—Demonstration—

Κόμπος: A Platform for Debugging Complex Concurrent Applications

Stefan Marr
Johannes Kepler University Linz
Austria

Carmen Torres Lopez
Vrije Universiteit Brussel
Belgium

Dominik Aumayr
Johannes Kepler University Linz
Austria

Elisa Gonzalez Boix
Vrije Universiteit Brussel
Belgium

Hanspeter Mössenböck
Johannes Kepler University Linz
Austria

Keywords Concurrency Models, Debugging, Concept Reification, Stepping, Replay

1. Introduction

With the omnipresence of multicore processors, developers are building more concurrent software. Much research has focused on providing programming models and abstractions to ease the contruction of concurrent systems such as actors, communicating sequential processes (CSP), or software transactional memory (STM). To address specific application requirements, developers start to cherry-pick the abstractions that fit the problems at hand [4]. This means, applications are built from a combination of concurrency abstractions such as threads, actors, CSP, or even STM.

However, when it comes to debugging and tooling support, the same revolution has not happened. To date, there is no software platform or ecosystem that provides tooling that addresses the challenges of combining concurrency models. Typically, applications are debugged on the level of the basic abstractions of a language. For Java or C#, this means complex applications are debugged by inspecting threads, reasoning about low-level memory accesses, or basic atomic operations, even if an application uses modern concurrency frameworks such as Akka for Scala, or the Task Parallel Library for .NET. This makes really difficult to get a good understanding of the different concurrent entities and to recreate the conditions that lead a system to exhibit a bug.

In this demonstration, we present Kómpos, a debugging platform for complex concurrent applications. Kómpos allows developers to focus on the high-level interactions between different kinds of concurrent entities, e.g. actors, STM and threads. It has been built into the SOMNS language implementation [2] featuring actors, CSP, fork/join, and a basic STM.

2. A Concurrent Debugger

The goal of concurrent debugger is to provide a dedicated support for managing concurrent entities and their execution. Typically a concurrent debugger is crafted for the concurrency model on which applications are built. Kómpos is a concurrent debugger for complex concurrent systems enabling developers to identify specific subsystems or their interactions to understand problematic program behavior better. Thus, it is crucial to identify the concurrent entities that communicate with each other. For example, when using actors, we want to see which actors exchange messages, and for a program using CSP, we want to know which processes are connected via channels, and which messages are exchanged.

This information should be easily accessible and allow developers to selectively explore the dynamic behavior of the system. This is in contrast to classic debuggers, which put the program code and current state first. While this information is of high importance for us as well, we argue for a *system-centric* view as a cornerstone of a concurrent debugger to enable interaction with the program on a higher level than on a per-entity basis, i.e., on the level of groups of concurrent entities constituting subsystems.

Another relevant aspect of a system-centric approach is that the options for interacting with a running system should go beyond simple stepping through instruction streams or line-based breakpoints as in classic online debuggers. In addition to classic stepping, the debugger should also enable developers to follow the high-level interactions. For example, a stepping debugger for actor and CSP programs should enable stepping and breakpoints at message-level granularity [1, 5].

<Programming> '17 April 3-6, 2017, Brussels, Belgium

3. Advanced Debugging Techniques

In addition to exploring a system-centric approach to debugging concurrent systems with Kómpos, we also explore advanced debugging techniques to help identify root cause of concurrency bugs. More concretely, the demo will show: (1) deterministic replay and (2) an assertion system for actor message protocols in SOMNS.

One key challenge of concurrency is the non-determinism introduced by scheduling of concurrent execution. To help developers reason about the execution of a concurrent program, SOMNS comes with support for recording execution traces which enables deterministic replay and debugging of recorded executions.

SOMNS also features support for defining protocols for actor interactions. It allows developers to specify constraints over the communication between actors to describe the intended order message, and e.g. delivery expectations about senders. Message protocol violations and incorrect assumptions about the systems' behavior can be then explored deterministically in the debugger.

4. Open Research Question

To realize and complete the vision behind Kómpos, much work remains to make the approach practical.

A Metalevel Interface for Concurrency Models. A debugger cannot support all possible concurrency models or libraries directly, because there are to many variations of them. Instead, we work on devising a metalevel interface that is generic enough to support various concurrency models, and provides mechanisms so that specific libraries or models can explicitly communicate to the debugger, how information or interactions should be made available. However, there are many open questions, including what the common elements are, and which elements are useful to identify concurrency issues. The goal is to avoid an additive or enumerative approach in supporting concurrency models, instead we require a generalized but customizable approach.

A Scalable Approach to a System-Centric View. A live graphical representation of the system implies many challenges for scalability to larger programs. For example, how can we group similar concurrent entities so that a developer can selectively explore different subsystems without being overwhelmed by the large number of entities belonging to unrelated subsystems. With techniques like low-overhead tracing, we believe that the debugging platform can then dynamically identify entities that behave homogeneously. A simple but likely useful approximation would be to take the lexical program location into account where a concurrent entity was created. Depending on the concurrency model, other high-level aspects could be taken into account. This could include for instance information about exchanged messages, communication partners, or common call stacks.

Another question is how to visualize relevant information. Especially in large systems, we need ways to *zoom* into specific subsystems, or even change abstraction levels. When tracking problematic interactions between concurrency abstractions, we might actually need to look at the underlying Java-level operations instead of focusing on the high-level exchange of actor messages or the interactions of transactions.

5. Related Work

Debuggers for high-level concurrency models have been investigated before. This includes for example debuggers for actor systems [1, 3] including the ScalaIDE.[1] There has also been work on debuggers for transaction systems [6]. However, to the best of our knowledge, there is currently no system that provides support for debugging of multiple high-level concurrency models.

6. Conclusion

This demonstration introduces Kómpos, a system-centric debugger for complex concurrent applications. Kómpos support focuses on the interactions between high-level concurrent entities. As a result, it supports developers not only by visualizing these interactions but also by enabling them to interact with the running program on the level of the concurrency abstractions, i.e., message sends, fork/join operations, or transactions.

Acknowledgments

This research is funded by a collaboration grant of the Austrian Science Fund (FWF) with the project I2491-N31 and the Research Foundation Flanders (FWO Belgium).

References

[1] E. Gonzalez Boix, C. Noguera, and W. De Meuter. Distributed debugging for mobile networks. *Journal of Systems and Software*, 90:76–90, April 2014.

[2] S. Marr and H. Mössenböck. Optimizing Communicating Event-Loop Languages with Truffle, 2015. Presented at AGERE!'15.

[3] T. Stanley, T. Close, and M. Miller. Causeway: A message-oriented distributed debugger. Technical report, HP Labs, 2009.

[4] S. Tasharofi, P. Dinges, and R. E. Johnson. Why Do Scala Developers Mix the Actor Model with other Concurrency Models? In *Proc. of ECOOP*, volume 7920 of *LNCS*, pages 302–326. Springer, 2013.

[5] C. Torres Lopez, S. Marr, H. Mössenböck, and E. Gonzalez Boix. Towards Advanced Debugging Support for Actor Languages: Studying Concurrency Bugs in Actor-based Programs, October 2016. Presented at AGERE!'16.

[6] F. Zyulkyarov, T. Harris, O. S. Unsal, A. Cristal, and M. Valero. Debugging Programs That Use Atomic Blocks and Transactional Memory. In *Proc of PPoPP*, pages 57–66. ACM, 2010.

[1] http://scala-ide.org/docs/current-user-doc/features.html