The ABS Simulator Toolchain

Rudolf Schlatte*, Einar Broch Johnsen, Eduard Kamburjan,
S. Lizeth Tapia Tarifa

University of Oslo, Oslo, Norway

Abstract

ABS is a language for behavioral modeling of distributed, time- and resource-sensitive communicating systems. ABS is based on an executable actor-based semantics with asynchronous method calls, with method call results being delivered via future variables. Data is modeled via a functional, side-effect-free layer of algebraic data types and parametric functions. Actor behavior is expressed in a sequential, imperative way, with explicit suspension points for in-actor cooperative scheduling. A declarative time and resource model allows modeling of time-sensitive actor behavior in a compositional way. A software product line language layer implements model variability via code deltas and feature models. This paper describes the toolchain that makes it possible to simulate ABS models, and lists the most important case studies done with ABS.

Keywords:
Distributed actor systems, executable models, time and resource behavior

1. Introduction

The ABS [1] modeling language targets the design, verification, and executable modelling of concurrent and distributed systems. It is an actor-based, object-oriented language with a Java-like syntax. The concurrency model of ABS is based on active objects [2]. ABS is supported by a range of analysis tools (see, e.g., [3]); this tool paper focuses on the simulation toolchain [4].

Section 2 introduces ABS, Section 3 describes the software architecture of the simulator and Section 4 discusses some implementation aspects. Section 6 lists the bigger case studies that used ABS; Section 7 concludes the paper.

*Corresponding author
Email addresses: rudi@ifi.uio.no (Rudolf Schlatte), einarj@ifi.uio.no (Einar Broch Johnsen), eduard@ifi.uio.no (Eduard Kamburjan), sltarifa@ifi.uio.no (S. Lizeth Tapia Tarifa)

2. Motivation and Background

Modeling is the process of capturing essential characteristics of a complex system, while leaving out irrelevant details. Each model implicitly has a purpose, reflected by the aspects of the real world that it regards as important. ABS is a modeling language that captures the essential characteristics of distributed, communicating systems of independent actors. These systems can be pure software systems, cyber-physical systems, or software-free systems.

ABS models capture the essential communication patterns, time behavior and resource consumption aspects of the modeled systems. The internal states of actors can be modeled at very different levels of detail, depending on the needs of the model. The following features of ABS are used to create such models:

Asynchronous method calls and first-class futures. The essential feature of a distributed system is that communication (sending a method call) and execution (scheduling an incoming call) are decoupled. The caller can continue execution until the result of a call is needed, and the callee can schedule calls from multiple callers as needed.

Process suspension and guarded commands. Inside an ABS actor, multiple processes execute in a cooperative manner, with only one process running at any given time. Processes suspend themselves when waiting for a method call result or waiting for a boolean condition over the actor state (e.g., waiting for a work item list to be non-empty).

Data structures and functions. Algebraic datatypes are used to model actor state and data that is passed between actors via method calls. Functions that calculate over such datatypes are side effect-free, which makes static analysis of ABS models easier.

Time and resource models. A process can suspend itself for a given amount of time, or require some amount of resources [5], which will also cause time to advance. In this way, any ABS model can be transformed into a *timed model* [6]. A short tutorial for ABS, including time and resource modeling, can be found in [7].

3. Software Framework

The toolchain described in this paper has the task of parsing and analyzing an ABS model, and (usually) translating it into an executable form. The compiler is used from the command line, but can be integrated into editors and IDEs [8]. A compiled model can be executed from the command line; which requires Erlang to be installed. Both the compiler and the required runtime components also come packaged in a docker image, so

no software installation is needed on the modeler's machine. Installation instructions are available at `https://abs-models.org/getting_started/local-install/`.

Basic editor support is available for a number of editors; the list of editors and installation instructions are available at `https://abs-models.org/getting_started/editor-support/`.

