Available online www.jsaer.com

Journal of Scientific and Engineering Research, 2020, 7(3):308-312

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

State Machines for RFQ Negotiation in Java: A Comparative Analysis

Ananth Majumdar

thisisananth@gmail.com

Abstract: This paper presents a comparative analysis of three prominent technologies for implementing state machines in Java for Request for Quote (RFQ) negotiation processes: Spring State Machine, AWS Step Functions, and Camunda. RFQ negotiations in financial markets require robust, scalable, and flexible systems to manage complex workflows. State machines have emerged as a powerful tool for handling these intricate processes. The study evaluates each technology based on three critical criteria: scalability, ease of implementation, and persistence capabilities. Through detailed examination and practical examples, we explore how each solution addresses the unique challenges of RFQ negotiation workflows, including the management of buyside and sellside states, and the handling of event-driven transitions. Our analysis reveals that while Spring State Machine offers a lightweight solution for simple, single-instance applications, it faces challenges in distributed scenarios. AWS Step Functions, with its event-driven capabilities excels in cloud-native environments, offering excellent scalability and seamless integration with other AWS services. Camunda emerges as a strong contender for complex RFQ systems, providing a balance of scalability, ease of implementation, and robust persistence, along with the benefits of BPMN support. The findings of this study provide valuable insights for developers and architects tasked with designing and implementing RFQ negotiation systems. We conclude that the choice of technology should be based on specific project requirements, existing infrastructure, and team expertise, with Camunda and AWS Step Functions standing out as strong options for complex, event-driven RFQ negotiation workflows.

Keywords: Request for Quote (RFQ), state machines in Java, Spring State Machine

Introduction

In the fast-paced world of financial markets, efficient and reliable negotiation processes are crucial. Request for Quote (RFQ) negotiations, a common practice in various financial instruments trading, require robust systems to manage their complex workflows. State machines have emerged as a powerful tool for managing these intricate processes, providing a structured approach to handling the various stages of negotiation. This paper aims to explore and compare three prominent technologies for implementing state machines in Java for RFQ negotiation: Spring State Machine, AWS Step Functions, and Camunda. We will evaluate these technologies based on their scalability, ease of implementation, and persistence capabilities, providing insights to help developers and architects make informed decisions when designing RFQ negotiation systems.

Understanding State Machines in RFQ Negotiation

A. Definition and Purpose of State Machines

A state machine is a computational model used to design systems that transition between a finite set of states. In the context of RFQ negotiations, state machines help manage the workflow by clearly defining the possible states of a negotiation and the allowed transitions between these states.

B. RFQ Negotiation Workflow

The RFQ negotiation process typically involves two sides: the buy side (the party requesting the quote) and the sellside (the party providing the quote). Let's examine the states involved in each side:

1. **Buyside States:**

RFQ SENT: The initial state when a new RFQ is generated.

QUOTING: The RFQ has been sent to potential dealers for pricing and is receiving quotes from the dealers.

QUOTE ACCEPTED: The buy side has accepted the quote.

RFQ COMPLETE: The dealer has confirmed the quote and the RFQ is complete.

RFQ CANCELLED: The buyside has cancelled the RFQ.

2. **Sellside States:**

RFQ QUOTING: The dealer is in the process of pricing the RFQ.

QUOTE SENT: The dealer has sent a quote to the buy side.

DEALER LAST LOOK: A dealer has provided a quote, and they have a final opportunity to confirm or withdraw. This state is reached when the buyside hits/lifts the quote provided by a dealer.

CONFIRMED: The deal has been confirmed by both parties.

RFQ CANCELLED: The buyside has cancelled the RFQ.

Additional state transitions:

QUOTE SENT -> RFQ QUOTING: This occurs if the dealer withdraws their quote.

QUOTE SENT -> QUOTE SENT: This happens when the dealer sends a new quote, replacing the previous one.

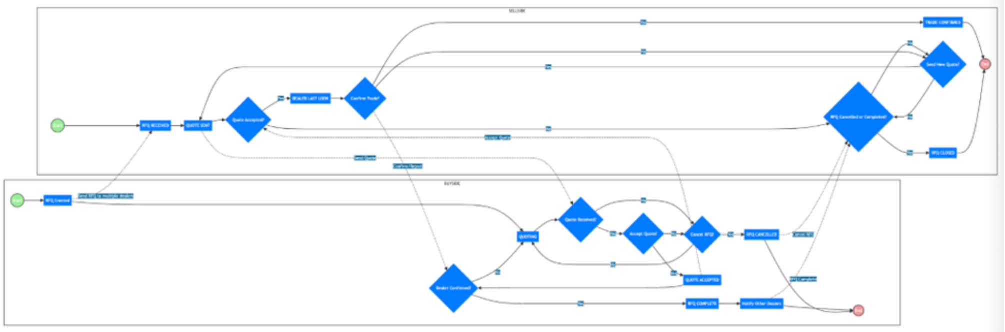


Fig. 1: Buyside and sellside state machine for RFQ negotiation

C. Importance of State Management in Financial Transactions

Proper state management is critical in financial transactions for several reasons:

1. Compliance: Ensuring that all steps in the negotiation process are followed correctly and can be audited.
2. Consistency: Maintaining a consistent view of the negotiation status across all involved parties.
3. Error handling: Providing clear pathways for handling exceptions and edge cases.
4. Performance: Enabling efficient processing of high volumes of concurrent negotiations.

