Adding C++ Support to mbeddr

Language Engineering to Build an IDE for C++

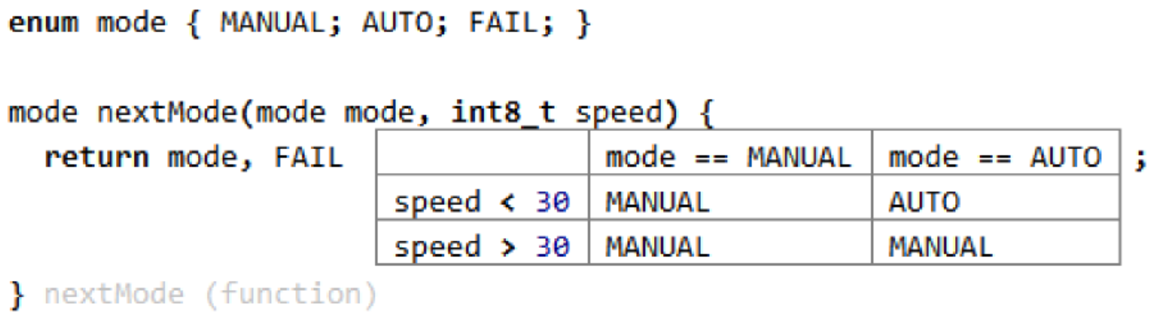
Master's Thesis

Presents: *Zaur Molotnikov*

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

Context

Example 1 : Decision Table

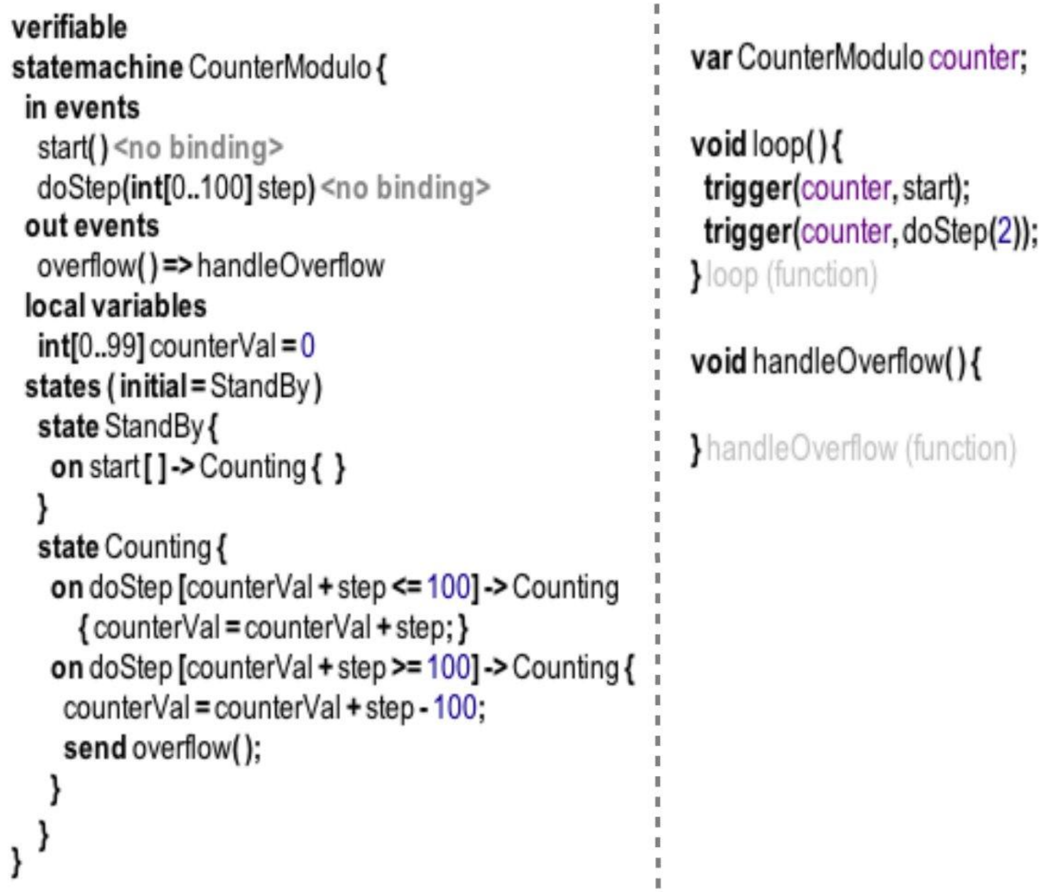


```
enum mode { MANUAL; AUTO; FAIL; }

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

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

