

rhysome®

EDA – SOA Interface Patterns

November 9, 2006

Robert Covington, CTO

8425 woodfield crossing boulevard | suite 345 |

indianapolis | in | 46240 | 317.252.2636 |

www.rhysome.com

Types of Events in an Enterprise

➤ Physical Events

- The power has been turned off
- The temperature is below 0 Celsius
- A tornado hits Evansville, IN

➤ Transactional Events

- A new customer has been added
- A claim has been filed
- A check has been received

➤ Aggregate Events

- The NASDAQ average has increased 20% over the last 48 hours

➤ Relational Events

- Auto claims filed in Louisiana in the last 7 days are 25% greater than for the previous week AND life insurance claims in Louisiana are 40% greater than for the previous week

➤ Complex Events

- Combinations of the above in an event hierarchy

Event Relationships

Causality

- An event is caused by another event if the second event could not have occurred without the occurrence of the first event.
- Examples describing causal relationships
 - The presence of heat causes water to boil.
 - My pushing of the accelerator caused the car to go faster.
- Causality relationships are dynamic:
 - May evolve over time as new events are introduced
 - May be difficult to predict utilizing linear methods
 - You can't always determine if someone has a fever by comparing their temperature to 98.6 F.

Event Relationships (continued)

Causality Types

- Observed causality
 - Event A has been observed to cause Event B
 - Examples
 - Changing a database schema affects a group of applications
 - Water at sea level boils at 100C
- Inferred causality
 - Based on statistical analysis, there is a high probability of a causality relationship between 2 events.
 - Example
 - When Bob turns 21, he is less likely to have an automobile accident
- Concurrency
 - Events which occurred at the same time but did not cause each other.
 - Example
 - I landed in San Francisco and the NYSE fell 30 points

Challenges of EDA for SOA

- Event Consumers must be able to accept events as they are presented
 - Events are are not necessarily transactional. They can not be repeated, rolled back, or re-derived.
 - Enormous volumes of fined grained detail may be involved.
 - ie. An intrusion detection system may need to trap every packet on the network and feed it into a system.
 - Are web services/ESBs well designed for handling hundred's of thousands/millions of fine grained events per second?
- EDAs are not request/reply. Interfaces are required to interface EDA to a request/reply system.
- EDAs require complete tracing of events to fully evaluate causality relationships.
- Many events require filtering and enrichments before they are useful.
 - 3 RFID scanners may read a single RFID. To be useful, the duplicates will need to be removed and the RFID tag will need to be enriched with the product information associated with that tag.
- An event driven architecture supports an operational sequence that is dependent on the operation of the business at any point in time, rather than a pre-planned process flow. This means that rules, business processes, etc. may need be reprocessed to take in account new information.

Messaging Patterns or EDA/SOA Interfaces

- Messaging patterns can provide a foundation for the method of Interfacing EDA and SOA systems
- For the purposes of this discussion, I will reference the Messaging Patterns that are described in the book:

