Efficiently Engineering IoT Architecture Languages—An Experience Report

Jörg Christian Kirchhof[1], Anno Kleiss[1], Judith Michael[1], Bernhard Rumpe[1] and Andreas Wortmann[2]

[1]*Software Engineering, RWTH Aachen University, Germany, https://se-rwth.de/*

[2]*Institute for Control Engineering of Machine Tools and Manufacturing Units (ISW), University of Stuttgart, Germany, https://www.isw.uni-stuttgart.de/*

Abstract

Engineering architecture description languages (ADLs) is complex. Yet, research and industry have developed over 120 ADLs for various purposes. Many of these languages share similar concepts and elements. Instead of creating a novel ADL for the IoT from scratch, we created the MontiThings IoT ADL through systematically reusing (parts of) various stand-alone languages. In this paper, we detail the MontiThings ADL family, its constituents, and language reuse mechanisms. Researchers and practitioners in the engineering of (IoT) ADLs can benefit from these insights to prevent creating another 120 ADLs from scratch.

Keywords

Internet of Things, Model-Driven Engineering, Architecture Description Language

1. Introduction

MontiThings is an ecosystem for the model-driven development, deployment, and analysis of Internet of Things (IoT) applications [1, 2, 3]. MontiThings can generate C++ code from its models and also provides the necessary scripts to containerize the code using Docker.

2. Language Features

MontiThings is developed using the MontiCore Language Workbench [4], leveraging the library of composable modeling languages [5] offered by MontiCore (*cf.* Fig. 1).

Type System MontiThings offers primitive types similar to Java or C++ (`byte`, `int`, `long`, `float`, `boolean`, `char`, `String`). Additionally, collections types can group objects (Set, List, Map). To facilitate working with sensor data, MontiThings enables using SI units as primitive types (*e.g.,* `kg`, `dB`, `km`, `s`, `°C`, . . .) by extending MontiCore's SI Unit language. MontiThings automatically converts the SI unit values if necessary (*e.g.,* `m/s` to `km/h`). Developers can specify their own types using class diagrams from MontiCore's CD4Analysis language.

MESS'22: International workshop on MDE for Smart IoT Systems, July 04–08, 2022, Nantes, France

0000-0002-8188-3647 (J. C. Kirchhof); 0000-0002-1378-3097 (A. Kleiss); 0000-0002-4999-2544 (J. Michael); 0000-0002-2147-1966 (B. Rumpe); 0000-0003-3534-253X (A. Wortmann)

© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

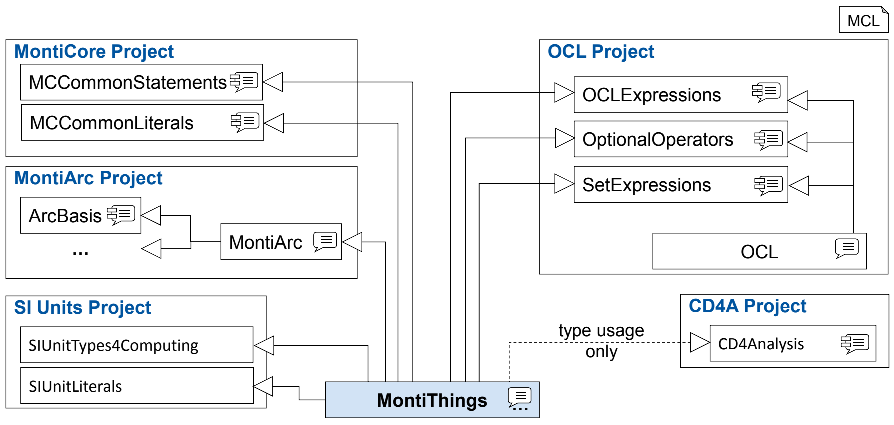

Figure 1: Overview of MontiThings' language inheritance hierarchy.

Behavior MontiThings components offer three modes of computation: initialization, *i.e.,* behavior executed when starting the component, cyclic behavior, *i.e.,* behavior executed in time intervals, and event-based behavior, *i.e.,* behavior executed in response to receiving a message. The three modes can be combined within the same component but only one behavior can be executed simultaneously per component to prevent race conditions. The behavior of MontiThings components can be specified in four ways: Composed components instantiate and connect subcomponents to specify their own behavior. Atomic components do not have subcomponents, but specify their behavior through a programming language embedded in the model (using MontiCore's `MCCommonStatements` language), statecharts (from MontiCore's statechart language) or handwritten C++ code.

