



Stile Architetture Event-Based


Outline

- Approfondimento sullo Stile Event-Based
- La topologia Mediator e quella Broker

- Rifer: **Software Architecture Patterns**
- By Mark Richards
- <http://www.oreilly.com/programming/free/software-architecture-patterns.csp>



Event-Based Style- Generalità

- The event-driven architecture pattern is a popular distributed asynchronous architecture pattern used to produce **highly scalable** applications. 
- It is also **highly adaptable** and can be used for small applications and as well as large, complex ones.
- The event-driven architecture is made up of highly **decoupled, single-purpose** event processing components that **asynchronously** receive and process events.
- Può essere realizzata con due diverse topologie: **Mediator e Broker.**





Topologia Mediator



- The mediator topology is useful for events that have **multiple steps** and require some level of orchestration to process the event.
- For *example*, a single event to place a stock trade might require you to first validate the trade, then check the compliance of that stock trade against various compliance rules, assign the trade to a broker, calculate the commission, and finally place the trade with that broker.
- All of these steps would require some level of orchestration to determine **the order of the steps** and which ones can be done **serially and in parallel**.

Topologia Mediator: componenti

- There are four main types of architecture components within the mediator topology: **event queues**, an **event mediator**, **event channels**, and **event processors**.
- The event flow starts with a client sending an event to an event queue, which is used to transport the event to the event mediator.
- The event mediator receives the initial event and orchestrates that event by sending additional asynchronous events to event channels to execute each step of the process. 
- Event processors, which listen on the event channels, receive the event from the event mediator and execute specific business logic to process the event. 

