné obě architektury aplikovat zároveň (jsou vzájemně kompatibilní). Aplikace implementované dle principů SOA mohou k interakcím mezi svými komponentami užít události. Naopak, mnoho obchodních aplikací využívajících EDA využije také SOA principů ve svém návrhu.

Chandy et al. (2010, kap. 9 sek. 2) dále zmiňují, že mnoho vývojových projektů implementujících EDA (a/nebo CEP systém) je zahájeno pod záštitou podnikové strategie pro SOA či BPM. Lze říci, že všechny tři zmíněné koncepty (EDA, SOA a BPM) jsou navzájem kompatibilní.

Událostmi řízená SOA je kombinací EDA a SOA dohromady, a zahrnuje v sobě všechny principy obou těchto architektur. Z tohoto pohledu je správné, aby podniky zahrnovaly práci na EDA do programů na zavedení SOA. Není například žádoucí, aby v podniku vznikaly oddělené a tedy konkurenční týmy. Naopak by někteří IT architekti pro SOA měli být vyškoleni a chápat principy EDA, MOM produktů či možnosti a přínosy zpracování komplexních událostí. Navíc by bylo chybou implementovat SOA strategii bez podpory událostmi řízené SOA (Chandy a Schulte 2010, kap. 9 sek. 2), jelikož EDA přináší další vyšší úroveň flexibility podniku, jak již bylo zmíněno v předešlých kapitolách.

Před uvedením následující kapitoly je vhodné zopakovat, že většina interakcí v SOA aplikacích je řízena požadavky. Konzument služby odesílá zprávu poskytovateli služby, a následně získává odpověď obsahující data či potvrzení, že služba byla provedena.

3.6.2.1 Událostmi řízená servisně orientovaná architektura (SOA 2.0)

Pokud aplikace splňuje zároveň principy návrhu dle SOA i EDA, tak se jedná o událostmi řízenou SOA (Chandy a Schulte 2010, kap. 9 sek. 1). Chandy et al. se zde odkazují na své vlastní definice návrhu aplikace dle principů SOA a EDA, jež byly také uvedeny v kapitolách 3.6.1.1, resp. 3.2.2. Tyto principy se ve své podstatě shodují s principy uváděnými jinými autory.

Většina SOA aplikací je řízena požadavky, ale jen některé jsou řízeny událostmi. Hlavním rozdílem je nesplnění EDA principů „komunikuje jednosměrně“ a „je bez příkazů”, které by se daly interpretovat jako “nevyžaduje žádnou zpětnou reakci“, resp. „nepředpokládá konkrétní akci, která by se měla udát při vyvolání požadavku“. Rozdílem je tedy fakt, že v případě řízení událostmi jsou jednotlivé komunikující strany odděleny i sémanticky.

Většinu EDA obchodních aplikací lze kvalifikovat tak, že splňují principy SOA. Interakce v EDA je modulární z důvodu oddělenosti producenta a konzumenta zpráv. Tito jsou navíc nahraditelní úplně jinou implementací, a lze je také hromadně distribuovat či sdílet. Přesto ne všechny EDA systémy splňují kritéria principů SOA. Například mohou být některá metadata rozhraní komponent realizovaných v rámci EDA skryta před jinými vývojáři, anebo může být také skryt samotný kanál užitý pro výměnu zpráv (Chandy a Schulte 2010, kap. 9 sek. 1).

Událostmi řízená SOA by měla být užita v případech, kdy komponenta konající jako první nevyžaduje odpověď z druhé komponenty. Výhodou je pak vyšší míra konzistence dat, šíření informací (information dissemination) a uvědomování si situace (situation-awareness) (Chandy a Schulte 2010, kap. 9 sek. 1). Požadavky řízená SOA by naopak měla být užita pro ty aspekty aplikace, kdy je vyžadována odpověď první komponentě.

Požadavky řízená SOA a událostmi řízená SOA sdílí mnoho aspektů, které vedou ke zvýšení efektivity při jejich společném zavádění v podniku. Mezi příklady lze uvést

nástroje pro řízení, průmyslové standardy, mechanismy zabezpečení, produkty pro výměnu zpráv, způsob implementace zpráv (XML) či formátu metadat (XSD[17]). Stejně tak ESB a MOM produkty, a samotné protokoly na výměnu zpráv (HTTP a SOAP), podporují komunikaci řízenou událostmi či požadavky. Výjimku ovšem tvoří produkty pro registraci a správu SOA služeb, které typicky nejsou plně připraveny na evidenci událostí, a musí být tedy upraveny dodatečnými vlastními rozšířeními.

Taylor et al. (2009, s. 185) uvádějí, že událostmi řízenou SOA lze realizovat za užití ESB produktů. Pro výměnu zpráv navrhuje užít webových služeb, a to jak pro producenty událostí (publish), tak i pro jejich konzumenty (subscribe). Taylor et al. ale dále poznamenávají, že vytvoření tohoto modelu sítě služeb vyžaduje vysokou úroveň řízení (governance) a zprostředkování (mediation) mezi všemi oddělenými komponentami, aby tato síť fungovala bezpečně a efektivně.

Pro shrnutí lze uvést, že zavedení událostmi řízené SOA (EDA+SOA) nemusí být tolik nákladné a může být i efektivní, pokud se vhodně využijí synergické efekty plynoucí ze vzájemné kompatibility obou přístupů.

3.6.2.2 Specifikace SOA služeb a událostí

Chandy et al. (2010, kap. 9 sek. 1) poznamenávají, že mnoho problémů při návrhu SOA služeb vyvstává také při návrhu událostí. V obou případech je zapotřebí definovat obsah požadavku, odpovědi a formát samotné zprávy (např. XSD/XML dokument). V případě SOA je nutné uvést také funkci, kterou daná služba vykonává. Událostmi řízené interakce pojem funkce nemají. Musí se naopak specifikovat JMS topic, a také další vlastnosti asociované se zprávou (událostí).

V SOA platí, že granularitu požadavky řízených rozhraní si definuje poskytovatel služby. Naproti tomu granularita události je dána samotným objektem události. Není znám návod pro určování vhodné granularity SOA rozhraní (Chandy a Schulte 2010, kap. 9 sek. 1), a podobně je tomu i při určování granularity událostí. Pro správné určení granularity je většinou zapotřebí znát zamýšlený obchodní proces a jednotlivá potřebná data. Při

[17] XML Schema Definition: Představuje implementaci specifikace XML Schema, která je doporučována organizací W3C.

určování objektu události by se návrh měl zaměřit na entitu, se kterou daná událost souvisí. Tento způsob návrhu událostí je tedy ve své podstatě podobný návrhu servisně orientovaných entitně zaměřených řídících služeb.

3.7 Technologie

Tato kapitola představuje nejdůležitější specifikace a technologie používané při implementaci událostmi řízené architektury, a vybrané specifikace související se servisně orientovanou architekturou. Kapitola se primárně zaměřuje na ty specifikace a technologie, které jsou otevřené a jsou akceptované širší veřejností.

3.7.1 Přehled

Základní principy událostmi řízené architektury mohou být realizovány i bez užití otevřených standardů jakým je například značkovací jazyk XML. V případech nutnosti zpracování příchozích událostí z různých hardwarových senzorů či proprietárních řešení se může dokonce jednat i o nejrůznější formy binárních dat, které je zapotřebí specificky interpretovat, tedy předem znát jejich vnitřní strukturu. Tato řešení jsou oprávněná například i v případech, kdy je zapotřebí zpracovávat enormní množství událostí, minimalizovat nároky na prostředky (procesor, paměť) či co nejvíce snížit dobu odezvy v rámci událostní infrastruktury. Jedná se ale většinou o řešení na míru (pro řešení konkrétního problému), a nelze tedy tento přístup ani generalizovat ani doporučit.

Dle studia dostupných zdrojů nebyla nalezena žádná specifikace, která by souvisela pouze s událostmi řízenou architekturou, tvořila její základ a byla akceptována širší veřejností. Taylor et al. (2009, s. 177) zmiňují například možnost užití specifikací WSDM[18] pro správu webových služeb. Tyto specifikace ovšem nebyly zahrnuty do většího množství produktů, a nelze je tedy nyní považovat za relevantní. Z pohledu využití EDA v oblasti BPM je také nejisté, které formáty výměny dat o událostech se stanou široce

[18] Web Services Distributed Management: Specifikace standardizační organizace OASIS pro správu webových služeb. Je tvořena dvěma části: Management Using Web services (MUWS) a Management Of Web Services (MOWS).

akceptovanými (Becker et al. 2012). Jako slibné se ovšem jeví nové komunikační protokoly uváděné v následující kapitole.

Z pohledu Událostmi řízené SOA je možné užít produktů typu ESB, které typicky poskytují dostatečné množství možností potřebných pro její implementaci. Producenti událostí mohou využívat webových služeb pro odesílání jednotlivých událostí, a konzumenti událostí se zase registrovat pro jejich odběr (dle modelu publish-subscribe). ESB podporuje asynchronní výměny zpráv a také funkce potřebné pro správu jednotlivých (webových) služeb. Pro implementaci EDA je možné využít pouze produktů typu MOM, typicky formou běžného asynchronního přenosu zpráv (založeného na JMS[19]) za užití modelu publish-subscribe. Je však možné pro realizaci užít produkt typu ESB, který v sobě typicky zahrnuje i potřebné vlastnosti MOM produktu, a který navíc umožňuje rozšíření svého užití i na další oblasti v podniku.

Pro servisně orientovanou architekturu je možné užít pouze takové technologie, které vyhoví základním požadavkům a vlastnostem na servisní orientaci. Stávající implementace SOA jsou proto často realizovány za pomoci specifikací, které jsou založeny na značkovacím jazyku XML a které umožňují standardizovanou výměnu zpráv mezi systémy. Specifikace pro XML, XML Schema, SOAP a WSDL[20] se řadí mezi základní a jsou dnes běžně používané a podporované. Funkcionality webových služeb jsou dále rozšiřovány specifikacemi hromadně označovanými jako WS-* (Erl 2009, s. 507) a Pulier et al. (2006, s. 21) mezi základní specifikace pro SOA řadí také UDDI[21]. Tyto specifikace ovšem nebudou již dále detailněji charakterizovány, jelikož přímo nesouvisí s EDA.

3.7.2 Přehled základních specifikací a technologií EDA a SOA 2.0

Následující tabulka představuje přehled základních specifikací a technologií, které jsou běžně užity při implementacích událostmi řízené architektury a událostmi řízené SOA.

---

[19] Java Messaging Specification: Specifikace vydávaná v rámci Java Community Process Program. Specifikuje způsob asynchronní výměny zpráv mezi komponentami.

[20] Web Services Definition Language: Specifikace konsorcia W3C pro definování webových služeb.

[21] Universal Description Discovery and Integration: Specifikace standardizační organizace OASIS pro registrování a vyhledávání existujících služeb.

Tabulka 5 - Přehled základních specifikací a technologií užitých v EDA a SOA 2.0

Zkratka	Základní charakteristika	Užití v architektuře
XML	Obecně rozšířený a podporovaný značkovací jazyk, který představuje standardní formát pro výměnu informací mezi systémy.	EDA, SOA, SOA 2.0
XML Schema	Specifikace pro definování struktury XML dokumentů. Nejvíce rozšířená implementace je v jazyce XSD, který je sám založen na XML.	EDA, SOA, SOA 2.0
JSON[22]	JavaScript Object Notation. Jednoduchý textově založený formát pro výměnu dat. Snadno strojově zpracovatelný.	EDA
SOAP	Simple Object Access Protocol. Specifikace přenosového protokolu využívaného pro přenos zpráv na bázi XML. Obvykle je pro účely přenosu zpráv webových služeb využíván v kombinaci s protokolem HTTP.	SOA, SOA 2.0
WSDL	Web Services Definition Language. Jazyk na bázi XML pro počítačově zpracovatelnou definici webových služeb. Zprávy webových služeb jsou definovány pomocí jazyka XSD a přenos obvykle zajišťuje protokol SOAP.	SOA, SOA 2.0
JMS	Java Message Service. Specifikace pro asynchronní výměnu zpráv mezi komponentami. Nezávislá na konkrétním dodavateli. Podporována v radě produktů.	EDA, SOA 2.0

Zdroj: autor, 2014

Následující tabulka představuje další slibné specifikace a technologie, které mohou být užity pro optimalizaci výměny zpráv mezi komponentami či pro zvýšení interoperability.

[22] JavaScript Object Notation : Otevřený standard pro výměnu dat (RFC 4627, ECMA-404).

Tabulka 6 - Přehled slibných specifikací a technologií pro užití v EDA

Zkratka	Základní charakteristika	Možnost užití
AMQP	Advanced Message Queuing Protocol. Otevřený standard (specifikace standardizační organizace OASIS). Podobné vlastnosti jako JMS, ovšem není definovaný na úrovni API, ale za užití specifikace formátu přenášených binárních dat. AMQP může sloužit jako přenosový protokol pro JMS API. Příslibem je větší interoperabilita než v případě JMS, tedy zvýšení možností užití z pohledu produkce a konzumace událostí. Výhodou oproti JMS API je také zabudovaná podpora pro routing zpráv. Dobrá podpora v produktech typu MOM.	EDA
STOMP	Streaming Text Oriented Messaging Protocol. Jednoduchý protokol na výměnu textových zpráv, podobný HTTP a nezávislý na užitém programovacím jazyku. Vhodný například pro zasílání jednoduchých zpráv či pro implementaci proudu událostí. Dobrá podpora především v open-source produktech typu MOM.	EDA
XMPP	Extensible Messaging and Presence Protocol. Protokol původně založený pro výměnu instantních zpráv (instant messaging), založený na XML (podpora rozšiřitelnosti), otevřený standard, možnost decentralizovaného nasazení (podpora adresování), pro komunikaci téměř v reálném čase. Z pohledu EDA umožňuje komunikaci formou modelu publish-subscribe, a také podporu implementace proudu událostí. Velké množství standardem pokrytých rozšíření. Nevhodný pro přenos binárních dat (nutný převod na base64 kódování) či pro zasílání malých zpráv (upovídaný protokol).	EDA

Zdroj: autor, 2014

3.7.3 Enterprise Service Bus

Enterprise Service Bus (ESB) představuje koncept sběrnice, která umožňuje propojení aplikací v lokální síti pomocí webových služeb (Cummins 2009, s. 13). ESB je decentralizovaný EAI middleware s podporou komunikačních standardů, který usnadňuje implementaci servisně orientované architektury. Smyslem integrace aplikací dle EAI konceptu je omezit přímé vazby mezi jednotlivými aplikacemi, což ESB plní vytvořením sdílené sběrnice služeb, která je jednotlivým aplikacím zpřístupněna. ESB typicky poskytuje celou řadu pokročilých funkcí, mezi něž lze zařadit:

- Nezávislost na konkrétním operačním systému či programovacím jazyku;
- Podpora standardizovaných technologií a specifikací s možností podpory proprietárních technologií při napojování na původní systémy;
- Podpora různých návrhových vzorů pro výměnu dat mezi systémy: Například synchronní a asynchronní výměny zpráv na bázi principů request/reply, fire-and-forget či publish/subscribe;
- Podpora pro směrování a adresování zpráv mimo jiné na základě obsahu zpráv či definovaných pravidel a zásad;
- Podpora pro transformace a mediace zpráv: Podpora pro různé technologické či aplikační adaptéry, při transformacích z různých komunikačních protokolů či při mapování mezi různými formáty zpráv;
- Pokročilé funkce při práci se službami: Validace a transformace zprávy, upřednostnění či pozdržení zpracování zprávy, agregace více služeb do nové služby, obohacování zprávy o dodatečné informace z jiných zdrojů, rozdělování a spojování více zpráv či podpora řešení výjimek;
- Správa služeb včetně jejich nasazení a verzování, monitorování, auditování, logování a měření jejich výkonu;
- Podpora zajištění kvality služeb včetně jejich bezpečnosti, zaručeného doručení zpráv, podpory transakcí či opožděného doručení zpráv v případě výpadku či přehlcení cílové aplikace,
- Realizace nových služeb složených užitím jiných již existujících služeb.

ESB představuje konkrétní prostředek pro implementaci servisně orientované architektury či Událostmi řízené SOA. Přidanou hodnotou je možnost iterativního zavedení servisně orientované integrace systémů. Zavedením jednotné metodiky pro implementaci, testování a nasazování služeb, a zároveň při využití již implementovaných pokročilých funkcí ESB, lze z dlouhodobého hlediska užitím ESB produktů minimalizovat investované prostředky.

Dalším aspektem při implementaci integrace aplikací je užití již existujících a praxí prověřených návrhových vzorů. Pro vnitropodnikovou integraci lze například využít společný (kanonický) datový model, který podporuje abstrakci a volnou vazbu mezi službami. Tento návrhový vzor lze přirovnat k servisně orientovanému principu návrhu entitně zaměřených řídících služeb, přičemž hlavním rozdílem je orientace na strukturu dat napříč celým podnikem. Výhodou je minimalizace vývoje nutných transformací zpráv mezi aplikacemi, kdy pro jednotlivé služby postačuje pouze jednou implementovat transformaci do společného datového modelu, ale také celkové zvýšení flexibility při zavádění změn v jednotlivých aplikacích. Tento princip je také aplikovatelný na zpracování příchozích událostí, kdy může docházet k překladu proprietárního formátu dat ze vstupního systému do kanonického datového modelu události (návrhový vzor adaptér). Tato událost je poté při dalším zpracování už v jednotném formátu.

Při masivním využívání služeb je nutné uvážit možné výkonnostní problémy způsobované užitím ESB produktu jako centrálního bodu, přes který probíhá veškerá komunikace (vně i vevnitř podniku). Při implementaci ESB je tedy vhodné včas řešit otázky související s horizontální škálovatelností.

3.8 Produkty

Tato kapitola představuje vybrané produkty, které podporují událostmi řízenou architekturu, servisně orientovanou architekturu či produkty, které jsou dále využity v rámci vlastní práce. Kritérii pro výběr bylo obecné povědomí o daném produktu, a zároveň povědomí o výrobci daného produktu. Uvedené produkty jsou seřazeny abecedně podle názvu.

3.8.1 Message-oriented-middleware produkty

Tato kapitola obsahuje seznam produktů označených jako Message-oriented-middleware, které mohou sloužit v rámci EDA pro výměnu událostí mezi komponentami.

Tabulka 7 – Přehled produktů typu Message-oriented-middleware

Společnost	Produkt	Základní charakteristika produktu
Apache Software Foundation	Apache ActiveMQ	Populární nekomerční open-source messaging produkt. Přístupný z mnoho programovacích jazyků a platforem. Podpora JMS 1.1 API a řady protokolů včetně AMQP, STOMP či MQTT. http://activemq.apache.org/
Apache Software Foundation	Apache Qpid	Nekomerční open-source messaging produkt, postavený na protokolu AMQP. Možné doplnit podporu JMS 1.1 API užitím Qpid JMS komponenty. http://qpid.apache.org/index.html
Aurea	Aurea Sonic MQ CAA	Komerční produkt. Podpora JMS 1.1. Vysoká dostupnost a podpora clusteringu. (Původní výrobce: Progress Software).
TIBCO	Enterprise Message Service	Komerční produkt. Podpora JMS 1.1. Enterprise messaging platforma – vysoká dostupnost a podpora clusteringu a škálovatelnosti.
Fiorano	FioranoMQ	Komerční produkt. Podpora JMS 2.0 a dalších protokolů. Enterprise messaging platforma. Vysoká propustnost v porovnání s jinými MOM produkty.
IBM	IBM WebSphere MQ	Komerční produkt. Možnost podpory JMS 1.1. Široká podpora nasazení v heterogenním prostředí, podpora specifických funkcí pro zabezpečení přenosu zpráv, Enterprise messaging platforma.
Microsoft	Microsoft Message Queuing (MSMQ)	Komerční produkt. Bez podpory JMS API. Vhodný pouze při implementaci EDA s produkty, které podporují MSMQ komunikaci.
Oracle	Open Message Queue	Nekomerční open-source produkt s podporou JMS 2.0. https://mq.java.net/
Pivotal Software	RabbitMQ	Open-source řešení s komerční podporou. Primárně založeno na AMQP, ale možnost pluginů pro podporu JMS 1.1 API, STOMP či MQTT protokolů. Podpora clusteringu a vysoké dostupnosti. https://www.rabbitmq.com/

IIT Software GmbH	SwiftMQ	Komerční produkt. Podpora JMS 1.1. Enterprise messaging platforma. Podpora upgrade za chodu. http://www.swiftmq.com/

Zdroj: autor, 2014

3.8.2 Přehled vybraných produktů pro implementaci EDA a SOA

Tato kapitola představuje vybrané řady produktů, které podporují událostmi řízenou architekturu a servisně orientovanou architekturu. Kritérii pro výběr bylo obecné povědomí o daném produktu, povědomí o výrobci daného produktu, a možnost integrace mezi SOA a EDA produktem ze stejné nabídky produktového portfolia daného výrobce. Do přehledu byl zařazen také produkt společnosti WSO2, který je v této práci dále užit. Uvedený seznam produktů je seřazen abecedně podle názvu společnosti.