```
;
} nextMode (function)
```

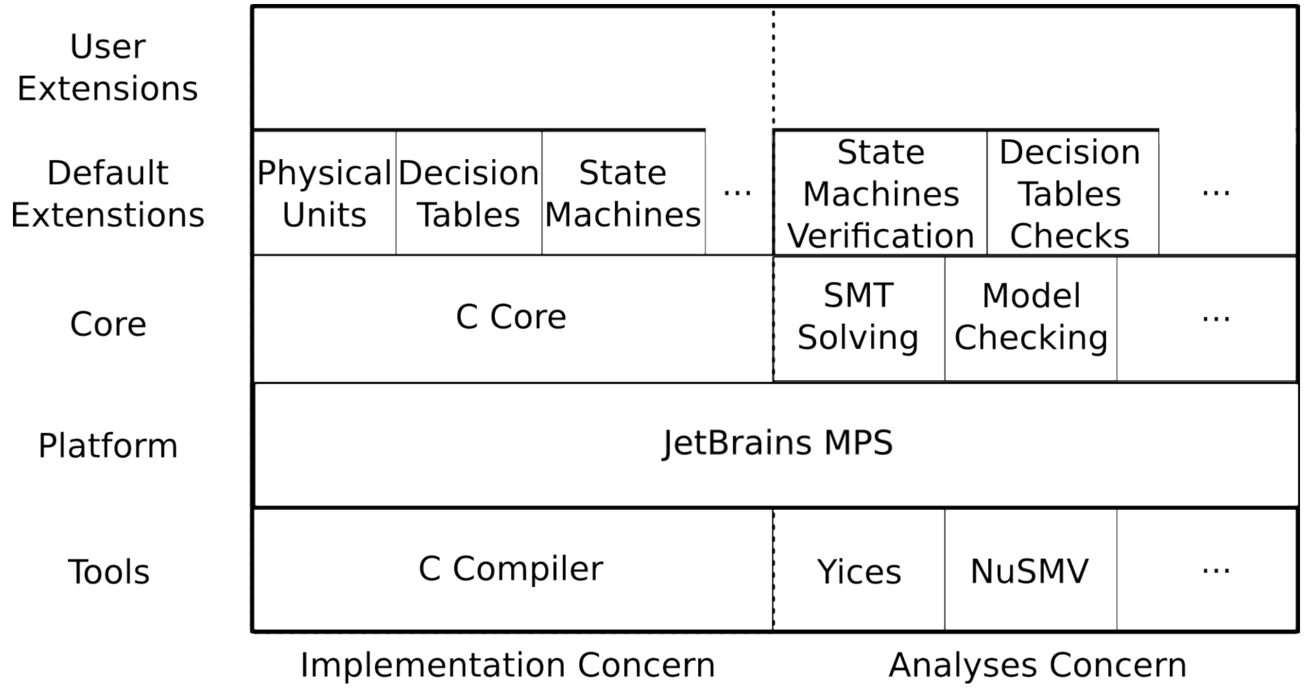
Example 2 : State Machine



```
verifiable
statemachine CounterModulo {
 in events
  start() <no binding>
  doStep(int[0..100] step) <no binding>
 out events
  overflow() => handleOverflow
 local variables
  int[0..99] counterVal = 0
 states ( initial = StandBy )
  state StandBy {
   on start [ ] -> Counting {  }
  }
  state Counting {
   on doStep [counterVal + step <= 100] -> Counting
      { counterVal = counterVal + step; }
   on doStep [counterVal + step >= 100] -> Counting {
    counterVal = counterVal + step - 100;
    send overflow();
   }
  }
}
```

```
var CounterModulo counter;

void loop() {
 trigger(counter, start);
 trigger(counter, doStep(2));
} loop (function)

