Abstract

Object orientation provides principles for abstraction and encapsulation. Abstraction is accomplished by high-level concepts of interfaces, classes, and objects. Many object-oriented languages have followed the blocking and synchronous model of messaging. *Asynchronous* message passing is one of the fundamental elements of the actor model. Unlike in object orientation, a message is not bound to any interface in the actor model.

Ongoing research focuses on combining object-oriented programming with the actor model and concurrency. In a concurrent setting with objects, multi-threading is a common approach to provide an object-oriented concurrency model. Integration of actor model and object orientation leads to the use of asynchronous method calls. However, neither of object-orientation and actor model support explicit control of pre-emption for messages or asynchronous method calls. In comparison to this rigid "run to completion"-execution style, coroutines are more flexible: they allow multiple entry points and can be used for collaborative pre-emption between tasks by giving up control with explicit yield or suspend instructions. Coroutines are not originally established in the object-oriented paradigm. Furthermore, interactions in both coroutines and multi-threading are blocking and synchronous. The main challenge is to generate production code from an actor-based language which supports asynchronous method calls and coroutine-style execution.

This thesis contributes to the intersection of object orientation, the actor model, and concurrency. We choose Java as the main target programming language and as one of the mainstream object-oriented languages. We formalize a subset of Java and its concurrency API. We create an abstract mapping from an actor-based modeling language, ABS, to the programming semantics of concurrent Java. The mapping ensures a correct translation from a model to its equivalent production-ready code. We provide the formal semantics of the mapping and runtime properties of the concurrency layer including deadlines and scheduling policies. We provide an implementation of the ABS concurrency layer as a Java API library and framework using Java 8 features. The implementation is generic and extensible as it can be used either as a standalone library or as part of a pluggable code generator. We design and implement a runtime monitoring framework, JMSeq, to verify the correct ordering of execution of methods in possibly interleaved threads through code

annotations in JVM. In addition, we design a large-scale monitoring system as a real-world application; the monitoring system is built with ABS concurrent objects and formal semantics that leverages schedulability analysis to verify correctness of the monitors.

Samenvatting

Objectgeoriënteerd programmeren biedt ondersteuning voor abstractie en encapsulatie. Abstractie wordt bereikt door de introductie van high-level concepten als interfaces, klassen en objecten. Veel objectgeoriënteerde talen gebruiken blokkerende, synchrone communicatie voor het zenden van berichten (messages) tussen objecten. *Asynchrone* communicatie (message passing) vormt een fundamenteel element van het zogenaamde actor model. Anders dan in objectoriëntatie is een message niet gebonden aan of beperkt tot een specifieke interface.

Huidig onderzoek focust op het combineren van objectoriëntatie met het actor model en concurrency (meerdere taken kunnen tegelijk actief zijn). Voor het uitbreiden van objectoriëntatie naar een concurrent omgeving is multi-threading een veelgebruikte aanpak. Integratie van objectoriëntatie met het actor model leidt tot het gebruik van asynchrone methode aanroepen. Echter, nog objectoriëntatie, nog het actor model biedt ondersteuning voor expliciete controle voor het tijdelijk onderbreken (pre-emption) van de verwerking van berichten of asynchrone methode aanroepen. In vergelijking met deze starre “run to completion”-executie stijl van programma’s zijn coroutines flexibeler: er zijn meerdere “entry points” voor de executie, en taken coëpereren voor controle over de processor, door tijdelijk controle op te geven met expliciete yield of suspend instructies. Coroutines zijn echter niet standaard in objectoriëntatie, en interacties met zowel coroutines als multi-threading zijn traditioneel synchroon en blokkerend. De belangrijkste uitdaging is de generatie van productie code uit een actor gebaseerde taal die ondersteuning biedt voor asynchrone methode aanroepen en een flexibele coroutine stijl van executie.

Dit proefschrift draagt bij aan de kennis over de combinatie van objectoriëntatie, het actor model en concurrency. We kiezen Java, als een volwassen en een van de meest gebruikte programmeertalen in productie omgevingen, als de target language voor code generatie vanuit een actor gebaseerde taal. We formaliseren een deel van Java en zijn concurrency library (API). Vervolgens creëren we een abstracte relatie (een functie) van de actor gebaseerde taal ABS naar concurrent Java. Deze abstracte functie garandeert een correcte vertaling van een abstract ABS model naar executeerbare

productie code. We geven de formele semantiek van de functie en de eigenschappen van de concurrency laag tijdens executie, inclusief deadlines en scheduling policies. Daarna implementeren we de concurrency laag van ABS als een Java API library, met behulp van Java 8. Deze implementatie is generiek en uitbreidbaar, en kan zowel gebruikt worden als standalone library, of als een inplugbare code generator. We ontwerpen en implementeren een monitoring framework genaamd JMSeq waarmee de correcte ordening van methode aanroepen, gespecificeerd met code annotaties, geverifieerd kan worden in een multi-threaded omgeving. Tevens ontwerpen we een grootschalig real-world monitoring systeem met ABS concurrent objecten en een formele semantiek. We gebruiken state-of-the-art technologie voor schedulability analyse om de correctheid van de monitors te verifiëren.