Všichni níže vybraní výrobci poskytují ucelenou obchodní řadu produktů, která umožňuje plnohodnotnou implementaci SOA. Realizaci servisně orientované architektury společně s EDA lze provést užitím některého z uvedených produktů pro SOA (z důvodu užití otevřených standardů a volné vazbě). Využití produktů jednoho výrobce ovšem přináší větší garanci funkční kompatibility a podpory ze strany daného výrobce, a tedy i větší pravděpodobnost snadnějšího zavádění do rutinního provozu.

Mezi výrobce by bylo možné zařadit i společnosti, které jsou známé v oblasti produktů pro servisně orientovanou architekturu. Jedná se o společnosti SOA Software a Progress Software. Tyto společnosti ovšem nemají v nabídce odpovídající produkt pro událostmi řízenou architekturu, a nesplnily tak stanovená kritéria.

Tabulka 8 – Přehled vybraných řad produktů pro implementaci EDA a SOA

Společnost	SOA produkt (řada)	EDA/CEP produkt (řada)	Stručné informace o produktech
IBM	IBM WebSphere	IBM InfoSphere Streams	SOA: Široká podpora SOA včetně registru služeb a pokročilé správy služeb. EDA: Nový produkt. Původní získán akvizicí (IBM WebSphere Business Events).

Microsoft	Microsoft BizTalk Server	Microsoft StreamInsight	SOA: Základní podpora s návazností na cloudové prostředí Windows Azure. EDA: Podpora CEP s vysokou propustností, integrováno s Microsoft prostředím pro vývoj (.NET).
Oracle	Oracle Fusion Middleware SOA Suite	Oracle Event-Driven Architecture (EDA) Suite	SOA, EDA: Ucelená podpora (včetně SOA Governance). Návaznost na další produkty společnosti. Jádrem EDA produkt Oracle Complex Event Processing (CEP).
Red Hat	Red Hat JBoss Enterprise SOA Platform	Red Hat JBoss BRMS	SOA: Jedna z nejmladších platforem implementující SOA. Open Source s komerční podporou. EDA produkt kombinuje užití různých produktů řady Drools.
Software AG	webMethods	Apama Real-time Analytics	Řada produktů předního výrobce v oblasti SOA. Podpora SOA Governance a BPM. Produkt EDA získán akvizicí v roce 2013 od společnosti Progress Software.
TIBCO	TIBCO ActiveMatrix	TIBCO BusinessEvents TIBCO StreamBase CEP	SOA: Ucelená podpora. Možnost optimalizace obchodních procesů pomocí zpracování komplexních událostí. Druhý produkt EDA získán akvizicí v roce 2013.
WSO2	WSO2 SOA & Integration Platform	WSO2 Big Data Analytics Platform	SOA: Ucelená podpora (včetně SOA Governance, ESB i BPM). Open Source s komerční podporou. EDA: Založeno na Apache Cassandra and Apache Hadoop. Hlavní produkt WSO2 Complex Event Processor.

Zdroj: autor, 2014

4 Vlastní práce

Následující kapitoly uvádí praktickou ukázku implementace EDA.

4.1 Praktická aplikace podle zásad EDA

Pro vlastní práci byly stanoveny následující cíle:

- Navrhnout základní prvky jednoduché IT architektury v souladu s principy událostmi řízené architektury a základními principy servisně orientované architektury.
- Navrhnout a implementovat ukázkové algoritmické obchodování na burze cenných papírů užitím EDA.
- Navrhnout a implementovat ukázkové servisně orientované služby (SOA) potřebné pro zpracování zadaných fiktivních pokynů pro nákup či prodej akcií.
- Pomocné cíle: Vygenerovat či jinak získat události potřebné při zpracování, a automatizovaně je zaslat do EDA infrastruktury.

Na úvod bude navržena jednoduchá IT architektura, která bude v souladu s principy EDA a SOA, a která bude zároveň v dlouhodobém časovém období poskytovat dostatečnou flexibilitu pro funkční i výkonnostní změny. Bude zvolena kombinace produktů a technologií vhodných pro realizaci vlastní práce. Řešení bude splňovat požadavky na stabilitu, vysokou dostupnost, škálovatelnost a podporu v rutinním provozu.

Hlavním cílem ukázkové aplikace EDA bude implementace algoritmického obchodování na burze cenných papírů za užití vyhodnocování komplexních událostí (téměř) v reálném čase. Pro zjednodušení bude simulované obchodování probíhat pouze na jednom akciovém trhu, a za užití pokynů typu limit (nákup/prodej za stanovenou cenu). Ukázkový algoritmus pro obchodování bude založen na strategii Trend following, jejímž základem je vhodné zpracování a vyhodnocení předchozího vývoje na daném trhu. Reakcí na zpracovanou komplexní událost bude iniciace zadání pokynu pro nákup či prodej konkrétního akciového titulu. Pokyny budou realizovány formou servisně orientovaných služeb skrze Enterprise Service Bus. Jelikož se jedná o ukázkovou aplikaci, tak cílové služby pouze zaevidují daný pokyn, a dále jej již nijak nebudou zpracovávat.

Ukázka si neklade za cíl realizaci všech funkčních a nefunkčních požadavků nutných pro reálné algoritmické obchodování, ale minimální předvedení zvoleného výseku funkcionality při splnění principů událostního řízení. Vlastní algoritmické obchodování bude ovšem realizováno za užití reálně užívaných algoritmů a principů, ovšem v jejich základní podobě.

4.2 Návrh IT architektury

V rámci následujících kapitol je vytvořen návrh jednoduché IT architektury, která splňuje základní principy událostmi řízené architektury a servisně orientované architektury.

4.2.1 Úvodní rozhodnutí o podobě architektury

Základním rysem flexibilní architektury je vytvoření prostředí, ve kterém mohou co nejjednodušeji vznikat, měnit se a zanikat služby jednotlivých IT systémů, a zároveň být přitom v co největší míře využívány ostatními IT systémy podniku. Životní cyklus jednotlivých služeb by v ideálním případě neměl negativně ovlivňovat okolní IT systémy, tedy způsobovat nutnost změn v těchto systémech či jejich nedostupnost. Zároveň by jednotlivé služby měly být v co největší míře nezávislé na nevyžádaných změnách, které přímo nesouvisí s funkcionalitou dané služby.

Mezi vybrané limitující faktory omezující flexibilitu lze zařadit:

- různorodost technologií a přístupů používaných v jednotlivých systémech,
- udržovatelnost systémů z pohledu vyvolaných změn,
- kontrola nad zabezpečením a kvalitou jednotlivých funkcí systémů,
- složitost z pohledu kontinuálního monitorování a vyhodnocování dostupnosti funkcí jednotlivých systémů.

Za tímto účelem je nutné nejenom zvolit vhodné produkty a vybudovat technickou infrastrukturu, ale také vytvořit vnitropodnikovou metodologii, která při správné aplikaci minimalizuje problémy spojené se spravováním různorodých IT systémů a jejich služeb. Základním rysem takové vnitropodnikové metodologie je definice pravidel pro návrh služeb a pro stanovení a evidenci jejich životního cyklu. Z pohledu nastavení základních rysů IT architektury je nutné zvolit takové technologie a produkty, které budou splňovat anebo podporovat výše zmíněné základní cíle. Typicky se jedná o produkty označované

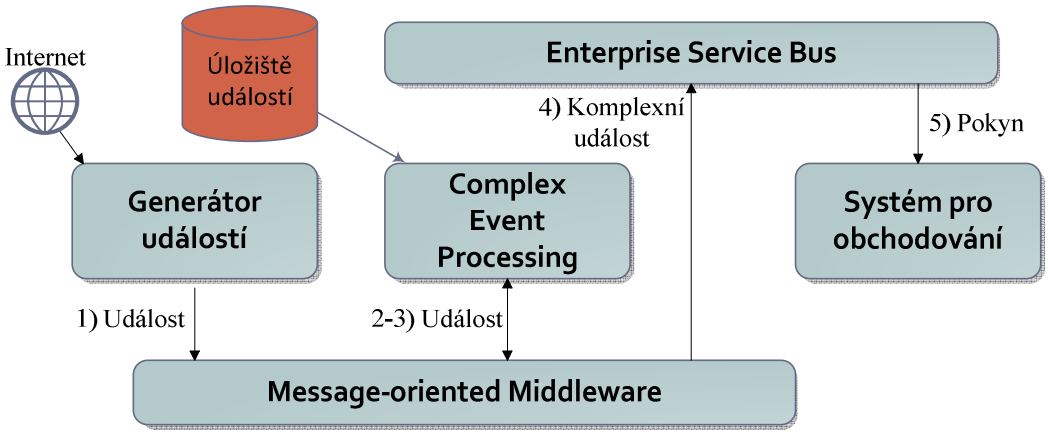
jako **SOA Governance**, které se primárně zaměřují na udržovatelnost IT architektury při masivním nasazení a využívání servisně orientovaných služeb. Z pohledu potřeby flexibilní integrace IT systémů lze využít produktů typu **Enterprise Service Bus**.

Cílem událostmi řízené architektury je naopak vytvoření prostředí, ve kterém jsou v reálném čase zpracovávány různorodé příchozí události, generovány komplexní události, a na jejich základě případně zahájena vhodná reakce. Charakteristickým aspektem je zde také volná vazba mezi producentem a konzumentem jednotlivých událostí. Možným řešením je vytvoření horizontálně škálovatelné infrastruktury pro přenos a zpracování jednotlivých zpráv. Tato bude také využívat vhodné kombinace technologií a produktů. Typicky se jedná o produkty typu **Message-oriented middleware** pro přenos zpráv, a **Complex Event Processing** pro jejich zpracování (v téměř reálném čase).

4.2.2 Základní koncepční návrh realizace

Následující schéma představuje základní koncepční návrh realizace cílů vlastní práce. Zároveň je zde naznačen i předpokládaný tok dat mezi jednotlivými prvky zamýšlené architektury.

Vlastní práce se primárně nezaměřuje na ukázku striktních principů implementace SOA. Z tohoto důvodu zde není užit produkt realizující SOA Governance. Pro ukázku využití servisně orientovaných služeb z událostmi řízeného prostředí zde postačuje užití produktu ESB. Tento produkt poskytuje dostatečné prostředky pro přístup ke službám, které jsou poskytovány třetími aplikacemi.



Obrázek 9 - Vlastní práce - Kontext ukázkové aplikace s naznačením datových toků

Zdroj: autor, 2014

4.2.3 Předpokládaný tok dat v ukázkové aplikaci

V rámci ukázkové aplikace bude probíhat následující výměna dat mezi jednotlivými komponentami. Aplikace “Generátor událostí“ průběžně získává data o vývoji na akciovém trhu pomocí poskytovaných API služeb společnosti Yahoo. Aplikace Generátor událostí sestaví zprávu do stanoveného formátu, a odešle ji do stanoveného JMS topic v Message Oriented Middleware systému.

CEP systém obdrží zprávu (událost) z JMS topic ze systému MOM, a vyhodnotí ji podle stanovených podmínek pro jednotlivé komplexní události. Pokud je podmínka splněna, tak systém CEP vygeneruje zprávu (událost) do příslušného JMS topic v MOM systému. Formát zprávy je zde opět předem stanoven.

ESB obdrží tuto zprávu (událost) z JMS topic MOM systému a provede příslušné akce pro její zpracování:

- transformace vstupních dat z JSON formátu do XML,
- zavolání vzdálené operace webové služby (služby pro zadání pokynu k obchodování do systému „Systém pro obchodování“) a
- následné zapsání výsledku operace do logovacího souboru ESB.

Systém pro obchodování reaguje na pokyny iniciované ze strany ESB. Vyvolaná operace provede realizaci pokynu pro nákup či prodej akcií vybraného akciového titulu. Pro potřeby ukázkové aplikace je pouze provedena fiktivní realizace, a to zalogováním vstupních a výstupních dat operace.

4.2.4 Volba produktů a technologií

Koncepční návrh a datové toky uvedené v předešlých kapitolách splňují zadání dle stanovených cílů vlastní práce. Následujícím krokem je zvolení vhodných produktů a technologií pro realizaci zamýšleného cíle. S ohledem na velikost podniku, ve kterém se zavádí příslušné technologie, lze zvolit jak volně dostupné produkty založené na Open Source, tak i uzavřené komerční produkty různých výrobců.

Pro účely realizace ukázkové aplikace jsou zvoleny vybrané produkty z produktové řady „WSO2 SOA & Integration Platform“ a „WSO2 Big Data Analytics Platform“. Obě platformy jsou založeny na otevřeném kódu a jsou volně šiřitelné pod Apache licencí.

Jednotlivé vybrané produkty splňují potřebné předpoklady pro realizaci architektury dle principů EDA a SOA.

Pro jednotlivé oblasti zamýšlené IT architektury jsou zvoleny následující konkrétní produkty a technologie:

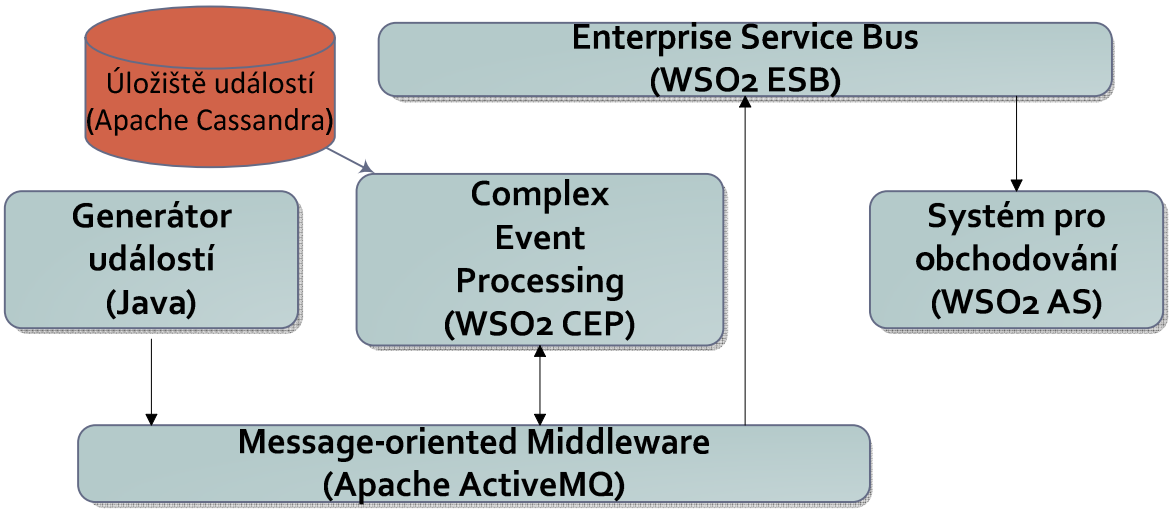
Tabulka 9 – Produkty zvolené pro realizaci ukázkové aplikace EDA

Oblast	Produkt	Technologie
Generátor událostí	Java (standalone aplikace)	HTTP, RSS, CSV, MultiThreading, CSV, JSON, JMS API (JMS topic)
Message-oriented Middleware (přenos zpráv)	Apache ActiveMQ 5.9.x	JMS API (JMS topic), JSON
Úložiště událostí	Apache Cassandra[23]	-
Zpracování komplexních událostí	WSO2 Complex Event Processing 3.0.x	Dotazovací jazyk Siddhi, JSON, XML, XPath, JMS API (JMS topic)
Enterprise Service Bus	WSO2 Enterprise Service Bus 4.8.x	XML, XML Schema, SOAP a WSDL, JMS API (JMS topic), JSON, transformace dat.
Systém pro obchodování	WSO2 Application Server 5.2.x	XML, XML Schema, SOAP, WSDL a JAX-WS, WAR

Zdroj: autor, 2014

[23] Produkt Apache Cassandra není ve výsledné ukázkové aplikaci užit. Zdůvodnění je možné nalézt v následující kapitole.

Následující obrázek představuje jednotlivé zvolené produkty v již uvedeném kontextu zamýšlené implementace.



Obrázek 10 - Softwarové produkty zvolené pro implementaci ukázkové aplikace

Zdroj: autor, 2014

4.3 Vymezení rozsahu práce

Z důvodu omezení délky vlastní práce byla učiněna následující rozhodnutí, která vymezují rozsah ukázkové aplikace EDA.

Je realizován modul „Generátor událostí“ pro načítání informací pomocí Yahoo Finance CSV API a Yahoo Finance News RSS pro konkrétní akciový titul. Modul pracuje ve více vláknech za účelem testování celkového paralelního zpracování událostí v jednotlivých systémech.

Ukázka implementace komplexních událostí umožní dostatečné předvedení výhod užití kombinace technologií EDA a SOA. V systému CEP jsou proto vytvořeny tři komplexní události pro vyhodnocování algoritmického obchodování strategie typu Trend following. Je realizován základní algoritmus vyhovující principu této strategie. Výčet implementovaných komplexních událostí:

- Reagující na růst poptávky za užití tří exponenciálních klouzavých průměrů tržních cen (krátkodobý, střednědobý, dlouhodobý).

- Reagující na pokles poptávky za užití tří exponenciálních klouzavých průměrů tržních cen (krátkodobý, střednědobý, dlouhodobý).
- Reagující na náhlý pokles poptávky v kombinaci s nedávným zveřejněním informace o právě obchodovaném podniku.

Reakcí na první komplexní událost je vyvolání pokynu pro nákup daného akciového titulu. Pro zbylé dvě komplexní události je naopak vyvolán pokyn pro prodej. Z důvodu zjednodušení ukázkové aplikace není v rámci vyhodnocování komplexních událostí zjišťováno ani nijak řešeno zda je akciový titul nakoupen, či v jakém množství, zda je aktuálně možné akciový titul nakoupit, či obecně jakékoli jiné vyhodnocování stávajícího portfolia. V případě reálné implementace by tato funkcionalita mohla být například součástí systému pro obchodování.

V rámci implementace je upřednostněno užití exponenciálního klouzavého průměru (EWMA[24]) oproti běžnému klouzavému průměru. Tento typ klouzavého průměru je užit především z důvodu rychlejší reakce na změny v trendu (Harper 2009; OnlineTradingConcepts.com 2012).

Pro potřeby splnění stanovených cílů je nutné realizovat rozšíření dotazovacího jazyka Siddhi, jež umožní výpočet exponenciálního klouzavého průměru.

Pro přenos zpráv mezi jednotlivými systémy je užit JMS topic. Pro zvýšení interoperability je pro JMS zprávu zvolen formát JSON, který je uložen v objektu typu TextMessage. Z pohledu zamýšleného ukázkového zpracování je formát JSON podporován jednotlivými zvolenými produkty a je tedy akceptovatelný.

Implementované koncové služby systému "Systém pro obchodování" jsou vystaveny pomocí proxy služeb na produktu ESB. Vstupem do těchto proxy služeb je JMS topic s JSON formátem zprávy. Vstupní JMS zpráva je nejprve vhodně transformována, a následně je vyvolána koncová služba. Výstup z této služby není na ESB nijak dále zpracováván. Fiktivní "Systém pro obchodování" loguje zavolanou koncovou službu, a neobsahuje žádnou další funkčnost či logiku.

[24] Zkratka EWMA: Exponentially Weighted Moving Average.

Omezení rozsahu implementace ukázkové aplikace:

- Není uvedeno ověření, zda návrh a implementace splňují náležitosti pro budování systémů dle architektur EDA a SOA. Práce tyto náležitosti ovšem splňuje.
- Nejsou uvedeny kroky vedoucí k instalaci a konfiguraci vývojového a cílového prostředí. Užitý postup je dostupný v instalačních materiálech jednotlivých produktů.
- Není užito úložiště (Apache Cassandra) pro uchování událostí. Zpracování tedy probíhá pouze v rámci paměti produktu WSO2 CEP, a není tedy například odolné vůči možným výpadkům. Pro splnění cílů práce toto však není zapotřebí.
- V rámci ukázkové aplikace jsou jednotlivé produkty užity pouze v jedné instanci, tedy bez využití klastru či rozložení zátěže (load balancing).

Během implementace jsou maximálně využívány pokročilé možnosti zvolených produktů pro generování zdrojových kódů a nastavení jednotlivých modulů. Generovaný kód není vložen jako součást práce, ale místo toho jsou uvedeny kroky vedoucí ke stejnému výsledku. Ukázky autorsky vytvořeného zdrojového kódu jsou vloženy jako přílohy na konci této práce.

4.4 Volba strategie pro realizaci obchodování

Algoritmické obchodování představuje oblast, která za posledních 10 let velmi nabyla na svém významu. Je odhadováno (Lee 2011; Riley 2013), že tento druh obchodování představuje desítky procent z celkového počtu všech obchodů v Evropě či USA. Podobné je tomu i na různých rozvojových trzích (Innovator 2011).

Předmětem této práce není zkoumání oblasti algoritmického obchodování jako celku, ale pouze zvolení a využití konkrétního přístupu pro realizaci simulace obchodování na burze cenných papírů. Existují různé druhy či strategie algoritmického obchodování (Treleaven et al. 2013). Lze zmínit například systematické obchodování, vysoko-frekvenční obchodování či ultra vysoko-frekvenční obchodování. U posledně jmenovaného probíhá vyhodnocení a realizace obchodu pod jednu milisekundu. Mezi strategie lze řadit například (Treleaven et al. 2013):

- Momentum (Trend following) – zjišťování momentu časových řad. Při rostoucím trendu lze nakupovat, při opačném prodávat.
- Mean reversion – předpoklad, že ceny se dlouhodobě neudržují u svého minima či maxima, ale vracejí k průměru.
- Statistical arbitrage – užití statistických metod pro vyhodnocování trhu. Mezi známé metody patří například strategie Pairs trading, kdy dva historicky korelované cenné papíry budou v budoucnu konvergovat.

