Abstract:

Traditionally, concurrency models fall into two broad categories: flexible and performant versus safe and manageable concurrency control. On one end of the spectrum there are shared-memory models such as threads and locks which are very flexible and performant but offer almost no safety guarantees. On the other end of the spectrum are isolated message-passing models which are often more stringent, favoring safety and liveness guarantees.

The actor model is such a message-passing concurrency model and because of the asynchrony of its communication mechanism and isolation of its different processes the actor model avoids issues such as deadlocks and low-level data races by construction. This facilitates concurrent programming, especially in the context of complex interactive applications where modularity, security and fault-tolerance are required. The trade-off is that the actor model sacrifices expressiveness and efficiency with respect to parallel access to shared state.

This dissertation aims to show that strict isolation is not a prerequisite to guarantee deadlock and race condition freedom, and that breaking with no-shared-state concurrency can improve the flexibility and performance of message-passing concurrency models
while still maintaining their safety guarantees.

In this dissertation, we give an overview of the issues that come with representing shared resources in modern actor systems and then formulate an extension to a the actor model that allows safe, expressive and efficient sharing of mutable state among otherwise isolated concurrent components. We propose domains as a set of novel language abstractions for encapsulating and sharing state. With the domain model, we introduce four types of domains, namely immutable, isolated, observable and shared domains. The design and implementation of the domain model is realized in the context of SHACL, a novel communicating event-loop actor language. We validate the usefulness of our model by applying it in practice through a case study in Scala.

Samenvatting:

Traditioneel kunnen concurrency modellen opgedeeld worden in twee categorieën: flexibele en performante versus veilig en beheersbare concurrencycontrol. Aan het ene uiteinde van het spectrum er zijn gedeeld-geheugen modellen zoals threads en locks die zeer flexibel en performant zijn maar bijna geen veiligheidsgaranties bieden. Aan het andere uiteinde van het spectrum zijn geïsoleerde message-passing modellen die vaak strikter zijn, ten voordele van de veiligheidsgaranties van het resulterende model.

Het actormodel is zo een message-passing model en omdat het actormodel asynchrone communicatie hanteert en de verschillende processen strikt geïsoleerd zijn vermijdt dit model gekende concurrencyproblemen zoals deadlocks en elementaire data races. Dit vergemakkelijkt concurrent programmeren, voornamelijk in de context van complexe interactieve applicaties waarbij eigenschappen zoals modulariteit, veiligheid en fouttolerantie belangrijk zijn. De keerzijde van de

medaille is dat dit model expressiviteit en efficiëntie opoffert betreffende parallelle toegang tot gedeelde informatie.

Deze verhandeling stelt zich tot doel aan te tonen dat strikte isolatie van de verschillende actoren geen vereiste is om deadlocks en data races te vermijden. De breuk maken met no-shared-state parallelisme kan de flexibiliteit en de performantie van het message-passing concurrency modellen verbeteren en tegelijkertijd hun veiligheidsgaranties bewaren.

In deze verhandeling starten we met een overzicht van de verschillende problemen die zich voordoen bij het voorstellen van gedeelde informatie in moderne actorsystemen. Nadien formuleren we een uitbreiding van het actormodel dat de programmeur in staat moet stellen om op een veilige, expressieve en efficiënte manier gedeelde informatie voor te stellen. We stellen domeinen voor als een set van verschillende taalabstracties voor het kapselen en delen van informatie. Met dit model stellen we vier verschillende types van domeinen voor, met name niet-aanpasbare, geïsoleerde, observeerbare en gedeelde domeinen. Het ontwerp en de implementatie van domeinen zijn gerealiseerd in de context van SHACL, een nieuwe communicating event-loop actor programmeertaal. We valideren ook de bruikbaarheid van ons model door het toe te passen in de praktijk door middel van een case study in Scala.