Enterprise Integration Patterns

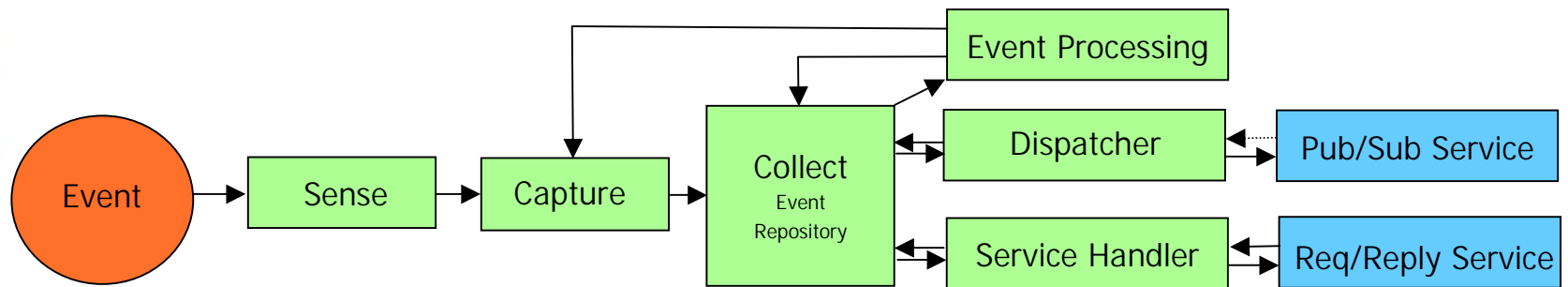
by Gregor Hohpe and Bobby Woolf

Published by Addison-Wesley

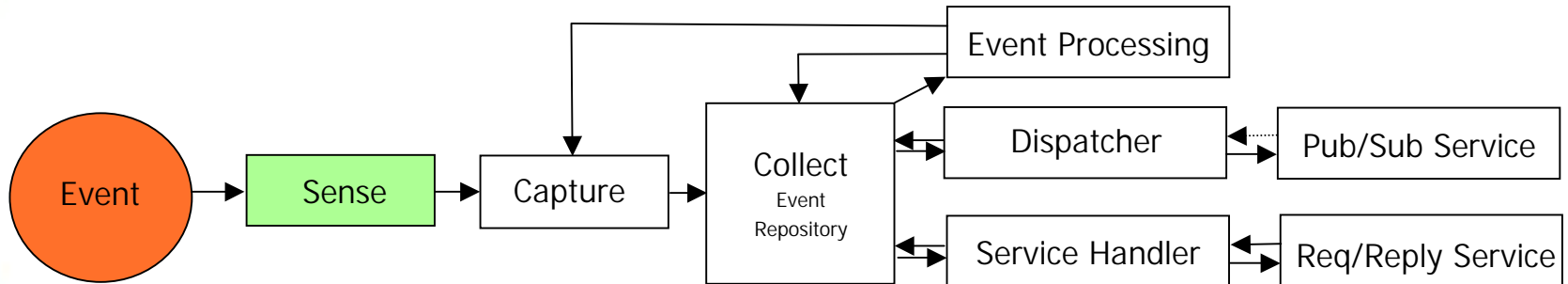
Copyright 2004, Pearson Education, Inc

ISBN 0-321-20068-3

EDA / SOA Interface Patterns

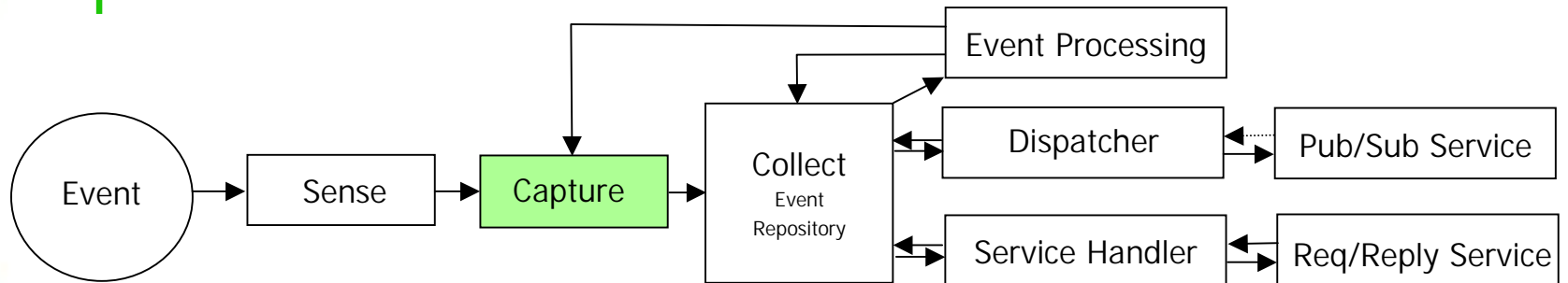


Sense



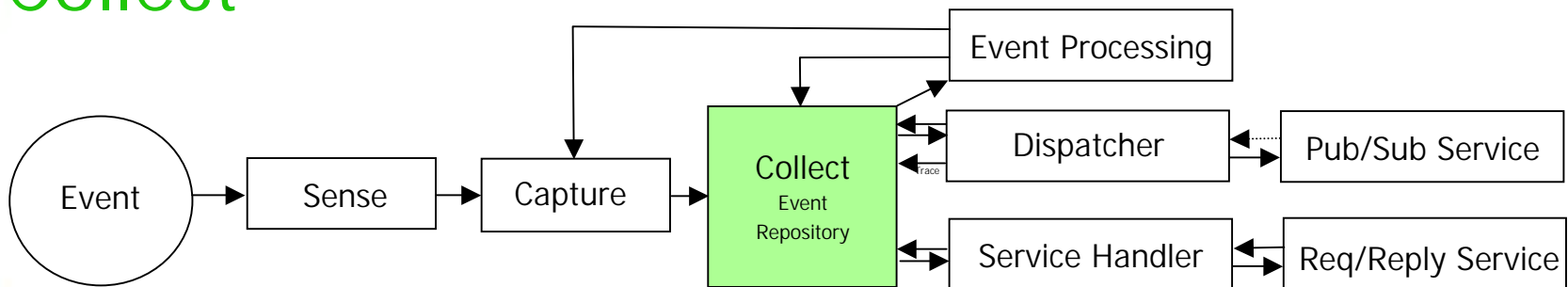
- Detects Events and generate event streams
- Examples of sensors
 - A hardware device invokes a CPU interrupt. A sensor application then responds to the interrupt and reads the value
 - A sensor application polls a sensor. When the value has changed a “significant” amount, an event is generated
