zühlke
empowering ideas

Programmers rejoice: QML makes business people understand

About me

My company zühlke empowering ideas

What I do at work

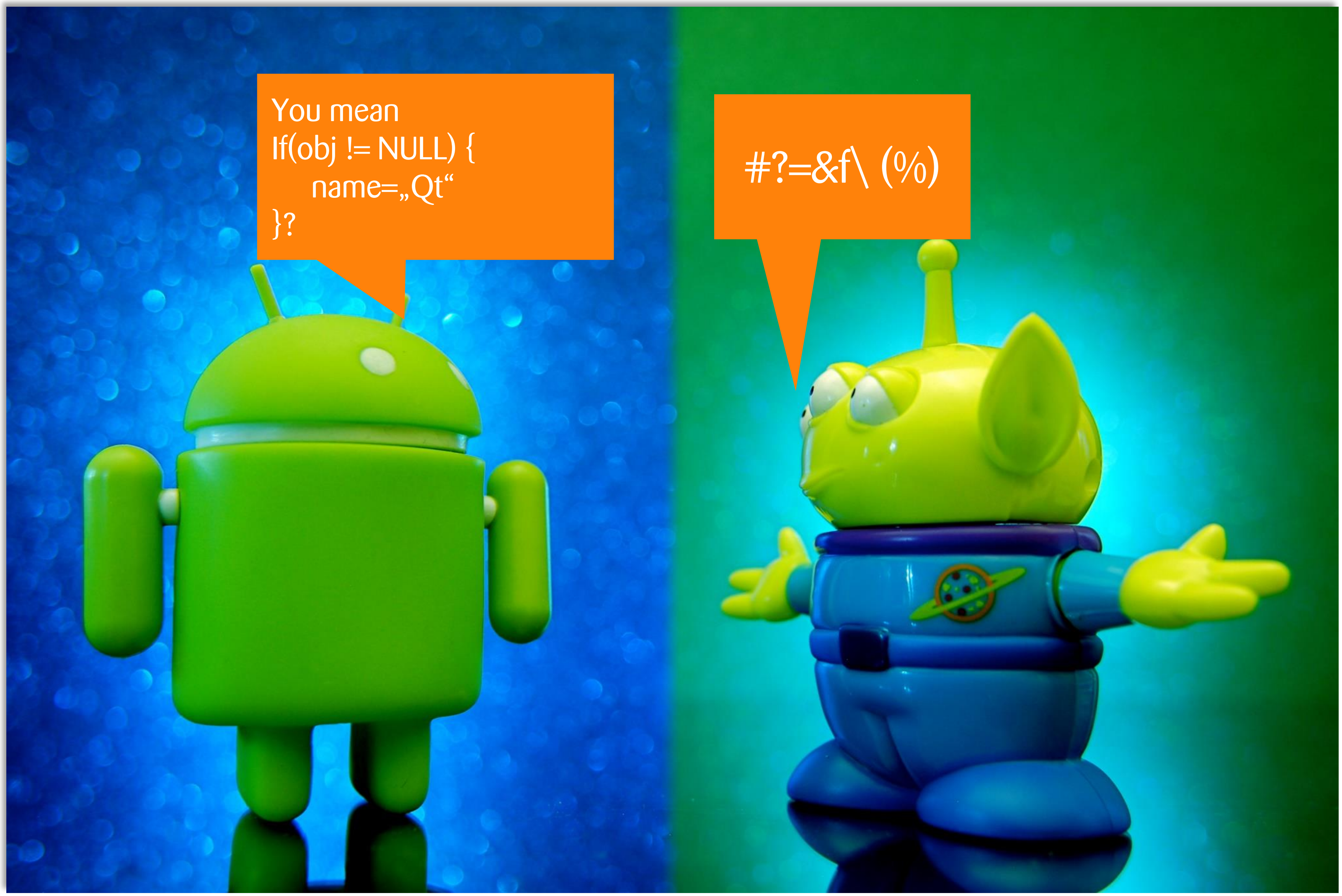
Where I live

What is it all about?