Stile Event-Based- Topologia Mediator

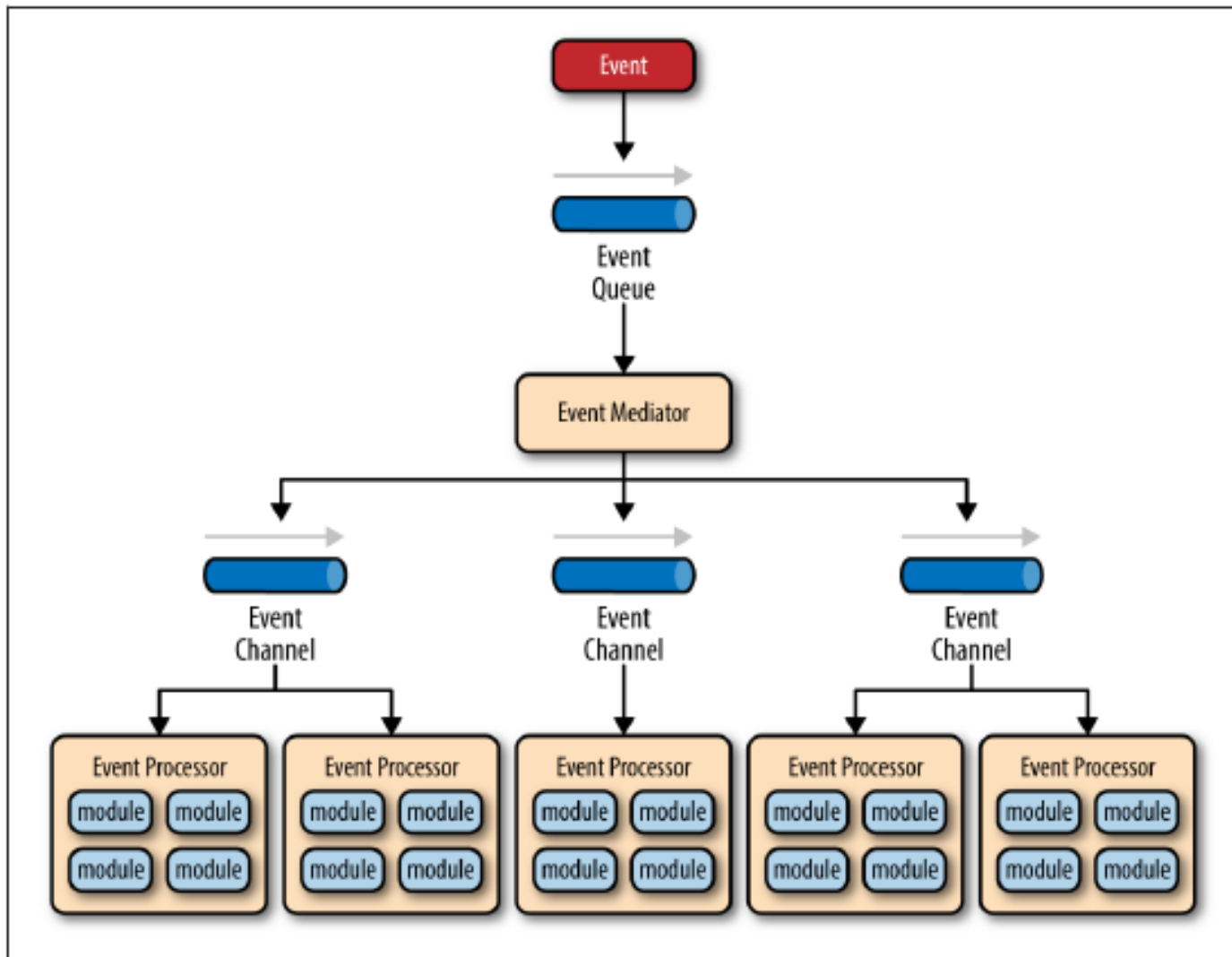


Figure 2-1. Event-driven architecture mediator topology

Dettagli ulteriori



- It is common to have anywhere from a dozen to several hundred event queues in an event-driven architecture.
- The pattern does not specify the implementation of the event queue component; it can be a message queue, a web service endpoint, or any combination thereof.
- There are two types of events : an **initial event** and a **processing event**.
- The initial event is the original event received by the mediator, whereas the processing events are ones that are generated by the mediator and received by the event-processing components.

Event-mediator component



- The event-mediator component is responsible for orchestrating the steps contained within the initial event. For each step in the initial event, the event mediator sends out a specific processing event to an event channel, which is then received and processed by the event processor.
- the event mediator doesn't actually perform the business logic necessary to process the initial event; rather, it knows of the steps required to process the initial event.


Event channels

- Event channels are used by the event mediator to asynchronously pass specific processing events related to each step in the initial event to the event processors.
- The event channels can be either **message queues** or **message topics**, although message topics are most widely used with the mediator topology so that processing events can be processed by multiple event processors (each performing a different task based on the processing event received). 
- Quale differenza fra **message queues** e **message topics**? (Pull vs. Publish-Subscribe)

Message Queue vs. Message Topic in JMS

- JMS (Java Message System) è un insieme di API Java per permettere a programmi su rete di scambiarsi messaggi
- Basata su JMS Consumer, JMS producer, JMS message, JMS queue, JMS topic ..
- JMS queue è usata per raccogliere messaggi inviati. I messaggi sono prelevati nell'ordine di arrivo e una volta letti vengono rimossi dalla coda
- JMS topic è un meccanismo di distribuzione per la pubblicazione di messaggi inviati a più client

Event processor components

- The event processor components contain the application business logic necessary to process the processing event. Event processors are **self-contained, independent, highly decoupled** architecture components that perform a specific task in the application or system.
- While the **granularity** of the event-processor component can vary from **fine-grained** (e.g., calculate sales tax on an order) to **coarse grained** (e.g., process an insurance claim), it is important to keep in mind that in general, each event-processor component should perform a single business task 

