Adding High-Level Concurrency to EScala

Jurgen M. Van Ham

ASCOLA, École des Mines de Nantes and Software Technology Group, Technische Universität Darmstadt
jurgen.van-ham@mines-nantes.fr

Abstract

On the one hand, languages like EventJava combine Event-Based Programming with concurrency. On the other hand, extending Aspect-Oriented Programming with concurrency has been studied as well. Seamlessly combining both styles with concurrency in a single language is possible with the right building blocks. We claim that the join is such a building block.

Categories and Subject Descriptors D.3.3 [*Programming Languages*]: Language constructs and Features—Concurrent programming structures

General Terms Languages, Experimentation

Keywords concurrency, declarative events, join, Aspect-Oriented Programming

1. The research problem and motivation

The observer pattern [5] is widely used to provide some form of Event-Based Programming (EBP) in an Object-Oriented Programming (OOP) style. However, its use results in boilerplate code in the sending *subjects*, which are coupled to the *observers* because of the observers interface, especially when an observer observes different kinds of updates. These problems are addressed by languages that support EBP: *events* are sent to their (receiving) *handlers*, which act as being part of an observer.

Whereas EBP events are explicitly triggered, the join points of *Aspect-Oriented Programming* (AOP) can be seen as implicitly-triggered events.

In Complex Event Processing [10] (CEP) events can be combined. Event expression is a *declarative* way to specify such a combination of events, which also supports transforming and filtering of events.

The language ESCALA [6] integrates events that can be used for EBP and AOP into Scala [11]. Its declarative events are not yet powerful enough to implement CEP, as they do not support streams and can only combine events originating from a single primitive event. This can be improved. Still, what currently exists already allows refining events, which can be used as an AOP pointcut language.

For the time being, concurrency has not been taken into account when designing EScala events. However, many programs do have a concurrent nature, for instance a program does not block all its other tasks during interaction with a user or another external party. Concurrency and EBP both can benefit from each other. Events help to structure (concurrent) programs. In the other way around, concurrency can enable asynchronous events whereby the sender of the event and its handler can proceed in parallel. Since AOP can be considered as EBP with implicit events the use of concurrency could benefit to communication and synchronization between the aspects and the base program. The use of concurrency could benefit to communication and synchronization between the aspects and the base program in AOP.

The standard approach to synchronization is low-level, based on a shared-memory model distinguishing communication and synchronization using locks and monitors. An alternative approach consists of relying on a shared-nothing model and joins [4] to handle both communication and synchronization. This approach has been followed, for instance, by Polyphonic C# [1] and `scalajoins` [7].

However, joins do not apply to events in Polyphonic C# but to function calls; events are not well integrated with object-oriented programming in EventJava, and neither of the languages offer an AOP-like mechanism with dynamic registration of handlers.

Our objective is to pursue the integration started by ESCALA by considering at the same time OOP, EBP, AOP and concurrency.

2. Background and related work

In terms of events, a (binary) join [4] can be seen as synchronizing two events (the event occurring first is blocked until the second occurs) and results in the triggering of a third event associated to the synchronization. When events have arguments the third event receives a combination of the arguments from the first two.

AOSD'12, March 25-30, 2012, Potsdam, Germany.
ACM 978-1-4503-1222-6/12/03.

EventJava [3] is a concurrent EBP which can correlate events in streams. It was used to study Scalable Efficient Correlation Detection [9], which compares different implementations and shows that joins can be efficient. The compared implementations include, on the one hand, languages like Polyphonic C#, on the other hand, libraries like the Joins Concurrency Library [12] for Visual Basic, the first implementation of joins as a library and `scalajoins` [7] a library for Scala.

Concurrency in AOP, which allows the base program and its aspects to be executed concurrently, was studied in CEAOP [2]. CEAOP is based on complex low-level synchronization.

While related work covers the parts which we try to combine, none of it combines all of them.