Lze obecně doplnit, že algoritmické obchodování je v průsečíku statistiky, počítačových věd, matematiky, financí a ekonomie. Je také nutné zdůraznit, že algoritmické obchodování nijak nezajišťuje zisk a může být značně rizikové (Bates 2013). Je zde také podezření, že způsobuje různé negativní efekty na trzích, na kterých působí (Chaboud et al. 2013). Z tohoto důvodu byl a je vliv tohoto druhu obchodování předmětem vyšetřování a různých regulací (U.S. Commodity Futures Trading Commission a U.S. Securities & Exchange Commission 2010; European Securities and Markets Authority 2012; Henderson 2014).

Pro účely této práce byla **zvolena strategie Trend following**, která patří mezi méně užívané, ale přesto dnes běžné metody algoritmického obchodování. Základní charakteristiku principů a aplikace této strategie z ekonomického hlediska představují například Berger (2014) či Davenport (2013). Z pohledu matematiky a statistiky se jedná především o zjišťování aktuálních trendů z časových řad. Rozhodování o konkrétním nákupu či prodeji na daném trhu je poté založeno na ekonomických teoriích.

Užité metody pro zjišťování trendů jsou založeny na různých prognostických metodách. Mezi ty výpočetně jednodušší patří užití klouzavých průměrů (Moving average; MA). Pro účely této práce bylo **zvoleno užití exponenciálních klouzavých průměrů** pro zjišťování trendů vývoje cen jednotlivých akciových titulů. Exponenciální klouzavé průměry typicky reagují rychleji na změnu vývoje než běžně klouzavé průměry (Twomey 2010). Je vhodné zmínit, že klouzavé průměry tvoří základ výpočtu dalších technických indikátorů jako je Bollinger Bands či MACD[25] (Norman 2012). Různé přístupy a

[25] Zkratka MACD: Moving Average Convergence-Divergence.

vyhodnocení efektivity investice je možné nalézt například v pracích Acara a Satchella (1997) či Baltase a Kosowského (2011). Baltas a Kosowski dále upozorňují na skutečnost, že při uplatnění této strategie musí investor počítat s větší volatilitou při obchodování. Jsou zde také uvedeny další různé metody pro zjišťování trendů, které vykazují vyšší návratnost investice než při užití klouzavých průměrů.

Trend following strategii je obecně vhodnější použít na trzích, které nevykazují přílišnou volatilitu a kde je zjištění trendů méně ovlivněno aktuálním děním. Jedná se především o peněžní trhy, trhy s obligacemi či komoditami, a to často formu budoucích kontraktů (futures contracts). Dle Wilcoxe a Crittendena (2005) je ovšem možné Trend following strategii užít také pro obchodování na akciových trzích, což je také záměrem této práce.

4.5 Návrh algoritmu pro realizaci obchodování

V předešlé kapitole byla zvolena strategie Trend following pro obchodování na akciovém trhu. Metodou pro zjišťování trendu byl zvolen výpočet exponenciálního klouzavého průměru. Dle Weissmana (2012) je například možné použít 200 denní jednoduchý klouzavý průměr. Pro rozhodování o nákupu či prodeji je vhodnější užít způsob, kdy je vypočteno více klouzavých průměrů pro různě dlouhá časová období (Nathan 2013). Při užití dvou a více klouzavých průměrů může být signálem pro nákup či prodej překřížení těchto průměrů (Murphy 2009; OnlineTradingConcepts.com 2012). Při obchodování je například možné užít tří klouzavých průměrů – 200 dnů pro dlouhodobé, 100 dnů pro střednědobé a 20 dnů pro krátkodobé období. Optimální typ a počet užitých klouzavých průměrů, a ani optimální délka pro ně užitých období, není nikde stanovena.

Pro účely praktické části této práce je stanoveno užití tří exponenciálních klouzavých průměrů – vždy jeden pro dlouhodobé, střednědobé a krátkodobé období. Z důvodu potřeby provedení základního ověření vlastního průběhu algoritmu je stanovena délka dlouhodobého období na 45 sekund, střednědobého na 30 sekund a krátkodobého na 15 sekund. Frekvence generování dat je stanovena na 250 milisekund pro jeden typ obchodované akcie.

- Komplexní událost 1 (identifikace rostoucího trendu)

- Pokud je klouzavý průměr pro krátkodobé a zároveň i pro střednědobé období nad dlouhodobým klouzavým průměrem, tak je generován pokyn pro nákup zpracovávané akcie. Není povoleno generování více jak jednoho pokynu během 15 sekund pro jeden akciový titul.

- Komplexní událost 2 (identifikace klesajícího trendu)
  - Pokud je klouzavý průměr pro krátkodobé a zároveň i pro střednědobé období pod dlouhodobým klouzavým průměrem, tak je generován signál pro povolení prodeje zpracovávané akcie. Není povoleno generování více jak jednoho signálu během 15 sekund pro jeden akciový titul.
- Komplexní událost 3 (krátkodobé šoky)
  - Pokud je exponenciální klouzavý průměr pro krátkodobé období menší o 2 a více procent než běžný klouzavý průměr pro krátkodobé období, tak je generován signál pro povolení prodeje zpracovávané akcie. Není povoleno generování více jak jednoho signálu během 15 sekund pro jeden akciový titul.

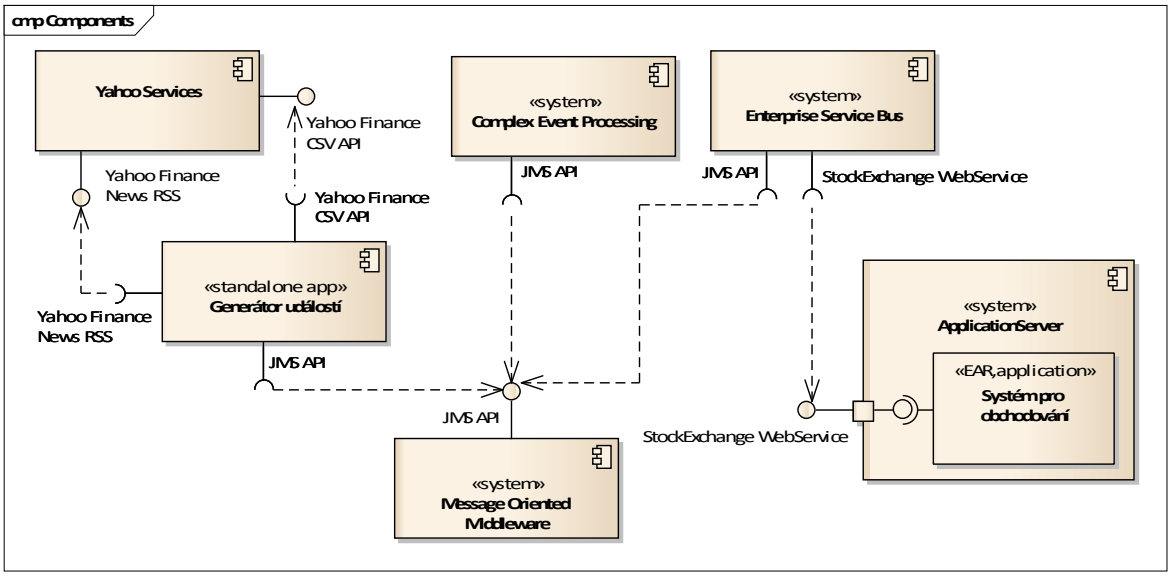
Uvedený algoritmus pro generování pokynů je plně realizován za užití příchozích událostí v rámci produktu WSO2 Complex Event Processing. Jedná se tedy o událostmi řízené zpracování proudu dat za užití komplexních událostí.

4.6 Ukázková aplikace

Následující kapitoly uvádějí postup návrhu a vytvoření ukázkové aplikace v událostmi řízeném prostředí. Návrh a implementace událostí a jejich následné užití v produktech WSO2 CEP, WSO2 ESB a WSO2 AS.

4.6.1 Komponentový diagram ukázkové aplikace

Následující obrázek představuje komponentový diagram zamýšlené ukázkové aplikace. Jednotlivé komponenty a rozhraní uvedené v komponentovém diagramu jsou podrobněji charakterizovány v následujících kapitolách.



Obrázek 11 - Komponentový diagram ukázkové aplikace

Zdroj: autor, 2014

4.6.2 Specifika implementačního prostředí

V rámci implementace zamýšleného řešení je nutné nainstalovat a nakonfigurovat jednotlivé zvolené produkty. V základní konfiguraci není možné tyto produkty používat na jednom serveru, jelikož by docházelo ke konfliktu užitých síťových portů. Produkty společnosti WSO2 umožňují parametrickou změnu užitých portů formou definování offsetu. Pro jednotlivé užité produkty je proto užit offset s odstupem 10 portů. Tato konfigurace již nevykazuje zmíněný konflikt.

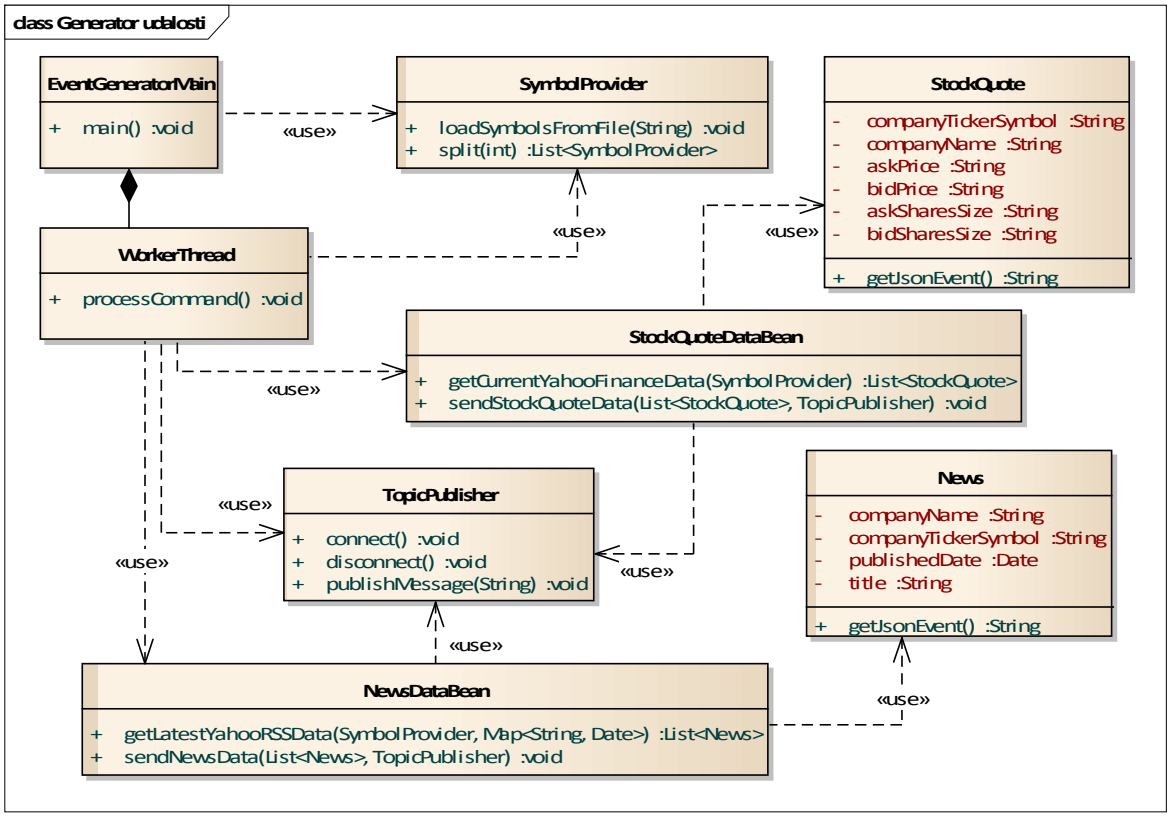
Jednotlivé zvolené produkty jsou také náročné na operační paměť. Jelikož jsou jednotlivé produkty WSO2 založeny na OSGi technologii[26], tak je možné je všechny provozovat v rámci jedné instance Java virtuálního stroje. V rámci vývoje ukázkové aplikace EDA nebyla tato konfigurace ovšem užita, a to z důvodu zvýšení průkaznosti výsledků realizovaného řešení.

[26] Vysvětlení: WSO2 produkty jsou založeny na specifikacích OSGi Alliance a jsou tedy plně modulární. Lze je tedy instalovat jako nezávislé moduly v rámci OSGi kontejneru, a tím zvolit požadovanou kombinaci produktů a minimalizovat nároky na potřebné zdroje.

4.6.3 Implementace aplikace Generátor událostí

Systém pro zpracování událostí vyžaduje na svém vstupu data, která by mohla být zpracována. Získávání aktuálních údajů z akciového trhu je realizováno prostřednictvím rozhraní, které je pro tyto účely zpřístupněno společností Yahoo. Uvedená funkcionalita je implementována v aplikaci Generátor událostí, který slouží pouze jako pomocná součást celkového řešení vlastní práce. Aplikace pracuje na principu periodického načítání aktuálních údajů z uvedených rozhraní, a následného vytváření příslušných JMS zpráv. V reálné implementaci by byla vhodnější přímá integrace s IT systémy jednotlivých akciových trhů, preferovaně formou asynchronních zpráv, a ideálně s co nejmenší latencí.

Aplikace je realizována v jazyce Java. Jednotlivé dotazy na aktuální ceny akcií jsou prováděny paralelně ve více vláknech, a zároveň také dávkově (dotaz na více akciových titulů v rámci jednoho volání). Následující diagram tříd představuje návrh implementace aplikace Generátor událostí:



Obrázek 12 - Diagram tříd aplikace Generátor událostí

Zdroj: autor, 2014

Aplikace Generátor událostí je spuštěna třídou EventGeneratorMain. Ta si ze souboru načte symboly pro jednotlivé akciové tituly, se kterými bude dále pracovat, za užití třídy SymbolProvider. Následně jsou rozděleny načtené symboly do skupin, a pro jednotlivé skupiny vytvořena vlákna pro další paralelní zpracování. Paralelní zpracování je realizováno třídou WorkerThread. Ta obsahuje cyklus pro periodické volání logiky pro získávání dat a jejich následné odeslání do JMS topic.

Třída StockQuoteDataBean realizuje funkční logiku pro získání údajů z akciových trhů (pomocí Yahoo Finance CSV API) a transformaci vstupních údajů do objektu StockQuote. Objekt je následně uložen do formátu JSON a výstup následně zaslán do JMS topic s názvem cz.czu.pef.topics.stockQuotes.

Třída NewsDataBean realizuje funkční logiku pro načítání a zpracování publikovaných údajů z Yahoo Finance News RSS. Jednotlivé údaje jsou převedeny do objektu News, který je následně uložen do formátu JSON a ten odeslán do JMS topic cz.czu.pef.topics.stockExchangeNews.

Zdrojové soubory aplikace je možné nalézt v příloze s názvem: Příloha B – Zdrojové soubory aplikace Generátor událostí.

4.6.4 Nastavení systému Message Oriented Middleware

V rámci vlastní práce je jako MOM užit produkt Apache ActiveMQ. Pro jeho užití postačuje mít nainstalovanou Java Runtime Environment (JRE) a provést úpravu konfigurace produktu dle návodu uvedeného v příloze s názvem: Příloha C – Konfigurace produktu Apache ActiveMQ.

4.6.5 Implementace systému pro zpracování komplexních událostí

Systém pro zpracování komplexních událostí, tedy vybraný produkt WSO2 CEP, tvoří jádro vlastní práce. Cílem implementace je zpracování proudů dat za užití komplexních událostí, a to v (téměř) reálném čase. V rámci předchozích kapitol byl stanoven cíl realizace tří komplexních událostí. Vstupem jsou příchozí události o vývoji ceny na akciových trzích a informace o publikovaných novinkách o daném konkrétním akciovém titulu.

Vývoj aplikací pro WSO2 CEP probíhá ve vývojovém prostředí založeném na produktu Eclipse IDE. Během implementace bylo ovšem zjištěno, že toto vývojové prostředí stále není v dobrém a použitelném stavu, především zde mnoho potřebných funkcí zcela chybí. V rámci práce bylo nutné zvolit alternativní přístup pro vývoj této části ukázkové aplikace. Jako efektivní řešení bylo zvoleno provedení manuální konfigurace v rámci administrační konzole běžícího produktu WSO2 CEP. Toto řešení bylo použito i v případě produktu WSO2 ESB, kde se vyskytl podobný problém s vývojovým prostředím.

Informace do produktu WSO2 CEP jsou zasílány z aplikace Generátor událostí, a to prostřednictvím MOM produktu Apache ActiveMQ. Nejprve je tedy zapotřebí provést napojení produktu WSO2 CEP na produkt Apache ActiveMQ. Návod na napojení je možné nalézt v příloze s názvem: Napojení na produkt Apache ActiveMQ. Výstupní události z produktu WSO2 CEP jsou také zasílány pomocí JMS topic.

Následující kapitoly uvádějí jednotlivé kroky nutné pro implementaci ukázkové aplikace. Předpokladem je již napojený produkt WSO2 CEP na produkt Apache ActiveMQ. Předpokladem pro vývoj rozšíření dotazovacího jazyka Siddhi je zde funkční vývojové prostředí Java založené na produktech Eclipse IDE a Maven.

4.6.5.1 Vývoj a registrace rozšíření jazyka Siddhi

Produkt WSO2 CEP nepodporuje výpočet agregovaných hodnot formou exponenciálního klouzavého průměru, který je potřebný pro realizaci zamýšlených komplexních událostí. Produkt ovšem poskytuje možnost rozšíření užitého dotazovacího jazyka Siddhi, a to formou implementace Java tříd a jejich registrace v produktu. Po registraci těchto tříd v produktu WSO2 CEP je možné tuto implementaci napřímo využít při sestavování jednotlivých dotazů.

Postup vývoje rozšíření jazyka Siddhi o podporu EWMA:

1.) V prostředí Eclipse IDE vytvořit běžný Maven projekt (nepoužít archetype).
2.) Upravit Maven konfigurační soubor pom.xml dle konfiguračního souboru přiloženého v příloze s názvem: Maven konfigurační soubor projektu.
3.) Založit Java třídu ExponentialMovingAverageAggregatorFactory. Třída slouží pro vytváření instancí agregační třídy a aktuálně podporuje pouze datový typ Double. Zároveň tato třída definuje název jmenného prostoru a název funkce,

jež dohromady budou určovat způsob vyvolání z dotazovacího jazyka Siddhi. Třída implementuje rozhraní OutputAttributeAggregatorFactory.

4.) Založit Java třídu ExponentialMovingAverageAggregatorDouble, která provádí vlastní výpočet hodnoty dle EWMA. Třída implementuje rozhraní OutputAttributeAggregator.
5.) Provést sestavení výsledného JAR souboru pomocí nástroje Maven.
6.) Nakopírovat výsledný JAR soubor do adresáře ${WSO2_CEP}//repository/components/lib/.
7.) Přidat následující řádek do souboru ${WSO2_CEP}/repository/conf/siddhi/siddhi.extension: cz.czu.pef.siddhi.extensions.ExponentialMovingAverageAggregatorFactory
8.) Restartovat produkt WSO2 CEP

Zdrojový kód výše uvedených Java tříd je možná nalézt v příloze s názvem: Rozšíření jazyka Siddhi.

4.6.5.2 Registrace vstupního proudu událostí StockQuoteData

V rámci produktu je zapotřebí nejprve zaregistrovat jednotlivé proudy událostí, které budou sloužit pro komunikaci uvnitř i vně CEP produktu. Pro komunikaci s okolními systémy je nutné definovat jednotlivé vstupní a výstupní proudy událostí a příslušné transformace dat. Pro transformaci vstupních zpráv do vnitřního formátu produktu CEP je pro každý proud událostí definován tzv. Builder. Pro transformaci výstupních zpráv z CEP produktu poté tzv. Formatter. Vnitřní proudy událostí využívají formátu zpráv, který je pro tento produkt nativní.

Jednotlivé níže uvedené kroky jsou realizovány v administrační konzoli produktu WSO2 CEP.

Postup vytvoření vstupního proudu událostí StockQuoteData včetně napojení na JMS topic a transformaci vstupních dat:

1.) V administrační konzoli navigovat do cesty: Home > Manage > Event Processor > Event Streams
2.) Po akci “Add Event Stream” zadat:
    a. Název proudu událostí: cz.czu.pef.dp.streams.StockQuoteData

b. Verze proudu událostí: 1.0.0
c. V sekci “Payload Data Attributes” přidat postupně v uvedeném pořadí atributy "TickerSymbol" (typ string), "CompanyName" (typ string), "AskPrice" (typ double), "BidPrice" (typ double), "AskSize" (typ double) a "BidSize" (typ double).