void handleOverflow() {

} handleOverflow (function)
```

mbeddr : Technology Stack



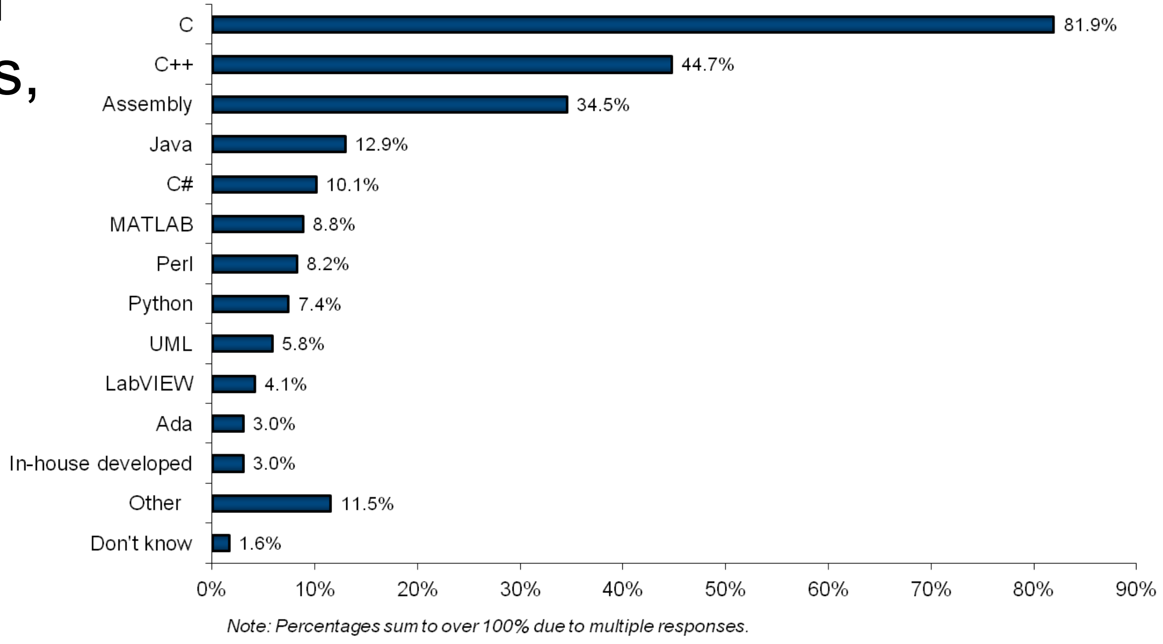
Safer C dialect for embedded development:

- only C core supported - “unsafe” constructions dropped;
- domain specific extensions with analyses

Problem

C++ Support

- C++ is popular among embedded system developers, but
- mbeddr does not support C++, so it makes sense to
- extend mbeddr to support C++



Source - VDC Research: http://blog.vdcresearch.com/embedded_sw/2010/09/what-languages-do-you-use-to-develop-software.html

Extending mbeddr

Language Engineering in Practice

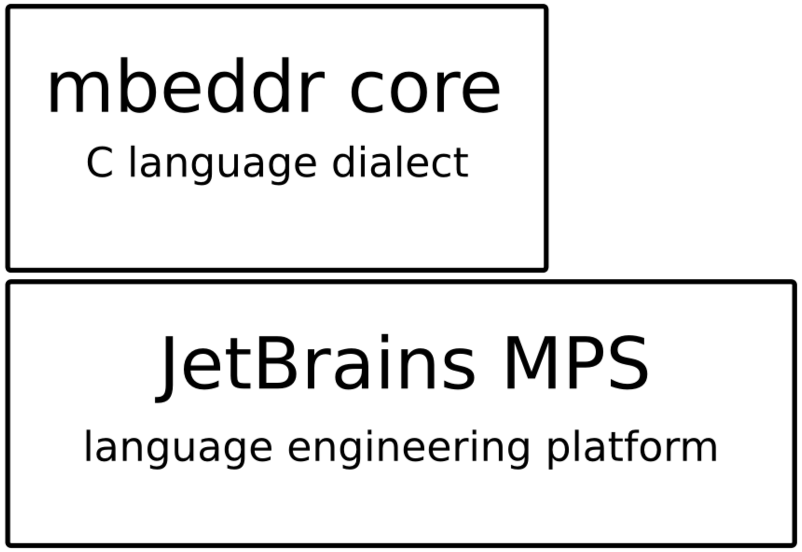
- **mbeddr core** is mainly a C programming language - all constructions are valid C++

mbeddr core