Agenda

- Motivation
- A real life example
- The hurdles of DSL creation
- What Qt has to offer
- PresentationSystem demo
- Extending Qml‘s type system
- When to use Qml for DSL creation and when not

A common problem in software development

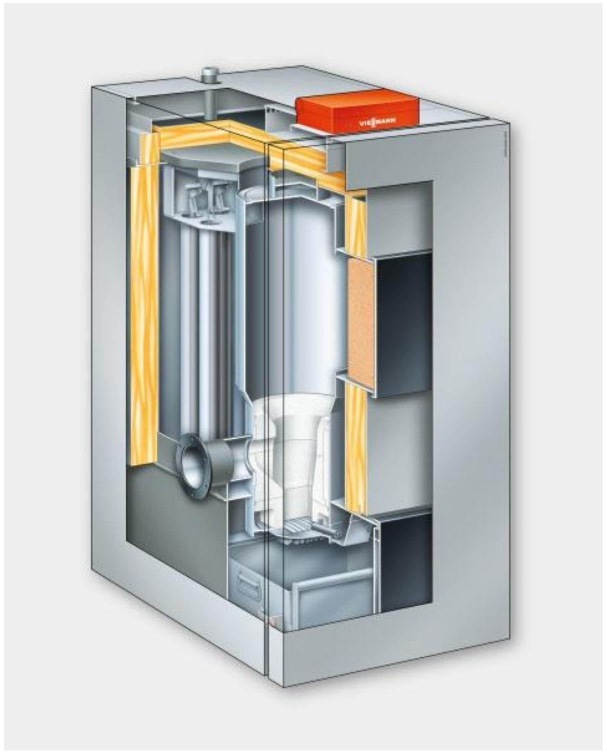


What if…

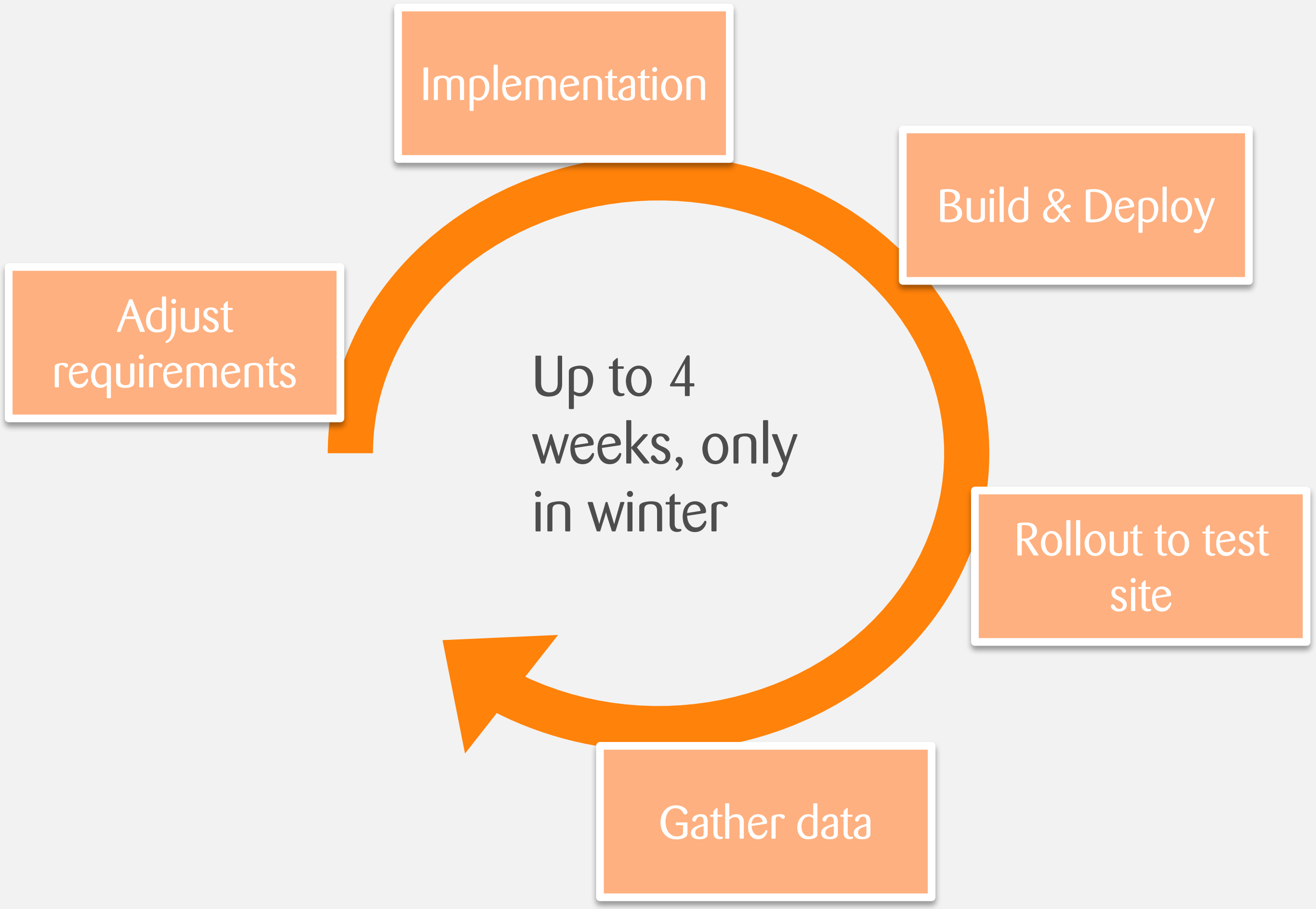
- Stakeholders and developers could share a common language?
- changes to the software could be made in that language, too?