3.) Zvolit akci “Add Event Stream”
4.) Zvolit akci "In-Flows" pro právě založený proud událostí "cz.czu.pef.dp.streams.StockQuoteData"
5.) Zadat následující údaje:
    a. Název: cz_czu_pef_eventBuilders_StockQuoteBuilder
    b. Zvolit vstupní adaptér “ActiveMQ-input“
    c. Název JMS topic: cz.czu.pef.topics.stockQuotes
    d. Zvolit možnost „json“ pro vstupní typ mapování
    e. Přidat mapování z JSON do vnitřních položek v sekci “JSON Mappings”:

Tabulka 10 - Charakteristika vstupní zprávy StockQuote a mapování z JSON formátu

Název atributu	Datový typ	JSON výraz	Charakteristika atributu
TickerSymbol	String	$.StockQuoteEvent.TickerSymbol	Kód akciového titulu
CompanyName	String	$.StockQuoteEvent.CompanyName	Název společnosti
AskPrice	Double	$.StockQuoteEvent.AskPrice	Aktuální cena nabídky
BidPrice	Double	$.StockQuoteEvent.BidPrice	Aktuální cena poptávky
AskSize	Double	$.StockQuoteEvent.AskSize	Aktuální objem nabídky
BidSize	Double	$.StockQuoteEvent.BidSize	Aktuální objem poptávky

Zdroj: autor, 2014

    f. Zvolit akci “Add Event Builder“

Výše uvedeným postupem vznikne vstupní proud událostí, který bude přijímat zprávy z JMS topic. Příchozí JMS zprávy jsou transformovány z formátu JSON do vnitřního formátu produktu WSO2 CEP pomocí vytvořeného sestavovače zpráv s názvem cz_czu_pef_eventBuilders_StockQuoteBuilder. Zadaný název JMS topic odpovídá názvu, který je užit v aplikaci „Generátor událostí“.

4.6.5.3 Registrace vstupního proudu událostí StockExchangeNews

Postup vytvoření vstupního proudu událostí StockExchangeNews včetně napojení na JMS topic a transformaci vstupních dat:

1.) V administrační konzoli navigovat do cesty: Home > Manage > Event Processor > Event Streams
2.) Po akci “Add Event Stream” zadat:
    a. Název proudu událostí: cz.czu.pef.dp.streams.StockExchangeNews
    b. Verze proudu událostí: 1.0.0
    c. V sekci “Payload Data Attributes” přidat postupně v uvedeném pořadí atributy “TickerSymbol” (typ double), “Title” (typ string) a PublishedDate (typ long).
3.) Zvolit akci “Add Event Stream”
4.) Zvolit akci "In-Flows" pro právě založený proud událostí "cz.czu.pef.dp.streams.StockExchangeNews"
5.) Zadat následující údaje:
    a. Název: cz_czu_pef_eventBuilders_StockExchangeNewsBuilder
    b. Zvolit vstupní adaptér “ActiveMQ-input“
    c. Název JMS topic: cz.czu.pef.topics.stockExchangeNews
    d. Zvolit možnost „json“ pro vstupní typ mapování
    e. Přidat mapování v sekci “JSON Mappings”:

Tabulka 11 - Charakteristika vstupní zprávy News a mapování z JSON formátu

Název atributu	Datový typ	JSON výraz	Charakteristika atributu
TickerSymbol	String	$.NewsEvent.TickerSymbol	Kód akciového titulu
Title	String	$.NewsEvent.Title	Nadpis publikace
PublishedDate	Long	$.NewsEvent.PublishedDate	Datum a čas publikace

Zdroj: autor, 2014

    f. Zvolit akci “Add Event Builder“

Výše uvedeným postupem vznikne vstupní proud událostí, který bude přijímat zprávy z JMS topic. Příchozí JMS zprávy jsou transformovány z formátu JSON do

vnitřního formátu produktu WSO2 CEP pomocí vytvořeného sestavovače zpráv s názvem cz_czu_pef_eventBuilders_StockQuoteBuilder.

4.6.5.4 Registrace pomocných vnitřních proudů událostí

Výstupem níže uvedených komplexních událostí je zpráva, která je vložena do vnitřního proudu událostí v rámci produktu WSO2 CEP. Poté je tato zpráva transformována do vhodného výstupního formátu a odeslána vně produktu. Nejprve je tedy zapotřebí vytvořit pomocné proudy událostí následujícím způsobem:

1.) V administrační konzoli navigovat do cesty: Home > Manage > Event Processor > Event Streams
2.) Po akci “Add Event Stream” zadat:
    a. Název: cz.czu.pef.dp.streams.internal.BidPriceLoweringStockQuotes
    b. Verze proudu událostí: 1.0.0
    c. V sekci “Payload Data Attributes” přidat postupně v uvedeném pořadí atributy “ TickerSymbol ” (typ string),“ CompanyName” (typ string),“ ShortTermAveragePrice” (typ double),“ MediumTermAveragePrice” (typ double) a “ LongTermAveragePrice ” (typ double).
3.) Zvolit akci “Add Event Stream”
4.) Opakovat kroky 1 až 3 také pro proud událostí:
    a. Název: cz.czu.pef.dp.streams.internal.BidPriceRaisingStockQuotes

4.6.5.5 Realizace exekučního plánu BidPriceStockAnalyzer

Produkt WSO2 CEP pro realizaci komplexních událostí využívá tzv. exekuční plány. Exekuční plán je instancí stroje pro zpracování komplexních událostí a je tedy odpovědný za vlastní zpracování jednotlivých událostí. Typický exekuční plán sestává ze sady vstupních a výstupních proudů událostí, a sady dotazovacích výrazů.

Postup vytvoření exekučního plánu BidPriceStockAnalyzer pro vyhodnocení dvou dlouhodobých komplexních událostí:

1.) V administrační konzoli navigovat do následující cesty a zvolit možnost: Home > Manage > Event Processor > Execution Plans
2.) Po zvolení možnosti “Add Execution Plan“ zadat následující základní údaje:

a. Název exekučního plánu: BidPriceStockAnalyzer
b. Popis exekučního plánu:
c. V sekci Query Expressions zaregistrovat vstupní proud „cz.czu.pef.dp.streams.StockQuoteData“ pod názvem “StockQuoteData“.
d. V sekci Exported Streams zaregistrovat pod názvem “BidPriceLoweringStockQuotes“ výstupní proud “cz.czu.pef.dp.streams.internal.BidPriceLoweringStockQuotes“.
e. V sekci Exported Streams zaregistrovat pod názvem “BidPriceRaisingStockQuotes“ výstupní proud “cz.czu.pef.dp.streams.internal.BidPriceRaisingStockQuotes“.
f. Vlastní dotazovací výrazy exekučního plánu v jazyce Siddhi (logika pro vyhodnocení komplexních událostí):

```
/* Seskupení a výpočet EWMA pro krátkodobé období. Uloženo do dočasného
proudu událostí. */
    from StockQuoteData[BidSize > 0 and BidPrice > 0]#window.time(15 sec)
as t1
    select t1.TickerSymbol,
           t1.CompanyName,
           cz_czu_pef:exponentialMovingAverage(t1.BidPrice) as
AveragePrice,
           count(t1.BidPrice) as CountData
    group by TickerSymbol
    insert into StockQuoteData_ShortTerm;

/* Seskupení a výpočet EWMA pro střednědobé období. Uloženo do dočasného
proudu událostí. */
    from StockQuoteData[BidSize > 0 and BidPrice > 0]#window.time(30 sec)
as t1
    select t1.TickerSymbol,
           t1.CompanyName,
           cz_czu_pef:exponentialMovingAverage(t1.BidPrice) as
AveragePrice,
           count(t1.BidPrice) as CountData
    group by TickerSymbol
    insert into StockQuoteData_MediumTerm;

/* Seskupení a výpočet EWMA pro dlouhodobé období. Uloženo do dočasného
proudu událostí. */
    from StockQuoteData[BidSize > 0 and BidPrice > 0]#window.time(45 sec)
as t1
    select t1.TickerSymbol,
           t1.CompanyName,
           cz_czu_pef:exponentialMovingAverage(t1.BidPrice) as
AveragePrice,
           count(t1.BidPrice) as CountData
    group by TickerSymbol
```

```
    insert into StockQuoteData_LongTerm;

/* Dotaz na hledání vzoru, kde hodnota EWA pro krátkodobé a střednědobé
období je vyšší než hodnota EWMA dlouhodobého období, a to v rámci 45
sekund. Výsledky pro jednotlivé akciové tituly jsou seskupeny. Emitování
komplexní události o rostoucím trendu je limitováno na jednu událost za
15 sekund. */
    from every
         t3 = StockQuoteData_LongTerm<1:> ->
         t2 = StockQuoteData_MediumTerm[t2.TickerSymbol ==
t3.TickerSymbol and t2.AveragePrice > t3.AveragePrice]<1:> ->
         t1 = StockQuoteData_ShortTerm[t1.TickerSymbol == t2.TickerSymbol
and t1.AveragePrice > t2.AveragePrice]<1:>
    within 45 sec
    select t1.TickerSymbol,
           t1.CompanyName,
           t1.AveragePrice as ShortTermAveragePrice,
           t2.AveragePrice as MediumTermAveragePrice,
           t3.AveragePrice as LongTermAveragePrice
    group by t1.TickerSymbol
    output last every 15 sec
    insert into BidPriceRaisingStockQuotes;

/* Dotaz na hledání vzoru, kde hodnota EWA pro krátkodobé a střednědobé
období je menší než hodnota EWMA dlouhodobého období, a to v rámci 45
sekund. Výsledky pro jednotlivé akciové tituly jsou seskupeny. Emitování
komplexní události o klesajícím trendu je limitováno na jednu událost za
15 sekund. */
    from every
         t3 = StockQuoteData_LongTerm<1:> ->
         t2 = StockQuoteData_MediumTerm[t2.TickerSymbol ==
t3.TickerSymbol and t2.AveragePrice < t3.AveragePrice]<1:> ->
         t1 = StockQuoteData_ShortTerm[t1.TickerSymbol == t2.TickerSymbol
and t1.AveragePrice < t2.AveragePrice]<1:>
    within 45 sec
    select t1.TickerSymbol,
           t1.CompanyName,
           t1.AveragePrice as ShortTermAveragePrice,
           t2.AveragePrice as MediumTermAveragePrice,
           t3.AveragePrice as LongTermAveragePrice
    group by t1.TickerSymbol
    output last every 15 sec
    insert into BidPriceLoweringStockQuotes;
```

Výše vytvořený exekuční plán vybírá akciové tituly, u kterých se změnil trend ceny poptávky z rostoucího na klesající, či opačně. Pro výpočet jsou užity 3 exponenciální klouzavé průměry (krátkodobý, střednědobý a dlouhodobý), které jsou vzájemně mezi sebou porovnávány.

Zpracování exekučního plánu probíhá nepřetržitě (kontinuálně) a je řízeno pouze příchozími událostmi z proudu událostí StockQuoteData. Jakmile je přijata událost z tohoto

proudu, tak je okamžitě zařazena ke zpracování v jednotlivých dotazech. To je umožněno tím, že se jedná o JMS topic a je tedy podporováno doručení zprávy k více zaregistrovaným odběratelům. Pro odlišení jednotlivých délek období (od krátkodobé po dlouhodobou) je užito jedné z vlastností jazyka Siddhi, která podporuje omezení "viděných" událostí dle času jejich produkce. Je zde také zapotřebí rozlišovat jednotlivé události podle akciového titulu, o kterém poskytují údaje. K tomuto slouží seskupení událostí podle atributu TickerSymbol, tedy podle kódu akciového titulu. V rámci úvodních výpočtů pro jednotlivá období je užito vytvořeného rozšíření jazyka Siddhi pro výpočet hodnoty EWMA. Konkrétně se jedná o užití agregační funkce s názvem „exponentialMovingAverage" ve jmenném prostoru "cz_czu_pef".

Další dva dotazy představují vlastní logiku pro vyhodnocení (prvních dvou) stanovených komplexních událostí. V daných dotazech je užito mnoho konstruktů a vlastností dotazovacího jazyka Siddhi tak, aby byly splněny stanovené podmínky. Je užito vlastnosti Siddhi pro vyhledávání vzorů (syntaxe: „->") po sobě jdoucích událostí, kde je hledána situace, kdy hodnota dlouhodobého EWMA je větší, resp. menší, než hodnoty střednědobé a krátkodobé. Vyhodnocení vzorů je navíc omezeno pouze na události, které se staly maximálně 45 sekund v minulosti. Dotaz je nutné sepsat takovým způsobem, aby se navzájem neovlivňovaly události pro různé akciové tituly, k čemuž slouží nejenom omezující podmínka (v hranatých závorkách), ale také následné seskupení nalezených vzorů za užití „group by". Dále dotaz zajišťuje, aby se minimalizovaly duplicitní (opakující se) vzory, a negenerovala se tak zbytečně vyšší zátěž při proudovém zpracování. To je zajištěno neomezeným požadavkem na počet výskytů vyhovujících událostí (syntaxe: "<1:>"). Při všech uvedených vyhovujících podmínkách by mohlo dojít k velmi častému emitování komplexní události. Z tohoto důvodu je omezeno emitování pouze na jeden výskyt každých 15 sekund. Je přitom emitována poslední známá vyhovující událost.

Všechny takto emitované události představují zprávu, která je stále ve vnitřním formátu produktu WSO2 CEP. Nyní je zapotřebí provést naformátování dané zprávy do potřebného výstupního formátu, který bude zpracováván externím systémem.

Postup pro vytvoření formátovaných výstupních zpráv z exekučního plánu BidPriceStockAnalyzer:

1.) V administrační konzoli navigovat do cesty: Home> Manage > Event Processor > Event Streams
2.) Zvolit akci "Out-Flows" pro proud událostí " cz.czu.pef.dp.streams.internal.BidPriceRaisingStockQuotes"
    a. Název: cz_czu_pef_formatters_BidPriceRaisingStockQuotes
    b. Zvolit výstupní adaptér “ActiveMQ-output“
    c. Zvolit výstupní proud “cz.czu.pef.topics.BidPriceRaisingStockQuotes“
    d. Zvolit možnost „json“ pro výstupní typ mapování

4.6.5.6 Realizace exekučního plánu BidPriceShockStockAnalyzer

Postup vytvoření exekučního plánu BidPriceShockStockAnalyzer pro vyhodnocení poslední komplexních událostí:

1.) V administrační konzoli navigovat do následující cesty a zvolit možnost: Home > Manage > Event Processor > Execution Plans
2.) Po zvolení možnosti “Add Execution Plan“ zadat následující základní údaje:
    a. Název exekučního plánu: BidPriceShockStockAnalyzer
    b. Popis exekučního plánu:
    c. V sekci Query Expressions zaregistrovat vstupní proud „cz.czu.pef.dp.streams.StockQuoteData“ pod názvem “StockQuoteData“.
    d. V sekci Query Expressions zaregistrovat vstupní proud „cz.czu.pef.dp.streams.StockExchangeNews“ pod názvem “StockExchangeNews“.
    e. V sekci Exported Streams zaregistrovat pod názvem “BidPriceLoweringStockQuotes“ výstupní proud “cz.czu.pef.dp.streams.internal.BidPriceLoweringStockQuotes“.
    f. Vlastní dotazovací výrazy exekučního plánu v jazyce Siddhi (logika pro vyhodnocení komplexních události):

```
/* Seskupení a výpočet SMA a EWMA pro krátkodobé období. Uloženo do
dočasného proudu událostí. */
    from StockQuoteData[BidSize > 0 and BidPrice > 0]#window.time(15 sec)
as t1
    select t1.TickerSymbol as TickerSymbol,
           t1.CompanyName as CompanyName,
```

```
            cz_czu_pef:exponentialMovingAverage(t1.BidPrice) as
ExpAveragePrice,
            avg(t1.BidPrice) as SimpleAveragePrice,
            count(t1.BidPrice) as CountData
    group by t1.TickerSymbol
    insert into StockQuoteData_ShortTermAverages;

/* Seskupení novinek podle akciového titulu. Jsou ponechány pouze
události z posledních 10 minut. Uloženo do dočasného proudu událostí. */
    from StockExchangeNews#window.externalTime(PublishedDate, 10 min) as
t1
    select t1.TickerSymbol
    group by t1.TickerSymbol
    insert into StockQuoteData_ShortTermNews;

/* Jednosměrné spojení seskupených událostí z akciového trhu se
seskupenými událostmi o novinkách pro stejný akciový titul. Je provedena
také selekce událostí, pro které EWMA je menší o 2 a více procent než
SMA. */
    from StockQuoteData_ShortTermAverages as t1 unidirectional
    join StockQuoteData_ShortTermNews as t2
    on (t1.TickerSymbol == t2.TickerSymbol and
((convert(t1.ExpAveragePrice, double) * 1.02) <
convert(t1.SimpleAveragePrice, double)))
    select t1.TickerSymbol,
           t1.CompanyName,
           t1.ExpAveragePrice as ShortTermAveragePrice,
           t1.ExpAveragePrice as MediumTermAveragePrice,
           t1.ExpAveragePrice as LongTermAveragePrice
    group by t1.TickerSymbol, t2.TickerSymbol
    insert into BidPriceLoweringStockQuotes;
```

První výše uvedený dotaz je obdobný jako v předešlé kapitole s tím rozdílem, že je navíc vypočtena i hodnota běžného klouzavého průměru.

Druhý dotaz seskupuje příchozí události o publikovaných novinkách, které se udály za posledních 10 minut.

Třetí uvedený dotaz je jádrem poslední stanovené komplexní události, jejímž cílem je zjišťování (i krátkodobě) prudkých poklesů cen, a následné vyvolání pokynu pro prodej akciového titulu. Je zde zapotřebí vložit pojistku proti falešnému vyvolání pokynu pro prodej, a tedy zvýšit odolnost algoritmu vůči běžné volatilitě. Z tohoto důvodu je začleněn další faktor v rozhodovacím procesu. Aby mohl být vydán pokyn pro prodej, tak musí být splněna podmínka publikování RSS zprávy v posledních 10 minutách o daném akciovém titulu. Jedná se o prostou ukázkovou podmínku, která slouží primárně pro demonstraci různorodých možností v ukázkové aplikaci.

Emitovanou komplexní událost je zapotřebí také vystavit pro externí systémy. To je zajištěno užitím existujícího výstupního proudu událostí BidPriceLoweringStockQuotes, který byl definován v předchozí kapitole. Sdílením vstupních i výstupních proudů událostí je také demonstrována flexibilita, kterou lze užít při návrhu událostmi řízené architektury.

4.6.6 Implementace systému pro obchodování

Pro potřeby ukázkové aplikace je zapotřebí vyvinout jednoduchou aplikaci, která vystaví služby pro zadávání pokynů pro nákup či prodej konkrétního akciového titulu. Implementace těchto služeb je tedy fiktivní, a slouží pouze pro demonstrativní účely ukázkové aplikace.

Postup vývoje a nasazení aplikace CZU_PEF_StockExchangeApp:

1.) V prostředí Eclipse IDE vytvořit běžný Maven projekt typu WAR.
2.) Vytvořit Java třídu implementující webovou službu.
3.) Sestavit WAR aplikaci za užití nástroje Maven.
4.) Nasadit aplikaci v administrační konzoli WSO2 AS

Zdrojový soubor zmíněné Java třídy je možné nalézt v příloze s názvem: Příloha F – Zdrojové soubory systému pro obchodování. Před užitím v produktu ESB je vhodné dodatečně ověřit dostupnost nasazené aplikace a její základní funkčnost.

4.6.7 Implementace Enterprise Service Bus

Sekundárním cílem ukázkové aplikace je představení možností synergií mezi událostmi řízenou architekturou a servisně orientovanou architekturou. Pro využití výhod plynoucích ze servisně orientované architektury je zapotřebí užít takových návrhových a technických prostředků, které nebudou bránit servisní orientaci. Z technického pohledu se především jedná o užití otevřených standardů založených na webových službách, a v neposlední řadě také možností souvisejících s užitím produktů typu ESB a SOA Governance. Z důvodu omezení rozsahu práce je užit pouze produkt WSO2 ESB.

Nejprve je zapotřebí napojit produkt WSO2 ESB na produkt Apache ActiveMQ. Potřebný návod a konfigurační soubory je možné nalézt v příloze s názvem: Příloha E – Zdrojové a konfigurační soubory ESB.