C language dialect

Extending mbeddr

Language Engineering in Practice

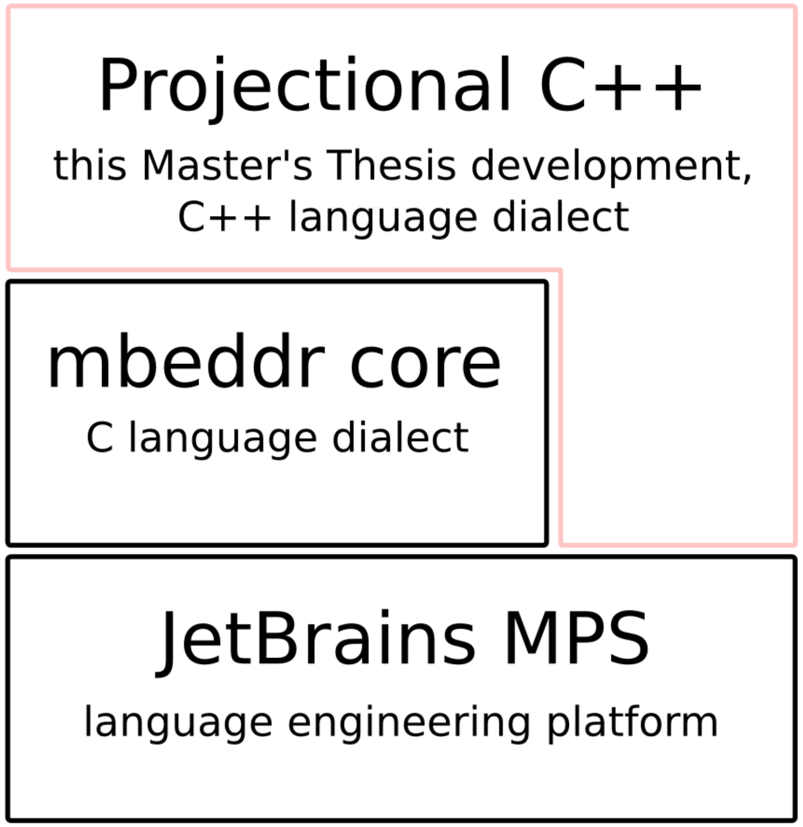
- **mbeddr core** is mainly a C programming language - all constructions are valid C++
- based on a language engineering framework **JetBrains MPS**



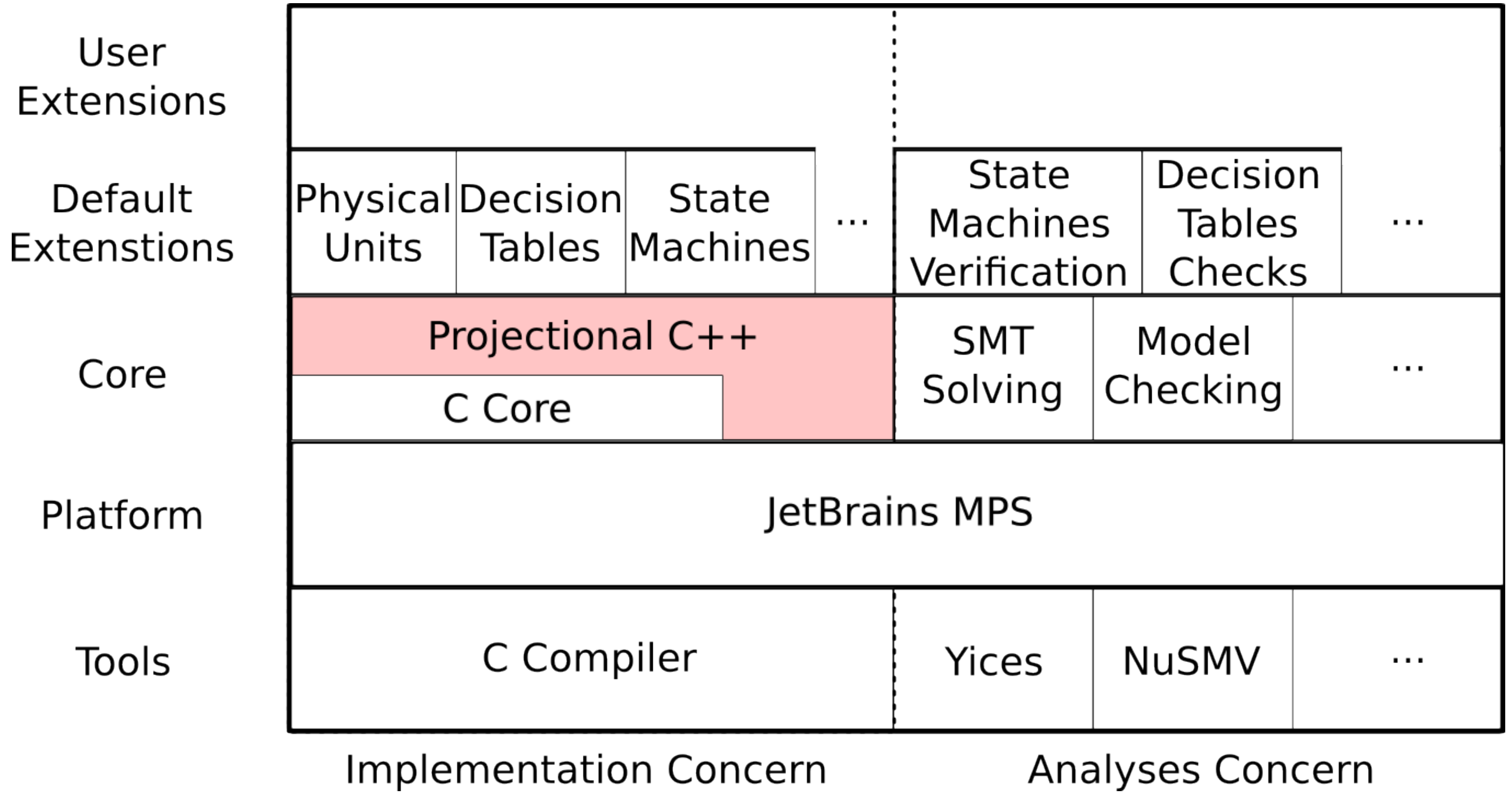
Extending mbeddr

Language Engineering in Practice

- **mbeddr core** is mainly a C programming language - all constructions are valid C++
- based on a language engineering framework **JetBrains MPS**
- to which we add C++ programming language, we call it ***Projectional C++***


Projectional C++
this Master's Thesis development, C++ language dialect
mbeddr core
C language dialect
JetBrains MPS
language engineering platform