A real life example

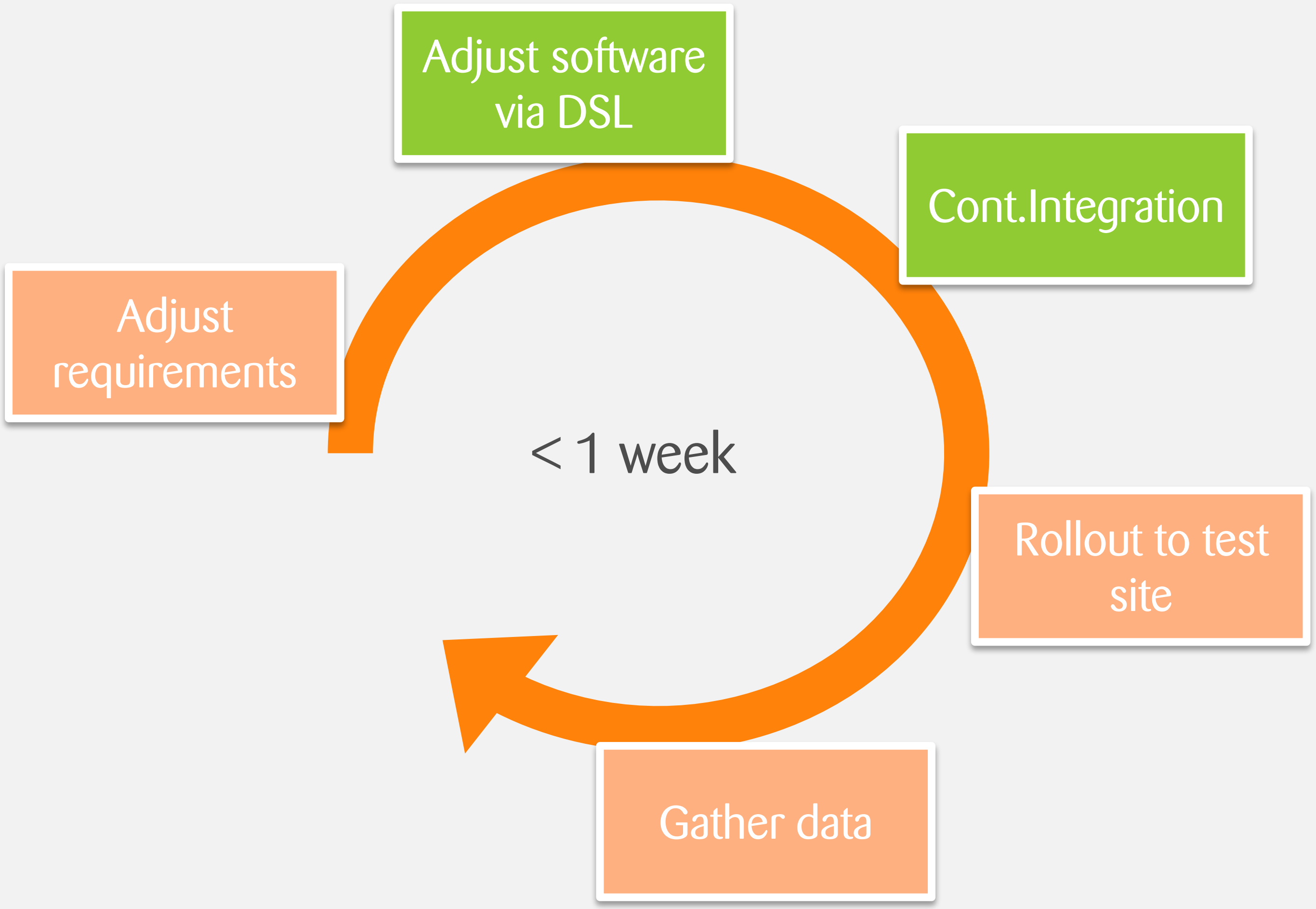
zühlke
empowering ideas



The heating system example. Before.



The heating system example. After.



Why...

…isn‘t there a DSL in every software project?

DSLs are hard to develop

Formal definition

- Grammar
- Schema
- Metamodel
- Abstract Syntax

Tooling

- Parser
- Validator
- Compiler
- Editor / IDE

DSL creation with XText

XText is a powerful, DSL-based language development framework
Steps to create a DSL using Xtext:

- Textual definition of syntax and semantic model for your DSL. Language: Grammar
- Configure generator that creates java packages and editor for your DSL. Language: MWE2
- Set up Maven builds for CI of your DSL
- Update Eclipse
- Use your DSL
- Not targeting Java? → Use Xtend for code generation

Programming languages compared

Comes with an extensible type system!

QML, WPF, Silverlight	Object oriented, declarative, special purpose, Model driven
C++, Java, C#	Object oriented, imperative,general purpose
C, COBOL	Procedure oriented, imperative
Assembler Code	Processor instruction set, no abstraction at all

DSL creation with Qml – what Qt has to offer

- Custom types can be made available in Qml
- Types used are backed by C++ classes
- CodeCompletion for custom types immediately available in Creator
- Easy to learn
- Effort needed to create a (simple) DSL is low
- No other programming languages needed!

Extending the Qml type system

- Any custom types can be registered with QML‘s type system (but must inherit from QObject)
- Different forms of registration are available to define the runtime behaviour of your types:
  - Creatable types
  - Uncreatable types
  - Interfaces
  - Singletons

The QmlPresentationSystem

- is a DSL for creating slide decks
- makes use of QML‘s extensible type system
- is easy to use (compared to programming a slide deck with C++ or any other general purpose language)

How to use your own custom types in Qml

1. Derive from QObject

```
Public class YourType : public Qobject {…}
```

2. Define Properties

```
Q_PROPERTY(DataElementIds::EnDataElementIds identifier
            READ getIdentifier MEMBER m_id)
```

3. Register type in Qml

```
qmlRegisterType<YourType>("com.your.namespace", 1, 0, „QmlTypeName");
```

How to use your own custom types in Qml

5. Run qmake

4. Import custom namespace in Qml file

```
import com.your.namespace 1.0
```

6. Ready to use custom type in Qml

```
YourType{
            displayName: „YourInstanceName"
            value: false
            identifier: YourId
}
```

Using singleton types

QObject and QJSValue types can be registered as singleton types.

