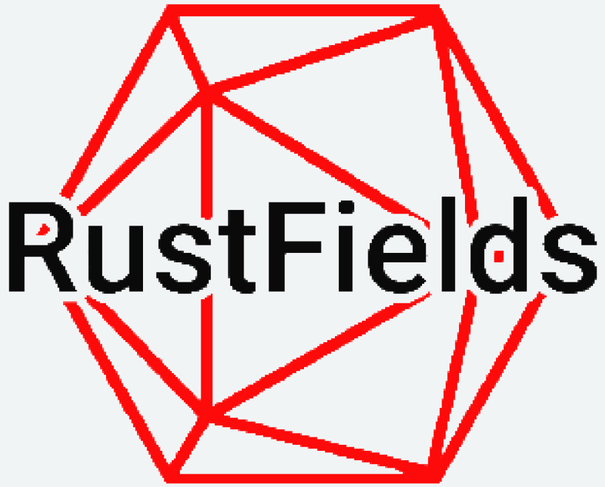


Angela Cortecchia – Leonardo Micelli – Paolo Penazzi – Filippo Vissani

Laboratorio di Sistemi Software – @UniBo

Introduction & Goals

This project was born from the willingness to extend ScaFi: a Scala-based library and framework for Aggregate Programming.

The goal of this project is to explore different solutions to make the field calculus available on thin devices.

Examples: ScaFi-core

```
class Example extends AggregateProgram:
  override type MainResult = Int
  new *
  override def main() =
    foldhood( init = 1)(_+_)( expr = 2)
```

Examples: RuFi-core

```
fn main() {
    let program : fn(RoundVM) → (...) =
        |vm: RoundVM|
            foldhood(
                vm,
                init: || 1,
                aggr: |a : i32, b : i32| a + b,
                expr: |vm1 : RoundVM| (vm1, 2));
    let (_result, _vm) = round( vm: RoundVM::new( context: init_context()), program);
}
```

Examples: ScaFi-fields

```
class Example extends FieldLanguageProgram:
  override type MainResult = Int
  new *
  override def main() =
    val f1 = nbrf(mid())
    val f2 = Field.lift(2)
    val f3 = for
      id <- f1
      n <- f2
    yield id + n
    foldhoodf[Int](_+_)(f3)
```