Následně je zapotřebí provést registrace proxy služeb v ESB produktu:

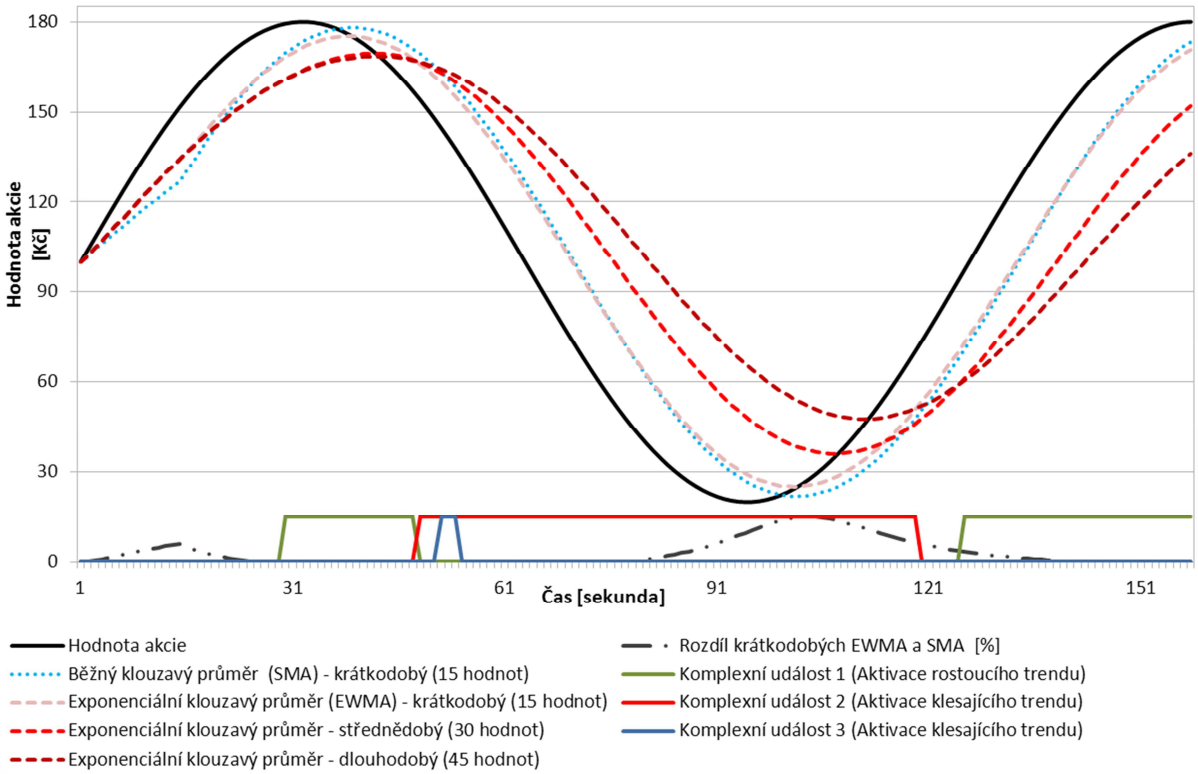
1.) V administrační konzoli ESB produktu navigovat do cesty: Home > Manage > Services > Add > Proxy Service > Proxy Service
2.) Proxy služba BidPriceRaisingProxy
    a. Zvolit typ služby “Logging Proxy“.
    b. Cílové URL nastavit na webovou službu vystavenou systémem pro obchodování (aplikací CZU_PEF_StockExchangeApp).
    c. Zvolit typ transportu „jms“.
    d. Zadat další konfigurační změny formou editace konfiguračního zdroje.
        i. Je zapotřebí především provést transformaci vstupních dat, sestavit SOAP zprávu a tu odeslat na cílovou webovou službu.
        ii. Celá konfigurace služby je k dispozici v příloze s názvem: Konfigurace proxy služby BidPriceRaisingProxy.
3.) Proxy služba BidPriceLoweringProxy
    a. Postup je shodný s postupem pro předešlou proxy službu.
    b. Celá konfigurace služby je k dispozici v příloze s názvem: Konfigurace proxy služby BidPriceLoweringProxy.

Výsledkem této implementace je vytvoření dvou proxy služeb, které přijímají události s JMS topic, do kterého jsou generovány zprávy z produktu WSO2 CEP. Vyvolávání služeb na Enterprise Service Bus je tedy řízeno na základě zpracování komplexních událostí, přičemž původce události nemá žádnou vazbu (ani sémantickou) na vyvolávanou službu. Toto schéma představuje značné přiblížení k modelu architektury označované jako SOA 2.0.

4.7 Testování v událostmi řízeném prostředí

Během vývoje ukázkové aplikace je průběžně prováděno funkční testování jednotlivých částí tak, aby byla ověřena správnost zamýšlené implementace. Pro účely testování správnosti vyhodnocování komplexních událostí je užito umělé generování vstupních dat, konkrétně formou zapnutí této funkce v aplikaci Generátor událostí. Hodnota akcií je vypočtena dle stanovené funkce sinus, a to s vypočtenou a vhodně omezenou frekvencí a amplitudou.

Následující graf představuje průběh vybraných klouzavých průměrů a intervaly aktivace jednotlivých implementovaných komplexních událostí, a to pouze v závislosti na vývoji hodnoty akcie. Uvedené hodnoty jsou fiktivní a slouží pouze pro představení řešení.



Obrázek 13 – Ukázka průběhu vybraných klouzavých průměrů a intervalů aktivace implementovaných komplexních událostí

Zdroj: autor, 2014

Pro funkční testování v událostmi řízené architektuře je klíčová schopnost generování potřebných událostí ve správný okamžik. Zde se projevuje negativní vlastnost EDA. Pro potřeby testování relativně jednoduché komplexní události je zapotřebí vyvinout specifický nástroj, který vhodným způsobem vytváří umělé vstupní události. Významnou roli zde představuje především schopnost vyvolat tuto událost ve správný okamžik tak, aby mohly být otestovány jak pozitivní, tak i negativní testovací scénáře.

Událostmi řízená architektura představuje prostředí, ve kterém není definováno, jaký bude mít dopad vyvolání konkrétní události. Tento dynamický determinismus (Taylor et al. 2009, s. 120) představuje jeden ze základních prvků EDA, který sice zvyšuje flexibilitu prostředí pro zavádění změn, ale který má i své negativní vlivy. Mezi tyto vlivy lze zařadit i zvýšenou náročnost testování (Howard 2013). Komplexní událost v událostmi

řízeném prostředí lze považovat za hybridní automat, jelikož obsahuje jak diskrétní, tak i spojité prvky (Henzinger 1996). Spojitým prvkem je minimálně čas vzniku události, na které je komplexní událost závislá a která může vzniknout v libovolný okamžik. V tomto případě se tedy jedná o podtřídu označovanou jako “Timed automata“ (Alur a Dill 1994). Alur et al. dále uvádějí, že z důvodu výskytu vlivu času, tedy závislosti na čase vyvolání události, je složitost algoritmu exponenciální s každou další přidanou časovou závislostí. V důsledku toho složitost testování komplexní události roste exponenciálně s každým dalším zpracovávaným proudem událostí, či obecněji časově závislou operací, která má na zkoumanou komplexní událost vliv. Přestože je možné zpracování za jistých podmínek řešit i v polynomiálním čase (Jurdziński et al. 2007), tak z praktického hlediska je náročnost testování stále enormní a tedy nevyhovující. Bez uvedení dalších podrobností lze dodat, že teoretický výzkum v této oblasti stále probíhá, a existuje celá řada prací zabývajících se technikami modelování a ověřování podobných typů automatů, či přímo návrhem a optimalizací metod testování v událostmi řízeném prostředí (Norström et al. 1999; Schiefer et al. 2006; Fribourg a Kühne 2011; Krause a Giese 2012; Norman et al. 2013).

Při testování je dále nutné brát v úvahu omezenou reálnou dobu pro vlastní vykonání testů, stejně tak i případné situace, kdy testovaným a požadovaným stavem je nepřicházející událost. Nelze například testovat situace, kdy je nutné reálně čekat po dobu delší než je doba vyčleněná na vlastní fázi testování. Tímto případem jsou i výpočty klouzavého průměru pro dlouhodobé a střednědobé období, které jsou zapotřebí v ukázkové aplikaci. V práci je sice užita technika simulující generování jednotlivých událostí ve zrychleném čase, ale ze striktního pohledu není systém otestován na situace, které nastávají při nezměněném časovém průběhu. Chování systému je minimálně ovlivněno jinou frekvencí generovaných událostí, vyšším paralelismem, hardwarovými možnostmi užitého prostředí, či aktuálním stavem cache a tedy efektivitou cacheování dat.

5 Zhodnocení výsledků

Vlastní práce se zaměřila na návrh základní IT architektury, která je v souladu s principy událostmi řízené architektury, a zároveň základními prvky servisně orientované architektury. Konkrétním cílem bylo navrhnout a implementovat ukázkové algoritmické obchodování na burze cenných papírů, které by probíhalo plně automatizovaně a v reálném čase. Z důvodu potřeby omezení rozsahu práce byly pro ukázkovou aplikaci upřesněny jednotlivé cíle, a přesněji tak vymezen rozsah práce.

Během studia odborné literatury nebyl identifikován způsob návrhu událostmi řízené architektury, který by byl obecně uznáván a doporučován. Tato práce pro svou ukázkovou aplikaci užívá nejnovější přístup SOA 2.0, který vychází z konceptu vhodného propojení EDA a SOA. Přínosem je úspěšné empirické ověření, že lze tyto dva architektonické přístupy navzájem kombinovat, a tedy reálně užít. Zároveň lze tímto potvrdit i tvrzení Taylora et al. (2009, s. 156), že existuje silné propojení mezi současnými způsoby implementace EDA a způsobem návrhu architektur dle principu servisní orientace, a že tyto dvě architektury sdílí mnoho stavebních prvků užitých při návrhu. Lze také souhlasit s tvrzením Chandyho et al. (2010, kap. 9 sek. 1), že SOA a EDA je možné aplikovat zároveň, a že jsou vzájemně kompatibilní a komplementární.

Během vlastní práce bylo zapotřebí nejprve zvolit vhodný produkt pro komplexní zpracování událostí, přičemž při výběru byly postupně zjišťovány nedostatky v přidružených vývojových nástrojích. Prvně uvažovaný produkt Oracle Complex Event Processing byl z důvodu chyb ve vývojovém nástroji zamítnut pro další použití. Pro finálně užitý produkt WSO2 CEP ovšem byla situace obdobná. Dle názoru autora této práce by v případě reálného užití uvedených komerčních produktů mohlo dojít ke zvýšeným nákladům a prodlevám způsobeným absencí efektivního vývojového nástroje.

Dalším zjištěním je absence standardizovaného či jinak jednotného dotazovacího jazyka, který by mohl být užit pro vyhodnocování komplexních událostí. Dotazovací jazyky poskytované jednotlivými produkty se odlišují jak ve své syntaxi, tak především v poskytovaných možnostech. Tato skutečnost může mít vliv především z dlouhodobého hlediska, kdy je podnik svázán s konkrétním výrobcem produktu, a je například také zapotřebí zaměstnávat úzce specializované odborníky.

Z důvodu potřeby vyhodnocování stanovených komplexních událostí bylo zapotřebí rozšířit možnosti dotazovacího jazyka Siddhi, který je užit v produktu WSO2 CEP. Za tímto účelem byl vyvinut kód pro novou agregační funkci sloužící pro výpočet exponenciálního klouzavého průměru. Tato nová funkce byla následně užita v rámci jednotlivých dotazů komplexních událostí.

Při realizaci jednotlivých komplexních událostí bylo zapotřebí provádět průběžné funkční testování, které by ověřovalo správnost implementace. Zde se projevila negativní vlastnost událostmi řízené architektury, kdy pro potřeby realizace relativně jednoduché sady dotazů bylo zapotřebí vyvinout a otestovat generátor dat, který by pro jednotlivé funkční kombinace poskytoval vhodná data. Bylo nutné nejenom připravit a generovat tato data, ale také je poskytovat v předem stanoveném pořadí a čase tak, jak to vyžadovaly jednotlivé podmínky (logika) dané testované komplexní události. Tento úkol významně převyšoval svou pracností a náročností vlastní implementaci stanovených komplexních událostí. Při reálném nasazení EDA v podniku by tato skutečnost především významně negativně ovlivňovala celkovou pracnost, a tedy i výsledné náklady změnových požadavků. Z důvodu závislosti na časovém faktoru lze na událostmi řízenou architekturu nahlížet jako na hybridní automat tak, jak jej definuje Henzinger (1996). Alur et al. (1994), resp. Jurdziński et al. (2007), uvádějí, že složitost pro tento typ automatu roste exponenciálně, resp. ve specifických případech polynomiálně. Podstatný je zde především počet časově závislých faktorů, tedy počet vstupních proudů událostí či počet časově závislých operací nad nimi vykonávanými. Chandy et al. (2010, kap. 7 sek. 1) uvádějí jako výhodu EDA volnost vazeb mezi jednotlivými prvky a s tím související celkovou agilnost této architektury, včetně vysoké flexibility změn. Stejně tak Taylor et al. (2009, s. 125) uvádějí, že EDA je založena na dynamickém determinismu, kdy není předem známo, jaký dopad v systému bude mít vyvolaná událost. Dle autora práce mají ovšem uvedené vlastnosti EDA i svůj negativní projev, a to právě v oblasti velkých nároků na analýzu a realizaci funkčního a zátěžového testování.

Na základě průběhu a výsledků vlastní práce lze doporučit užití EDA pouze ve specifických případech, kdy je například zapotřebí zpracování údajů v co nejkratším čase či pokud je očekáváno, že zadání bude velmi komplikované realizovat za užití běžně užívaných návrhových technik a prostředků. Tento přístup tedy není doporučen jako

obecně vhodný pro budování IT systémů. Před zavedením EDA v podniku je doporučeno také zvážit celkovou komplexitu zamýšleného cílového řešení, a podle toho provést rozhodnutí o užití či neužití tohoto přístupu.

V rámci vlastní práce bylo implementováno vyhodnocování vstupních údajů o vývoji cen jednotlivých akciových titulů, a to formou nepřetržitého zpracovávání proudů vstupních dat. Pro omezení složitosti ukázkové aplikace byl realizován jednoduchý algoritmus pro rozhodování o nákupu či prodeji daného akciového titulu, založený na strategii následování trendů (trend following). Z technologického hlediska lze přesto říci, že se podařilo ověřit použitelnost navrhovaného IT řešení pro stanovené účely. Událostmi řízená architektura je dostatečnou platformou pro realizaci automatizovaného algoritmického obchodování, a stávající produkty poskytují dostatečné možnosti pro tuto či podobné realizace.

Dále lze zhodnotit vlastní průběh implementace daného řešení, během kterého nebyly zaznamenány další negativní vlivy. Realizace řešení byla ovšem časově náročná, jelikož bylo zapotřebí odborným způsobem provést návrh užitých produktů a technologií tak, aby byly vzájemně v souladu a nedocházelo k problémům způsobeným nekompatibilitou. Navrhované kombinace technologií a produktů užité v ukázkové aplikaci lze doporučit i pro realizaci skutečných projektů. Autorovi práce byly k užitku jak předchozí zkušenosti související s tvorbou bakalářské práce na téma Servisně orientovaná architektura, tak především z pracovní činnosti v IT oboru.

Závěr

Jedním z cílů této práce bylo poskytnout ucelený přehled architektonických přístupů, které souvisí s tvorbou software dle konceptu událostmi řízené architektury. Úvodní kapitoly se této problematice věnují z pohledu historického vývoje jednotlivých přístupů, a představují užité způsoby modelování informačních systémů a způsoby jejich vzájemné interakce. Za tímto účelem byl představen vývoj základní klient-server komunikace v souvislosti s objektovými přístupy, a následné snahy o standardizaci komunikačních technologií. Vyvrcholením těchto snah bylo vytvoření přístupu integrace podnikových aplikací (EAI), který ovšem ve výsledku byl limitován užitím různých proprietárních technologií a v důsledku toho docházelo k nízké flexibilitě změn. Nevýhody EAI byly potlačeny v konceptu servisně orientované architektury, která využívá otevřené standardy a specifické způsoby analýzy a návrhu servisně orientovaných služeb. SOA v kombinaci s dalšími přístupy a technikami zvyšuje agilnost a flexibilitu podniku. Práce dále představuje základní možné synergie při využití kombinace přístupů SOA, EDA a obchodních procesů BPM. Práce zmiňuje možnosti související se zpracováním velkého objemu dat, a tedy i vztah k horizontální škálovatelnosti. Z tohoto důvodu byly také představeny pojmy Cloud Computing a BigData, přičemž oba byly identifikovány jako vhodné pro realizaci rozsáhlých systémů založených na konceptu EDA.

Druhým cílem práce bylo formulovat základní teze, které by definovaly událostmi řízenou architekturu jako celek. Za tímto účelem bylo nejprve nutné uvést definice a charakteristiky pojmů, které tvoří základ pro definici EDA. Byly zde uvedeny různé pohledy na chápání pojmu událost, a pojmy jako jsou proud událostí či komplexní událost. Při formulaci definice EDA byl analyzován pohled několika různých citovaných autorů, přičemž bylo identifikováno, že jednotliví autoři užívají různou terminologií. Ta je v některých případech zavádějící až konfliktní. Z tohoto důvodu bylo zapotřebí během studia literatury vhodně interpretovat definice jednotlivých autorů, a v práci je následně představit jednotným způsobem. Na závěr je uvedeno vlastní shrnutí, které sumarizuje definici EDA v souladu s pohledy jednotlivých autorů.

Práce pokračuje analýzou možných oblastí využití zkoumaného konceptu, a to s cílem představení možných přínosů, rizik či omezení. Byly uvedeny příklady oborů a

oblastí, ve kterých je zkoumaný koncept uplatněn a využíván. Dále byly uvedeny možné přínosy v oblastech jako je operativní či strategické řízení podniku, a s tím související vazba na business intelligence. Možná rizika a omezení byla analyzována z několika pohledů, a to včetně uvedení případných negativních dopadů na uživatele či provozovatele IT systémů. Jako možný dopad byla zmíněna možnost zahlcení koncových uživatelů nerelevantními událostmi, a doručení nesprávné, opožděné či neúplné informace.

Dalším cílem práce bylo uvedení různých pohledů na událostmi řízenou architekturu. Prvním pohledem bylo uvedení zevrubné charakteristiky vybraných vnitřních součástí, které tvoří základ zkoumaného konceptu. Byly představeny možné modely pro přenos zpráv a také charakteristiky základních operací, které tvoří základ architektury. Byly také charakterizovány pojmy kontext události a vzor události, které jsou běžně užity při zpracování komplexních událostí.

Navazující kapitola se zabývala pohledem na základní kroky zavádění EDA v podniku. Byly uvedeny jednotlivé fáze a možná kritéria pro zavádění, a to od úvodních rozhodnutí o způsobu a rozsahu implementace, přes úvodní volbu vhodných technologií a pilotní projekt, až po vlastní implementaci.

Následně byl uveden pohled na různé možnosti návrhu EDA, tedy různé způsoby, jakými lze tuto architekturu modelovat. Byly uvedeny možnosti pro typ vazeb mezi komponentami, způsob zpracování událostí, užití komplexních událostí, užití stavů zpracování, či různé typy producentů a konzumentů událostí. Práce dále pokračovala uvedením konkrétního způsobu celkového návrhu, počínaje návrhem sítě komponent a toku událostí, návrhem komplexních událostí a konče návrhem jejich hierarchie.

Dalším cílem práce bylo uvedení vztahu EDA k servisně orientované architektuře. Nejprve byly uvedeny základní vlastnosti SOA, její definice a základní principy servisní orientace. Dále byl charakterizován pojem Událostmi řízená servisně orientovaná architektura (SOA 2.0), a následně představeny výhody a synergie toho přístupu. Následovalo základní porovnání rozdílů v navrhování servisně orientovaných služeb a objektů událostí.

Práce dále uvedla přehled specifikací a technologií, které tvoří základ EDA a SOA 2.0 či které mají přímý vztah k těmto konceptům. V návaznosti na SOA zde byly také představeny vlastnosti produktu typu Enterprise Service Bus.

V závěru přehledu řešené problematiky byl uveden výčet nejvíce propracovaných softwarových produktů, pomocí kterých lze EDA a SOA 2.0 v podniku implementovat.

Vlastní práce se zaměřila na implementaci ukázkového algoritmického obchodování na burze cenných papírů, které probíhá plně automatizovaně a v téměř reálném čase, a které je implementováno dle principů EDA. Byla navržena vhodná IT architektura, a to včetně uvedení základního koncepčního návrhu, předpokládaného toku dat, a volby vhodných softwarových produktů a technologií. Návrh architektury proběhl s vědomím možné budoucí potřeby vysoké dostupnosti, stability a také horizontální škálovatelnosti. Praktické ověření proběhlo za užití produktů Apache ActiveMQ, WSO2 Complex Event Processing, WSO2 Enterprise Service Bus a WSO2 Application Server.

Byl navržen a implementován ukázkový automatizovaný způsob vyhodnocování aktuálního dění na akciovém trhu, který je založen na základní ekonomické a prognostické teorii. Pro vyhodnocování byly užity tři exponenciální klouzavé průměry (EWMA) pro různě dlouhá časová období, a ty mezi sebou vzájemně vhodně porovnávány. Pro demonstraci možností EDA byl v rámci ukázkové aplikace implementován další způsob vyhodnocování, jehož cílem je zrychlení detekce poklesu hodnoty akcií. Konkrétně bylo užito porovnání EWMA s běžným klouzavým průměrem (v krátkodobém období), v kombinaci s detekcí čerstvě publikovaných zpráv o předmětné obchodované společnosti.

V rámci vlastní práce bylo zapotřebí nainstalovat, nakonfigurovat a vhodně propojit jednotlivé produkty. Dále pak bylo nutné navrhnout a vyvinout řadu modulů a částí, které tvoří vlastní jádro ukázkové aplikace. Zároveň byly navrženy a implementovány ukázkové servisně orientované služby, které slouží pro fiktivní zadávání pokynů pro nákup či prodej konkrétních akciových titulů. V neposlední řadě bylo zapotřebí implementovat modul pro načítání aktuálních údajů o vývoji na akciovém trhu a o publikovaných novinkách, a celé řešení řádně odladit a otestovat.