Projectional C++ in mbeddr Technology Stac

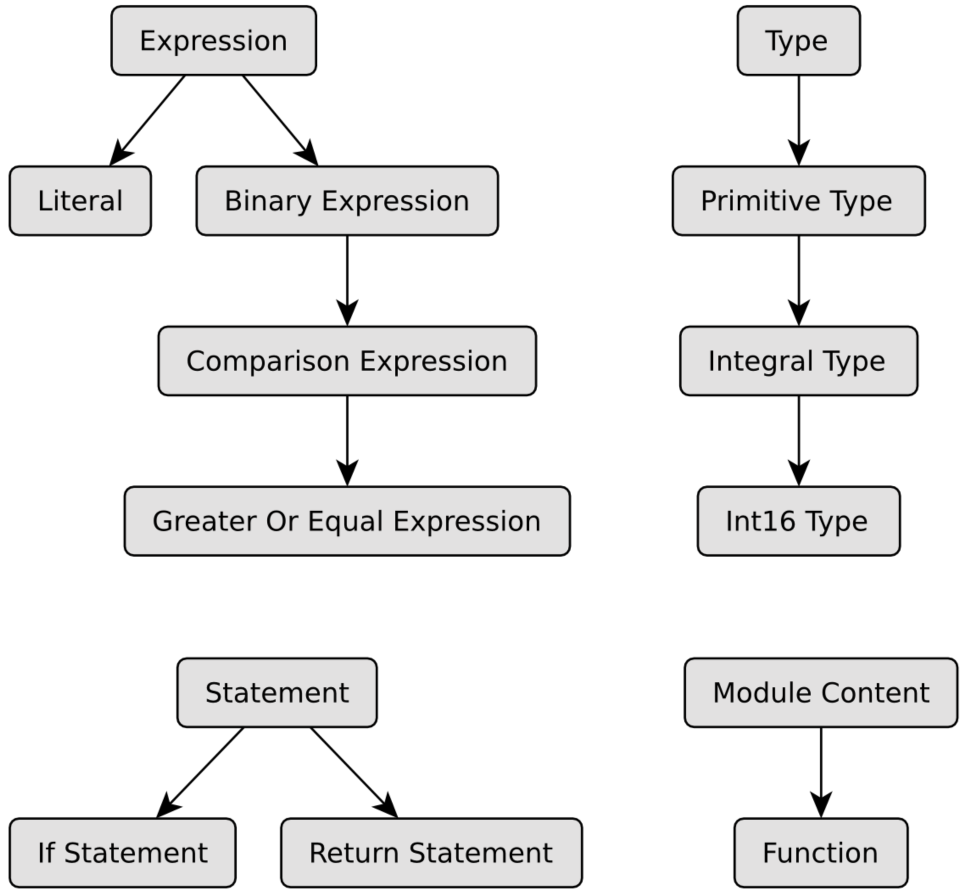


Approach

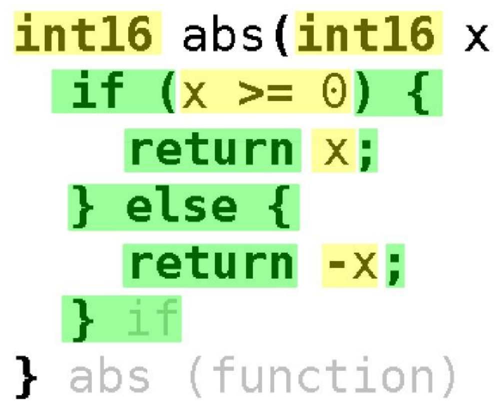
Meta-Model Hierarchies

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

- Language syntax is a meta-model
- Model is the code
- Code is *projected*

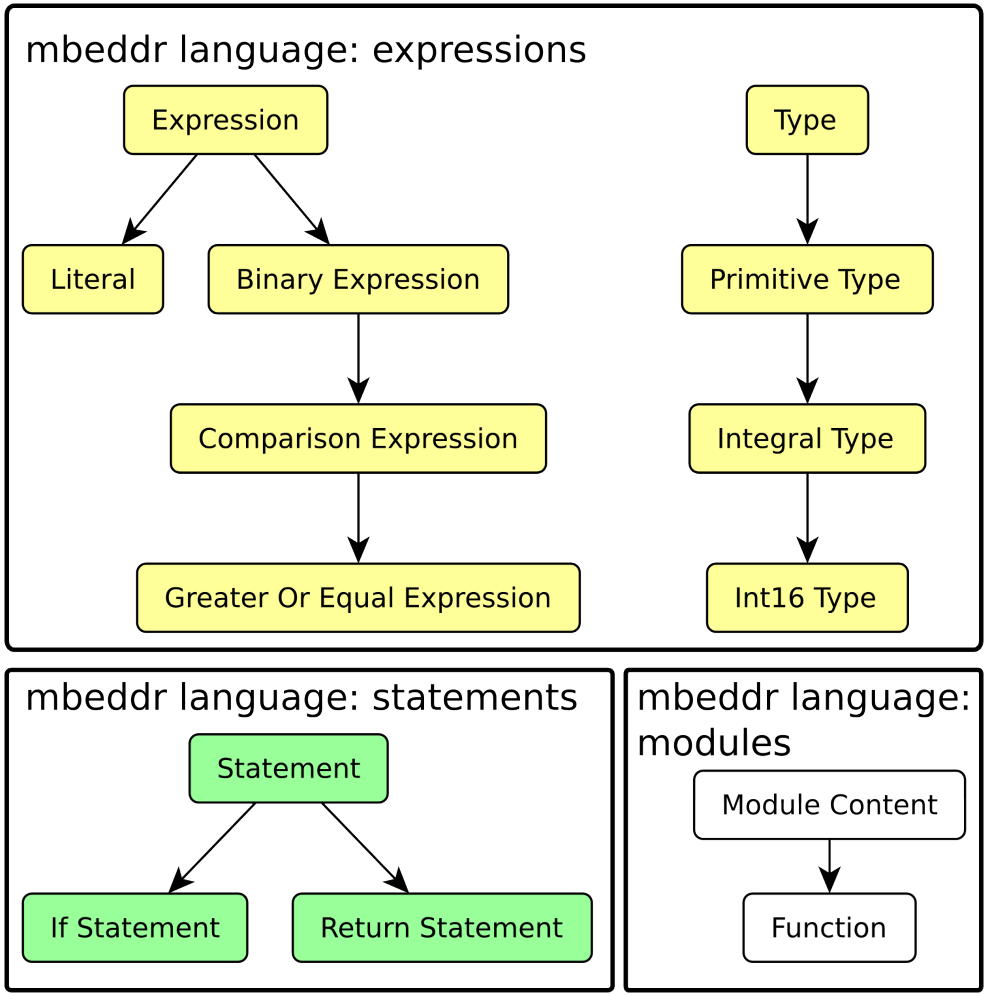


Language Modularity



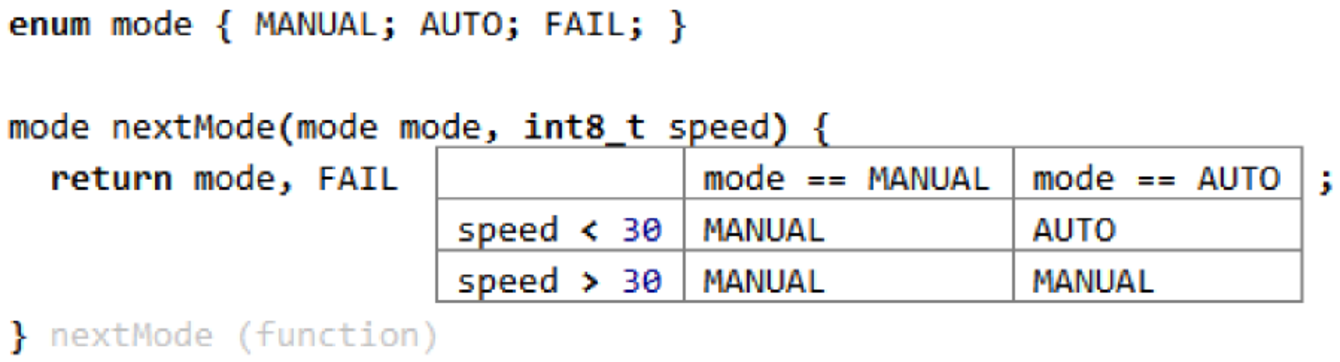
```
int16 abs(int16 x) {
  if (x >= 0) {
    return x;
  } else {
    return -x;
  } if
} abs (function)
```

- statements language uses expressions
- modules language uses expressions and statements languages



Language Extensibility

- state machines language extends expressions language



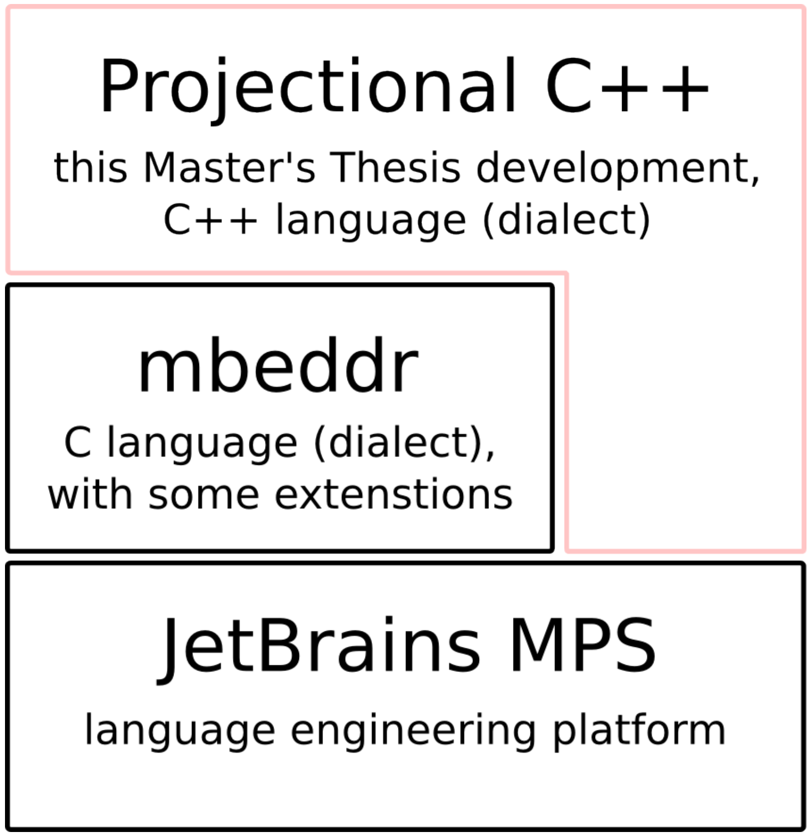
```
enum mode { MANUAL; AUTO; FAIL; }

