Integrating Systems with Event Driven Architecture

Eoin Woods
www.eoinwoods.info

About Me

- Software architect at UBS Investment Bank
  - responsible for synthetic equity platform
- Software architect for ~10 years
- Author of “Software Systems Architecture” book with Nick Rozanski
- IASA and BCS Fellow, IET member, CEng

Agenda

- Integrating Systems
- Event Driven Architecture
- Event Driven Integration
- Case Study
- Conclusion

Integrating Systems

Integrating Systems

- We integrate systems all the time
  - for accidental reasons (evolution, tactical solutions, …)
  - deliberately (partitioning, system simplicity, …)
- We do it in a small number of standard ways
  - file transfer (classic EAI – Informatica et al)
  - shared databases and replication
  - messaging (e.g. send entity changes via messages)
  - events (sending events between systems, usually via messages)
- File transfer is probably the most popular
  - messaging probably next
  - integration using events is quite rare
- This talk is about integration using events
  - why we should do it more often

EAI Approaches

- File Transfer
  - classic ETL (extract file, transform file, move file, load file)
  - simple, but timeliness, duplication and effort are challenges
  - ETL tools give you a DSL for ETL (e.g. Informatica)
    - primitives for extract/load, split, transform, filter, aggregate, match, …
- Shared Database
  - read only shared database (e.g. reference data distribution)
  - read/write shared database shared between systems
  - simple & consistent (single copy of the data)
  - data ownership, scalability, schema evolution are major challenges
- Database Replication
  - simple, reliable, largely transparent to applications
  - consistency and timeliness are the main problems

EAI Approaches

- Messaging
  - broadcast or point-to-point exchange of information
  - messages can contain documents (entities), requests or events
  - allows (near) real-time integration and batch style integration
  - persistent or transient messaging
  - good infrastructure , scalable, performant, reliable, extensible
  - challenges are complexity, message semantics, synchronization and big data
- Events
  - architectural style where communication via events not entities
  - usually a specialisation of Messaging
  - the focus of this talk (much more later)

Event Driven Architecture

Event Driven Architecture (EDA)

- From the Event Processing Technical Society …

  [EDA is] *an architectural style in which some of the components are event driven and communicate by means of events.*

  [Event Driven means] *the behaviour of a device, software module or other entity whose execution is in response to the arrival of events from external or internal sources*

