Technische Universität
München

Fakultät für
Informatik

Master's Thesis

Adding C++ Support to mbeddr

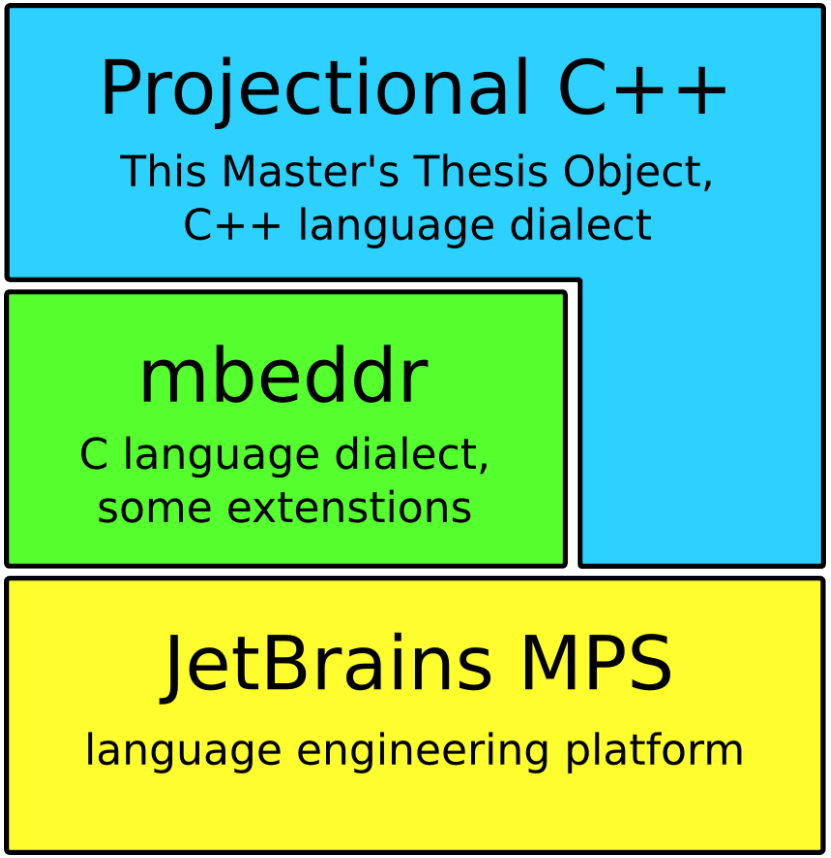
Language Engineering for C++ over the mbeddr Project C implementation

Presents: Zaur Molotnikov
Advisor: Dr. rer. nat. Daniel Ratiu
Supervisor: PD Dr. rer. nat. habil. Bernhard Schätz

Adding C++ to mbeddr

Layered architecture

- MPS: *building IDEs**
- mbeddr (Fortiss, Itemis): *better C flavor + extensions for embedded development*
- Why to support C++: *new programming paradigms, mbeddr users want it*