V závěru práce bylo možné ověřit názory některých citovaných autorů o výhodách plynoucích ze vzájemného užití přístupů EDA a SOA. Byla také potvrzena deklarovaná

vlastnost událostmi řízené architektury, a to flexibilita změn. Bylo uvedeno zjištění, že tato vyšší flexibilita je doprovázena negativním efektem v podobě vyšší složitosti a náročnosti na funkční a zátěžové testování. Následně bylo uvedeno, že zde zkoumaný architektonický přístup není doporučen jako obecně vhodný pro budování IT systémů. Je také doporučeno vždy důkladně zvážit negativní dopady, především pak již uvedenou vyšší náročnost realizace.

Téma událostmi řízené architektury je velmi rozsáhlé, a to ani ne tak z důvodu své komplexnosti, ale především z důvodu nutnosti uvedení vhodného kontextu a různých návaznosti na další architektonické přístupy. Práce si sice vytkla za cíl charakterizovat pouze základní aspekty EDA a následně implementovat relativně jednoduchou ukázkovou aplikaci, ale i přesto byl výrazně přesažen stanovený rozsah práce. Důvodem byla nutnost navrhnout a implementovat jednotlivé části ukázkové aplikace tak, aby mohly být ověřeny deklarované vlastnosti zkoumané architektury.

Práce samotná nepokrývá celou oblast událostmi řízené architektury a i nadále by ji bylo možné v tomto ohledu rozšiřovat. Předně by bylo možné práci rozšířit o detailní rozbor postupů souvisejících s analýzou a návrhem komplexních událostí, a to i s ohledem na možnosti v oblasti detekce vzorů událostí či užití kontextů událostí. Dále by bylo možné provést výzkum a posléze i návrh na standardizaci dotazovacího jazyka užitého pro vyhodnocování komplexních událostí. Stejně tak by bylo možné zkoumat možnosti v oblasti zefektivnění vývojových nástrojů a jednotlivých produktů, a to i v souvislosti s uvedenou vyšší náročností na funkční a zátěžové testování. V neposlední řadě by bylo možné se zaměřit na další přístupy a technologie, které souvisí s možnostmi elasticity a horizontální škálovatelnosti. Jako příklady lze uvést možnost zpracování událostí přímo v cloudovém prostředí s cílem minimalizace zpoždění reakce, či analýzu možných synergií s oblastí označovanou jako BigData.

Seznam použitých zdrojů

ACAR, Emmanuel a Stephen E. SATCHELL, 1997. A theoretical analysis of trading rules: an application to the moving average case with Markovian returns. *Applied Mathematical Finance* [online]. roč. 4, č. 3, s. 165–180 [vid. 15. březen 2014]. ISSN 1350-486X. Dostupné z: doi:10.1080/135048697334791

AGRAWAL, Divyakant, 2009. The Reality of Real-Time Business Intelligence. In: Malu CASTELLANOS, Umesh DAYAL a Timos SELLIS, ed. *Business Intelligence for the Real-Time Enterprise* [online]. B.m.: Springer Berlin Heidelberg, Lecture Notes in Business Information Processing, 27, s. 75–88 [vid. 14. únor 2014]. ISBN 978-3-642-03421-3, 978-3-642-03422-0. Dostupné z: http://link.springer.com/chapter/10.1007/978-3-642-03422-0_6

ALUR, Rajeev a David L. DILL, 1994. A theory of timed automata. *Theoretical Computer Science* [online]. 25.4., roč. 126, č. 2, s. 183–235 [vid. 22. březen 2014]. ISSN 0304-3975. Dostupné z: doi:10.1016/0304-3975(94)90010-8

ALVES, Alexandre, Robin J. SMITH a Lloyd WILLIAMS, 2013. *Getting Started with Oracle Event Processing 11g*. Birmingham: Packt Publishing, Limited. ISBN 978-1-84968-454-5.

BALTAS, Akindynos-Nikolaos a Robert KOSOWSKI, 2011. *Trend-following and Momentum Strategies in Futures Markets* [online] [vid. 15. březen 2014]. Dostupné z: http://www.efmaefm.org/0EFMAMEETINGS/EFMA%20ANNUAL%20MEETINGS/2012-Barcelona/papers/EFMA2012_0485_fullpaper.pdf

BARLOW, Mike, 2013. *Real-Time Big Data Analytics: Emerging Architecture*. 1 edition. B.m.: O'Reilly Media. ISBN 978-1-44936-421-2.

BATES, John, 2013. High Frequency Trading: Clear and Present Danger? *Huffington Post* [online]. [vid. 15. březen 2014]. Dostupné z: http://www.huffingtonpost.com/john-bates/high-frequency-trading-cl_b_4393276.html

BECKER, Jörg, Martin MATZNER, Oliver MÜLLER a Marcel WALTER, 2012. A Review of Event Formats as Enablers of Event-Driven BPM. In: Florian DANIEL, Kamel BARKAOUI a Schahram DUSTDAR, ed. *Business Process Management Workshops* [online]. B.m.: Springer Berlin Heidelberg, Lecture Notes in Business Information Processing, 99, s. 433–445 [vid. 14. únor 2014]. ISBN 978-3-642-28107-5, 978-3-642-28108-2. Dostupné z: http://link.springer.com/chapter/10.1007/978-3-642-28108-2_42

BERGER, Serge, 2014. Trend following explained: How to stay in a trending asset. *TradingFloor.com* [online] [vid. 12. březen 2014]. Dostupné z: http://www.tradingfloor.com/posts/trend-following-explained-stay-trending-asset-741651789

BOTZER, David, 2007. *Real-Time Middleware Seminar 2007 - Event-Driven approach for Real-Time Enterprise* [online] [vid. 14. únor 2014]. Dostupné z: https://www.research.ibm.com/haifa/Workshops/rt2007/present/Event-Driven_approach_for_Real-Time_Enterprise.pdf

CLARK, Tony a Balbir S. BARN, 2011. Event driven architecture modelling and simulation. In: [online]. B.m.: IEEE, s. 43–54 [vid. 23. únor 2014]. ISBN 978-1-4673-0412-2, 978-1-4673-0411-5, 978-1-4673-0410-8. Dostupné z: doi:10.1109/SOSE.2011.6139091

CUMMINS, Fred A, 2009. *Building the agile enterprise: with SOA, BPM and MBM*. Amsterdam: Elsevier/Morgan Kaufmann. ISBN 978-0-12-374445-6.

DAVENPORT, Douglas, 2013. Trend Following Explained. *Money and Markets - Financial Advice | Financial Investment Newsletter* [online] [vid. 15. březen 2014]. Dostupné z: http://www.moneyandmarkets.com/trend-following-explained-51696

DECARE SYSTEMS IRELAND, 2013a. *Ad Hoc Data Changes in an Event Driven Architecture* [online]. [vid. 23. únor 2014]. Dostupné z: http://blog.decaresystems.ie/2013/11/15/ad-hoc-data-changes-in-an-event-driven-architecture/

DECARE SYSTEMS IRELAND, 2013b. *Problems with Event Driven Architecture* [online]. [vid. 23. únor 2014]. Dostupné z: http://blog.decaresystems.ie/2013/11/12/problems-with-event-driven-architecture/

EMMERICH, Wolfgang, 2000. *Engineering distributed objects*. Chichester; New York: John Wiley & Sons. ISBN 0-471-98657-7.

ERL, Thomas, 2009. *SOA: servisně orientovaná architektura : kompletní průvodce*. Brno: Computer Press. ISBN 978-80-251-1886-3.

ETZION, Opher a Peter NIBLETT, 2010. *Event processing in action*. Greenwich, 74° w. long: Manning. ISBN 978-1-93518-221-4.

EUROPEAN SECURITIES AND MARKETS AUTHORITY, 2012. *Guidelines - Systems and controls in an automated trading environment for trading platforms, investment firms and competent authorities* [online] [vid. 15. březen 2014]. Dostupné z: https://www.esma.europa.eu/content/Guidelines-Systems-and-controls-automated-trading-environment-trading-platforms-investment-f

FAISON, Ted, 2006. *Event-based programming taking events to the limit*. Berkeley, CA.; New York: Apress ; Distributed to the Book trade worldwide by Springer-Verlag New York. ISBN 978-1-59059-643-2.

FREMANTLE, Paul, 2008. *An interesting problem in Event Driven Architecture | Paul Fremantle's Blog* [online]. [vid. 23. únor 2014]. Dostupné z: http://pzf.fremantle.org/2008/09/interesting-problem-in-event-driven.html

FRIBOURG, Laurent a Ulrich KÜHNE, 2011. Parametric Verification and Test Coverage for Hybrid Automata Using the Inverse Method. In: Giorgio DELZANNO a Igor POTAPOV, ed. *Reachability Problems* [online]. B.m.: Springer Berlin Heidelberg, Lecture Notes in Computer Science, 6945, s. 191–204 [vid. 22. březen 2014]. ISBN 978-3-642-24287-8, 978-3-642-24288-5. Dostupné z: http://link.springer.com/chapter/10.1007/978-3-642-24288-5_17

GARTNER, 2009. *Gartner Says SOA Is Evolving Beyond Its Traditional Roots* [online] [vid. 3. únor 2014]. Dostupné z: http://www.gartner.com/newsroom/id/927612

HANDY, Alex, 2010. SOA's dead; long live SOA. *SD Times: Software Development News* [online] [vid. 3. únor 2014]. Dostupné z: http://www.sdtimes.com/content/article.aspx?ArticleID=34062&page=1

HARPER, David, 2009. Exploring The Exponentially Weighted Moving Average. *Investopedia* [online] [vid. 16. březen 2014]. Dostupné z: http://www.investopedia.com/articles/07/ewma.asp

HEFFNER, Randy, 2011. *Forrester Research : Research : SOA Adoption 2010: Still Important, Still Strong* [online] [vid. 3. únor 2014]. Dostupné z: http://www.forrester.com/SOA+Adoption+2010+Still+Important+Still+Strong/fulltext/-/E-RES59058

HENDERSON, Richard, 2014. *Algo trading systems in FINRA sights | news | Trading & Execution | thetradenews.com* [online] [vid. 15. březen 2014]. Dostupné z: http://www.thetradenews.com/news/Asset_Classes/Equities/Algo_trading_systems_in_FINRA_sights.aspx?l=tl

HENZINGER, Thomas A., 1996. The theory of hybrid automata. In: *Eleventh Annual IEEE Symposium on Logic in Computer Science, 1996. LICS '96. Proceedings*: *Eleventh Annual IEEE Symposium on Logic in Computer Science, 1996. LICS '96. Proceedings* [online]. s. 278–292. ISBN 0-8186-7463-6. Dostupné z: doi:10.1109/LICS.1996.561342

HOFF, Todd, 2011. *High Scalability - High Scalability - Facebook's New Realtime Analytics System: HBase to Process 20 Billion Events Per Day* [online] [vid. 6. únor 2014]. Dostupné z: http://highscalability.com/blog/2011/3/22/facebooks-new-realtime-analytics-system-hbase-to-process-20.html

HOWARD, Tom, 2013. Testing Event-Driven Architectures. *Windy Road* [online]. [vid. 22. březen 2014]. Dostupné z: http://windyroad.com.au/2013/03/20/testing-event-driven-architectures/

HUANG, Xiao-Hui, Shi-Hong ZOU, Ling-Wei CHU, Shi-Duan CHENG a Wen-Dong WANG, 2009. An Event-Driven Fault Localization Algorithm Based on Incremental Bayesian Suspected Degree. *An Event-Driven Fault Localization Algorithm Based on Incremental Bayesian Suspected Degree* [online] [vid. 6. únor 2014]. Dostupné z: http://en.cnki.com.cn/Article_en/CJFDTOTAL-DZYX200906055.htm

CHABOUD, Alain, Ben CHIQUOINE, Erik HJALMARSSON a Clara VEGA, 2013. *Rise of the Machines: Algorithmic Trading in the Foreign Exchange Market* [online]. SSRN Scholarly Paper ID 1501135. Rochester, NY: Social Science Research Network [vid. 15. březen 2014]. Dostupné z: http://papers.ssrn.com/abstract=1501135

CHANDY, K. Mani a W. Roy SCHULTE, 2010. *Event processing designing IT systems for agile companies* [online]. New York: McGraw Hill [vid. 3. únor 2014]. ISBN 978-0-07163-349-9. Dostupné z: http://www.amazon.com/gp/product/B002TGNIZM

INNOVATOR, 2011. Brokers Upgrade To Algorithmic Trading For FI Clients in India. *All About Ultra High-Frequency Trading: Algorithmic and High Speed Trading Strategies* [online]. [vid. 15. březen 2014]. Dostupné z: http://ultrahighfrequencytrading.com/2011/05/18/brokers-upgrade-to-algorithmic-trading-for-fi-clients-in-india/

JURDZIŃSKI, Marcin, François LAROUSSINIE a Jeremy SPROSTON, 2007. Model Checking Probabilistic Timed Automata with One or Two Clocks. In: Orna GRUMBERG a Michael HUTH, ed. *Tools and Algorithms for the Construction and Analysis of Systems* [online]. B.m.: Springer Berlin Heidelberg, Lecture Notes in Computer Science, 4424, s. 170–184 [vid. 23. březen 2014]. ISBN 978-3-540-71208-4, 978-3-540-71209-1. Dostupné z: http://link.springer.com/chapter/10.1007/978-3-540-71209-1_15

KRAUSE, Christian a Holger GIESE, 2012. *Quantitative modeling and analysis of service-oriented real-time systems using interval probabilistic timed automata*. Potsdam: Universitätsverlag Potsdam. ISBN 978-3-86956-171-4.

LAPKIN, Anne, 2012. *Hype Cycle for Big Data, 2012* [online] [vid. 4. únor 2014]. Dostupné z: https://www.gartner.com/doc/2100215/hype-cycle-big-data-

LEE, Sang, 2011. Algorithmic Trading in FX: Ready for Takeoff? *Aite Group* [online] [vid. 15. březen 2014]. Dostupné z: http://www.aitegroup.com/report/algorithmic-trading-fx-ready-takeoff

LEHONG, Hung a Jackie FENN, 2012. Key Trends to Watch in Gartner 2012 Emerging Technologies Hype Cycle. *Forbes* [online] [vid. 4. únor 2014]. Dostupné z: http://www.forbes.com/sites/gartnergroup/2012/09/18/key-trends-to-watch-in-gartner-2012-emerging-technologies-hype-cycle-2/

LINDEN, Markus a Sebastian NEUHAUS, 2010. Event-Driven Business Intelligence Architecture for Real-Time Process Execution in Supply Chains. In: *Business information systems 13th International Conference, BIS 2010, Berlin, Germany, May 3-5, 2010. Proceedings* [online]. Berlin; Heidelberg: Springer, s. 280–290. ISBN 978-3-642-12814-1. Dostupné z: http://link.springer.com/chapter/10.1007%2F978-3-642-12814-1_24

LUCKHAM, David, 2007. *The power of events: an introduction to complex event processing in distributed enterprise systems*. Boston, Mass. [u.a.: Addison-Wesley. ISBN 978-0-20172-789-0.

LUCKHAM, David a Roy SCHULTE, 2011. *Event Processing Glossary – Version 2.0* [online]. [vid. 5. únor 2014]. Dostupné z: http://www.complexevents.com/2011/08/23/event-processing-glossary-version-2-0/

MARGULIUS, David L., 2002. Dawn of the real-time enterprise. *InfoWorld* [online] [vid. 14. únor 2014]. Dostupné z: http://www.infoworld.com/t/platforms/dawn-real-time-enterprise-726

MINER, Donald, 2012. *MapReduce design patterns*. Sebastopol, CA: O'Reilly. ISBN 978-1-44932-717-0.

MURPHY, Casey, 2009. Moving Averages: How To Use Them. *Investopedia* [online] [vid. 15. březen 2014]. Dostupné z: http://www.investopedia.com/university/movingaverage/movingaverages2.asp

NATHAN, Narendra, 2013. Tips to use moving averages indicator to make stock buying & selling decisions. *The Economic Times* [online] [vid. 15. březen 2014]. Dostupné z: http://articles.economictimes.indiatimes.com/2013-02-11/news/37039398_1_averages-short-term-trend-long-term-investors

NORMAN, Craig, 2012. Technical Indicators: Bollinger Bands & Moving Average Convergence-Divergence :: Student Investor. *Student Investor* [online] [vid. 15. březen 2014]. Dostupné z: http://www.studentinvestor.com/articles/2012/02/02/technical-indicators-bollinger-bands-and-moving-average-convergence-divergence-

NORMAN, Gethin, David PARKER a Jeremy SPROSTON, 2013. Model checking for probabilistic timed automata. *Formal Methods in System Design* [online]. 1.10., roč. 43, č. 2, s. 164–190 [vid. 22. březen 2014]. ISSN 0925-9856, 1572-8102. Dostupné z: doi:10.1007/s10703-012-0177-x

NORSTRÖM, Christer, Anders WALL a Wang YI, 1999. Timed Automata As Task Models for Event-Driven Systems. In: *Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications* [online]. Washington, DC, USA: IEEE Computer Society, s. 182–189 [vid. 22. březen 2014]. RTCSA '99. ISBN 0-7695-0306-3. Dostupné z: doi:10.1109/RTCSA.1999.811218

O'REILLY MEDIA, INC., 2012. *Big Data Now: 2012 Edition - O'Reilly Media* [online] [vid. 3. únor 2014]. Dostupné z: http://www.oreilly.com/data/free/big-data-now-2012.csp

ONLINETRADINGCONCEPTS.COM, 2012. Exponential Moving Average - Technical Analysis. *Online Trading Concepts* [online] [vid. 15. březen 2014]. Dostupné z: http://www.onlinetradingconcepts.com/TechnicalAnalysis/MAExponential.html

PULIER, Eric a Hugh TAYLOR, 2006. *Understanding enterprise SOA*. Greenwich, Conn. : London: Manning ; Pearson Education [distributor]. ISBN 1-932394-59-1.

RILEY, Danny, 2013. *When the algos eat the algos: The birth, death and rebirth of algorithmic trading | MrTopStep.com* [online] [vid. 15. březen 2014]. Dostupné z: http://mrtopstep.com/algos-eat-algos-birth-death-rebirth-algorithmic-trading/

SCHIEFER, Josef, Gerd SAURER a Alexander SCHATTEN, 2006. Testing Event-Driven Business Processes. *Journal of Computers* [online]. 1.11., roč. 1, č. 7 [vid. 22. březen 2014]. ISSN 1796-203X. Dostupné z: doi:10.4304/jcp.1.7.69-80

SIMONOV, Mikhail, 2013. Event-Driven Communication in Smart Grid. *IEEE Communications Letters* [online]. 6., roč. 17, č. 6, s. 1061–1064. ISSN 1089-7798. Dostupné z: doi:10.1109/LCOMM.2013.043013.122798

SVOBODA, Jiří, 2009. Cloud Computing [online]. roč. 2009, č. 2, Systémová integrace [vid. 3. únor 2014]. ISSN 1210-9479. Dostupné z: http://www.cssi.cz/cssi/cloud-computing

TAYLOR, Hugh, Angela YOCHEM, Les PHILLIPS a Frank MARTINEZ, 2009. *Event-Driven Architecture: How SOA Enables the Real-Time Enterprise*. B.m.: Pearson Education. ISBN 978-0-321-32211-1.

TRELEAVEN, Philip, Michal GALAS a Vidhi LALCHAND, 2013. Algorithmic trading review. *Communications of the ACM* [online]. 1.11., roč. 56, č. 11, s. 76–85 [vid. 15. březen 2014]. ISSN 00010782. Dostupné z: doi:10.1145/2500117

TWOMEY, Brian, 2010. *Simple Vs. Exponential Moving Averages* [online] [vid. 15. březen 2014]. Dostupné z: http://www.investopedia.com/articles/trading/10/simple-exponential-moving-averages-compare.asp

U.S. COMMODITY FUTURES TRADING COMMISSION a U.S. SECURITIES & EXCHANGE COMMISSION, 2010. *Findings Regarding the Market Events of May 6, 2010: Report of the Staffs of the CFTC and SEC to the Joint Advisory Committee on Emerging Regulatory Issues - marketevents-report.pdf* [online]. 30. září 2010. [vid. 15. březen 2014]. Dostupné z: http://www.sec.gov/news/studies/2010/marketevents-report.pdf

WEISSMAN, Richard, 2012. *Trend Following Kept Simple: The 200-Day Simple Moving Average - Commentary* [online] [vid. 12. březen 2014]. Dostupné z: http://news.cqg.com/commentary/2012/04/trend-following-kept-simple-the-200-day-simple-moving-average.html

WILCOX, Cole a Eric CRITTENDEN, 2005. *Does Trend following work on stocks* [online] [vid. 15. březen 2014]. Dostupné z: http://www.cis.upenn.edu/~mkearns/finread/trend.pdf

Přílohy

Tato kapitola obsahuje jednotlivé přílohy diplomové práce. Jednotlivé zde uvedené zdrojové a konfigurační soubory jsou uvedeny ve zkrácené, ale plně funkční, podobě.

Příloha A – Licence užitých děl

Tato kapitola obsahuje jednotlivé licence užitých děl tak, jak je jimi stanoveno.

Event Processing Glossary – Version 2.0

Pro užití práce Event Processing Glossary (Luckham a Schulte 2011) je zapotřebí uvést plné změní oznámení o autorských právech, a URI odkaz na PDF soubor.

