ATCT: a Java Framework that Offers New Approach to Developing Asynchronous Processes

Serguei Mourachov
Velare Technologies, Inc.
48203 – 595 Burrard Street
Vancouver, BC, V7X 1N8
1.604.837.9786
smourachov@velare.com

ABSTRACT

The development of modern loosely coupled distributed applications requires extensive use of asynchronous processes.

The ability to manipulate execution context could simplify development of such applications, helping to separate business logic from handling asynchrony.

This paper describes a framework that implements Execution Context Reification for Java Virtual Machine (JVM). The framework uses built-in secondary bytecode interpreter that provides access to Execution Context as a first class serializable object. Asynchronous Transfer of Control Threading (ATCT) mechanism is used to manage the execution process using well-known thread semantics. The framework allows the process to be suspended for unlimited amount of time without locking system threads. Next, the process can be instructed to resume execution from the point where it was stopped. Described approach will allow to simplify development of asynchronous processes by enabling use of sequential programming style.

Categories and Subject Descriptors

D.1.3 [Concurrent Programming]; D.1.4 [Sequential Programming]; D.1.5 [Object-oriented Programming];

General Terms

Algorithms, Design

Keywords

Asynchronous processes, execution context reification, Java framework.

1. INTRODUCTION

Most languages lack a representation of the execution context as an accessible object. However, this is much desired functionality that would allow one to implement innovative approaches to application development simplifying many programming tasks. One of the more important problems is programming asynchronous processes. Asynchronous processes are typically implemented by programming a state machine that manages the process flow, driven by asynchronous messages or events. Access to the execution context as a first class object, could simplify the development of such processes transforming relatively complex event-driven programming style to well-known sequential approach. Some programming languages offer an abstraction to the execution context, using semantics of Continuations. However, this semantics is not well understood by the masses of developers preventing wide use of continuation based approach.

OOPSLA'03, October 26–30, 2003, Anaheim, California, USA.
ACM 1-58113-751-6/03/0010.

2. THREADS

Thread abstraction is offered in many modern programming languages. Java and C# for example, offer an abstraction of Thread as a first class object, encapsulating functionalities such as suspend and resume. However, Thread's capacity to remain suspended is limited by the inability of capturing its execution stack and recreating it in another thread to continue execution, starting at the point of capture. As developers are very acquainted with the semantics of threads, its extension to provide Execution Context Reification functionality, could receive a better acceptance by the developer community than using continuations.

3. ATC THREAD

ATCT framework provides ATCThread class that in general follows threads semantics and adds the capability to access the execution context as a first class serializable object. Additionally, ATCThread provides a mechanism of Asynchronous Transfer of Control (ATC) to manage process execution. ATCThread is a serializable Java object and as such can be easily persisted or moved to another location. This makes it particularly useful for implementation of long running transactions, load balancing, and improving application scalability and robustness. ATCThread is a 'cornerstone' class of the framework that manages the execution context and is used to access most of the functionality of the framework. It encapsulates the byte code execution engine that maintains the execution stack and makes it available for manipulation.

4. BYTECODE EXECUTION ENGINE

The framework's built-in bytecode execution engine uses a mixed execution model, interpreting only parts of the byte code and delegating the rest of the execution to JVM. The byte code execution engine makes a distinction between three types of methods: methods that it interprets and offers the Execution Context Reification functionality for, methods which execution is delegated to the JVM and methods that command the execution engine to interrupt interpretation returning the control to the code that started ATCThread. ATCT manages the execution stack only for the methods it interprets and offers a flexible mechanism for defining the criteria according to which the method discrimination

is performed. The execution engine takes an efficient approach to executing byte code by delegating execution to the JVM whenever possible. In addition, such operations as creation of objects, garbage collection and memory management are delegated to the JVM as well.