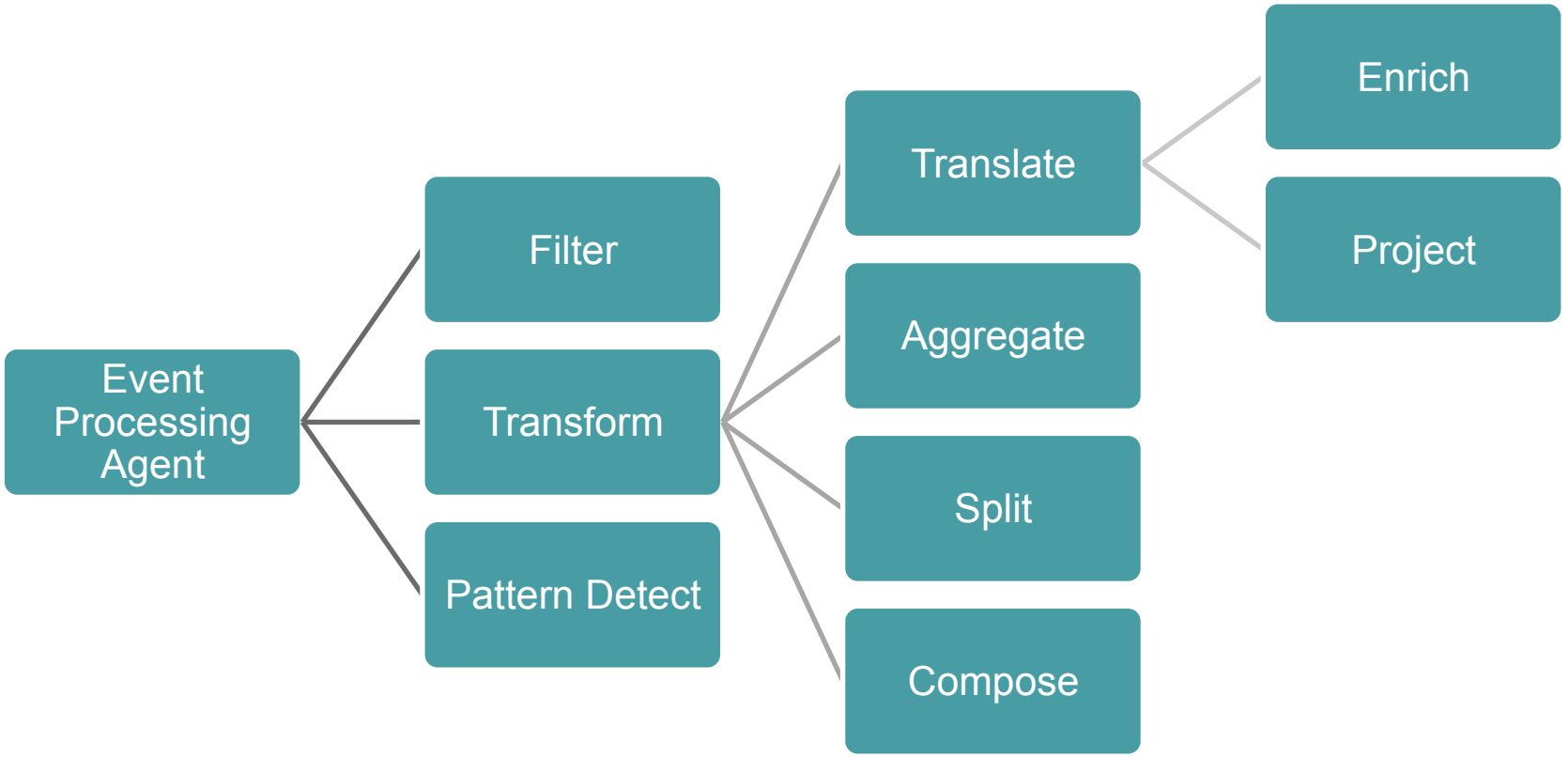
  [An Event is] *an object that represents encodes or records an event, generally for the purpose of computer processing*