Next we will review three options for creating state machines in Java - Spring State Machine, AWS Step Functions and Camunda.

Spring State Machine

Spring State Machine is a framework for application developers to use state machine concepts in their applications. It's part of the larger Spring ecosystem, which is widely used in Java enterprise applications.

Key features:

- Hierarchical state machine structure
- Event-driven architecture
- Spring Integration support

- Easy integration with other Spring projects

Implementing an RFQ negotiation system using Spring State Machine involves defining the states, transitions, and events that correspond to the negotiation process. The framework allows for creating a hierarchical state machine, which can be beneficial for modeling complex negotiation scenarios.

1. **Single instance performance:**

Spring State Machine performs well for single-instance applications, efficiently handling state transitions and events.

2. **Distributed state:**

For distributed scenarios, Spring State Machine offers a preview feature using ZooKeeper for maintaining distributed state. However, this feature is still in development and may not be suitable for production environments.

Spring State Machine is relatively easy to implement, especially for developers already familiar with the Spring ecosystem. It provides a fluent API for defining states and transitions, and integrates seamlessly with other Spring components.

Persist feature allows users to save a state of a state machine itself into an external repository and later reset a state machine based on serialized state. Spring State Machine offers several options for persisting state: In-memory state storage (default), Redis-based persistence and
custom persistence implementations

However, implementing robust persistence, especially in a distributed environment, can be challenging and may require additional development effort.

AWS Step Functions

AWS Step Functions is a serverless workflow service that allows you to coordinate multiple AWS services into serverless workflows. It provides a visual interface for designing and monitoring workflows.

Key features:

- Serverless architecture
- Visual workflow designer
- Integration with AWS services
- Built-in error handling and retry mechanisms

Implementing an RFQ negotiation system using AWS Step Functions involves defining the workflow as a series of states and transitions in Amazon States Language (ASL), a JSON-based language.

AWS Step Functions is highly scalable, leveraging AWS's serverless infrastructure. It can handle a large number of concurrent executions without requiring manual scaling.

The visual designer and integration with other AWS services make it relatively easy to implement workflows. However, it may require familiarity with AWS services and the Amazon States Language.

State persistence is handled automatically by AWS Step Functions, which maintains the execution state of each workflow instance.

AWS step functions support event based state transition using EventBridge.

EventBridge (Amazon CloudWatch Events) Integration: Step Functions can be configured to listen for specific EventBridge events, which can trigger state transitions.

Camunda

Camunda is an open-source workflow and decision automation platform. It supports both BPMN (Business Process Model and Notation) for process workflows and DMN (Decision Model and Notation) for business decision automation.

Key features:

- BPMN and DMN support
- Embeddable in Java applications
- REST API for remote process management
- Flexible deployment options (on-premises, cloud, hybrid)

Implementing an RFQ negotiation system using Camunda involves defining the process using BPMN, which can be done visually or programmatically.

Camunda offers excellent scalability options:

1. SQL-compatible database storage allows for easy scaling of the persistence layer.
2. The process engine can be deployed across multiple nodes for load balancing.
3. Camunda's architecture supports high throughput and concurrent process executions.

Camunda provides a good balance between ease of use and flexibility:

It supports visual BPMN modelers for process design. It can be embedded in existing Java applications

