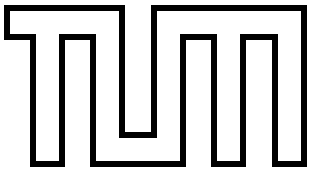


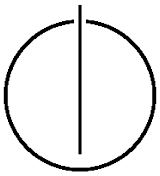
FAKULTÄT FÜR INFORMATIK

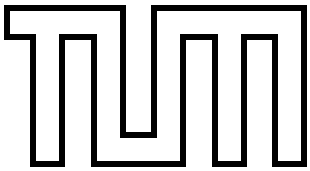
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master Thesis in Informatics

Adding C++ Support to MBEDDR

Zaur Molotnikov





FAKULTÄT FÜR INFORMATIK

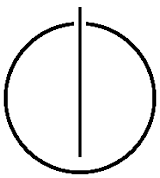
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master Thesis in Informatics

Adding C++ Support to MBEDDR

C++ Unterstützung für MBEDDR

Author:	Zaur Molotnikov
Supervisor:	Dr. Bernhard Schätz
Advisor:	Dr. Daniel Ratiu
Date:	September 16, 2013



Ich versichere, dass ich diese Masterarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 16. September 2013 Zaur Molotnikov

Acknowledgments

If someone contributed to the thesis... might be good to thank them here.

Abstract

An abstracts abstracts the thesis!

Contents

1 Introduction

In embedded programming the C++ programming language is widely spread, [4]. Being a general purpose programming language, C++ does not provide, however, any special support for an embedded systems programmer.

By changing the language itself, together with a tool set for it, it is possible to get a better environment for a dedicated domain, for example, specifically for embedded programming.

The first possible approach is dropping some language features, to get the language, which is simpler. As an example, a subset of C++, called *Embedded C++* can be brought, [2]. The approach taken in Embedded C++ is omitting very many core features of C++ off, allows for a higher degree of optimizations by compiler possible. Embedded C++ was intended to allow higher software quality through better understanding of the limited C++ by programmers, higher quality of compilers, better suitability for the embedded domain, [1]. This approach, however, has been criticized by the C++ community, specifically for the inability of the limited language to take advantage of the C++ standard library, which requires the C++ language features, absent in Embedded C++, [5].

The second approach to modify a language to get it more suitable for the embedded development, consists of extending the language with constructions specific to the domain. Such approach is taken, for example, in the *MBEDDR Project*, to improve on the C programming language, [6]. Extensions to C language developed in *MBEDDR Project* include state machines and decision tables.

A special language engineering environment is used to support modular and incremental language development in *MBEDDR Project*, *JetBrains MPS*. The language under development is split in a special class-like items, called concepts. As an example of a concept expression can be taken. Over the inheritance mechanisms, it is possible to extend languages, providing new concepts as children of the existing ones. For example, expression concept can be extended to support new sort of expressions, e.g. decision tables.

Building a general purpose programming language in a language engineering environment brings a basis to develop domain specific extensions to a well-known general purpose language.

Additionally to the language modification, the *Integrated Development Environment (IDE)* can be improved to support domain specific development. Various analyses can be built in into the code editor in order to detect inconsistencies, or, simply, "dangerous" constructs, and inform the programmer. Certain code formatting, or standard requirements could be enforced as well, and many more.

A mixture of the two approaches could be used in an attempt to achieve a "better" C++ for embedded development. A special *IDE* can be created together with a new C++ language flavor, which supports the embedded C++ programmer. This is the problem to be solved by this Master Thesis.

The new language together with the new *IDE* can later serve as a basis for extending the

C++ programming language with domain specific constructs for embedded programming. Creation of these extensions lies out of scope for this Master Thesis, and is left for further research.

The approach taken in this work goes further into exploring the language modularity on the basis of *JetBrains MPS*Ẇhile building the C++ programming language itself with the goal of embedded domain specific extensions in mind, the C++ itself is being built itself as an extension to the C programming language, provided by the *MBEDDR Project*.

Although C++ is a separate from C language, the high degree of similarity allows to make use of the C programming language, implemented by the *MBEDDR Project* as a foundation. Not only reuse of the basic C is achieved, but also the embedded extensions from the *MBEDDR Project* are immediately supported by the newly built C++.

This work explores further the support, provided by *JetBrains MPS* for the modular language construction, [3], and reviews it from the architectural point of view.

The C++ programming language is provided with a number of automations and analyses for it. The automations include code generation and structuring. They are implemented as a programming on the *Abstract Syntax Tree (AST)* in a Java-like programming language. The analyses and automations intend to improve quality, security and better understanding of the code.

In the *JetBrains MPS Application Programming Interface (API)* it is not explicitly defined, when the analyses and checks take place, how much of the computational resource they can take advantage of. This may affect the overall *IDE* performance, as the analyses complexity may be high. The question of analyses run-time and complexity is raised and discussed in this work.

2 Foundations

The C++ programming language developed through out this Master Thesis is based on two technologies, which are introduced in this chapter. The first technology is the *JetBrains MPS* language engineering environment, which provides the core foundations and means for incremental language construction. The second technology is the *MBEDDR Project*

2.1 Jetbains MPS

2.2 MBEDDR Project

3 Technologies in Use

4 Projectional C++ Implementation

5 Evaluation

6 Conclusion

Appendix

Glossary

API Application Programming Interface. 2

AST Abstract Syntax Tree. 2

concept is a class-like item, representing a modularity base of a language, e.g. expression, statement, method, etc.. 1

Embedded C++ is a language subset of the C++ programming language, intended to support embedded software development. 1

IDE Integrated Development Environment. 1, 2

JetBrains MPS is a language engineering environment allowing to construct incrementally defined domain specific languages. 1–3

MBEDDR Project is a JetBrains MPS based language workbench, representing C language and domain specific extensions for the embedded software development. 1–3

Bibliography

[1] Embedded C++. Objectives behind limitting c++, http://www.caravan.net/ec2plus/objectives/ppt/ec2ppt03.html.

[2] Embedded C++. Official website, http://www.caravan.net/ec2plus/.

[3] Zaur Molotnikov Daniel Ratiu, Markus Voelter and Bernhard Schaetz. Implementing modular domain specific languages and analyses. In *Proceedings of the 9th Workshop on Model-Driven Engineering, Verification and Validation*, 2012.

[4] VDC Research. Survey on embedded programming languages, http://blog.vdcresearch.com/embedded_sw/2010/09/what-languages-do-you-use-to-develop-software.html.

[5] Bjarne Stroustrup. Quote on embedded c++, http://www.stroustrup.com/bs_faq.html#ec++.

[6] Markus Voelter, Daniel Ratiu, Bernhard Schätz, and Bernd Kolb. mbeddr: an extensible c-based programming language and ide for embedded systems. In *SPLASH*, pages 121–140, 2012.