  From the Event Processing Glossary – v1.1

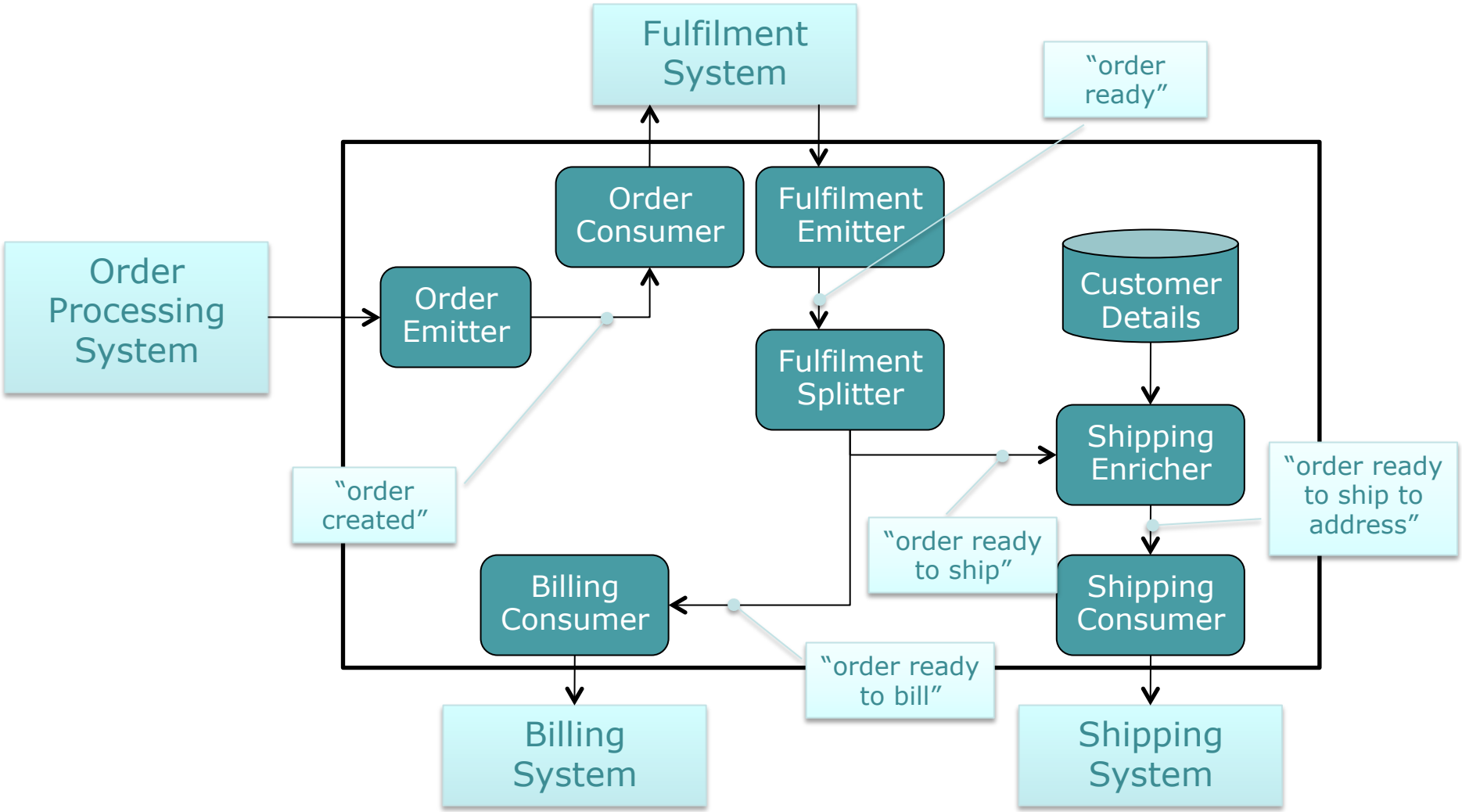
EDA - Overview

- A simple idea, using a standard set of concepts
  - Event producers and consumers
  - Event processing agents
  - Events
    - raw events and derived events
    - events contain a set of header fields and a body (structured or opaque)
  - Event channel
  - Shared state
- These concepts form a standard reference model
  - an Event Processing Network (EPN)
- The work of an EPN can be described as
  - Detect, Decide and Respond
    - or Sense, Analyse and Act

EDA – Event Processing Agents



EDA – Example Network



Types of Event Processing

- Simple event processing
  - events (typically) created for state changes or external occurrences
  - events processed one by one by “downstream” system(s)
  - often all that is needed for system integration problems
- Event Stream Processing (ESP)
  - filtering and processing of streams of (related) events (e.g. prices)
  - allows EPN to recognise events of note & to create derived measures or events (e.g. avg value threshold)
- Complex Event Processing (CEP)
  - complex events are those derived from other events (members)
  - CEP allows EPN to recognise relationships between events
    - e.g. impending vehicle arrivals that will cause overload at depot

EDA – Strengths and Weaknesses

- Very strong decoupling (+)
  - when done well and the rules are obeyed
- Very good scalability (+)
  - inherently scalable and distributed architectural style
- Very flexible extension and evolution (+)
  - again, if you stick to the rules
- Can be complicated to implement (-)
  - a lot of state, a lot of pieces, unfamiliar, need tight semantics
- Causality and traceability can be difficult to achieve (-)
  - makes tracking problems difficult, needs to be built in from start
- Monitoring is crucial to provide visibility (-)
  - otherwise failure isn't evident, things just "don't happen"

