

Cover Page



Universiteit Leiden



The handle <http://hdl.handle.net/1887/97598> holds various files of this Leiden University dissertation.

Author: Serbânescu, V.

Title: Software development by abstract behavioural specification

Issue Date: 2020-06-10

Leiden Institute of Advanced Computer Science

Software Development by Abstract Behavioural Specification

Vlad-Nicolae Şerbănescu

2020

Software Development by Abstract Behavioural Specification

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Leiden
op gezag van Rector Magnificus Prof. mr. C.J.J.M. Stolker
volgens besluit van het College voor Promoties
te verdedigen op woensdag 10 juni 2020
klokke 10:00 uur

door

Vlad-Nicolae Şerbănescu

geboren te Boekarest, Roemenië
in 1989

PhD Committee

Promotor:

Prof. dr. F.S. de Boer

Co-promotor:

Dr. M.M. Jaghoori

other members:

Prof. dr. F. Arbab

Prof. dr. M.M. Bonsangue

Prof. dr. A. Plaat

Prof. dr. F. Pop

Dr. R. Schlatte

Politehnica University of Bucharest, Romania

University of Oslo, Norway



The work in this thesis has been carried out at the Center for Mathematics and Computer Science (CWI) in Amsterdam and Leiden Institute of Advanced Computer Science (LIACS) at Leiden University. This research was supported by the European projects FP7-610582 ENVISAGE (Engineering Virtualized Services) and FP7-612985 UPSCALE (From Inherent Concurrency to Massive Parallelism through Type-based Optimizations).

Cover Design: Cristian-Ștefan Neacșu

Contents

1	Introduction	1
1.1	Research Context	1
1.2	Research Objectives	3
1.3	Research Challenges	4
1.4	Main Contributions and Thesis Outline	5
2	Asynchronous Programming in the Java Virtual Machine (JVM)	8
2.1	Java Concurrency Utilities	10
2.2	Scala Abstractions for Concurrent Programming	14
2.3	Coroutine Support	17
2.4	Summary	18
3	Abstract Behavioural Specification Language	19
3.1	Programming to Interfaces in ABS	20
3.2	Algebraic Data Types (ADT)	20
3.3	Asynchronous Model	23
3.3.1	Concurrent Object Groups (COGs)	23
3.3.2	Await and Get Construct	23
3.4	Synchronous Calls	25
3.5	Timed ABS	25
3.6	Deployment Components	26
3.6.1	Resource Types	26
3.6.2	Resource Consumption	27
4	ABS Runtime in Java	28
4.1	Evolution of the Runtime Schemes	28
4.1.1	Every asynchronous call is a thread and each actor has a lock	29
4.1.2	Every actor is a thread pool.	30
4.1.3	Every system has a thread pool.	30
4.1.4	Fully asynchronous environment.	31
4.2	Implementation Details	33
4.2.1	Task Generation and Completion	33
4.2.2	Task Scheduling and Execution	34
4.2.3	Symbolic Time	39
4.2.4	Resource Modeling	39

5	From ABS to Java	42
5.1	Compiler vs Library approach	42
5.2	The "Bloody Battle of Backends"	45
5.2.1	Original Java Backend	45
5.2.2	ABS-API using Java 8	47
5.2.3	ProActive Backend for ABS	55
6	ABS Runtime as a Library API	57
6.1	JAAC API through an Example Program	58
6.1.1	Library Methods.	58
6.1.2	Call Stack and Priorities.	60
6.2	ASCOOP Scala API	61
6.2.1	Actors Typed with Interfaces	61
6.2.2	Actors with Integrated Futures	64
7	ABS to Scala Compiler	66
7.1	Translating Core ABS and its Extensions to the Proposed Runtime	66
7.1.1	Objects as Actors in COGs	67
7.1.2	Asynchronous Communication	68
7.2	Translation of ABS to Scala	69
7.2.1	Sweeping Parameters	69
7.2.2	Compiling Algebraic Data Types and Pattern Matching	70
7.2.3	Translating the Built-in Types of ABS	70
7.3	Compiler Correctness	74
7.3.1	ABS Operational Semantics	75
7.3.2	ABS-SPAWN	78
7.4	The GAC Language	83
8	Case Studies and Benchmarks	86
8.1	Cooperative Scheduling Benchmarks	86
8.1.1	Coroutine "Heavy" Benchmark	86
8.1.2	NQueens Benchmark	88
8.2	Benchmarking the ASCOOP Library	90
8.2.1	Message Passing Overhead	91
8.2.2	Actors Overhead	93
8.2.3	CPU and Memory benchmarks	93
8.3	Map Reduce	96
8.4	Cache coherency	98
8.5	German Railway Example	99
9	Conclusions and Future Work	102
	Summary	104
	Samenvatting	105
	Acknowledgements	107
	Bibliography	109

List of Figures

4.1	Basic process-oriented approach	29
4.2	Actor-as-executor approach	30
4.3	System as a thread pool	31
4.4	Full data-oriented approach	32
5.1	ABS-API Class Diagram	49
5.2	Actor Model	50
5.3	Message encapsulation	50
5.4	Actor State Diagram	51
5.5	Thread Creation and Scheduling	52
5.6	Future Flow	54
5.7	ABS new in ProActive	56
5.8	ABS asynchronous method call in ProActive	56
7.1	Parameter Sweeping for a continuation triggered when encountering a given await	69
7.2	Syntax for ABS statements.	75
7.3	Translation of ABS into ABS-SPAWN	79
7.4	Execution of an Await Statement	82
7.5	Scheduling a Suspended Statement	82
7.6	ABS guarded command statements.	83
8.1	Performance figures for Coroutine Overhead	88
8.2	Results for the N-Queens problem using two different coroutine approaches	89
8.3	Performance figures Ping Pong Benchmark	91
8.4	Performance figures Fibonacci Benchmark	94
8.5	Performance figures Eratosthenes Sieve Benchmark	95
8.6	Performance figures Nqueens Benchmark	96
8.7	Execution time of DNA-matching ABS application	97
8.8	Results for the simulation of the memory system	100
8.9	The railway model through the visualization tool	101