4. Implementation

The toolchain is implemented in Java using the antlr4 parser [9] and the JastAdd AST (Abstract Syntax Tree) framework [10], and compiled via the gradle build system. The binary artifact is a single self-contained `.jar`-file that can run on any system with a Java runtime environment. Models are transpiled into Erlang [11] and executed on the Erlang BEAM VM, which is well-suited for simulations of highly-concurrent systems such as ABS models.

5. Usage Example

A minimal program in ABS that just prints `Hello world!` looks as follows:

```
module Hello;
{
  println("Hello world!");
}
```

When saved in a file `hello.abs`, this code can be compiled with the command `absc -e hello.abs`, which generates a `gen/` subdirectory. When running the compiler with the `-v` ("verbose") argument, some additional information about the compilation process is printed to the terminal. After compilation, the model is run with `gen/erl/run`, which prints the expected greeting.

A slightly larger example of an ABS model can be seen at `https://abs-models.org/documentation/examples/monty-hall/`. The model on that page shows simple communication patterns between actors resulting in a non-trivial result, and comes with extensive commentary.

6. Case Studies

Case studies that have been carried out using ABS include:

Fredhopper Cloud Services provide search and targeting facilities for retailers as a service (SaaS) [12]. Their software is under constant development and relies on automated configuration, continuous monitoring and testing. ABS models allow changes to be analyzed before deployment, including low-level effects such as resource consumption on virtual machines. For this case study, a general replay tool for logs from the production system was built [13]. The replay tool interacts with the simulator via the Model API and enables simulating real-world scenarios from Fredhopper's system logs. This was used to validate the correlation between the model and the actual system, and to predict the effects of changes in the system.

ABS-YARN [14] is a configurable ABS model for applications running on Apache's Hadoop YARN framework. Hadoop is a popular cloud infrastructure system and YARN (Yet Another Resource Negotiator) provides job scheduling and cluster resource management for Hadoop configurations. Simulation shows that the ABS framework can accurately reflect the behavior of YARN to efficiently compare different deployment decisions. This work was extended to Hadoop Spark Streaming [15] and to compare instance purchasing options provided by Amazon Web Services [16].

The HyVar toolchain is a framework that targets context-dependent software upgrades for car ECUs (Electronic Control Units). The framework collects information from a fleet of cars, analyzes this information to decide how to update the software running on the different cars, and sends software updates back to the cars when needed. The objective of the ABS simulations was to efficiently analyze the *scalability* of the toolchain [17].

Multicore Memory Systems. This case study focused on how a model of multicore memory systems [18], formalized in structural operational semantics, could be implemented in ABS as a simulator of memory operations on multicore architectures. The focus of this work was on correctness preserving transformations of operational rules with pattern matching synchronization conditions at the SOS level to a decentralized and asynchronous actor model [19]. The cooperative scheduling of ABS was crucial to ensure a granularity of process interleaving which corresponded to the SOS model.

Kubernetes. This case study develops an ABS model of a containerized orchestration system for cloud-native microservices, which are loosely-coupled collections of services designed to adapt to traffic in very fine-grained and flexible ways. The model focused on resource consumption and scaling of microservices deployed and managed by Kubernetes [20]. This work performed experiments on HPC4AI, a cluster for deploying high-performance applications, in order to compare the observed behavior of real systems to corresponding observations of instances of the model [21].

Compugene is a case study that models transcription of mRNA in com-

putational biology[1]. The time model of ABS was crucial to model biological processes, in particular to capture the effect of different rates of degradation in neighboring cells. The simulation is used to compare the mathematical model (expressed in ABS) with experimental results, with the aim to reduce the need for the more time-consuming experiments.

FormbaR is a case study that models the railway operations rulebooks of the main German railway infrastructure company [22]. Using an executable model, changes in the rulebooks can be prototyped with quick feedback cycles for their maintenance. The time model is needed to faithfully mirror train driving. The simulator was leveraged for analyzing the effects of rule changes. A special visualization tool is used to interact with the simulation without modifying the ABS code and to summarize the results of the execution [23].