mode nextMode(mode mode, int8_t speed) {
  return mode, FAIL |            | mode == MANUAL | mode == AUTO | ;
                    | speed < 30 | MANUAL         | AUTO         |
                    | speed > 30 | MANUAL         | MANUAL       |
} nextMode (function)
```

Views on a Language

- A language is defined in MPS in views on it:
  - **Structure** view - meta-model structure
  - **Behavior** view - methods, like in a Java class
  - **Editor** view - the way to input and edit a model
  - **Constraints** view - context sensitive limitations
  - **Type system** view - for typed languages
  - **Generators** view - used to lower abstraction level
  - **Analyses** view - for warnings and errors, informing
  - **TextGen** view - to generate to text
  - **Intentions** view - provide user-callable automations

Approach

- Describing a new language in JetBrains MPS through *views* on it,
- with the use of *language modularity* and
- *language extensibility*,
- to add C++ to mbeddr C language.


Projectional C++
this Master's Thesis development, C++ language (dialect)
mbeddr
C language (dialect), with some extenstions
JetBrains MPS
language engineering platform

Contribution

Challenges

C1: Is it in general possible to extend mbeddr C to C++? *Will mbeddr be flexible enough?*

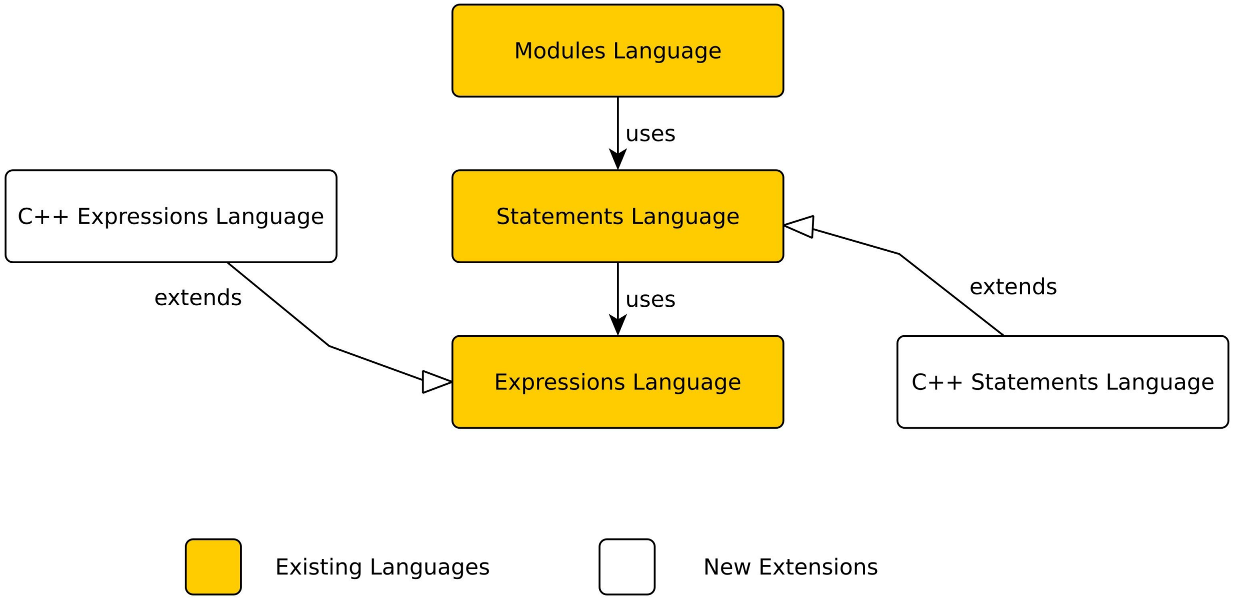
C2: Is it possible to make a “safer” C++ dialect? *Like mbeddr C is a safer C dialect.*

C3: Templates in C++ bear pure textual nature! *A contradiction with the projectional approach.*

*C1 - C3 are Challenges 1 - 3.

C1: Extending C to C++

- Practically proven to be possible
  - One-side-awareness challenge: mbeddr should not be aware of Projectional C++



C2: Better dialect of C++?

- **Dropping language features**
  - C++ is valuable with the standard library (STL), but
  - STL requires *all* C++ language features, thus
  - dropping “unsafe” language features is not the way!
- **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 and overrides have no syntax in C++, added:

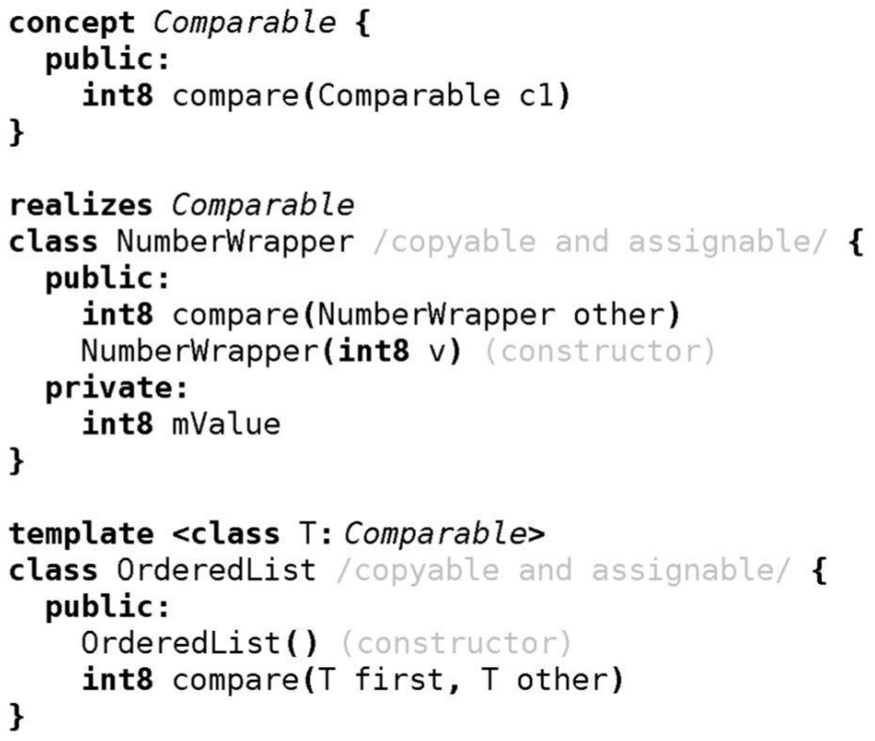
```
abstract class Widget /copyable and assignable/ {
  public:
    explicit Widget(Widget* parent) (constructor)
    pure virtual Size getDimensions()  = 0
}