Esempio- Mediator

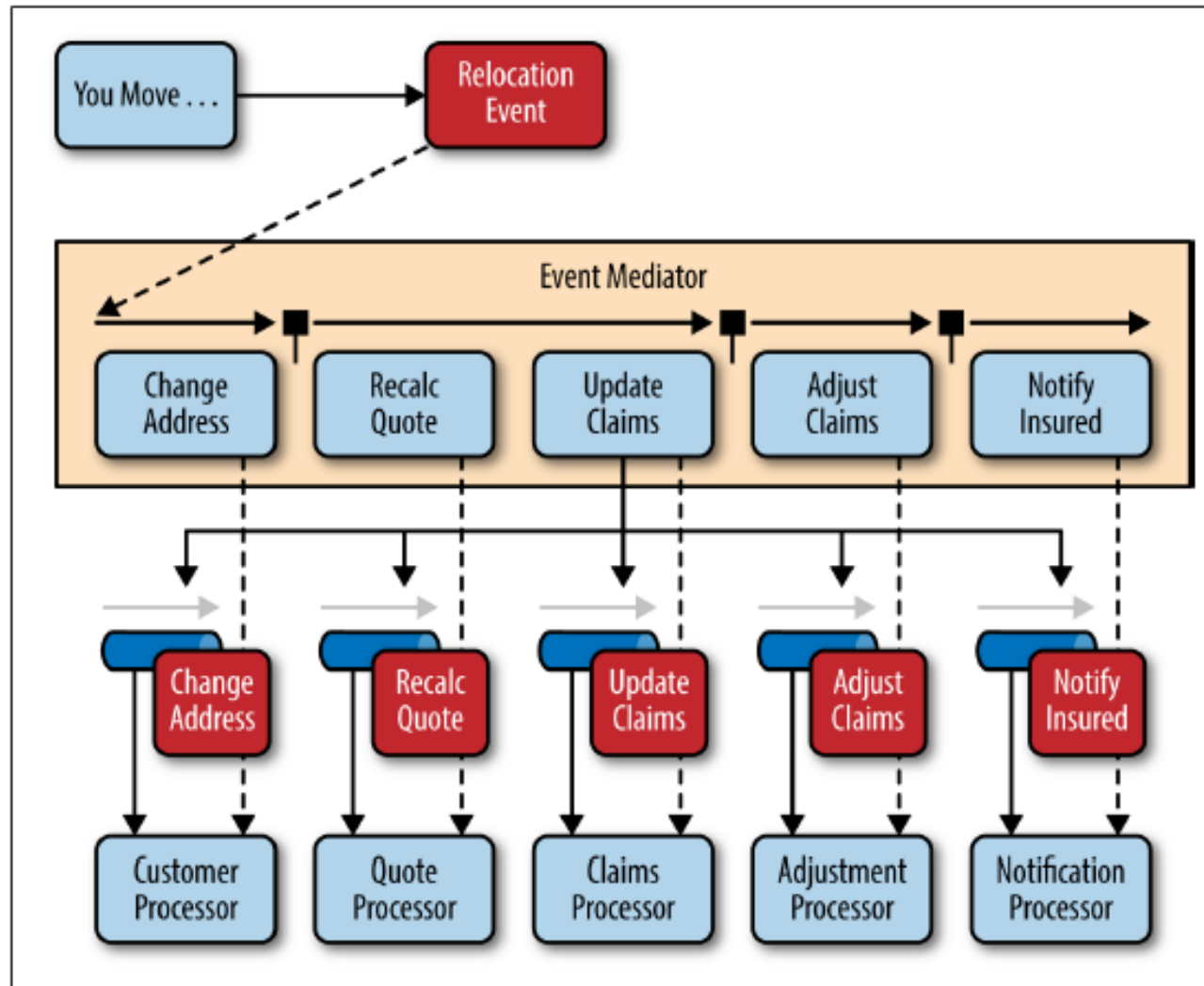


Figure 2-2. Mediator topology example


Implementazioni dell'Event Mediator

- Using open source integration hubs such as **Spring Integration**, **Apache Camel**, or **Mule ESB**. Event flows in these open source integration hubs are typically implemented through **Java code**.
- For more sophisticated mediation and orchestration, you can use **BPEL** (business process execution language) coupled with a BPEL engine such as the open source Apache ODE. BPEL is a standard XML-like language that describes the data and steps required for processing an initial event.
- For very large applications requiring much more sophisticated orchestration (including steps involving human interactions), you can implement the event mediator using a business process manager (BPM) such as jBPM.

