Partial Actor Continuations

Efficient and Extensible Request Routing for Event-Driven Architectures

Stefan Plantikow
Zuse Institute Berlin (ZIB)
Takustrasse 7
14195 Berlin, Germany
plantikow@zib.de

ABSTRACT

The request routing logic between the different stages in event-driven architectures is often distributed over different portions of the source code. This can make it hard to change and understand the flow of events in the system.

The article presents an approach that allows writing request routing logic as a set of routing scripts. Requests are executed step-wise according to their script by sending partial continuations that encapsulate their respective request's current execution state to stages for local processing and optional forwarding of follow-up continuations. The implementation of a simple domain specific language for routing scripts for the scala actor library is described and evaluated. The results show that request routing with partial actor continuations performs equally or better to using a separate stage for request routing logic for scripts of at least 3 sequential steps.

Categories and Subject Descriptors

H.2.4 [**Information Systems**]: Systems—*Concurrency*; D.1.3 [**Software**]: Programming Techniques—*Concurrent Programming*; D.3.3 [**Programming Languages**]: Language Constructs and Features—*Concurrency*; D.2.11 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Extensibility*

Keywords

Request Routing, Event-Driven Architecture, Partial Continuation, Actor Model, Scala

1. INTRODUCTION

Using a staged, event-driven architecture is an approach to the design of server software that can provide high degrees of concurrency and throughput. This is achieved by structuring the software as a set of stages that communicate exclusively via event queues and by optionally performing admission control on each queue.

DEBS 2009 Nashville, Tennessee USA

// TODO Examples of staged architectures are ... // TODO XtreemFS example

Depite their benefits, event-driven architectures can lead to a distribution of application logic over different stages. Since the implementation of each stage will usually reside in a different portion of the source code, the dynamic routing of requests through different stages is not described by a single source location.

This reduces the understandability of the system and in turn makes it harder to modify the request routing logic. Additionally, it makes it difficult to add new request types without changing the source code of existing stages and redeploying the system.

// Problem definition
// Article overview

2. PRELIMINARIES

// Actor Model
// Continuations
// Partial Continuations (?)

3. REQUEST ROUTING

// Dissect Problem

Application logic of staged architectures can be split into three categories. We call processing logic, the part of application logic that necessarilyy must be executed at a specific stage in order to access a resource that is only

// Stage Logic vs Routing Logic

3.1 Ping-Pong-Approach

A straightforward way to deal with this problem is to transfer the execution of the request routing logic to a separate stage. Requests enter the system as separate events at such a routing stage. This stage then continously forwards events to some stage, waits for a reply, and upon receipt, decides how to continue based on this and previous reply-events for the request. In the following, we will call this the ping-pong-approach.

While this approach allows to write the event flow portion of the application logic in a single stage, it has two disadvantages

// Statefulness

3.2 Partial Actor Continuations

// PAC - Approach, requires introducing continuations somewhere

4. SOFLEUSE: A DSL FOR REQUEST ROUTING

// Scala and Actors (briefly)
// Plays
// Example
// Describe execution

5. IMPLEMENTATION

// Scala and Actors (as needed for implementation)
// CPS-Transform

6. EVALUATION

// Describe settting
// Present results
// Discuss in detail esp w regard to actor library

7. SUMMARY

8. ACKNOWLEDGMENTS