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()
}
```

C3: Templates?

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

  but

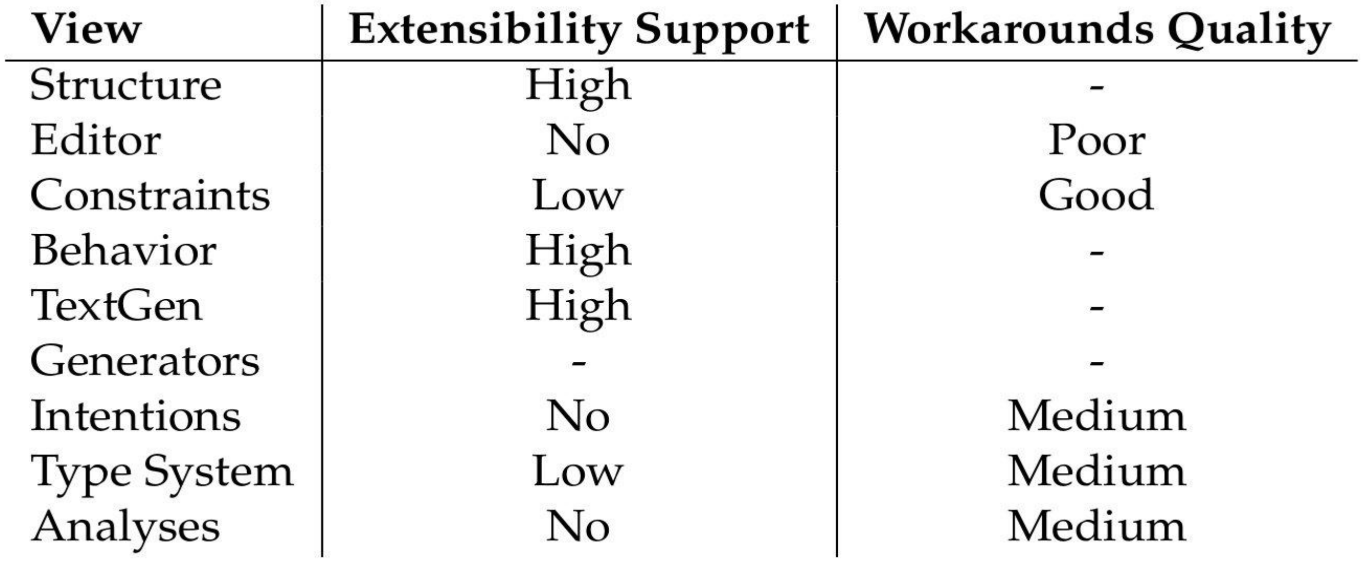
  - absent in C++
  - special importer needed
  - additional user work
  - code duplication