*Integrated Development Environments

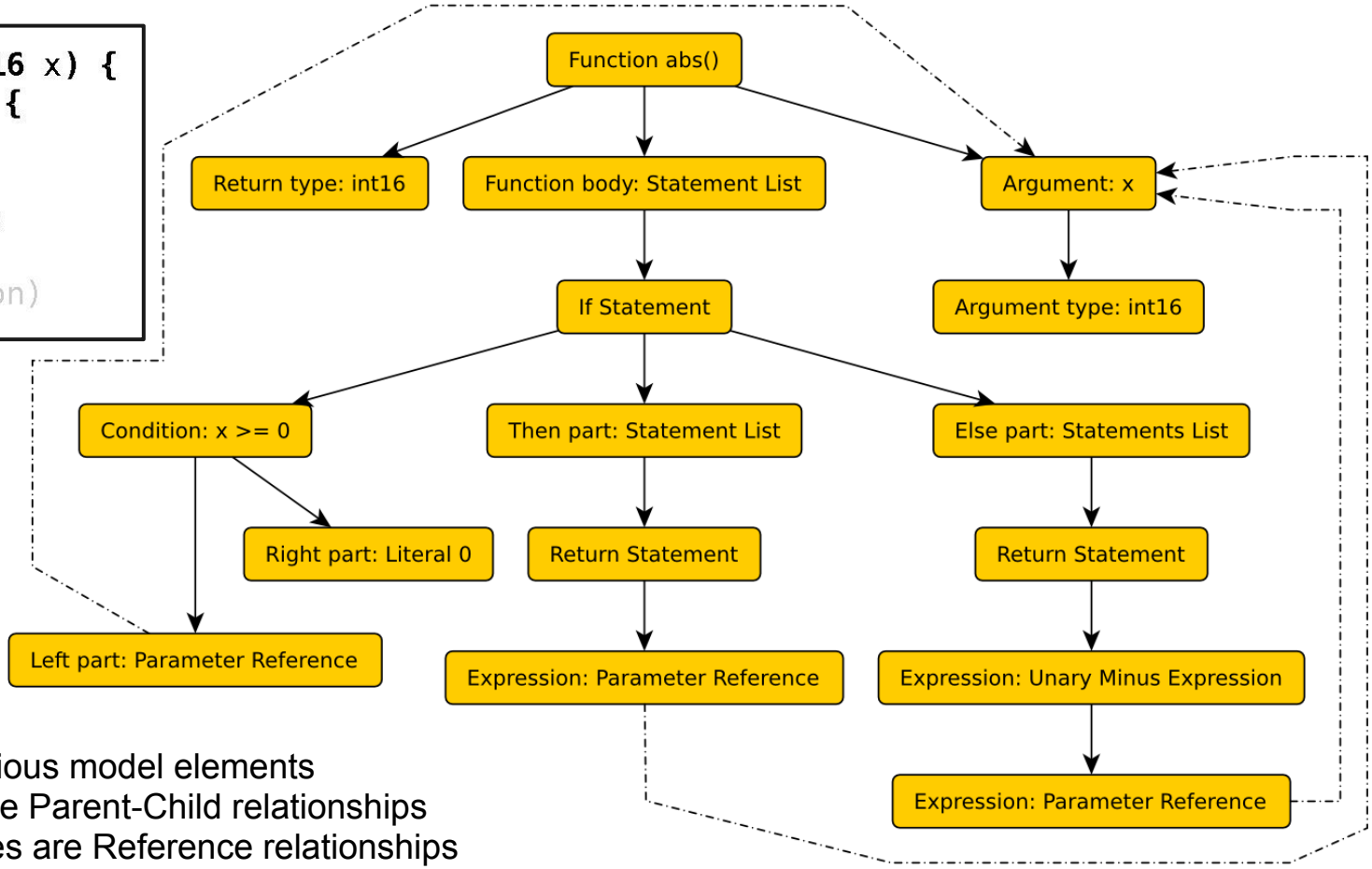
mbeddr and MPS

- mbeddr C language code sample
- Usually IDEs work with text code
- MPS works with abstract syntax trees instead
- Abstract syntax trees are *projected* as text

```
int16 abs(int16 x) {
  if (x >= 0) {
    return x;
  } else {
    return -x;
  } if
} abs (function)
```