EDA – Common Problems

- Inadvertent Coupling
  - events can morph into requests … suddenly coupling emerges
- Failure Recovery
  - recovery can be complicated: system state? in-flight events?
  - design recovery early with monotonic consumers and replay
- Latency
  - processing work unit can take longer than with a monolithic system
- Event Semantics
  - just defining a set of fields is fatal … what does the event *mean*?
- Accessing Reference Data
  - many EPAs need to access reference data – this can become a new bottleneck or consistency problem

Event Driven Integration

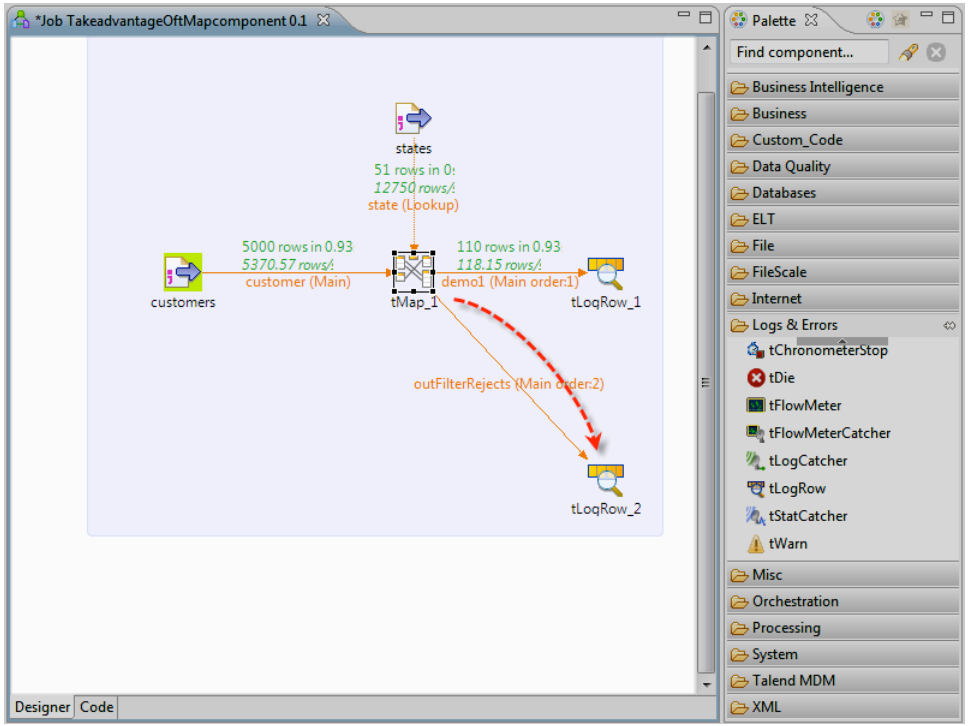
Integration Today

- The most common approach to EAI is via ETL
  - extract entities (rows) from one database
  - transform as needed and load into other database(s)
  - typically using a product like Informatica, Ab Initio or Talend
- Strengths of traditional ETL
  - simple, universal mechanism (files) with powerful tools available
  - relatively easy development, easy to monitor and debug
- Problems with traditional ETL
  - usually results in lots of “point to point” links (fragile)
  - nearly always couples systems, even if implicitly (e.g. file formats)
  - the ETL hub can be a scalability bottleneck
  - usually needs to be batch based (so poor timeliness)
  - ETL tools can be expensive (though so can hand coding!)

Integration Today

However, ETL is a perfectly viable approach

particularly with tool support …



But could we do better?

An Integration Alternative

- What are we looking for?
  - ease of build => good tools, standard technologies, simple ideas
  - ease of evolution => low coupling, high independence
  - scalability => workload partitioning, no shared elements
  - data integrity => no replication/duplication, real time updates
- What do we need to do?
  - recognise changes in source systems
    - encode and dispatch them as system neutral structures
  - transform changes
    - split, aggregate, compose, filter, match, enrich, …
  - propagate changes
    - apply the changes to systems in a system specific way
- What would allow this?
  - Events!