URI odkaz na PDF soubor:

http://www.complexevents.com/wp-content/uploads/2011/08/EPTS_Event_Processing_Glossary_v2.pdf

Oznámení o autorských právech:

Permission to copy and display this glossary in any medium without fee or royalty is hereby granted, provided that you include the copyright notice as shown herein and link or URL to the Material on the EPTS website.

DISCLAIMERS:

THIS MATERIAL IS PROVIDED "AS IS," AND THE EVENT PROCESSING TECHNICAL SOCIETY (EPTS) AND ITS MEMBERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THIS MATERIAL ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

TO THE FULLEST EXTENT PERMITTED BY APPLICABLE LAW, THE EPTS AND ITS MEMBERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THIS MATERIAL OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

No other rights are granted by implication, estoppel or otherwise.

Příloha B – Zdrojové soubory aplikace Generátor událostí

Tato příloha obsahuje třídy implementované při realizaci aplikace Generátor událostí. Redundantní prvky ve zdrojovém kódu byly odstraněny.

Třída EventGeneratorMain

```
package cz.czu.pef.dp.eventGenerator;

import java.util.concurrent.*;

public class EventGeneratorMain {
  private static final long MAX_DURATION_MS = 60 * 60 * 1000L; // 1 hour
  private static final int MAX_SIZE_SYMBOL_PROVIDER_STOCK_QUOTES = 20;
  private static final int MAX_THREADS_STOCK_QUOTES = 15;
  private static final int MAX_SIZE_SYMBOL_PROVIDER_NEWS = 3;
  private static final int MAX_THREADS_NEWS = 3;
  private static final boolean STOCK_QUOTE_USE_FAKE = false;
  private static final boolean NEWS_USE_FAKE = false;

  public static void main(String[] args) {
    long exitTimestamp = System.currentTimeMillis() + MAX_DURATION_MS;
    ExecutorService executorStockQuote =
Executors.newFixedThreadPool(MAX_THREADS_STOCK_QUOTES);
    ExecutorService executorNews =
Executors.newFixedThreadPool(MAX_THREADS_NEWS);

    try {
      SymbolProvider symbolProviderLoaded = new SymbolProvider();
      symbolProviderLoaded.loadSymbolsFromFile(SymbolProvider.FILENAME);

      for (SymbolProvider sp :
symbolProviderLoaded.split(MAX_SIZE_SYMBOL_PROVIDER_STOCK_QUOTES)) {
        Runnable worker = new WorkerThread(sp, true /*stockQuote*/, false
/*news*/, exitTimestamp, STOCK_QUOTE_USE_FAKE, NEWS_USE_FAKE);
        executorStockQuote.execute(worker);
      }

      for (SymbolProvider sp :
symbolProviderLoaded.split(MAX_SIZE_SYMBOL_PROVIDER_NEWS)) {
        Runnable worker = new WorkerThread(sp, false /*stockQuote*/, true
/*news*/, exitTimestamp, STOCK_QUOTE_USE_FAKE, NEWS_USE_FAKE);
        executorNews.execute(worker);
      }

      Thread.sleep(MAX_DURATION_MS);
      // Disable new tasks from being submitted
      executorStockQuote.shutdown();
      executorNews.shutdown();
      Thread.sleep(5000L);
      // Cancel currently executing tasks
      executorStockQuote.shutdownNow();
      executorNews.shutdownNow();
    }
    catch (InterruptedException e) {
```

```
      executorStockQuote.shutdownNow();
      executorNews.shutdownNow();
      Thread.currentThread().interrupt();
    }
    finally {
      executorStockQuote.shutdownNow();
      executorNews.shutdownNow();
    }
  }
}
```

Třída WorkerThread

```
package cz.czu.pef.dp.eventGenerator;

import java.util.*;
import cz.czu.pef.dp.eventGenerator.pojo.*;
import cz.czu.pef.dp.jms.*;

public class WorkerThread implements Runnable {
  public static String TOPIC_STOCK_QUOTES = "cz.czu.pef.topics.stockQuotes";
  public static String TOPIC_NEWS = "cz.czu.pef.topics.stockExchangeNews";

  public static final long DELAY_MS = 250L;

  private SymbolProvider symbolProvider;
  private boolean stockQuoteEnabled;
  private boolean newsEnabled;
  private long exitTimestamp;
  private boolean useStockQuoteFakeData;
  private boolean useNewsFakeData;

  public WorkerThread(SymbolProvider symbolProvider, boolean
stockQuoteEnabled, boolean newsEnabled, long exitTimestamp, boolean
useStockQuoteFakeData, boolean useNewsFakeData) {
    this.symbolProvider = symbolProvider;
    this.stockQuoteEnabled = stockQuoteEnabled;
    this.newsEnabled = newsEnabled;
    this.exitTimestamp = exitTimestamp;
    this.useStockQuoteFakeData = useStockQuoteFakeData;
    this.useNewsFakeData = useNewsFakeData;
  }

  @Override
  public void run() {
      processCommand();
  }

  private void processCommand() {
    TopicPublisher stockQuotePublisher = new
TopicPublisher(TOPIC_STOCK_QUOTES);
    TopicPublisher newsPublisher = new TopicPublisher(TOPIC_NEWS);

    try {
      if (stockQuoteEnabled) {
        stockQuotePublisher.connect();
      }
      if (newsEnabled) {
```

```
        newsPublisher.connect();
      }

      StockQuoteDataBean stockQuoteDataBean = new StockQuoteDataBean();
      NewsDataBean newsDataBean = new NewsDataBean();

      Map<String, Date> newsMap = Collections.synchronizedMap(new
HashMap<String, Date>());

      while (System.currentTimeMillis() <= exitTimestamp) {
        if (stockQuoteEnabled) {
          try {
            List<StockQuote> stockQuotes;
            if (useStockQuoteFakeData) {
              stockQuotes =
stockQuoteDataBean.getFakeStockQuoteData(symbolProvider);
            }
            else {
              stockQuotes =
stockQuoteDataBean.getCurrentYahooFinanceData(symbolProvider);
            }
            stockQuoteDataBean.sendStockQuoteData(stockQuotes,
stockQuotePublisher);
          }
          catch (Exception e) { … }
        }

        if (newsEnabled) {
          try {
            List<News> newsList;
            if (useNewsFakeData) {
              newsList = newsDataBean.getFakeNewsData(symbolProvider);
            }
            else {
              newsList =
newsDataBean.getLatestYahooRSSData(symbolProvider, newsMap);
            }
            newsDataBean.sendNewsData(newsList, newsPublisher);
          }
          catch (Exception e) { … }
        }

        // Pojistka proti spamovani Yahoo API
        long delayMs = DELAY_MS;
        if (newsEnabled && delayMs < 3000L) {
            delayMs = 3000L;
        }
        if (stockQuoteEnabled && delayMs < 250L) {
            delayMs = 250L;
        }
        Thread.sleep(delayMs);
      }
    }
    catch (InterruptedException e) {
      Thread.currentThread().interrupt();
    }
    finally {
      if (stockQuoteEnabled) { stockQuotePublisher.disconnect(); }
```

```
      if (newsEnabled) { newsPublisher.disconnect(); }
    }
  }
…
}
```

Třída SymbolProvider

```
package cz.czu.pef.dp.eventGenerator;

import java.io.*;
import java.nio.charset.Charset;
import java.util.*;

public class SymbolProvider {
  public static final String FILENAME = "current_symbols.txt";

  StringBuffer symbolsConcatenated;
  List<String> symbolsList;

  public SymbolProvider(String singleCompanyTickerSymbol) {
    symbolsConcatenated = new StringBuffer();
    symbolsList = new ArrayList<String>();
    symbolsConcatenated.append(singleCompanyTickerSymbol);
    symbolsList.add(singleCompanyTickerSymbol);
  }

  public void loadSymbolsFromFile(String filename) {
    FileInputStream fis = null;
    InputStreamReader isr = null;
    BufferedReader br = null;

    symbolsConcatenated = new StringBuffer();
    symbolsList = new ArrayList<String>();

    try {
      List<String> symbols = new ArrayList<String>();
      fis = new FileInputStream(new File(filename));
      isr = new InputStreamReader(fis, Charset.forName("UTF-8"));
      br = new BufferedReader(isr);
      while (true) {
        String symbol = br.readLine();
        if (symbol == null)
          break;
        symbol = symbol.trim();
        if ("".equals(symbol))
          continue;
        if (symbol.startsWith("#")) {
          continue;
        }
        symbols.add(symbol);
        if (symbolsConcatenated.length() != 0)
          symbolsConcatenated.append("+");
        symbolsConcatenated.append(symbol);
        symbolsList.add(symbol);
      }
    }
```

```
    catch (IOException e) { throw new RuntimeException(e); }
    finally {
      if (br != null)
        try { br.close(); } catch (IOException e) { throw new
RuntimeException(e); }
      if (isr != null)
        try { isr.close(); } catch (IOException e) { throw new
RuntimeException(e); }
      if (fis != null)
        try { fis.close(); } catch (IOException e) { throw new
RuntimeException(e); }
    }
    if (symbolsConcatenated.length() == 0) {
      throw new RuntimeException("No symbol found.");
    }
  }

  public void addSymbol(String symbol) {
    if (symbolsConcatenated == null) {
      symbolsConcatenated = new StringBuffer();
    }
    if (symbolsList == null) {
      symbolsList = new ArrayList<String>();
    }
    if (symbolsConcatenated.length() == 0) {
      symbolsConcatenated.append(symbol);
    }
    else {
      symbolsConcatenated.append("+").append(symbol);
    }
    symbolsList.add(symbol);
  }

  public StringBuffer getSymbolsConcatenated() {
    return symbolsConcatenated;
  }
  public List<String> getSymbolsList() {
    return symbolsList;
  }

  public List<SymbolProvider> split(int maxSizePerSymbolProvider) {
    List<SymbolProvider> spList = new ArrayList<SymbolProvider>();
    SymbolProvider sp = new SymbolProvider();
    for (String symbol : symbolsList) {
      if (sp.symbolsList != null && sp.symbolsList.size() >=
maxSizePerSymbolProvider) {
        spList.add(sp);
        sp = new SymbolProvider();
      }
      sp.addSymbol(symbol);
    }
    if (sp.getSymbolsList().size() > 0) {
      spList.add(sp);
    }
    return spList;
  }
…
}
```

Třída StockQuoteDataBean

```
package cz.czu.pef.dp.eventGenerator;

import java.io.*;
import java.net.*;
import java.util.*;
import au.com.bytecode.opencsv.CSVParser;
import cz.czu.pef.dp.eventGenerator.pojo.StockQuote;
import cz.czu.pef.dp.jms.TopicPublisher;

public class StockQuoteDataBean {
  public void sendStockQuoteData(List<StockQuote> stockQuotes,
TopicPublisher publisher) {
    for (StockQuote sq : stockQuotes) {
      String msg = sq.getJsonEvent();
      publisher.publishMessage(msg);
    }
  }

  public List<StockQuote> getCurrentYahooFinanceData(SymbolProvider
symbolProvider) {
    InputStream stream = null;
    BufferedInputStream bis = null;
    InputStreamReader isr = null;
    BufferedReader br = null;

    List<StockQuote> stockQuotesList = new ArrayList<StockQuote>();
    try {
      URL url = new URL("http://finance.yahoo.com/d/quotes.csv?s=" +
symbolProvider.getSymbolsConcatenated().toString() + "&f=snab2bb3b2b6");

      stream = url.openStream();
      bis = new BufferedInputStream(stream);
      isr = new InputStreamReader(bis);
      br = new BufferedReader(isr);

      int total = 0;
      int skipped = 0;
      while (true) {
        String line = br.readLine();
        if (line == null || "".equals(line.trim())) {
          break;
        }
        total++;

        CSVParser p = new CSVParser(',', '"');
        String[] tokenArr = p.parseLine(line);

        StockQuote sq = new StockQuote();

        sq.setCompanyTickerSymbol(tokenArr[0]);
        sq.setCompanyName(tokenArr[1]);
        sq.setAskPrice(tokenArr[2], tokenArr[3]);
        sq.setBidPrice(tokenArr[4], tokenArr[5]);
        sq.setAskSharesSize(tokenArr[6]);
        sq.setBidSharesSize(tokenArr[7]);
```

```
        stockQuotesList.add(sq);
      }
      return stockQuotesList;
    }
    catch (MalformedURLException e) { throw new RuntimeException(e); }
    catch (IOException e) { throw new RuntimeException(e); }
    finally {
      if (br != null)
        try { br.close(); }
        catch (IOException e) { throw new RuntimeException(e); }
      if (isr != null)
        try { isr.close(); }
        catch (IOException e) { throw new RuntimeException(e); }
      if (bis != null)
        try { bis.close(); }
        catch (IOException e) { throw new RuntimeException(e); }
      if (stream != null)
        try { stream.close(); }
        catch (IOException e) { throw new RuntimeException(e); }
    }
  }

  private static final long fakeStartTime = System.currentTimeMillis();

  private double currentFakePrice(String tickerSymbol) {
   final double hashCode = (double) tickerSymbol.hashCode();
   final double basePrice = hashCode % 100 + 80.0d;
   double frequencyMultiplier = ((double) (hashCode % 5)) + 1d;
   double amplitudeMultiplier = ((double) (hashCode % 300)) / 10.0d + 1d;

   final double degrees = ((System.currentTimeMillis() - fakeStartTime) /
1000.0d) * frequencyMultiplier;
   final double radians = (Math.PI / 180.0d) * degrees;
   double price = basePrice + amplitudeMultiplier * Math.sin(radians);
   if (price < 1.0d) price = 1.0d;
   return price;
  }

  public List<StockQuote> getFakeStockQuoteData(SymbolProvider
symbolProvider) {
    List<StockQuote> stockQuotes = new ArrayList<StockQuote>();

    for (String symbol : symbolProvider.getSymbolsList()) {
      StockQuote sq = new StockQuote();

      double price = currentFakePrice(symbol);

      sq.setAskPrice("" + price, "" + price);
      sq.setBidPrice("" + price + 1, "" + price + 1);
      sq.setAskSharesSize("999");
      sq.setBidSharesSize("999");
      sq.setCompanyTickerSymbol(symbol);
      sq.setCompanyName(symbol + " company name");
      stockQuotes.add(sq);
    }

    return stockQuotes;
```

```
  }
…
}
```

Třída NewsDataBean

```
package cz.czu.pef.dp.eventGenerator;

import java.io.IOException;
import java.net.*;
import java.text.SimpleDateFormat;
import java.util.*;
import com.sun.syndication.feed.synd.*;
import com.sun.syndication.io.*;
import cz.czu.pef.dp.eventGenerator.pojo.News;
import cz.czu.pef.dp.jms.TopicPublisher;

public class NewsDataBean {

  public void sendNewsData(List<News> newsList, TopicPublisher pub) {
    for (News sq : newsList) {
      String msg = sq.getJsonEvent();
      pub.publishMessage(msg);
    }
  }

  public List<News> getLatestYahooRSSData(SymbolProvider symbolProvider,
Map<String, Date> newsMap) {
    if (newsMap == null) {
      throw new IllegalArgumentException("newsMap is null");
    }
    List<News> newsListAll = new ArrayList<News>();
    for (String companyTickerSymbol : symbolProvider.getSymbolsList()) {
      try {
        Date lastNewsDate = newsMap.get(companyTickerSymbol);
        Date newLastDate = new Date();
        List<News> newsList =
getCurrentYahooRSSDataForTicketSymbol(companyTickerSymbol, lastNewsDate);
        newsListAll.addAll(newsList);

        newsMap.put(companyTickerSymbol, newLastDate);
      }
      catch (Exception e) {
        // skipping errors => continue with another symbol
      }
    }
    return newsListAll;
  }

  private List<News> getCurrentYahooRSSDataForTicketSymbol(String
companyTickerSymbol, Date lastNewsDate) {
    List<News> newsList = new ArrayList<News>();

    XmlReader reader = null;
    try {
      URL url = new URL("http://finance.yahoo.com/rss/headline?s=" +
companyTickerSymbol + "&lg=us");
```

```
      reader = new XmlReader(url);

      SyndFeedInput sfi = new SyndFeedInput();
      SyndFeed feed = sfi.build(reader);

      for (@SuppressWarnings("unchecked")
      Iterator<SyndEntry> i = feed.getEntries().iterator(); i.hasNext();)
{
        SyndEntry entry = (SyndEntry) i.next();

        if (lastNewsDate == null ||
lastNewsDate.before(entry.getPublishedDate())) {
          News news = new News();
          news.setCompanyTickerSymbol(companyTickerSymbol);
          news.setCompanyName(companyTickerSymbol);
          news.setTitle(entry.getTitle());
          news.setPublishedDate(entry.getPublishedDate());
          newsList.add(news);
        }
      }
      return newsList;
    } catch (MalformedURLException e) { throw new RuntimeException(e); }
    catch (IOException e) { throw new RuntimeException(e); }
    catch (IllegalArgumentException e) { throw new RuntimeException(e); }
    catch (FeedException e) { throw new RuntimeException(e); }
    finally {
      if (reader != null) {
        try { reader.close(); }
        catch (IOException e) {
          System.err.println(e.getMessage());
          // skipping error => continues with another symbol
        }
      }
    }
  }

  public List<News> getFakeNewsData(SymbolProvider symbolProvider) {
    List<News> newsList = new ArrayList<News>();

    final Date time = new Date((System.currentTimeMillis() - 1 * 1000) /
1000 * 1000); // 1 second old news
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss.SSSZ");

    for (String symbol : symbolProvider.getSymbolsList()) {
      News news = new News();
      news.setCompanyTickerSymbol(symbol);
      news.setCompanyName(symbol + " company name");
      news.setPublishedDate(time);
      news.setTitle(symbol + " - SOME FAKE NEWS TITLE, time = " +
sdf.format(time));
      newsList.add(news);
    }

    return newsList;
  }
}
```

Třída TopicPublisher

```
package cz.czu.pef.dp.jms;

import javax.jms.*;
import javax.naming.*;
import java.util.Properties;

public class TopicPublisher {
  private String INITIAL_CONTEXT_FACTORY =
"org.apache.activemq.jndi.ActiveMQInitialContextFactory";
  private String CONNECTION_STRING = "tcp://localhost:61616";

  private boolean isConnected = false;
  private String topicName;
  private String initialContextFactory = INITIAL_CONTEXT_FACTORY;
  private String connectionString = CONNECTION_STRING;
  private String factoryName = "TopicConnectionFactory";

  private TopicSession topicSession = null;
  private javax.jms.TopicPublisher topicPublisher = null;
  private InitialContext ctx = null;

  protected TopicPublisher() {
  }

  public TopicPublisher(String initialContextFactory, String
connectionString, String factoryName, String topicName) {
    this.initialContextFactory = initialContextFactory;
    this.connectionString = connectionString;
    this.factoryName = factoryName;
    this.topicName = topicName;
  }

  public TopicPublisher(String topicName) {
    this.topicName = topicName;
  }

  public void connect() {
    if (isConnected) {
      return;
    }
    Properties properties = new Properties();
    TopicConnection topicConnection = null;
    properties.put("java.naming.factory.initial", initialContextFactory);
    properties.put("connectionfactory.QueueConnectionFactory",
connectionString);
    properties.put("topic." + topicName, topicName);

    try {
      ctx = new InitialContext(properties);
      TopicConnectionFactory topicConnectionFactory =
(TopicConnectionFactory) ctx.lookup(factoryName);
      if (topicConnectionFactory == null) {
        throw new RuntimeException("topicConnectionFactory is null");
      }
      topicConnection = topicConnectionFactory.createTopicConnection();
```

```
      topicSession = topicConnection.createTopicSession(false,
Session.AUTO_ACKNOWLEDGE);
      if (topicSession == null) {
        throw new RuntimeException("topicSession is null");
      }
      Topic topic = (Topic) ctx.lookup(topicName);
      if (topic == null) {
       throw new RuntimeException("Topic name "+topicName+" not found.");
      }
      topicPublisher = topicSession.createPublisher(topic);
      isConnected = true;
    }
    catch (JMSException e) {
      throw new RuntimeException("Error in JMS operations", e);
    }
    catch (NamingException e) {
      throw new RuntimeException("Error in initial context lookup", e);
    }
  }

  public void disconnect() {
    if (!isConnected) {
      return;
    }
    if (topicPublisher != null) {
      try { topicPublisher.close(); topicPublisher = null; }
      catch (JMSException e) { throw new RuntimeException(e); }
    }
    if (topicSession != null) {
      try { topicSession.close(); topicSession = null; }
      catch (JMSException e) { throw new RuntimeException(e); }
    }
    if (ctx != null) {
      try { ctx.close(); ctx = null; }
      catch (NamingException e) { throw new RuntimeException(e); }
    }
    isConnected = false;
  }

  public void publishMessage(String messageStr) {
    if (!isConnected) {
      throw new RuntimeException("isConnected is false");
    }
    if (topicSession == null) {
      throw new RuntimeException("topicSession is null");
    }
    TextMessage textMessage;
    try {
      textMessage = topicSession.createTextMessage(messageStr);
      textMessage.setJMSExpiration(120000L); // default expiration
      topicPublisher.publish(textMessage);
    }
    catch (JMSException e) {
      throw new RuntimeException(e);
    }
  }
}
```