Abstract Syntax Tree

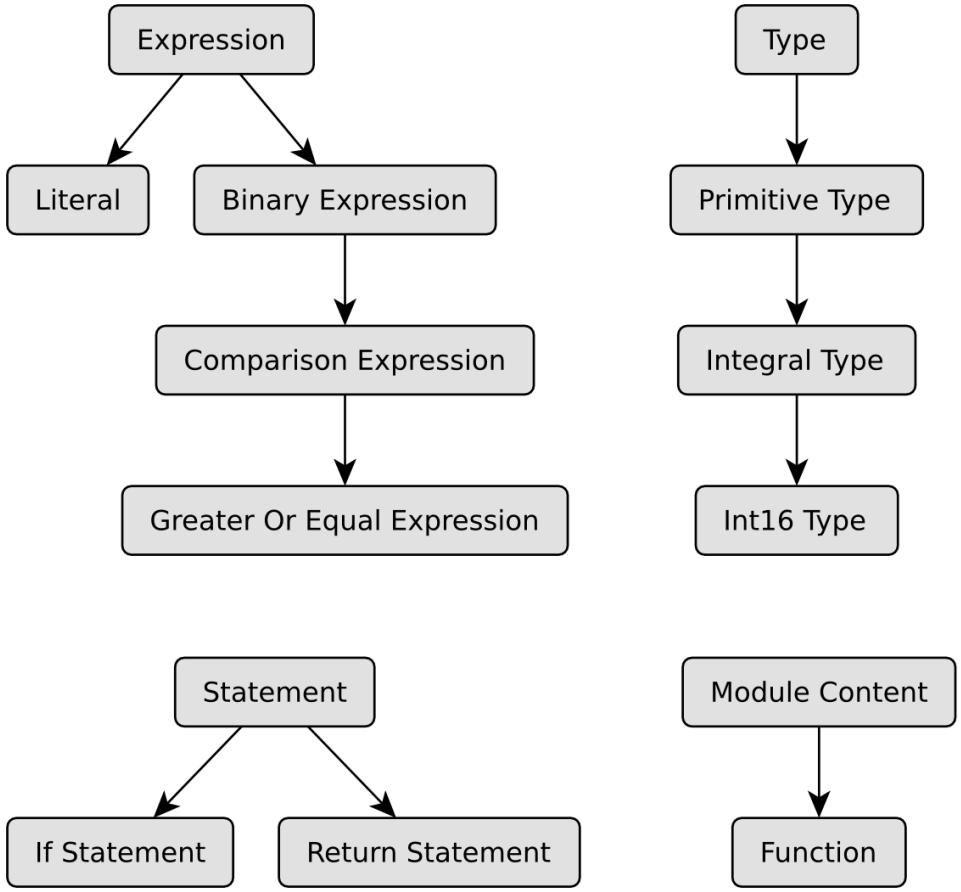
```
int16 abs(int16 x) {
  if (x >= 0) {
    return x;
  } else {
    return -x;
  } if
} abs (function)
```



Nodes are various model elements
Solid edges are Parent-Child relationships
Dash-dot edges are Reference relationships