Event Driven Integration

- Strong analogy between file-based EAI and EDA
  - Both connect systems in a non-intrusive manner, but event driven integration transmits events, not records

EAI Concept	EDA Concept
Data Source	Event Emitter / Source
Data Sink or Destination	Event Consumer
Transformation (split, match, filter, aggregate, combine, enrich,…)	Event Processing Agent (split, match, filter, aggregate, combine, enrich,…)

EDA avoids some EAI problems

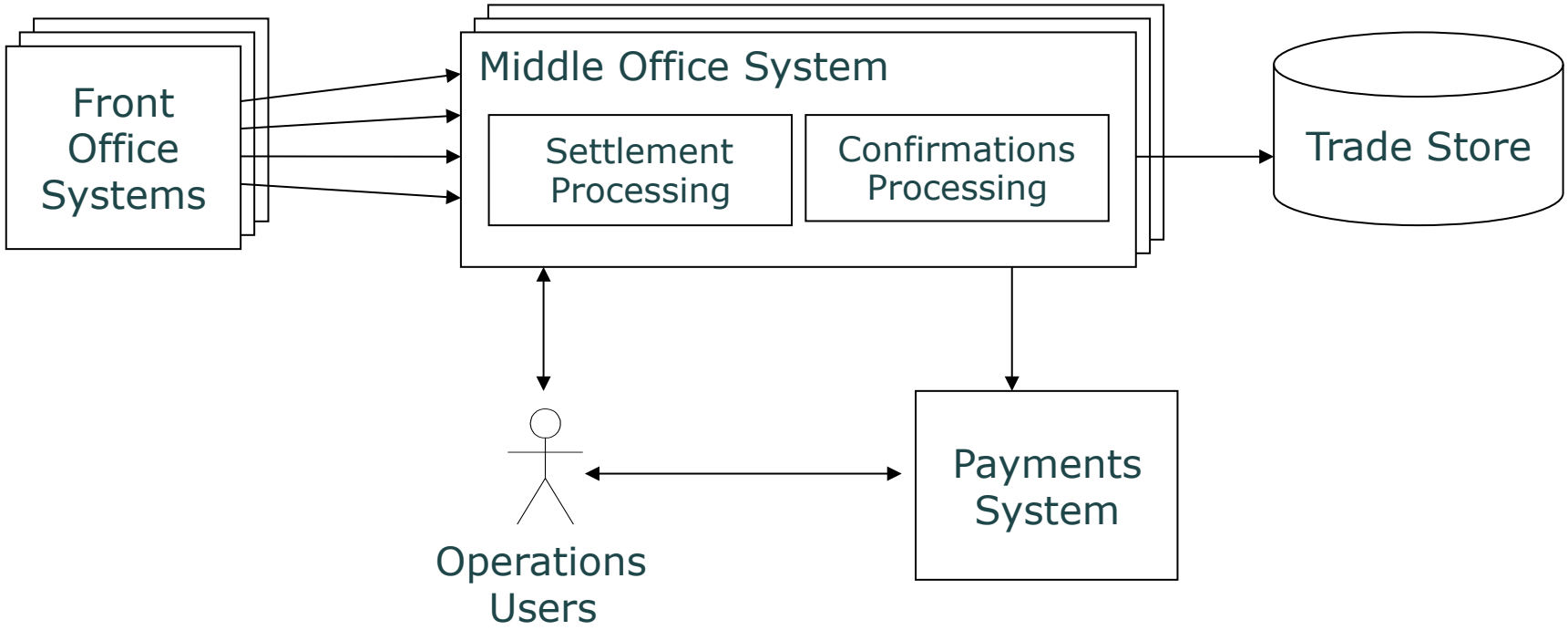
- Data formats don't leak out and couple systems
- no point-to-point connections
  - easier change and extension
- easy workload partitioning for scalability
- little or no sharing between systems
  - allowing separate evolution
- events are pushed, not pulled
  - near real-time updates