Registering types that are defined in C++:

```
int qmlRegisterSingletonType(const char * uri, int versionMajor, int versionMinor,
const char * typeName, QJSValue(* ) ( QQmlEngine *, QJSEngine * ) callback)
```

```
int qmlRegisterSingletonType(const char * uri, int versionMajor, int versionMinor,
const char * typeName, QObject *(* ) ( QQmlEngine *, QJSEngine * ) callback)
```

Using singleton types

Registering types that are defined in Qml:

```
int qmlRegisterSingletonType(const QUrl & url,
const char * uri, int versionMajor, intversionMinor, const char * qmlName)
```

Using uncreatable types

int qmlRegisterUncreatableType(const char * *uri*, int *versionMajor*, int *versionMinor*, const char * *qmlName*, const QString & *message*)

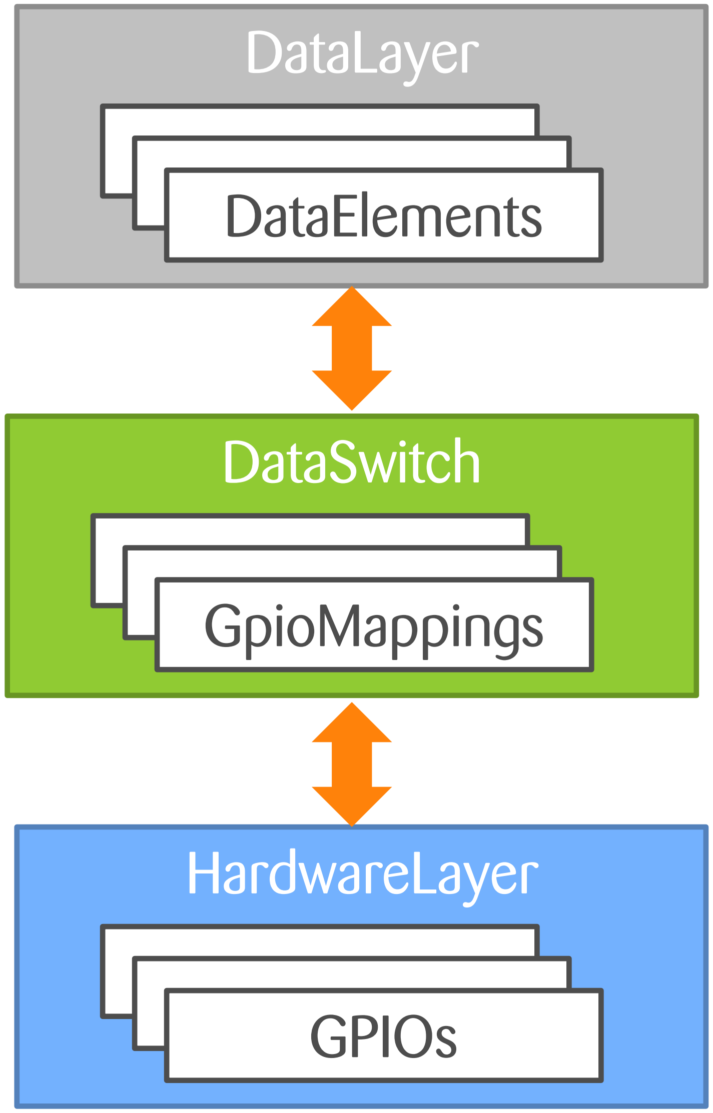
Note:
To use enums in Qml, they must be wrapped in a class.