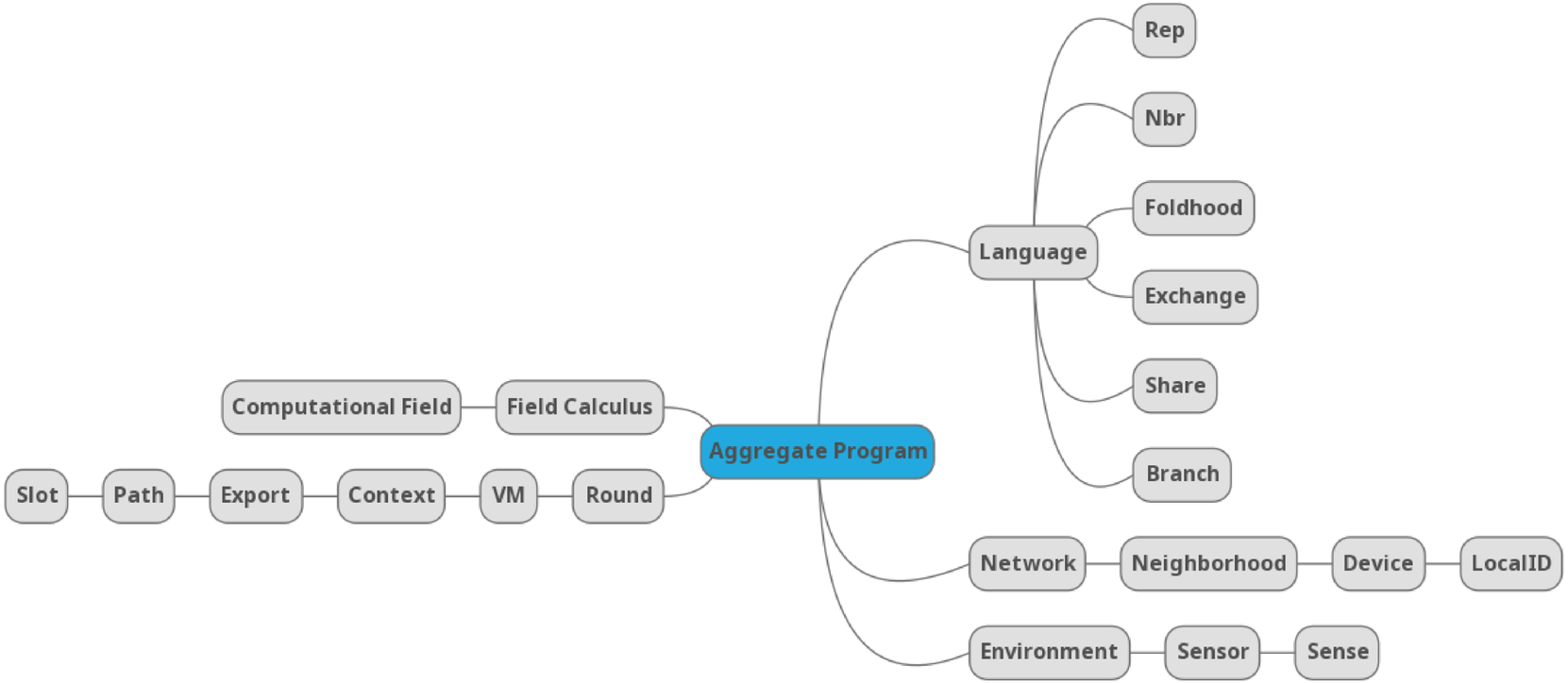
Development Process

- Agile Methodology & Domain-Driven Design
- Series of knowledge-crunching sessions to kickstart the design
- Split implementation into weekly sprints
- The team agreed on working with a semplified version of the Git Flow

Knowledge Crunching

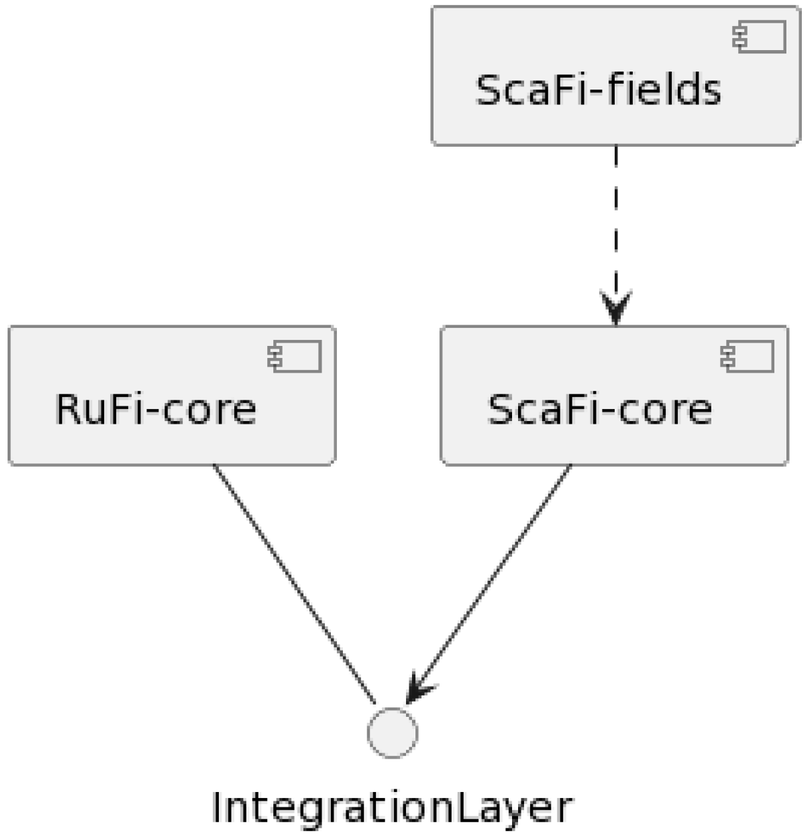
- The first knowledge-crunching sessions were about the main project goals
- Then we focused on highlighting the fundamental concepts of the domain
- We ended up with a ubiquitous language for the domain