Implementing EDI

Ad Hoc	• send messages that look like events • ad hoc event handling in each application
Custom	• standardise model and workflow • implement custom framework
BPM/ESB	• many products offer tools that can be used for event processing
CEP Engine	• complex event processors • also useful tools for simpler cases

Case Study

Case Study Starting Point



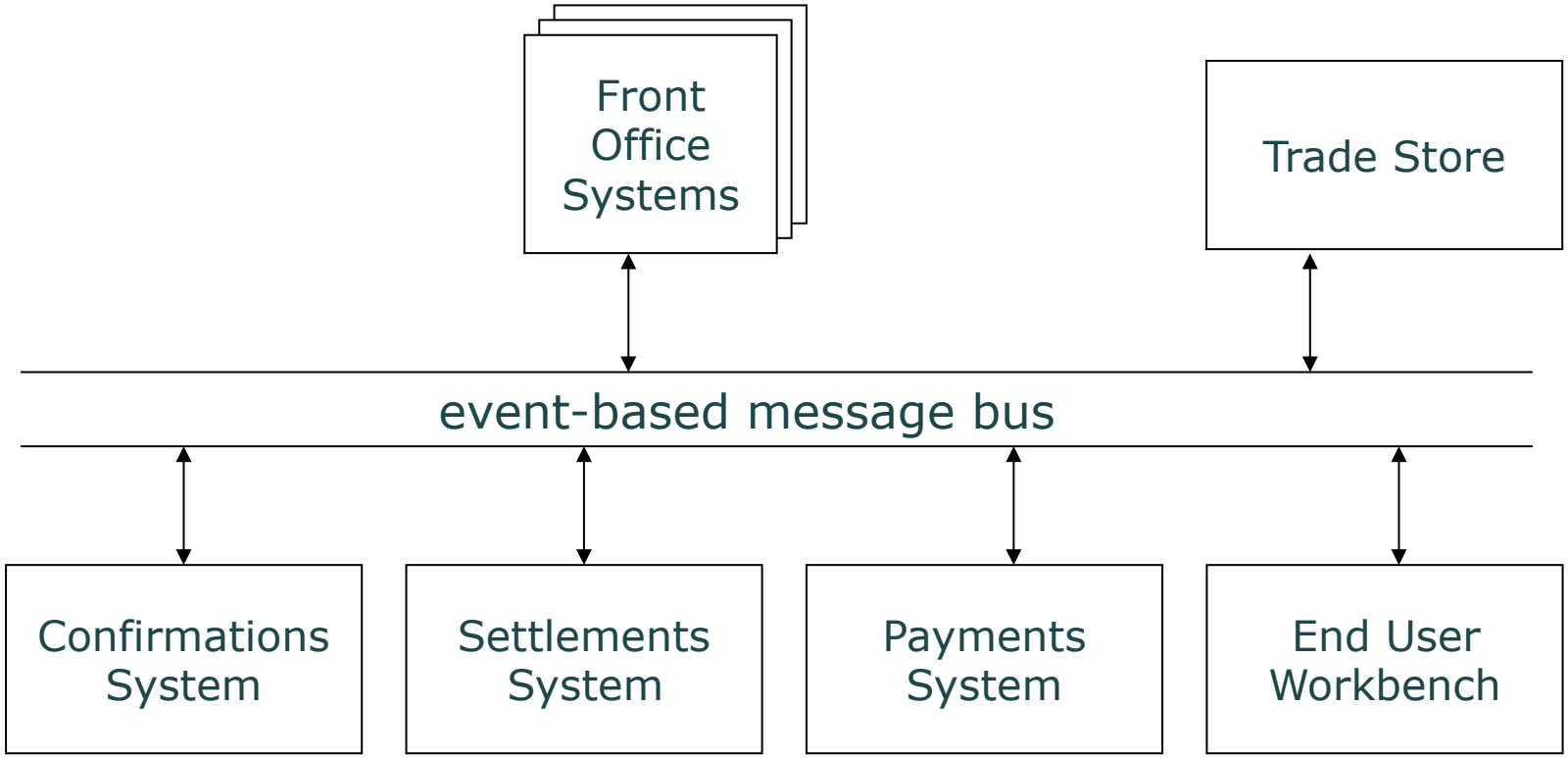
Case Study Background

- Capital Markets domain (Fixed Income Middle Office)
  - existing monolithic system had served well but become difficult to scale and evolve
  - a new middle office environment for FI rates products needed
  - didn't want a single large system for the replacement
- Future State Architecture and Build project
  - decision made to build 5 independent systems
    - all 5 must cooperate to implement the trade lifecycle
  - very strong desire to avoid technical or organisational coupling
    - avoid shared release cycle, no shared database, no point to point links
  - no special purpose EAI or EDA software used
    - but ended up using events for integration
    - and created a custom UML state chart evaluation library

Case Study Architecture Decisions

- Messaging identified as a key technology
  - desire for independence, lack of coupling, scalability
- Previous experience pointed to this not being enough
  - coupling and complexity can quickly occur with ad-hoc messaging
- Event based messaging identified as the solution
  - the systems all broadcast trade lifecycle events
  - message subscriptions route events to the right processor(s)
- The architectural implications were significant
  - no request/response interactions
  - event emitters independent of event consumers
  - no shared entity store that everything reads and writes
  - systems use state charts to track trade lifecycle
  - very large events use hybrid approach (events are state refs)

The Replacement



Case Study Results

- Smaller systems, better defined responsibilities
  - easier evolution
- Avoided coupling
  - well defined neutral (evolving) event model
- No direct dependencies between systems
  - no shared entity stores
- Development team independence
  - separate development and release lifecycles
  - clear coordination points (event model & lifecycle)

Changes to Design Thinking

- Announce don't ask