Expressions and Literals MontiThings' expressions are mainly built on top of the expressions provided by MontiCore out of the box, *i.e.,* assignments, mathematical, and boolean operators (*e.g.,* +, −, <=, or `||`). Additionally, MontiThings' reuses expressions from object constraint language (OCL) such as `@pre`, set expressions such as union or intersection of sets or checking if a set contains a given element. MontiCore's literal grammars offer MontiThings the capability to create numbers, strings, or boolean values. The `SIUnitLiterals` enable creating numbers with international system of units (SI) types such as `17 km/h`. To instantiate classes from class diagrams, MontiThings uses object diagrams using a JSON-like syntax.

3. Discussion, Recommendations, and Conclusion

MontiThings defines about 710 lines of grammar in 14 grammars and reuses 4371 lines of grammar from 46 grammars from the MontiCore project. We therefore assumed that language libraries can substantially reduce the effort required to implement new (IoT) architecture description languages (ADLs). To test this assumption, we tried to re-implemented Ericsson's IoT ADL *Calvin* [6, 7, 8]. We were able to parse (slightly adapted) Calvin models using only existing language components adapted with about 50 lines of grammar to match Calvin's syntax. Another 15 lines of grammar were needed to add CalvinScript, which is not covered by MontiCore's language library. The main limitation to this approach is that deviating from the language library's infrastructure (*e.g.,* type system) may require a non-negligible amount of work. As re-implementing the Calvin's Python-like type system would have implied significant changes, we kept MontiThings' type system. Thus, the Calvin models needed to be adapted to contain explicit type names. Nevertheless, this experiment suggests that future IoT ADLs could reuse models of existing IoT ADLs with only limited effort. Providing parsers for other IoT ADLs can lower the entry barrier for modelers who are already invested in another IoT ADL.

The intensive reuse of existing language components enables us to reduce the effort for creating a new language. Language engineers can extend already mature and tested languages but the approach still allows to add needed domain-specific extensions. Thus, we recommend language engineers to not build new IoT languages from scratch but utilize language libraries and consider the option of providing parsers for existing IoT ADLs.

Source Code

MontiThings is available on GitHub: https://github.com/MontiCore/montithings

Acknowledgments

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy—EXC 2023 Internet of Production—390621612. Website: https://www.iop.rwth-aachen.de

References

[1] J. C. Kirchhof, B. Rumpe, D. Schmalzing, A. Wortmann, MontiThings: Model-driven Development and Deployment of Reliable IoT Applications, Journal of Systems and Software 183 (2022) 111087.

[2] J. C. Kirchhof, A. Kleiss, B. Rumpe, D. Schmalzing, P. Schneider, A. Wortmann, Model-Driven Self-Adaptive Deployment of Internet of Things Applications with Automated Modification Proposals, ACM Transactions on Internet of Things (In Press, 2022, DOI: https://doi.org/10.1145/3549553).

[3] J. C. Kirchhof, L. Malcher, B. Rumpe, Understanding and Improving Model-Driven IoT Systems through Accompanying Digital Twins, in: E. Tilevich, C. De Roover (Eds.), Proceedings of the 20th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE '21), ACM SIGPLAN, 2021, pp. 197–209.

[4] K. Hölldobler, O. Kautz, B. Rumpe, MontiCore Language Workbench and Library Handbook: Edition 2021, Aachener Informatik-Berichte, Software Engineering, Band 48, Shaker Verlag, 2021. URL: https://www.monticore.de/handbook.pdf.

[5] A. Butting, R. Eikermann, K. Hölldobler, N. Jansen, B. Rumpe, A. Wortmann, A Library of Literals, Expressions, Types, and Statements for Compositional Language Design, Special Issue dedicated to Martin Gogolla on his 65th Birthday, Journal of Object Technology 19 (2020) 3:1–16.

[6] O. Angelsmark, P. Persson, Requirement-Based Deployment of Applications in Calvin, in: I. Podnar Žarko, A. Broering, S. Soursos, M. Serrano (Eds.), Interoperability and Open-Source Solutions for the Internet of Things, Springer International Publishing, Cham, 2017, pp. 72–87.

[7] P. Persson, O. Angelsmark, Calvin – Merging Cloud and IoT, Procedia Computer Science 52 (2015) 210 – 217. 6th Int. Conf. on Ambient Systems, Networks and Technologies (ANT).

[8] P. Persson, O. Angelsmark, Kappa: Serverless iot deployment, in: Proceedings of the 2nd International Workshop on Serverless Computing, WoSC '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 16–21.