Universiteit Antwerpen

# Translating statecharts and class diagrams to DEVS

**Sam Pieters**

Principal Adviser: Hans Vangheluwe
Assistant Adviser: Joeri Exelmans

Dissertation Submitted in May 2024 to the
Department of Mathematics and Computer Science
of the Faculty of Sciences, University of Antwerp,
in Partial Fulfillment of the Requirements
for the Degree of Master of Science.

# Contents

# List of Figures

# List of Tables

# Nederlandstalige Samenvatting

Nederlandse samenvatting komt hier.

# Acknowledgements

Write Acknowledgements here

# Abstract

Write Abstract here

# CHAPTER 1

# Introduction

Blablabla

## 1.1 Definitions

Some introductory section

# CHAPTER 2

# Background

## 2.1 Blabla

### 2.1.1 Bliep using references

? was not mentioned in chapter ??

# CHAPTER 3

# Methodology

The methodology employed in this thesis outlines the systematic approach taken to translate System of Communicating Concurrent Deterministic Finite-State Machines (SCCD) into Discrete Event System Specification (DEVS). This section provides an overview of the strategies, tools, and processes utilized to achieve the translation objectives.

The translation of SCCD to DEVS involves converting models represented in one formalism into another while preserving their essential behavioral characteristics. Such translation is critical for interoperability, model verification, and simulation interoperability between systems designed using different modeling paradigms.

In this chapter, we present the overarching methodology for translating SCCD to DEVS, detailing the steps involved, challenges encountered, and the rationale behind the chosen approaches. Additionally, we discuss alternative approaches considered during the development process, providing insights into why certain strategies were pursued while others were discarded.

The methodology encompasses various stages, including initial approach exploration, tool selection, implementation workflow, validation, and verification. Each stage plays a crucial role in ensuring the accuracy and effectiveness of the translation process.

Furthermore, the methodology emphasizes the importance of evaluating the translated DEVS models against predefined criteria to assess their fidelity to the original SCCD specifications. By adhering to a systematic methodology, this research aims to contribute to the advancement of model translation techniques and facilitate seamless integration between different modeling formalisms.

This chapter serves as a guide to the methodology adopted in this thesis, providing readers with a comprehensive understanding of the processes involved in translating SCCD to DEVS.

## 3.1 Translation Tools

The translation of System of Communicating Concurrent Deterministic Finite-State Machines (SCCD) to Discrete Event System Specification (DEVS) involves the use of specialized tools and techniques to facilitate the conversion process. In this section, we discuss the evolution of the translation tools employed in this research, from initial parsing methods to the adoption of a visitor pattern for seamless translation.

Initially, the translation process began with a straightforward approach of parsing the SCCD models represented in XML format line by line. While this method provided basic functionality, it quickly became apparent that it was not efficient or robust enough to handle the complexities of SCCD models and their translation to DEVS.

Upon further investigation, it was discovered that the SCCD compiler (TODO: reference) already implemented a visitor pattern for translating XML files into Python and Java representations. Leveraging this existing pattern proved to be a significant breakthrough in the translation process. By utilizing the visitor pattern, which allows for the traversal of complex data structures without modifying the structure itself, we were able to streamline the translation process and improve its efficiency and accuracy.

The visitor pattern provided a structured and modular approach to translating SCCD models to DEVS, enabling clear separation between the translation logic and the underlying model representation. This decoupling of concerns allowed for easier maintenance, extensibility, and reuse of the translation code.

An additional advantage of adopting the visitor pattern for XML to DEVS translation was its compatibility with the existing compiler infrastructure of SCCD. The modular nature of the visitor pattern facilitated seamless integration of the DEVS translation functionality into the existing compiler framework of SCCD. This integration allowed for the incorporation of DEVS translation as a standard feature within the SCCD compiler, providing users with a comprehensive toolset for model development and analysis.

By adopting the visitor pattern for XML to DEVS translation, we were able to overcome many of the limitations encountered with the previous parsing methods. The visitor pattern facilitated a more systematic and reliable approach to translating SCCD models to DEVS, ultimately enhancing the

quality and fidelity of the translated models.

In summary, the evolution of translation tools from basic XML parsing to the adoption of the visitor pattern reflects the iterative nature of the research process, wherein insights gained from experimentation and exploration lead to the refinement and improvement of translation techniques.

## 3.2 Considered Translation Approaches

### 3.2.1 First approach: Implementing everything into one AtomicDEVS

### 3.2.2 Second approach: Setting every component to AtomicDEVS

The first approach was to set every possible component (Controller, ObjectManager and classes) to an AtomicDEVS model. This proved to be impossible with the standard atomicDEVS because in SCCD, statecharts can be created at runtime. The classic AtomicDEVS does not allow to create AtomicDEVS objects at runtime and thus mapping statecharts to a corresponding AtomicDEVS is impossible.

# CHAPTER 4

# Implementation

Blablabla

## 4.1 Definitions

Some introductory section

# CHAPTER 5

# Examples

Blablabla

## 5.1 Definitions

Some introductory section

# CHAPTER 6

# Conclusions

Example of todo: TODO: Write conclusions here.

# Bibliography

Serge Demeyer, Stéphane Ducasse, and Oscar Nierstrasz. *Object-Oriented Reengineering Patterns*. Square Bracket Associates, 2008.

# Appendices

# APPENDIX A

## An appendix

An Appendix is just like another chapter.