  - don't ask for data, announce your lifecycle events
  - key change in thinking, forces loose coupling
- Send events not entities
  - transmit entity lifecycle events, not entities themselves
- Don't assume event processing is perfect
  - use "failsafe" processing to spot missing/extra events
- Monitoring is essential
  - needs to be built in from the start

Case Study Lessons Learned

- Event model is key to reliability and evolution
  - needs strong semantics and support for event model evolution
- Monitoring and fail-safe processing need from the start
  - don't drop unrecognised events, flag unexpected/missing events
- Asynchronous processing can cause problems
  - race conditions can occur if assumptions made about propagation
- System-wide view is necessary
  - no one system knows the whole story – "end to end" view needed
- Partitioning needs to be considered carefully
  - what information needs to be where? how to partition cheaply
- Designing and developing using events needs practice
  - request/response is very ingrained in most of us

In Conclusion

Summary

- Integrating systems is a necessity
  - sometimes a good choice, sometimes not, always needed
- Traditional file-based EAI has been an enduring approach
  - but it has limitations (coupling, scaling, effort, evolution, …)
- Other approaches have proven even more problematic
  - ad-hoc messaging has potential but gets messy quickly
  - shared database is seductive but change is hard in practice
- Event driven integration has strengths of other approaches
  - familiar processing models from EAI
  - familiar transport technologies from enterprise messaging
  - familiar neutral schema idea from shared database

Summary (ii)

- Event driven integration solves many traditional problems
  - builds on many of the strengths of other approaches
  - drives coupling out of the system
  - scalability across the integration infrastructure
  - flexible options for change and extension
  - timely consistency while avoiding shared data store
- Naturally brings its own challenges too
  - sophisticated approach, needing careful design
  - event model needs to be done early and to be extensible
  - asynchronous nature can lead to race conditions
  - needs monitoring and failsafe processing
  - event causality and tracing can be difficult (harder debugging)