7. Conclusions

ABS has been used for nearly 10 years by academic and industrial researchers to model and analyze real-world systems. In this paper, we discussed the ABS language and the simulator toolchain that enables the simulation of ABS models. ABS continues to be used and developed for research into language features and semantics, as a modeling language for real-world distributed systems, and as a basis for research into distributed systems.

Acknowledgments

The development of the ABS language and toolchain was supported by the EU projects HATS (FP7 ICT 231620), Envisage (FP7 610582), and HyVar (H2020 644298), and by the Research Council of Norway via the Sirius Centre for Research-based Innovation. We also wish to thank all individual developers for their contributions, large and small.

[1] E. B. Johnsen, R. Hähnle, J. Schäfer, R. Schlatte, M. Steffen, ABS: A core language for abstract behavioral specification, in: B. Aichernig, F. S. de Boer, M. M. Bonsangue (Eds.), Proc. FMCO 2010, Vol. 6957 of LNCS, Springer, 2011, pp. 142–164. doi:10.1007/978-3-642-25271-6_8.

[2] F. S. de Boer, V. Serbanescu, R. Hähnle, L. Henrio, J. Rochas, C. C. Din, E. B. Johnsen, M. Sirjani, E. Khamespanah, K. Fernandez-Reyes, A. M. Yang, A survey of active object languages, ACM Comput. Surv. 50 (5) (2017) 76:1–76:39. doi:10.1145/3122848.

[1] https://www.compugene.tu-darmstadt.de

[3] R. Hähnle, The abstract behavioral specification language: A tutorial introduction, in: Proc. FMCO 2012, Vol. 7866 of LNCS, Springer, 2012, pp. 1–37. doi:10.1007/978-3-642-40615-7_1.

[4] R. Schlatte, V. Stolz, L. Tveito, ABS Tools, version 1.9.3 (May 2021). doi:10.5281/zenodo.6867529.

[5] E. B. Johnsen, R. Schlatte, S. L. Tapia Tarifa, Integrating deployment architectures and resource consumption in timed object-oriented models, Journal of Logical and Algebraic Methods in Programming 84 (1) (2015) 67–91. doi:10.1016/j.jlamp.2014.07.001.

[6] J. Bjørk, F. S. de Boer, E. B. Johnsen, R. Schlatte, S. L. Tapia Tarifa, User-defined schedulers for real-time concurrent objects, Innovations in Systems and Software Engineering 9 (1) (2013) 29–43. doi:10.1007/s11334-012-0184-5.

[7] R. Schlatte, E. B. Johnsen, E. Kamburjan, S. L. T. Tarifa, Modeling and analyzing resource-sensitive actors: A tutorial introduction, in: COORDINATION 2021, Vol. 12717 of LNCS, Springer, 2021, pp. 3–19. doi:10.1007/978-3-030-78142-2_1.

[8] J. Doménech, S. Genaim, E. B. Johnsen, R. Schlatte, EasyInterface: A toolkit for rapid development of guis for research prototype tools, in: FASE, Vol. 10202 of LNCS, Springer, 2017, pp. 379–383. doi:10.1007/978-3-662-54494-5_22.

[9] T. J. Parr, R. W. Quong, ANTLR: A predicated-*LL(k)* parser generator, Softw. Pract. Exp. 25 (7) (1995) 789–810. doi:10.1002/spe.4380250705.

[10] T. Ekman, G. Hedin, The JastAdd system - modular extensible compiler construction, Sci. Comput. Program. 69 (1-3) (2007) 14–26. doi:10.1016/j.scico.2007.02.003.

[11] J. Armstrong, Erlang, Commun. ACM 53 (9) (2010) 68–75. doi:10.1145/1810891.1810910.