5. METHOD DISCRIMINATION MECHANISM

Method discrimination mechanism allows the execution engine to distinguish between the three types of methods. Each ATCThread can have an associated instance of MethodDiscriminator. A custom implementation of MethodDiscriminator can be based on a variety of approaches, for example on a result of applying a regular expression to the names of the methods being called. It can also be based on the package name, class name, or any other reflective information that is available from the context. In ATCT, default MethodDiscriminator uses subclasses of *Throwable* in method signatures to discriminate between methods.

6. ASYNCHRONOUSLY INTERRUPTIBLE METHODS

Dedicated methods, executed by the ATCT execution engine, are called Asynchronously Interruptible (AI) methods. Execution context reification functionality is only available for the methods interpreted by the execution engine. Because the ATCT Framework interprets AI methods using its built-in execution engine, an AI method will be executed slower in comparison to the same method being executed by the JVM. As discussed earlier, it is important to maintain a good ratio between AI methods and the methods delegated to the JVM.

7. ATC METHODS

ATC methods are used to initiate Asynchronous Transfer of Control. When the execution engine, during execution of AI method, encounters call to such method it returns control to the code that started an instance of ATCThread. At this moment ATCThread will be suspended and will contain execution context, relevant to the stack of AI methods being executed. In the suspended state, the ATCThread can be preserved or sent to another location and resume interpretation when desired. The ATCThread can be resumed using its resume method to continue execution of the AI method. ATC methods can return data that can be used inside the AI method after an ATC thread is resumed.

8. TECHNOLOGY USES

8.1 Asynchronous process programming

Business processes logic often can be easily represented in pseudo code using a sequential (linear) style. However, this is different from how they are programmed, due to the asynchronous nature of communication between participating parties and the limitations of mainstream programming languages.

Most often, implementing such a process involves creating a state machine that helps to manage flow processing and state transitions, triggered by asynchronous events. The business logic of such implementations is exposed to a high risk of being obfuscated by the code that performs state machine management and event handling routines.

One of the more strong advantages of ATCT technology is seen in enabling programming of asynchronous processes in a sequential manner. In this case, the code that implements business process may be as simple to understand as a pseudo-code used to define the business flow.

8.2 Using common programming languages for implementing workflow and process composition

Normally traditional programming languages cannot be used for workflow development. That's why commonly used workflow engines usually provide special flow definition languages that don't have power and flexibility available in general-purpose programming languages. ATCT ability to suspend and resume the execution at any time enables the implementation of a process flow composition using well-known OOP techniques. Such useful workflow patterns as parallel split, simple merge, multi merge and synchronization can be implemented gracefully in Java using the ATCT framework.

8.3 Mobile Agents

Mobile agents may become more "mobile" if they have the ability to preserve its state at any time and continue execution on a different machine. Due to the assistance provided by the framework in handling the complexity of preserving and restoring an agent's state, moving an agent from one machine to another can also be used to create highly sophisticated distributed applications. For example in some situations it would be more appropriate to transfer code (mobile agent) to the secure data source than move sensitive data to the unsecured server.

8.4 Web Application Flow

Web applications often implement multi step processes such as performing banking transaction, filling application forms etc. Implementations of such processes using ATCT allows creating more manageable code explicitly reflecting business process flow.

8.5 Gaming

In game programming, there is often a requirement to manage thousands of objects, where each object exhibits a particular behavior. Normally, proprietary scripting languages are developed to describe the object's process flows, with a preemptive approach to context switching. ATCT can be easily used in such applications, using all the power of standard Java programming including OOP, modularity and exception mechanisms.

9. LEVERAGING EXISTING TOOLS AND SKILLS

ATCT offers a powerful extension to Java allowing use of existing language features and tools, thus leveraging the investments companies made in IDEs, UML tools and development processes. Such important features of IDEs as code complete and contextual help for APIs remain available to the developers without need to change tools. In addition, ATCT allows developers to use well-known OOP approaches such as inheritance, encapsulation, abstract behavior, as well as design patterns for programming business processes.

10. MANAGEABILITY

Code that takes advantage of the ATCT framework carries clarity of pseudo code into the real implementation, allowing greater level of comprehension and reducing the probability of introducing subtle bugs during the implementation and maintenance phases.