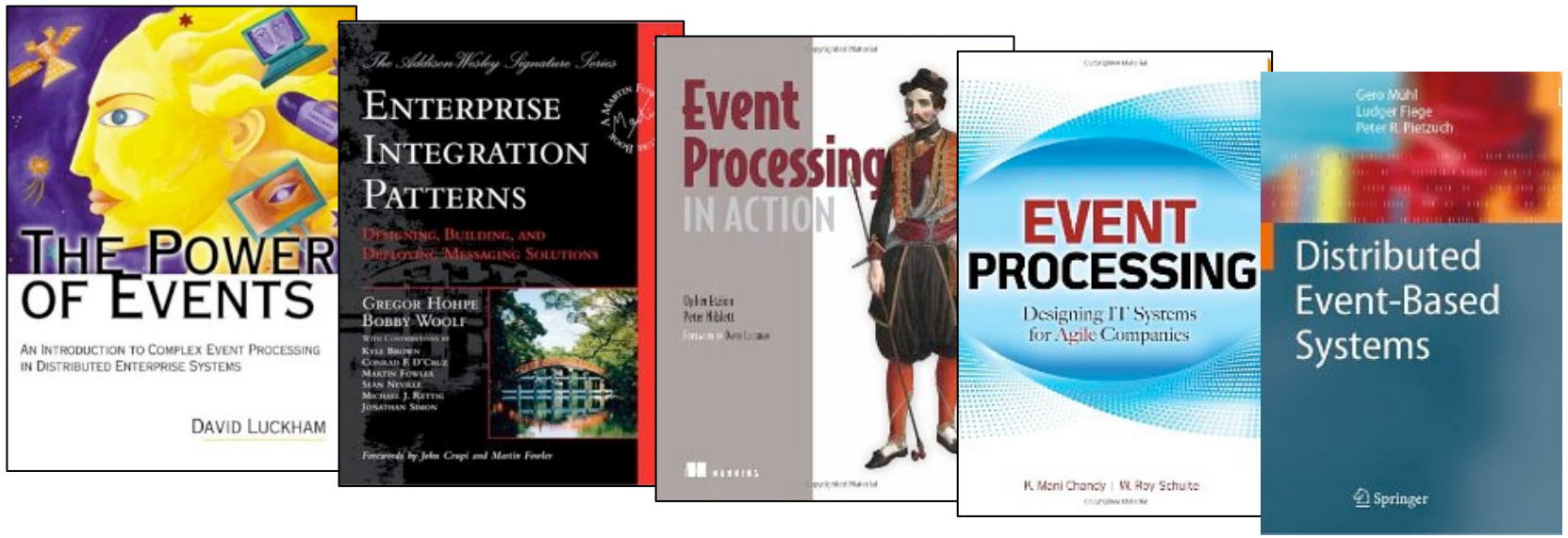
More Information

- Event Processing Technical Society
  - articles, Event Processing Glossary, event listings, (quiet) forums
  - http://www.ep-ts.com
- David Luckham's site
  - articles, news, pointers to research, event listings
  - http://www.complexevents.com
- Opher Etzion's blog and presentations
  - http://epthinking.blogspot.com
  - http://www.slideshare.net/opher.etzion
- Tim Bass' blog and presentations
  - http://www.thecepblog.com
  - http://www.slideshare.net/TimBassCEP
- Jack van Hoof's blog
  - http://soa-eda.blogspot.com
- Brenda Michelson's EDA Overview
  - http://tinyurl.com/eda-overview

More Information

- Some of the vendor people's blogs
  - Tibco (Paul Vincent) - http://tibcoblogs.com/cep
  - Oracle CEP - http://blogs.oracle.com/CEP
  - Apama (Progress) - http://apama.typepad.com
  - Streambase – http://streambase.typepad.com

- Some representative books on the topic



Questions and Comments?

Eoin Woods
www.eoinwoods.info
contact@eoinwoods.info