Concept Hierarchies

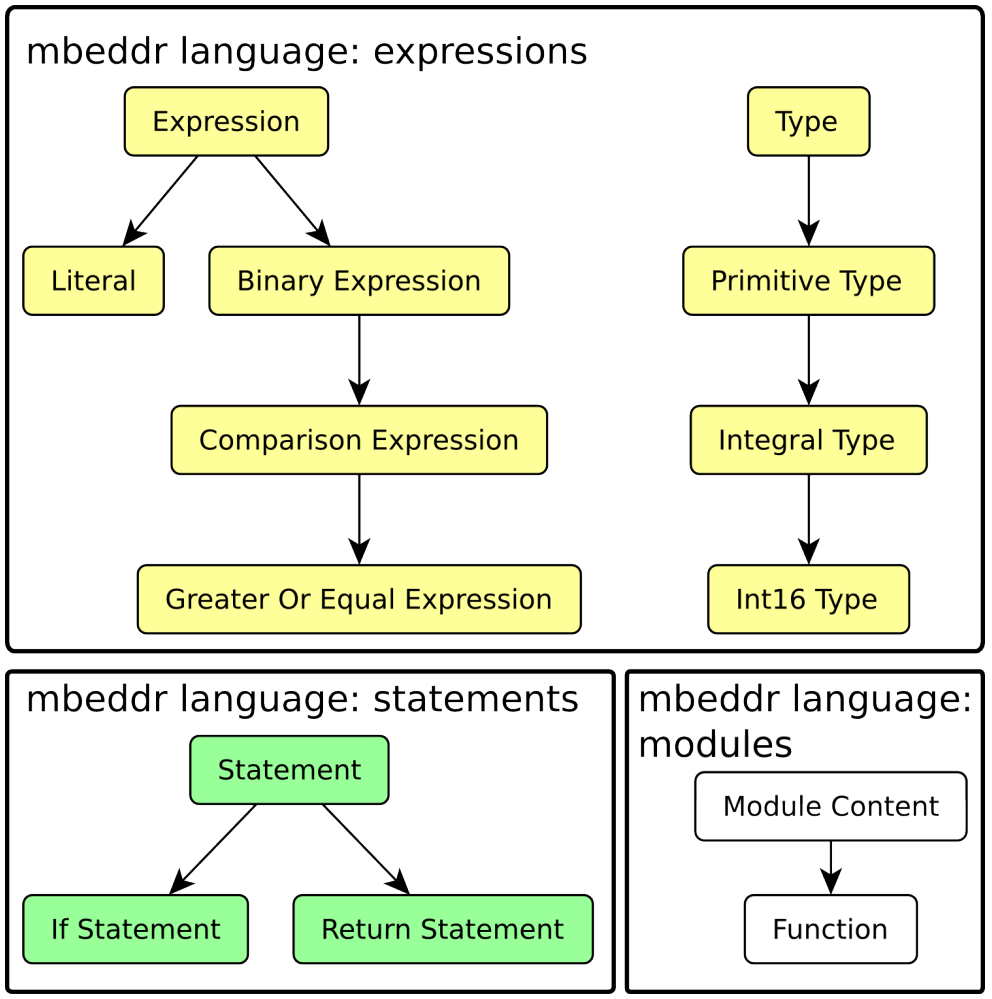
- *Concepts* are types of nodes, similar to classes, MPS term
- *Concepts* form inheritance hierarchies
- A node of a child *concept* can be used in place of a parent *concept* node



Language Modularity

```
int16 abs(int16 x) {
  if (x >= 0) {
    return x;
  } else {
    return -x;
  } if
} abs (function)
```

- MPS language is a set of *concepts*
- mbeddr represents many MPS languages
- but an IDE for one language - namely C



Language Extensibility

State Machine Expression (mbeddr)

- Extends an Expression parent *Concept*
- Features an analysis

```
enum mode { MANUAL; AUTO; FAIL; }

mode nextMode(mode mode, int8_t speed) {
  return mode, FAIL
```

	mode == MANUAL	mode == AUTO
speed < 30	MANUAL	AUTO
speed > 30	MANUAL	MANUAL

```
} nextMode (function)
```

Taken from [7], Ratiu 2012

Defining a *Concept*

- A *concept* is defined in views on it
  - **Structure** view - properties and relationships*
  - **Behavior** view - add methods like to a Java class*
  - **Editor** view - the way to input and edit an instance
  - **Constraints** view - add context sensitive limitations
  - **Type system** view - for typed languages
  - **Non-type-system** checks - for warnings and errors
  - **TextGen** view - to generate to text
  - **Intentions** view - provide user-callable automations