- Topologia Broker



Topologia Broker

- Il flusso dei messaggi è distribuito attraverso i componenti di event processing in uno stile a catena, attraverso un lightweight message broker (e.g., ActiveMQ, HornetQ, etc.). 
- Questa topologia è utile quando si ha un flusso di eventi relativamente semplice e non è necessaria una orchestrazione centrale degli eventi .
- Due soli tipi di componenti: Broker e Event Processor Component

Architettura basata su Broker

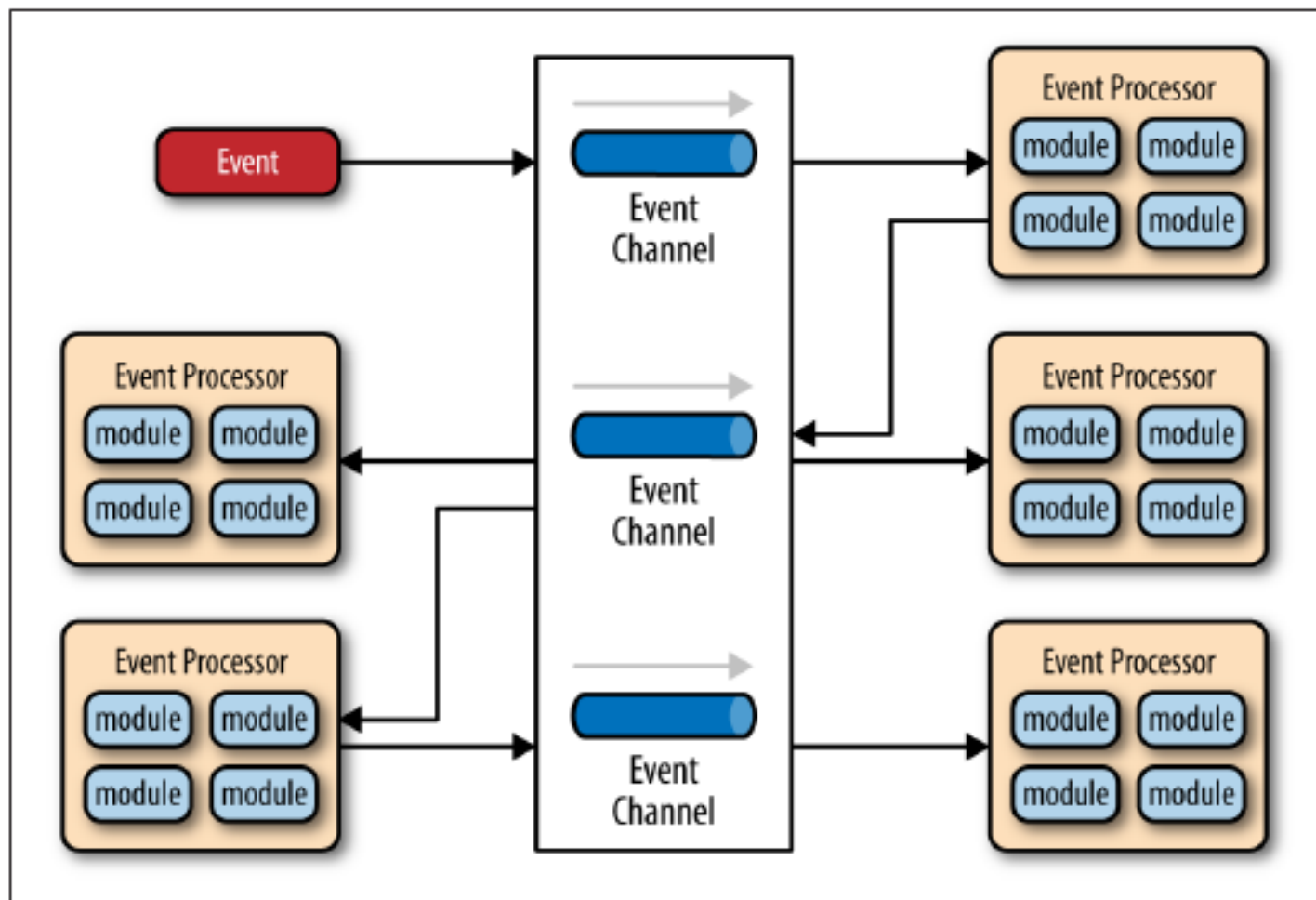


Figure 2-3. Event-driven architecture broker topology

La gestione degli eventi

- Each event-processor component is responsible for processing an event and publishing a new event indicating the action it just performed.
- Siccome non c'è nessun mediatore, I processori di evento sono responsabili di prelevare un evento a cui sono interessati.
- Attraverso un nuovo evento, pubblicato dal processore di eventi quando avrà finito, si innescherà un flusso di ulteriori azioni.