Třída News

```
package cz.czu.pef.dp.eventGenerator.pojo;

import java.util.Date;

public class News {
  private String companyTickerSymbol;
  private String companyName;
  private String title;
  private Date publishedDate;

  public String getCompanyTickerSymbol() {
    return companyTickerSymbol;
  }
  public void setCompanyTickerSymbol(String companyTickerSymbol) {
    this.companyTickerSymbol = companyTickerSymbol;
  }
  public String getCompanyName() {
    return companyName;
  }
  public void setCompanyName(String companyName) {
    this.companyName = companyName;
  }
  public String getTitle() {
    return title;
  }
  public void setTitle(String title) {
    this.title = title;
  }
  public Date getPublishedDate() {
    return publishedDate;
  }
  public void setPublishedDate(Date publishedDate) {
    this.publishedDate = publishedDate;
  }
  public String getJsonEvent() {
    StringBuffer sb = new StringBuffer();
    return sb.append("{ NewsEvent: ").append("{ TickerSymbol:
\"").append(companyTickerSymbol).append("\",").append(" CompanyName:
\"").append(companyName + "\",").append(" Title: \"").append(title +
"\",").append(" PublishedDate:
").append(publishedDate.getTime()).append(" } }").toString();
  }
}
```

Třída StockQuote

```
package cz.czu.pef.dp.eventGenerator.pojo;

public class StockQuote {
  private String companyTickerSymbol;
  private String companyName;
  private String askPrice;
  private String bidPrice;
  private String askSharesSize;
  private String bidSharesSize;
```

```
  public String getCompanyTickerSymbol() {
    return companyTickerSymbol;
  }
  public void setCompanyTickerSymbol(String companyTickerSymbol) {
    this.companyTickerSymbol = companyTickerSymbol;
  }
  public String getCompanyName() {
    return companyName;
  }
  public void setCompanyName(String companyName) {
    this.companyName = companyName;
  }
  public String getAskPrice() {
    return askPrice;
  }
  public void setAskPrice(String askPrice, String askPriceRealTime) {
    // skip invalid data (N/A ... Not Available)
    if (askPriceRealTime != null && !"0.00".equals(askPriceRealTime) &&
!"N/A".equals(askPriceRealTime)) {
      this.askPrice = askPriceRealTime;
      return;
    }
    if ("0.00".equals(askPrice) || "N/A".equals(askPrice)) {
      this.askPrice = null;
    }
    else { this.askPrice = askPrice; }
  }

  public String getBidPrice() {
    return bidPrice;
  }
  public void setBidPrice(String bidPrice, String bidPriceRealTime) {
    // skip invalid data (N/A ... Not Available)
    if (bidPriceRealTime != null && !"0.00".equals(bidPriceRealTime) &&
!"N/A".equals(bidPriceRealTime)) {
      this.bidPrice = bidPriceRealTime;
      return;
    }
    if ("0.00".equals(bidPrice) || "N/A".equals(bidPrice)) {
      this.bidPrice = null;
    }
    else { his.bidPrice = bidPrice; }
  }
  public String getAskSharesSize() {
    return askSharesSize;
  }
  public void setAskSharesSize(String askSharesSize) {
    if ("0.00".equals(askSharesSize) || "N/A".equals(askSharesSize)) {
      this.askSharesSize = null;
    }
    else { this.askSharesSize = askSharesSize; }
  }
  public String getBidSharesSize() {
    return bidSharesSize;
  }
  public void setBidSharesSize(String bidSharesSize) {
    if ("0.00".equals(bidSharesSize) || "N/A".equals(bidSharesSize)) {
```

```
      this.bidSharesSize = null;
    }
    else { this.bidSharesSize = bidSharesSize; }
  }

  public String getJsonEvent() {
    StringBuffer sb = new StringBuffer();
    return sb.append("{ StockQuoteEvent: ").append("{ TickerSymbol:
\"").append(companyTickerSymbol).append("\",").append(" CompanyName:
\"").append(companyName + "\",").append(" AskPrice: ").append(new
Double(askPrice)).append(", ").append(" BidPrice: ").append(new
Double(bidPrice)).append(", ").append(" AskSize: ").append(new
Double(askSharesSize)).append(", ").append(" BidSize: ").append(new
Double(bidSharesSize)).append(" } }").toString();
  }
}
```

Obsah konfiguračního souboru current_symbols.txt

Tento soubor obsahuje v jednotlivých řádcích seznam zkratek akciových titulů, pro které jsou aplikací Generátor událostí získávány informace z Yahoo služeb. Příklad obsahu souboru:

```
AAPL
GOOG
MSFT
YHOO
```

Příloha C – Konfigurace produktu Apache ActiveMQ

Pro plynulý tok dat v rámci výměny JMS topic zpráv je nutné provést následující konfiguraci, které vypne výchozí chování, kdy je uměle omezována produkce zpráv.

V souboru ${APACHE_ACTIVEMQ}/conf/activemq.xml je zapotřebí přidat následující XML fragment do XML elementu v cestě „<broker> <destinationPolicy> <policyMap> <policyEntries>“:

```
<policyEntry topic="cz.czu.pef.>" producerFlowControl="false"
memoryLimit="100mb">
      <dispatchPolicy>
            <strictOrderDispatchPolicy />
      </dispatchPolicy>
      <!-- A value of -1 disables the discarding of messages. -->
      <pendingMessageLimitStrategy>
            <constantPendingMessageLimitStrategy limit="-1"/>
      </pendingMessageLimitStrategy>
      <!--  5 minutes worth -->
      <subscriptionRecoveryPolicy>
            <timedSubscriptionRecoveryPolicy recoverDuration="300000" />
      </subscriptionRecoveryPolicy>
</policyEntry>
```

Příloha D – Zdrojové a konfigurační soubory systému pro zpracování komplexních událostí

Tato kapitola obsahuje přílohy pro konfiguraci produktu WSO2 CEP.

Napojení na produkt Apache ActiveMQ

Postup napojení produktu WSO2 CEP na messaging produkt Apache ActiveMQ:

1.) Zkopírovat následující soubory z adresáře ${APACHE_ACTIVEMQ}/lib/ do adresáře ${WSO2_CEP}/repository/components/lib/:
   - activemq-client-5.9.0.jar
   - activemq-broker-5.9.0.jar
   - geronimo-j2ee-management_1.1_spec-1.0.1.jar

2.) Restartovat produkt WSO2 CEP

3.) Přihlásit se do administrační konzole produktu WSO2 CEP

4.) Přidat vstupní JMS adaptér:
   - Navigovat do “Home > Configure > Event Processor Configs > Input Event Adaptors“.
   - Zvolit volbu „Add Input Event Adaptor“ a nastavit jednotlivé hodnoty dle následujícího XML fragmentu:

```
<?xml version="1.0" encoding="UTF-8"?>
<inputEventAdaptor name="ActiveMQ-input" statistics="disable"
trace="disable" type="jms"
xmlns="http://wso2.org/carbon/eventadaptormanager">
      <property
name="java.naming.provider.url">tcp://localhost:61616</property>
      <property name="transport.jms.SubscriptionDurable">false</property>
      <property name="transport.jms.UserName">admin</property>
      <property
name="java.naming.factory.initial">org.apache.activemq.jndi.ActiveMQIniti
alContextFactory</property>
      <property
name="transport.jms.ConnectionFactoryJNDIName">TopicConnectionFactory</pr
operty>
      <property name="transport.jms.Password">admin</property>
      <property name="transport.jms.DestinationType">topic</property>
</inputEventAdaptor>
```

5.) Přidat výstupní JMS adaptér:
   - Navigovat do “Home > Configure > Event Processor Configs > Output Event Adaptors“.

- Zvolit volbu „Add Output Event Adaptor“ s provést stejné nastavením jako v předchozím kroku.

Rozšíření jazyka Siddhi

Tato kapitola obsahuje implementaci tříd rozšiřujících možnosti jazyka Siddhi. Jsou zde uvedeny třídy zavádějící nový agregovaný typ realizující výpočet exponenciálního klouzavého průměru.

Třída ExponentialMovingAverageAggregatorFactory realizuje návrhový vzor Factory a slouží pro vytváření instancí objektu ExponentialMovingAverageAggregatorDouble.

Třída ExponentialMovingAverageAggregatorDouble realizuje výpočet exponenciálního klouzavého průměru z proudu dat, který je poskytován produktem WSO2 CEP.

Maven konfigurační soubor projektu

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
      <modelVersion>4.0.0</modelVersion>
      <groupId>cz.czu.pef.wso2.extensions</groupId>
      <artifactId>Stribrny-wso2-extension</artifactId>
      <version>1.0.0-SNAPSHOT</version>

      <dependencies>
            <dependency>
                  <groupId>org.wso2.siddhi</groupId>
                  <artifactId>siddhi-core</artifactId>
                  <version>2.0.0-wso2v4</version>
            </dependency>
      </dependencies>

      <repositories>
        <repository>
            <id>wso2-maven2-repository</id>
            <name>WSO2 Maven2 Repository</name>
            <url>http://dist.wso2.org/maven2</url>
        </repository>
      </repositories>
</project>
```

Třída ExponentialMovingAverageAggregatorFactory

```
package cz.czu.pef.siddhi.extensions;

import org.wso2.siddhi.core.exception.*;
import
org.wso2.siddhi.core.query.selector.attribute.factory.OutputAttributeAggr
egatorFactory;
import
org.wso2.siddhi.core.query.selector.attribute.handler.OutputAttributeAggr
egator;
import org.wso2.siddhi.query.api.definition.Attribute;
import org.wso2.siddhi.query.api.definition.Attribute.Type;
import org.wso2.siddhi.query.api.extension.annotation.SiddhiExtension;

@SiddhiExtension(namespace = "cz_czu_pef", function =
"exponentialMovingAverage")
public class ExponentialMovingAverageAggregatorFactory implements
OutputAttributeAggregatorFactory {

  public OutputAttributeAggregator createAttributeAggregator(Type[]
types) {
    if (types.length > 1) {
      throw new QueryCreationException("Exponential Moving Average can
only have one parameter");
    }
    Attribute.Type type = types[0];
    switch (type) {
      case STRING:
      case INT:
      case LONG:
      case FLOAT:
      case BOOL:
        throw new OperationNotSupportedException("Exponential Moving
Average not supported for type " + type.toString());
      case DOUBLE:
        return new ExponentialMovingAverageAggregatorDouble();
      default:
    }
    throw new OperationNotSupportedException("Exponential Moving Average
not supported for " + type);
  }
}
```

Třída ExponentialMovingAverageAggregatorDouble

```
package cz.czu.pef.siddhi.extensions;

import
org.wso2.siddhi.core.query.selector.attribute.handler.OutputAttributeAggr
egator;
import org.wso2.siddhi.query.api.definition.Attribute;

public class ExponentialMovingAverageAggregatorDouble implements
    OutputAttributeAggregator {
  private double emaCurrentValue;
  private int count = 0;
  private static final Attribute.Type type = Attribute.Type.DOUBLE;

  public Attribute.Type getReturnType() {
    return type;
  }
  public OutputAttributeAggregator newInstance() {
    return new ExponentialMovingAverageAggregatorDouble();
  }
  public void destroy() {
  }

  public Object processAdd(Object obj) {
      double value = (Double)obj;
      if (count <= 0) {
        emaCurrentValue = value;
        count++;
        return emaCurrentValue;
      }
      count++;

      // Zvolena alpha = 2 / (N+1)
      double alpha = 2.0d / ((double)count + 1.0d);

// EMA(current) =EMA(previous) + alpha * (price(current) - EMA(previous))
        double newValue = emaCurrentValue + alpha * (value -
emaCurrentValue);
        emaCurrentValue = newValue;
        return emaCurrentValue;
  }

  public Object processRemove(Object obj) {
    count--;
    if (count <= 0) {
      emaCurrentValue = 0;
      return 0;
    }
    // Odstranovana hodnota ma jiz v EWMA maly vyznam => nedelej nic.
    return emaCurrentValue;
  }
  public double getCurrentValue() {
    return emaCurrentValue;
  }

}
```

Příloha E – Zdrojové a konfigurační soubory ESB

Tato kapitola obsahuje přílohy pro konfiguraci produktu WSO2 ESB.

Napojení na produkt Apache ActiveMQ

Postup napojení produktu WSO2 ESB na messaging produkt Apache ActiveMQ:

1.) Zkopírovat následující soubory z adresáře ${APACHE_ACTIVEMQ}/lib/ do adresáře ${WSO2_ESB}/repository/components/lib/:
   - activemq-client-5.9.0.jar
   - activemq-broker-5.9.0.jar
   - geronimo-j2ee-management_1.1_spec-1.0.1.jar

2.) Restartovat produkt WSO2 ESB

3.) Přihlásit se do administrační konzole produktu WSO2 ESB

4.) Navigovat do cesty: Home > Manage > Service Bus > Message Stores

5.) Po vybrání volby “Add JMS Message Store“ použít následující konfiguraci:

```
<messageStore name="ActiveMQ"
class="org.apache.synapse.message.store.impl.jms.JmsStore"
xmlns="http://ws.apache.org/ns/synapse">
       <parameter
name="java.naming.factory.initial">org.apache.activemq.jndi.ActiveMQIniti
alContextFactory</parameter>
       <parameter
name="java.naming.provider.url">tcp://localhost:61616</parameter>
       <parameter
name="store.jms.connection.factory">TopicConnectionFactory</parameter>
       <parameter name="store.jms.username">admin</parameter>
       <parameter name="store.jms.password">admin</parameter>
       <parameter name="store.jms.JMSSpecVersion">1.1</parameter>
</messageStore>
```

Povolení JMS přenosů v konfiguraci ESB

Užití JMS je nutné povolit v konfiguraci produktu, a to vložením následujících fragmentů konfigurace do konfiguračního souboru v cestě ${WSO2_ESB}/repository/conf/axis2/axis2.xml.

1.) Fragment konfigurace pro povolení JMS přenosů (nutné odkomentovat sekci transportReceiver):

```
<transportReceiver name="jms"
class="org.apache.axis2.transport.jms.JMSListener">
      <parameter name="myTopicConnectionFactory" locked="false">
```

```
            <parameter name="java.naming.factory.initial"
locked="false">org.apache.activemq.jndi.ActiveMQInitialContextFactory</pa
rameter>
            <parameter name="java.naming.provider.url"
locked="false">tcp://localhost:61616</parameter>
            <parameter name="transport.jms.ConnectionFactoryJNDIName"
locked="false">TopicConnectionFactory</parameter>
            <parameter name="transport.jms.ConnectionFactoryType"
locked="false">topic</parameter>
      </parameter>

      <parameter name="myQueueConnectionFactory" locked="false">
            <parameter name="java.naming.factory.initial"
locked="false">org.apache.activemq.jndi.ActiveMQInitialContextFactory</pa
rameter>
            <parameter name="java.naming.provider.url"
locked="false">tcp://localhost:61616</parameter>
            <parameter name="transport.jms.ConnectionFactoryJNDIName"
locked="false">QueueConnectionFactory</parameter>
            <parameter name="transport.jms.ConnectionFactoryType"
locked="false">queue</parameter>
      </parameter>

      <parameter name="default" locked="false">
            <parameter name="java.naming.factory.initial"
locked="false">org.apache.activemq.jndi.ActiveMQInitialContextFactory</pa
rameter>
            <parameter name="java.naming.provider.url"
locked="false">tcp://localhost:61616</parameter>
            <parameter name="transport.jms.ConnectionFactoryJNDIName"
locked="false">QueueConnectionFactory</parameter>
            <parameter name="transport.jms.ConnectionFactoryType"
locked="false">queue</parameter>
      </parameter>
</transportReceiver>
```

2.) Fragment konfigurace pro povolení JMS transakcí:

```
<transaction timeout="30000">
      <parameter
name="java.naming.factory.initial">org.apache.activemq.jndi.ActiveMQIniti
alContextFactory</parameter>
      <parameter
name="java.naming.provider.url">tcp://localhost:61616</parameter>
      <parameter
name="UserTransactionJNDIName">UserTransaction</parameter>
      <parameter
name="TransactionManagerJNDIName">TransactionManager</parameter>
</transaction>
```

3.) Restart WSO2 ESB produktu

Konfigurace proxy služby BidPriceRaisingProxy

```
<?xml version="1.0" encoding="UTF-8"?>
<proxy xmlns="http://ws.apache.org/ns/synapse"
       name="BidPriceRaisingProxy"
       transports="jms"
       statistics="disable"
       trace="enable"
       startOnLoad="true">
   <target>
      <inSequence>
         <payloadFactory media-type="xml">
            <format>
               <cz:buyLimitOrder xmlns:cz="cz.czu.pef.dp">
                  <tickerSymbol xmlns="">$1</tickerSymbol>
                  <limitPrice xmlns="">$2</limitPrice>
               </cz:buyLimitOrder>
            </format>
            <args>
               <arg evaluator="json"
expression="$.event.payloadData.TickerSymbol"/>
               <arg evaluator="json"
expression="$.event.payloadData.ShortTermAveragePrice"/>
            </args>
         </payloadFactory>
         <log level="full"/>
         <header name="Action" value="&#34;&#34;"/>
      </inSequence>
      <outSequence>
         <log level="full"/>
      </outSequence>
      <endpoint>
         <address
uri="http://localhost:9773/CZU_PEF_StockExchangeApp/services/c_z_u__p_e_f
__stock_exchange_app" format="soap11"/>
      </endpoint>
   </target>
   <parameter name="transport.jms.ContentType">
      <rules>
         <jmsProperty>contentType</jmsProperty>
         <default>application/json</default>
      </rules>
   </parameter>
   <parameter
name="transport.jms.ConnectionFactory">myTopicConnectionFactory</paramete
r>
   <parameter name="transport.jms.DestinationType">topic</parameter>
   <parameter
name="transport.jms.Destination">cz.czu.pef.topics.BidPriceRaisingStockQu
otes</parameter>
   <description/>
</proxy>
```

Konfigurace proxy služby BidPriceLoweringProxy

```
<?xml version="1.0" encoding="UTF-8"?>
<proxy xmlns="http://ws.apache.org/ns/synapse"
       name="BidPriceLoweringProxy"
       transports="jms"
       statistics="disable"
       trace="enable"
       startOnLoad="true">
   <target>
      <inSequence>
         <payloadFactory media-type="xml">
            <format>
               <cz:sellLimitOrder xmlns:cz="cz.czu.pef.dp">
                  <tickerSymbol xmlns="">$1</tickerSymbol>
                  <limitPrice xmlns="">$2</limitPrice>
               </cz:sellLimitOrder>
            </format>
            <args>
               <arg evaluator="json"
expression="$.event.payloadData.TickerSymbol"/>
               <arg evaluator="json"
expression="$.event.payloadData.ShortTermAveragePrice"/>
            </args>
         </payloadFactory>
         <log level="full"/>
         <header name="Action" value="&#34;&#34;"/>
      </inSequence>
      <outSequence>
         <log level="full"/>
      </outSequence>
      <endpoint>
         <address
uri="http://localhost:9773/CZU_PEF_StockExchangeApp/services/c_z_u__p_e_f
__stock_exchange_app" format="soap11"/>
      </endpoint>
   </target>
   <parameter name="transport.jms.ContentType">
      <rules>
         <jmsProperty>contentType</jmsProperty>
         <default>application/json</default>
      </rules>
   </parameter>
   <parameter
name="transport.jms.ConnectionFactory">myTopicConnectionFactory</paramete
r>
   <parameter name="transport.jms.DestinationType">topic</parameter>
   <parameter
name="transport.jms.Destination">cz.czu.pef.topics.BidPriceLoweringStockQ
uotes</parameter>
   <description/>
</proxy>
```

Příloha F – Zdrojové soubory systému pro obchodování

Přílohy obsažené v této kapitole slouží pro vývoj systému sloužícího k fiktivnímu zpracování pokynů na akciovém trhu.

Třída implementace webové služby CZU_PEF_StockExchangeApp

```
package cz.czu.pef.dp;

import javax.jws.*;

@WebService(serviceName = "CZU_PEF_StockExchangeApp",
targetNamespace="cz.czu.pef.dp",
      name="CZU_PEF_StockExchangeApp",
      portName="CZU_PEF_StockExchangeAppPort")
public class CZU_PEF_StockExchangeApp {
  private static Long orderIdCounter = new Long(1);

  @WebMethod(operationName = "buyLimitOrder", action="buyLimitOrder")
  @WebResult(name="orderId")
  public long buyLimitOrder(@WebParam(name = "tickerSymbol") String
tickerSymbol,
                @WebParam(name = "limitPrice") Double limitPrice,
                @WebParam(name = "amount") Double amount) {
    long myId;
    synchronized (orderIdCounter) {
      myId = orderIdCounter++;
    }
    System.out.println("buyLimitOrder("+myId+"): tickerSymbol=" +
tickerSymbol + ", " + limitPrice + ", " + amount);
    return myId;
  }

  @WebMethod(operationName = "sellLimitOrder", action="sellLimitOrder")
  @WebResult(name="orderId")
  public long sellLimitOrder(@WebParam(name = "tickerSymbol") String
tickerSymbol,
                @WebParam(name = "limitPrice") Double limitPrice,
                @WebParam(name = "amount") Double amount) {
    long myId;
    synchronized (orderIdCounter) {
      myId = orderIdCounter++;
    }
    System.out.println("sellLimitOrder("+myId+"): tickerSymbol=" +
tickerSymbol + ", " + limitPrice + ", " + amount);
    return myId;
  }
}
```