Drawbacks you have to deal with

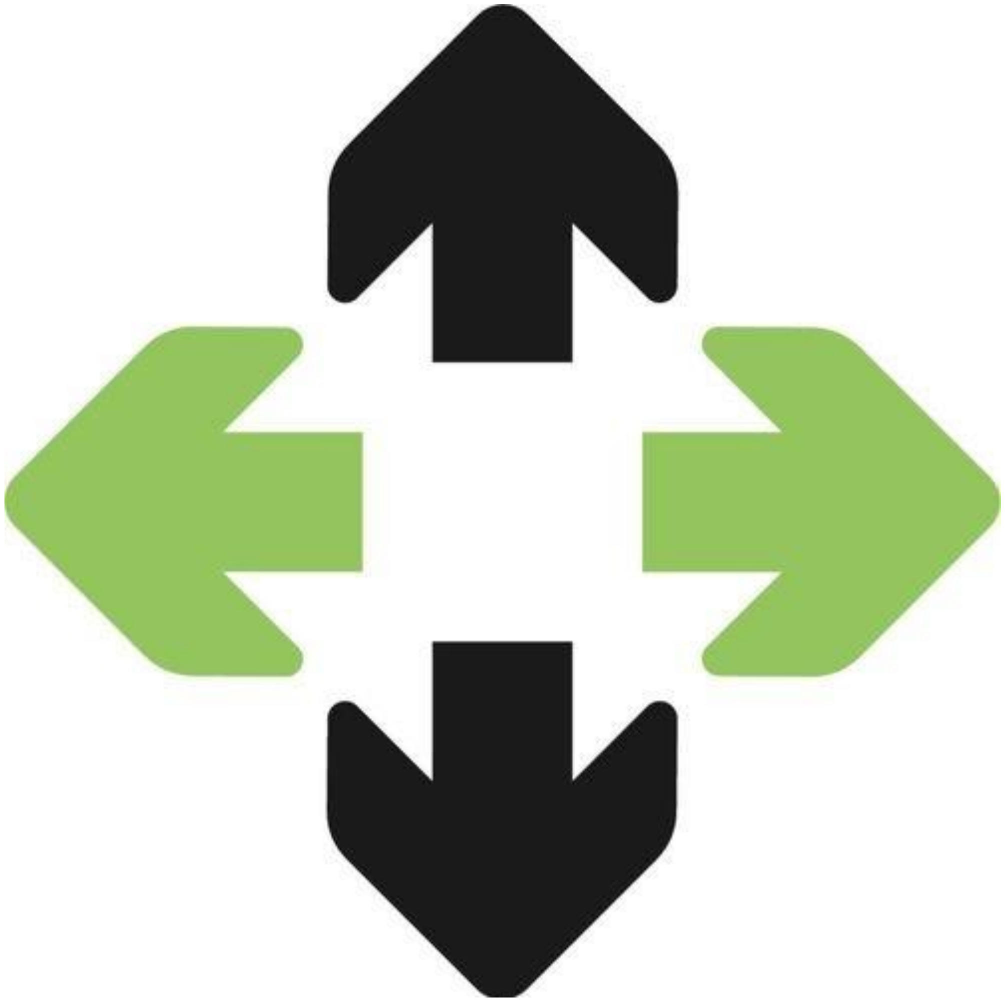
- Mingling with QtQuick types and QObject properties
- No integrated code generator for creation of non-Qt-code
- Editor is not always as smart as it could be
- Fixed syntax

See how it works

The sample application



When is a Qml based DSL the right choice?



What is a domain?

Stakeholders‘ domains	Your domains
Heating system	System composition
Drive control	StateMachines
Accounting	UIs
Routing	Data layer
…	…

Simply said: Everything is a domain

Think about a QML based DSL, when

- other tools would require too much effort
- simple DSL features are needed
- no code generation is required
- the DSL will mainly be used to define static aspects

Think about using other tools

- when you want your own syntax / semantics
- code generation for different languages is required
- when you want to have a clean DSL (without artifacts from QObject)

Q&A / Discussion