[12] E. Albert, F. S. de Boer, R. Hähnle, E. B. Johnsen, R. Schlatte, S. L. Tapia Tarifa, P. Y. H. Wong, Formal modeling and analysis of resource management for cloud architectures: an industrial case study using Real-Time ABS, Service Oriented Computing and Applications 8 (4) (2014) 323–339. doi:10.1007/s11761-013-0148-0.

[13] N. Bezirgiannis, F. S. de Boer, S. de Gouw, Human-in-the-loop simulation of cloud services, in: ESOCC, Vol. 10465 of LNCS, Springer, 2017, pp. 143–158. doi:10.1007/978-3-319-67262-5_11.

[14] J.-C. Lin, I. C. Yu, E. B. Johnsen, M.-C. Lee, ABS-YARN: A formal framework for modeling Hadoop YARN clusters, in: FASE, Vol. 9633 of LNCS, Springer, 2016, pp. 49–65. doi:10.1007/978-3-662-49665-7_4.

[15] J.-C. Lin, M. Lee, I. C. Yu, E. B. Johnsen, A configurable and executable model of Spark Streaming on Apache YARN, Int. J. Grid Util. Comput. 11 (2) (2020) 185–195. doi:10.1504/IJGUC.2020.105531.

[16] E. B. Johnsen, J.-C. Lin, I. C. Yu, Comparing AWS deployments using model-based predictions, in: ISoLA (2), Vol. 9953 of LNCS, 2016, pp. 482–496. doi:10.1007/978-3-319-47169-3_39.

[17] J.-C. Lin, J. Mauro, T. B. Røst, I. C. Yu, A Model-Based Scalability Optimization Methodology for Cloud Applications, in: IEEE SC2, IEEE Computer Society, 2017, pp. 163–170. doi:10.1109/SC2.2017.32.

[18] S. Bijo, E. B. Johnsen, K. I. Pun, S. L. Tapia Tarifa, A formal model of data access for multicore architectures with multilevel caches, Sci. Comput. Program. 179 (2019) 24–53. doi:10.1016/j.scico.2019.04.003.

[19] N. Bezirgiannis, F. S. de Boer, E. B. Johnsen, K. I. Pun, S. L. Tapia Tarifa, Implementing SOS with active objects: A case study of a multicore memory system, in: FASE, Vol. 11424 of LNCS, Springer, 2019, pp. 332–350. doi:10.1007/978-3-030-16722-6_20.

[20] K. Hightower, B. Burns, J. Beda, Kubernetes: Up and Running, O'Reilly, 2017.

[21] G. Turin, A. Borgarelli, S. Donetti, E. B. Johnsen, S. L. Tapia Tarifa, F. Damiani, A formal model of the Kubernetes container framework, in: ISoLA, Vol. 12476 of LNCS, Springer, 2020, pp. 558–577. doi:10.1007/978-3-030-61362-4_32.

[22] E. Kamburjan, R. Hähnle, S. Schön, Formal modeling and analysis of railway operations with active objects, Sci. Comput. Program. 166 (2018) 167–193. doi:10.1016/j.scico.2018.07.001.

[23] E. Kamburjan, J. Stromberg, Tool support for validation of formal system models: Interactive visualization and requirements traceability, in: F-IDE@FM, Vol. 310 of EPTCS, 2019, pp. 70–85. doi:10.4204/EPTCS.310.8.

Required Metadata

Current code version

Nr.	Code metadata description	
C1	Current code version	v1.9.3
C2	Permanent link to code/repository used for this code version	`https://github.com/abstools/abstools/releases/tag/v1.9.3`
C3	Permanent link to Reproducible Capsule	`https://hub.docker.com/repository/docker/abslang/absc`
C4	Legal Code License	BSD-3-Clause License
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Java 11, Erlang $\geq$ 23
C7	Compilation requirements, operating environments & dependencies	MacOS, Linux, Windows
C8	Link to developer documentation/-manual	`https://abs-models.org/`
C9	Support email for questions	abs-info@abs-models.org

Table 1: Code metadata (mandatory)