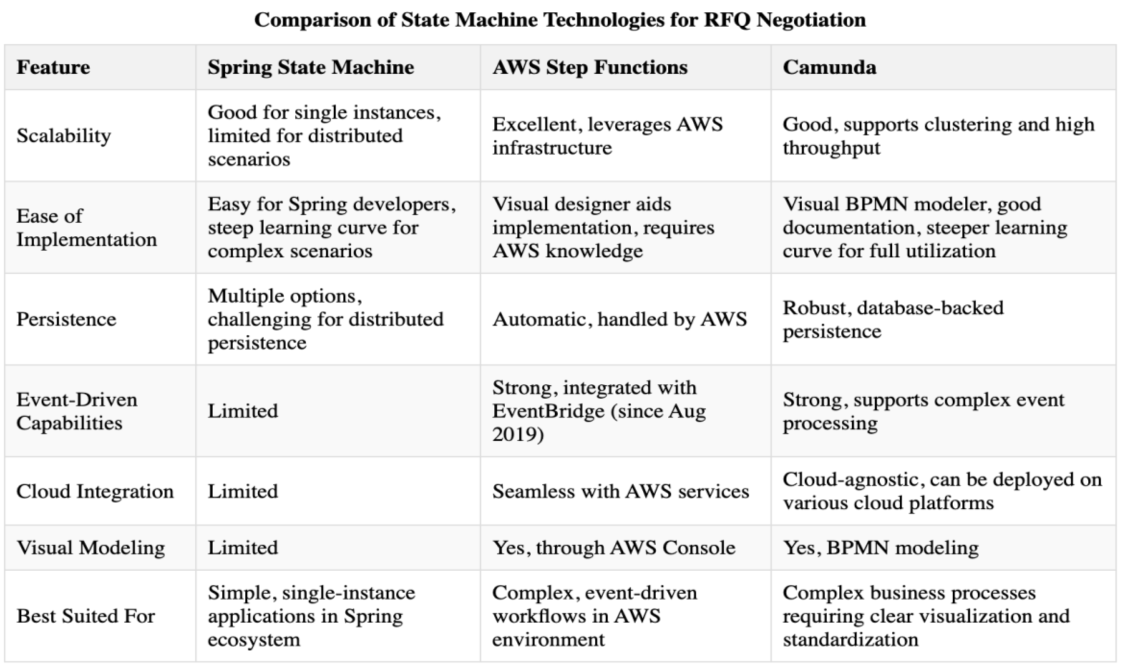
It has comprehensive documentation and community support.

Camunda uses a relational database for persisting process state, supporting various databases including PostgreSQL, MySQL, Oracle, and Microsoft SQL Server.

The support for BPMN is a significant advantage of Camunda:

It uses standardized notation understood by both technical and business stakeholders. It has a rich set of elements for modeling complex business processes. It helps in clear visualization of the entire RFQ negotiation workflow

Comparative Analysis



Comparison of State Machine Technologies for RFQ Negotiation

Feature	Spring State Machine	AWS Step Functions	Camunda
Scalability	Good for single instances, limited for distributed scenarios	Excellent, leverages AWS infrastructure	Good, supports clustering and high throughput
Ease of Implementation	Easy for Spring developers, steep learning curve for complex scenarios	Visual designer aids implementation, requires AWS knowledge	Visual BPMN modeler, good documentation, steeper learning curve for full utilization
Persistence	Multiple options, challenging for distributed persistence	Automatic, handled by AWS	Robust, database-backed persistence
Event-Driven Capabilities	Limited	Strong, integrated with EventBridge (since Aug 2019)	Strong, supports complex event processing
Cloud Integration	Limited	Seamless with AWS services	Cloud-agnostic, can be deployed on various cloud platforms
Visual Modeling	Limited	Yes, through AWS Console	Yes, BPMN modeling
Best Suited For	Simple, single-instance applications in Spring ecosystem	Complex, event-driven workflows in AWS environment	Complex business processes requiring clear visualization and standardization

Fig. 2: Comparative analysis of state machines

Conclusion

After analyzing Spring State Machine, AWS Step Functions, and Camunda for implementing RFQ negotiation state machines in Java, we can draw the following conclusions:

- For simple, single-instance applications with moderate scalability needs, Spring State Machine provides a lightweight and easily integrable solution, especially for teams already using the Spring ecosystem.
- For cloud-native applications leveraging AWS services, AWS Step Functions offers excellent scalability, integration capabilities, and support for event-driven scenarios. Its ability to handle event-based state transitions makes it well-suited for complex RFQ negotiation systems, especially those already integrated with the AWS ecosystem.
- Camunda emerges as a strong contender for complex RFQ negotiation systems, offering a good balance of scalability, ease of implementation, and robust persistence. Its support for BPMN provides clear

visualization and standardization of the negotiation process, which can be beneficial for both technical implementation and business stakeholder communication.

For large-scale, complex RFQ negotiation systems that require high scalability, robust persistence, and clear process visualization, Camunda appears to be the most suitable option among the three. However, the final choice should depend on specific project requirements, existing infrastructure, and team expertise.

Future considerations in this space may include advancements in distributed state management for Spring State Machine, enhanced event-driven capabilities for AWS Step Functions, and continued improvements in Camunda's cloud-native offerings.

As state machine technologies evolve, it will be crucial to stay updated on new features and capabilities that could further enhance RFQ negotiation systems, always keeping in mind the critical requirements of scalability, ease of implementation, and reliable persistence.

References

[1]. Spring State Machine Documentation: https://docs.spring.io/spring-statemachine/docs/current/reference/
[2]. AWS Step Functions Developer Guide: https://docs.aws.amazon.com/step-functions/
[3]. Camunda Documentation: https://docs.camunda.org/
[4]. Business Process Model and Notation (BPMN) Specification: https://www.omg.org/spec/BPMN/2.0/
[5]. "Designing Event-Driven Systems" by Ben Stopford, O'Reilly Media, 2018
[6]. "Enterprise Integration Patterns" by Gregor Hohpe and Bobby Woolf, Addison-Wesley Professional, 2012