Ubiquitous Language



Design

The overall architecture of the project is the following:

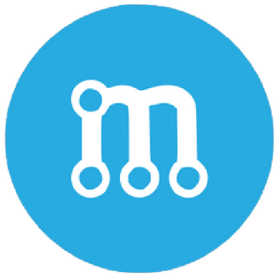


DevOps: Repository Management

- We created a GitHub organization: RustFields
- Each project has its repository
- Before merging code on the main/master branch, feature branches need to be approved by other team members using the Pull Request service
- To favor a linear history we opted to use the rebase method for merging feature branches

DevOps: Repository Management

We used bots like **Renovate**, **Mergify** and **Semantic Release**
to ease the repository management



DevOps: Build Automation

We used the standard build systems for each projects:

- **Cargo** for the Rust project
- **Sbt** for the Scala projects
- **Gradle** for the others projects

sbt

DevOps: Continuous Integration

- The team decided to use the GitHub Actions service for the CI/CD
- All projects have a common workflow called dispatcher, which is used as a filter to disable the branch builds for some branches
- Each project has its specific workflow based on the build systems, languages and targets platform

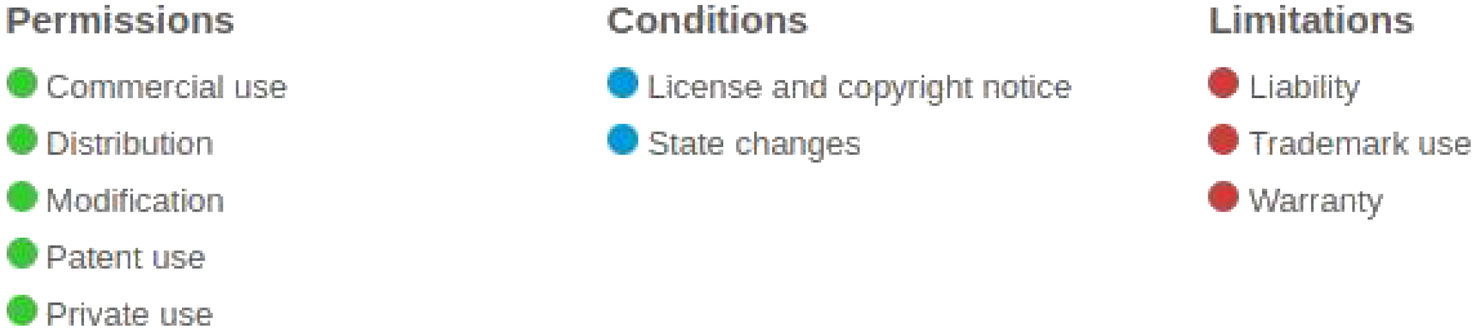
All workflows have the following common jobs after the dispatcher:

DevOps: Versioning and Releasing

- The team adopted the conventional-commit convention
- To ensure that the commits made are conformant to the required pattern, the team used Git Hooks
- Thanks to the use of Semantic Release it was possible to automate all the versioning and releasing work

License

Every repository in the organization is endowed with the Apache License 2.0.



Future Work

- Complete the integration layer between Scala and Rust
- Find a way to standardize the messages exchanged by devices in the network
- Explore a solution that makes use of Exchange or Share instead of the current constructs
- Expand the test suite of each project
- Expand the Rust core

Thanks for the attention!