 - A sensor application uses a Wire Tap Pattern to observe information flowing between existing communication channels including in a messaging backbone, database transactions, etc.
 - An application sends an event to a sensor application
- Sensors may collect:
 - State
 - Context (ie. Temporal and spatial properties)
 - Data
 - Metadata
 - Cause

Capture



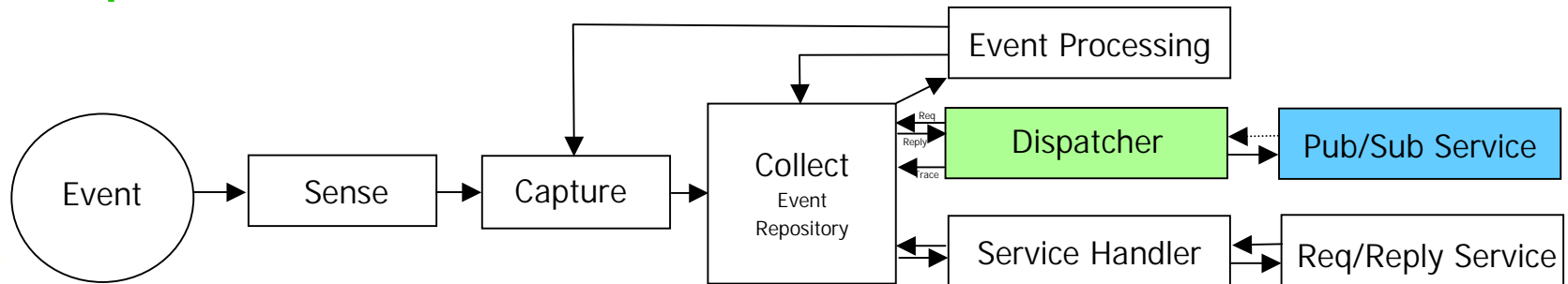
- Receive event streams from multiple sensors, generate standard Event Objects
- Uses the following message transformation patterns at a simple level:
 - Normalizer
 - Content Filter
- Ideally associates a unique key to the Event Message allowing for the core event data to be immutable with additional elements to be linked to the event object (ie. Event traces) using the Claim Check Pattern