3. Approach and uniqueness

It is of course possible to write concurrent programs in ESCALA by using the Scala libraries for concurrency but combining declarative events and low-level concurrency is not trivial. We have started by looking at how to add, at the user level, joins to define declarative events relying on explicit events and `scalajoins`. We found out that this is indeed feasible but requires quite a lot of code and incurs some overhead. This overhead comes from the fact that `scalajoins` events are of a different nature than the ESCALA events. An adapter [5] bridges this difference. Moreover, this new operator can be defined as a standard ESCALA operator, providing the user with a simple conceptual model. We plan to apply the same approach to other features such as asynchronous events.

The join for events was already implemented in `scalajoins`, however, integrating this operation into a language which supports declarative events and supports AOP as well is unique. It makes it possible to localize synchronization concerns of both objects and aspect instances.

4. Results and Contributions

We enriched expressions for declarative events in ESCALA with a join. To this end, we modified `EventsLib`, the library that implements events in ESCALA. In particular, we added event queueing so that events can wait for their counterpart.

With joins, provided asynchronous events are available, we can then implement the classical bounded buffer used as an example by Polyphonic C# using declarative events and event handlers. This results in a different style of programming as joins and their resulting actions are separated in different constructs that can be independently specialized and dynamically bound. Von Itzstein [8] shows various applications of the joins with method calls which can be implemented with events and handlers.

Using implicit events also makes it easy to program concurrent aspects. Synchronization becomes an identifiable part of the program instead of being scattered as calls to a complex `Monitor` singleton.

Finally, the integration of these concurrent events can continue with a study where this implemented model will be integrated into the tasks of Scala by combining it with threads and actors.

References

[1] Nick Benton, Luca Cardelli, and Cédric Fournet. Modern concurrency abstractions for C#. *ACM Transactions on Programming Languages and Systems*, 26(5):769–804, 2004.

[2] Rémi Douence, Didier Le Botlan, Jacques Noyé, and Mario Südholt. Concurrent aspects. In *Proceedings of the 5th international conference*, GPCE '06, pages 79–88. ACM, 2006.

[3] Patrick Eugster and K. Jayaram. EventJava: An extension of Java for event correlation. In Sophia Drossopoulou, editor, *ECOOP 2009 Object-Oriented Programming*, volume 5653 of *Lecture Notes in Computer Science*, pages 570–594. Springer Berlin / Heidelberg, 2009.

[4] Cédric Fournet and Georges Gonthier. The join calculus: A language for distributed mobile programming. In *Applied Semantics, International Summer School, APPSEM 2000*, pages 268–332. Springer-Verlag, 2002. Advanced Lectures.

[5] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[6] Vaidas Gasiūnas, Lucas Satabin, Mira Mezini, Angel Núñez, and Jacques Noyé. EScala: Modular event-driven object interactions in Scala. In *Proceedings of the 10th International Conference on AOSD 2011*. ACM Press, March 2011.

[7] Philipp Haller and Tom Van Cutsem. Implementing joins using extensible pattern matching. In Doug Lea and Gianluigi Zavattaro, editors, *10th International Conference on Coordination Models and Languages (COORDINATION 2008)*, volume 5052 of *LNCS*, pages 135–152. Springer-Verlag, June 2008.

[8] G.S Von Itzstein. *Introduction of High Level Concurrency Semantics in Object Oriented Languages*. PhD thesis, January 2005.

[9] K. Jayaram and Patrick Eugster. Scalable efficient composite event detection. In Dave Clarke and Gul Agha, editors, *Coordination Models and Languages*, volume 6116 of *Lecture Notes in Computer Science*, pages 168–182. Springer Berlin / Heidelberg, 2010.

[10] David C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[11] Martin Odersky, Lex Spoon, and Bill Venners. *Programming in Scala*. Artima, 2008.

[12] Claudio V. Russo. The joins concurrency library. In Michael Hanus, editor, *PADL*, volume 4354 of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2007.