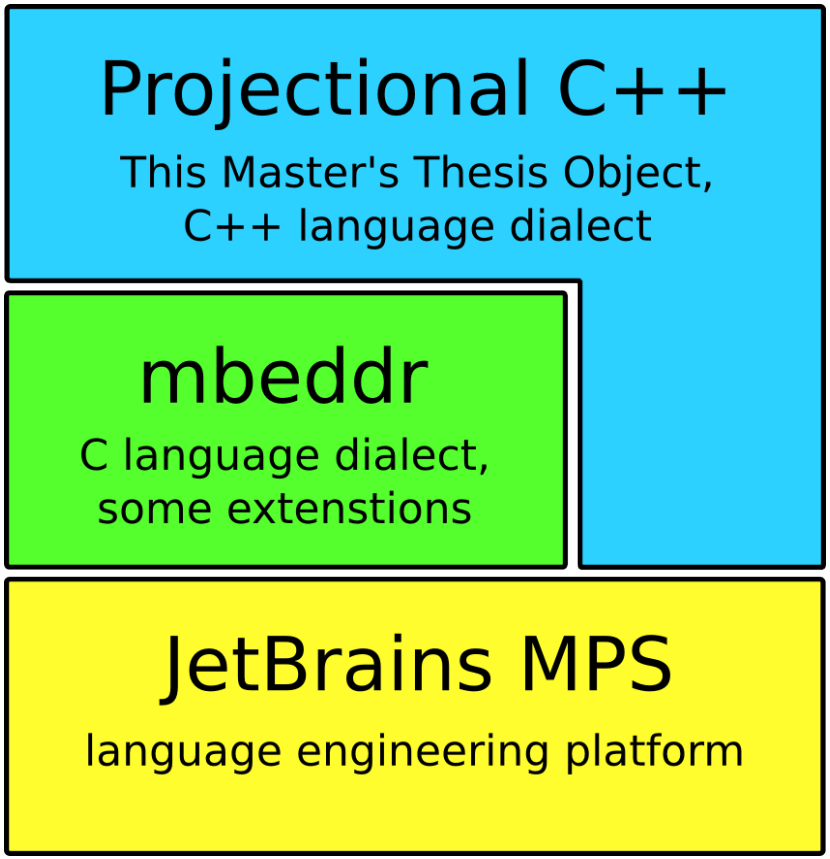
Some other views exist, they are not related to this work

* These views are similar to a class definition: fields and methods.

Adding C++ to mbeddr

Back on the track:

- The goal is to
- with the use of *language modularity* and
- *language extensibility*
- add C++ to mbeddr C language
- using JetBrains MPS platform

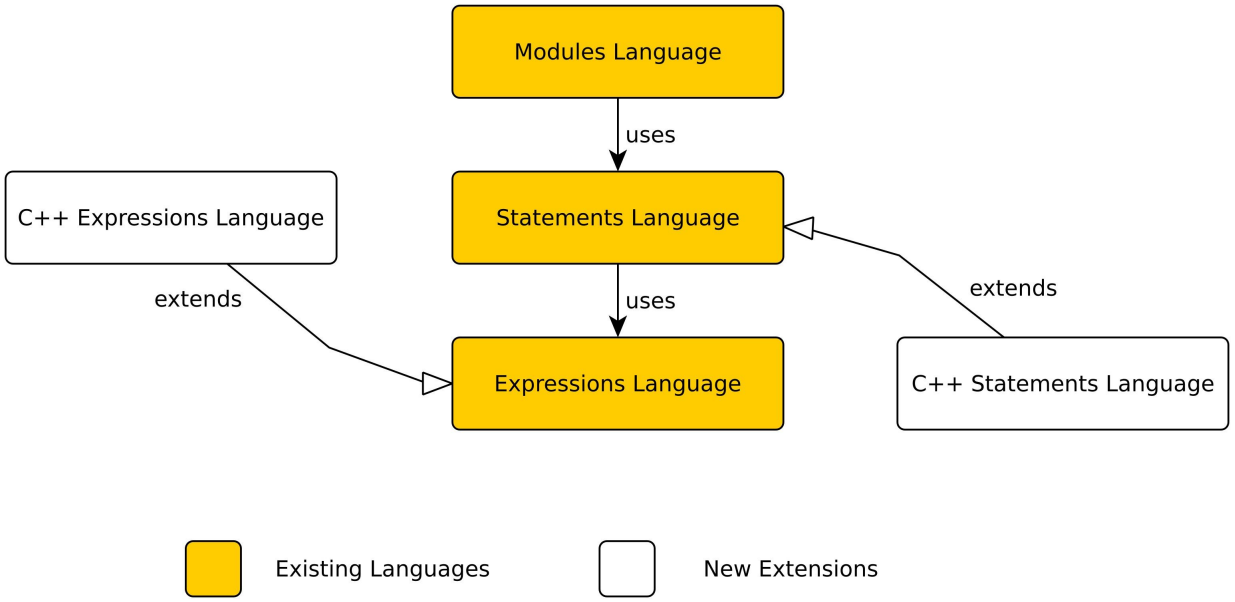


Questions, challenges...