Esempio: Il flusso eventi nella topologia Broker

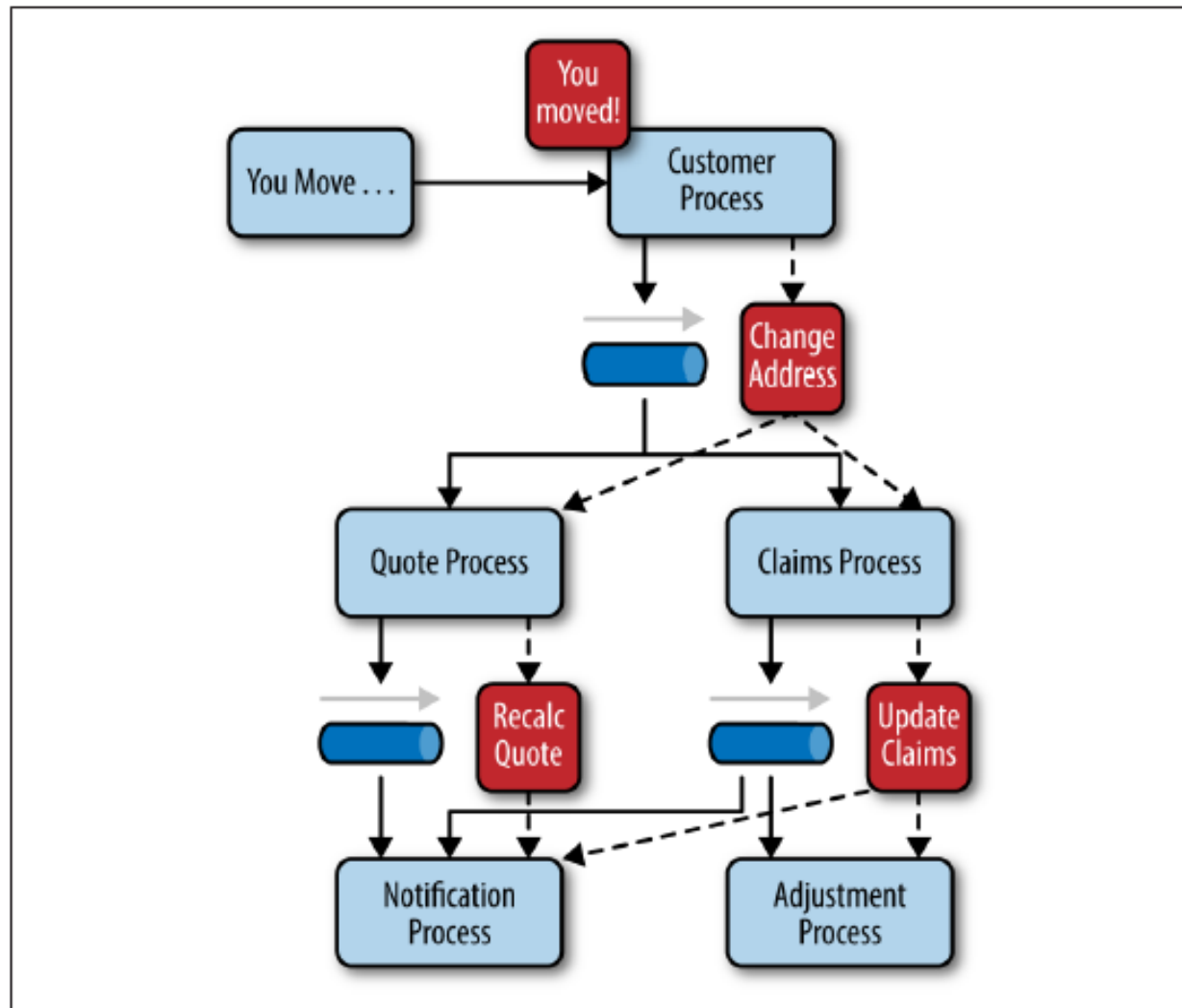






Figure 2-4. Broker topology example

Considerazioni finali

- Lo stile event-driven è complesso da implementare, a causa della sua natura asincrona.
- Devono essere affrontati vari problemi delle architetture distribuite, come:
- *remote process availability*, 
- *lack of responsiveness*,
- *broker reconnection logic* in caso di broker or mediator failure 
- Essendo uno stile fortemente disaccoppiato e distribuito, è difficile implementare lavoro transazionale! 


Necessità di event-processor component contracts

- Essendo I vari componenti disaccoppiati, è importante che essi abbiano e rispettino un contratto, che specifichi:
 - data values
 - data format
- che vengono passati agli altri processori di evento
- It is vitally important when using this pattern to settle on a **standard data format** (e.g., XML, JSON, Java Object, etc.) and establish a contract versioning policy right from the start. 

Analisi di qualità dello stile Event-Based

Evolvibilità

Overall agility

Rating: High 

Analysis: Overall agility is the ability to respond quickly to a constantly changing environment. Since event-processor components are single-purpose and completely decoupled from other event processor components, changes are generally isolated to one or a few event processors and can be made quickly without impacting other components.

Analisi dello Stile

Ease of deployment

Rating: High

Analysis: Overall this pattern is relatively easy to deploy due to the decoupled nature of the event-processor components. The broker topology tends to be easier to deploy than the mediator topology, primarily because the event mediator component is somewhat tightly coupled to the event processors: a change in an event processor component might also require a change in the event mediator, requiring both to be deployed for any given change.


Testability

Rating: Low

Analysis: While individual unit testing is not overly difficult, it does require some sort of specialized testing client or testing tool to generate events. Testing is also complicated by the asynchronous nature of this pattern.

Analisi dello Stile

Performance

Rating: High 

Analysis: While it is certainly possible to implement an event-driven architecture that does not perform well due to all the messaging infrastructure involved, in general, the pattern achieves high performance through its asynchronous capabilities; in other words, the ability to perform decoupled, parallel asynchronous operations outweighs the cost of queuing and dequeuing messages.

Scalability

Rating: High

Analysis: Scalability is naturally achieved in this pattern through highly independent and decoupled event processors. Each event processor can be scaled separately, allowing for fine-grained scalability.

Analisi dello Stile

Ease of development

Rating: Low

Analysis: Development can be somewhat complicated due to the asynchronous nature of the pattern as well as contract creation and the need for more advanced error handling conditions within the code for unresponsive event processors and failed brokers.