Collect



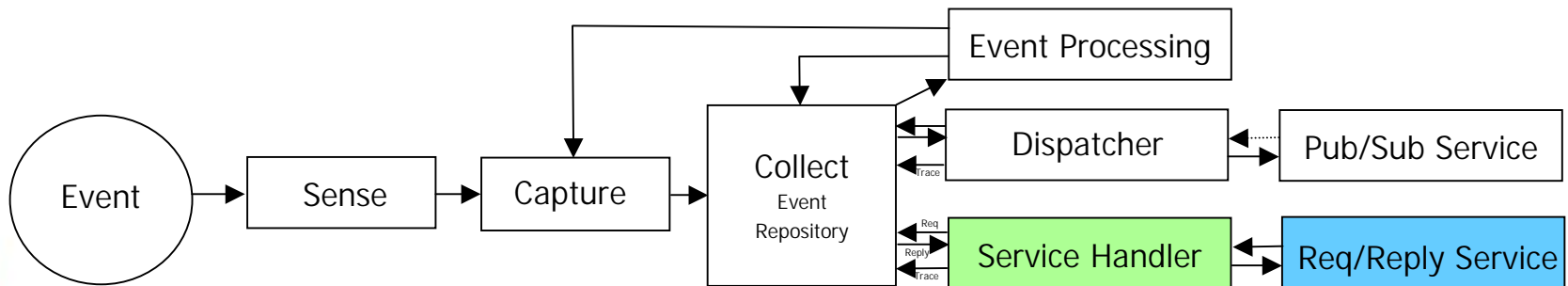
- Queue or Persist Event Objects into a State Repository
- State Repository
 - May be memory based (storing a finite window of events) and/or persisted to disk depending on architectural needs. (performance, required history of events to determine patterns, reliability, etc.)
 - Configurable retention policies
 - Capable of processing simple queries to extract a single or multiple events.
 - Stores events and event traces.
- May utilize the resequencer pattern to temporarily sequence events
- Adds pattern sequences to event trace.
- Optional use of the filter pattern to remove duplicate messages
- Optional use of the aggregator pattern to consolidate multiple event objects into a single event message
- Incoming event objects are received with the Event Consumer pattern
- Outgoing messages utilize request/reply interface pattern.

Dispatch



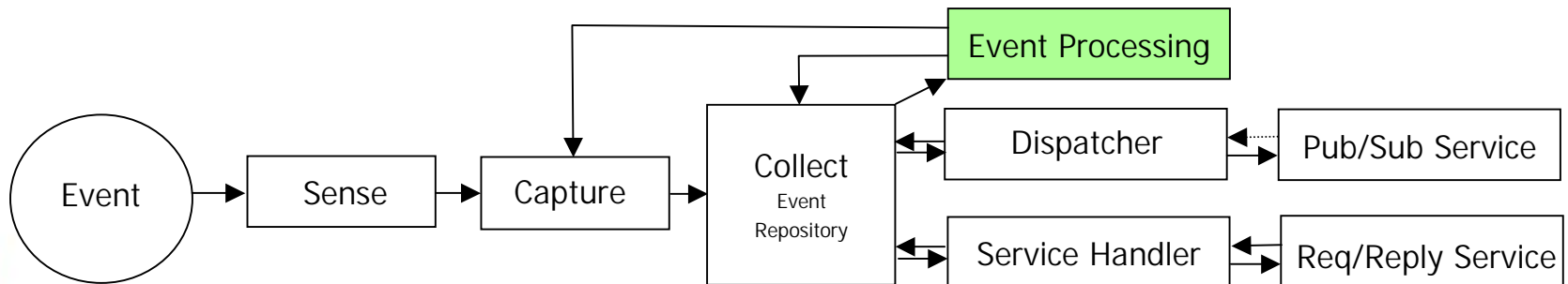
- Send an event to an event consumer using a Publish/Subscribe Pattern
- Interfaces to the State Repository using a request/reply pattern. This prevents the dispatcher from being overloaded from high volume event streams
- Consumers will receive events based on the following patterns:
 - Competing Consumer
 - Message Dispatcher
 - Selective Consumer
- Records dispatches in the Trace Log of the Event using the Claim-Check pattern.

Service Request Handler



- Provides an interface for request/reply systems to the state repository.
- Utilizes the Service Pattern to interface a Service to the State Repository.
- Records replies in the Trace Log of the Event using the Claim-Check pattern

Event Processing Engine



- Provides declarative analysis of event objects.
- Utilizing techniques such as pattern matching, causality analysis, declarative rules engines, agents, Event hierarchies, Event Stream Processing and neural networks to detect situations and in some cases predict potential outcomes.
- Generates virtual events which can feedback into the Event Processing language an event hierarchy.
- Suited for applications where procedural event processing (ie. BPM) is not practical.
 - Applications where permeating through every possible scenario would be impractical to create or manage.
- The Event Processing Engine may tune the Capture filter to change which events are captured.
 - Once a "threat" has been determined, you may increase the monitoring of certain types of events.

Service

- Consider using an adaptation of the Composed Message Processor pattern to handle aggregated messages.
 - Common event objects are aggregated during the collection phase (State Repository)
 - The aggregated event objects are dispatched to a single service.
 - The receiving service utilizes the splitter pattern to divide the payload back into individual events for execution.
- When applicable, this pattern can reduce dispatcher load.