1. Is it in general possible to extend mbeddr C to C++? *Will mbeddr be flexible enough?*
2. Is it possible to make a “better” C++ flavor? *Like mbeddr C is a better C.*
3. Templates in C++ bear pure textual nature! *A contradiction with concepts, defined in advance.*
4. How well does MPS support “complex” extensions? *Practice of language extensibility.*

Q1: Extending C to C++

- Practically proven to be possible



- One-side-awareness challenge

Q2: Better flavor of C++?

- Cutting out language features like mbeddr - impossible - no STL support
- Adding language features
  - Analyses to improve understanding *(abstract class)*
  - Information, made explicit *(override)*
  - Code generation, automations *(getter and setter)*
  - Naming conventions made explicit *(naming of fields)*

And...

- Projectional C++ is a base for future extensions. *Signals, design patterns, more?*

Example:

- Abstract classes, pure virtual functions, overrides have no syntax in C++, added:

```
abstract class Widget /copyable and assignable/ {
  public:
    explicit Widget(Widget* parent) (constructor)
    pure virtual Size getDimensions()  = 0
}

abstract class Button : public Widget /copyable and assignable/ {
  public:
    Button() (constructor)
    pure virtual boolean isPressed()  = 0
}

class PushButton : public Button /copyable and assignable/ {
  public:
    PushButton() (constructor)
    virtual Size getDimensions() overrides Widget::getDimensions()
    virtual boolean isPressed() overrides Button::isPressed()
}
```

Q3: Templates?

- Implemented through “C++ concepts”
- Have a number of advantages and disadvantages
  - explicit
  - checkable

  but
  - absent in C++
  - special importer
  - additional work
  - code duplication

- Another approach?

```
concept Comparable {
  public:
    int8 compare(Comparable c1)
}

realizes Comparable
class NumberWrapper /copyable and assignable/ {
  public:
    int8 compare(NumberWrapper other)
    NumberWrapper(int8 v) (constructor)
  private:
    int8 mValue
}

template <class T: Comparable>
class OrderedList /copyable and assignable/ {
  public:
    OrderedList() (constructor)
    int8 compare(T first, T other)
}
```

Q4: MPS Extensibility?

View	Extensibility Support	Workarounds Quality
Structure	High	-
Editor	No	Poor
Constraints	Low	Good
Behavior	High	-
TextGen	High	-
Generators	-	-
Intentions	No	Medium
Type System	Low	Medium
Analyses	No	Medium

- MPS can provide better support for extensibility

Lessons Learned

- Rebuilding a Language in Projection - Guidelines
- Analyses and Complexity - Future MPS Improvements

Rebuilding a Language in Projection

Few principles discovered may apply to every language reconstructed:

- Target semantics - pure virtual functions, exts
- Store more information - overrides
- Configuration is a part of source - naming
- Hide redundant syntax - braces, etc.
- Make syntax human readable - pure virtuals
- Show core, hint on details - friend function
- Perform analyses - abstract classes

Analyses and Complexity

- Analyses were found to be useful, however
  - MPS does not support them explicitly
  - Computational complexity can be high enough
- Propositions for MPS evolution, APIs for analyses:
  - When analysis start?
  - Which scope do they have?
  - Result caching needed?
  - Prioritisation, concurrency limitations?
  - Informing user - can be improved and
  - Common solutions offered for reuse

Future Work

- Complete language support + STL
- Investigating language use
- Importer, templates
- Debugger
- Extensions on top of Projectional C++
- JetBrains MPS Evolution

Thank you!

Thank you for your attention!

You are welcome to ask questions:

Zaur Molotnikov

zaur@zaurmolotnikov.com