```
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)
}
```

Lessons Learned

Meta-Model 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

- Meta-model extensibility depends on meta-meta-model (MPS) design
- MPS can provide better support for extensibility

Making a Language Safer

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 - preventive and informative

Language Tooling

- Analyses were found to be useful, however
  - MPS does not support them explicitly!
  - Computational complexity can be very high!

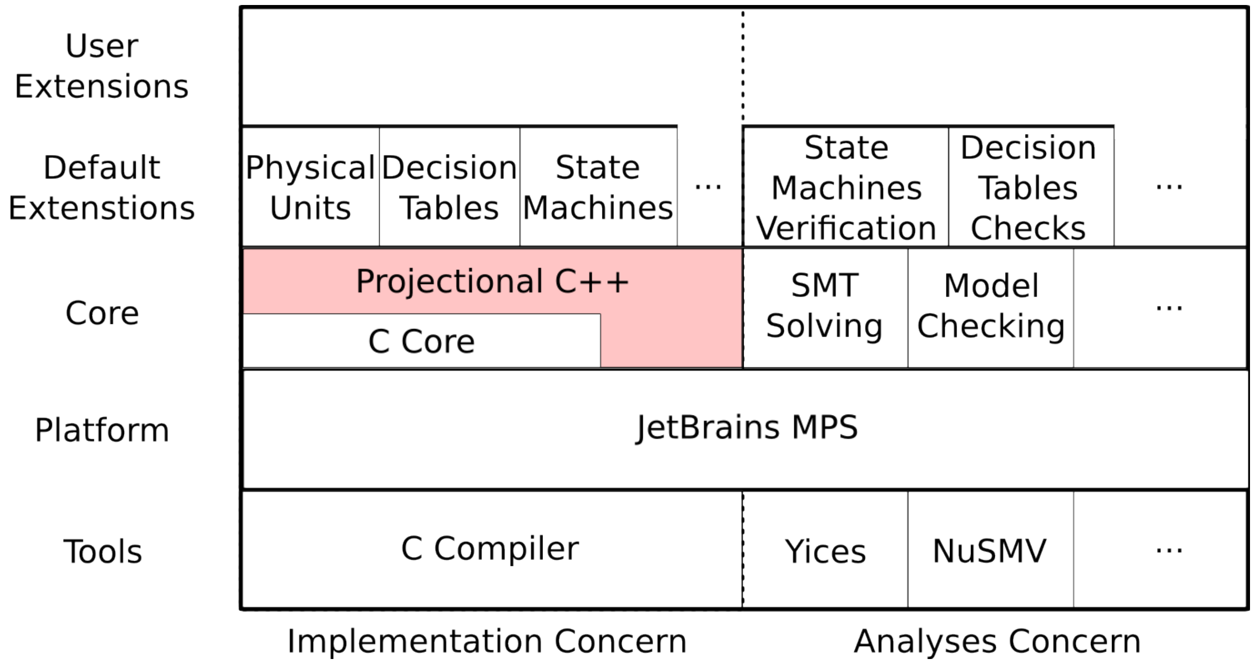
- Propositions for MPS evolution - APIs for analyses:
  - When does an analysis start?
  - Which scope does it have?
  - Is result caching needed?
  - Prioritisation, concurrency limitations?
  - Informing the user - can be improved and
  - Common solutions offered for reuse

Future Work

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

Thank you!

Thank you for attention! Questions now!



Zaur Molotnikov, zaur@zaurmolotnikov.com