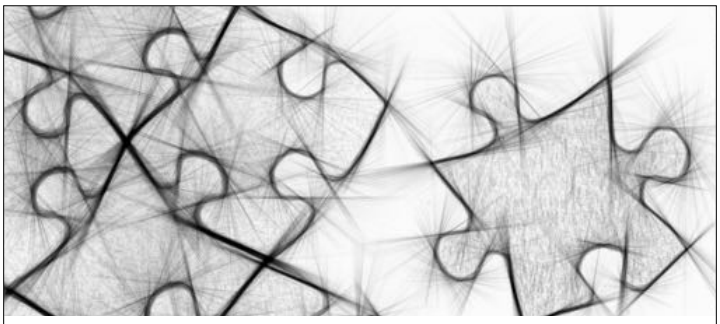


Towards Scalable Software Composition

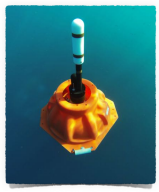
Sébastien Mosser
Research seminar, McMaster University
15/03/2021

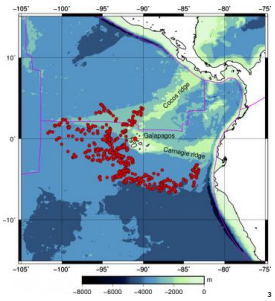
Sébastien Mosser

***Snowboarding** since 1995. **Composing** things since 2007.*

- 19-...: Associate Professor, UQAM
- 12-18: "Maître de Conférences", Univ. Côte d'Azur
- 11-12: Research Scientist, SINTEF
- 10-11: Postdoc, Inria Lille Nord-Europe
- 07-10: PhD student, Université de Nice

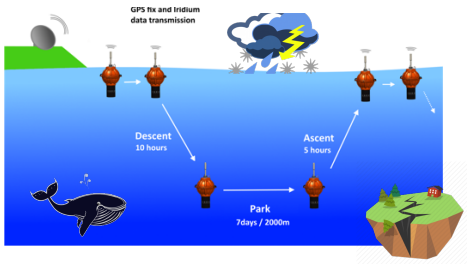
Example of Industrial Collaboration: MERMAID

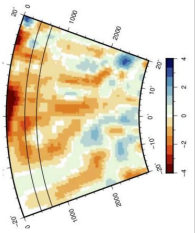




- Project **started in 2002** by Guust Nolet
- **Acoustic passive monitoring** of oceans
- Now **3rd generation** buoys deployed

Running Software, 20,000 leagues under the seas





Ok, but where is the composition?

Running Software, 20,000 leagues under the seas

"Can you build an AppStore for our MERMAIDs?"
- G. Nolet, 2016

- MERMAIDs are ***expensive***, and often ***idle***
  - *Oceans are full of challenges (80% unexplored)*
    - *Monitoring whales, plastic pollution, salinity, ..*
  - ***Compose** "data collection campaigns"*
- **A MERMAID is not a smartphone**
  - *Legacy ad hoc code (no operating system)*
  - *Hostile environment (salt, pressure), energy, ..*

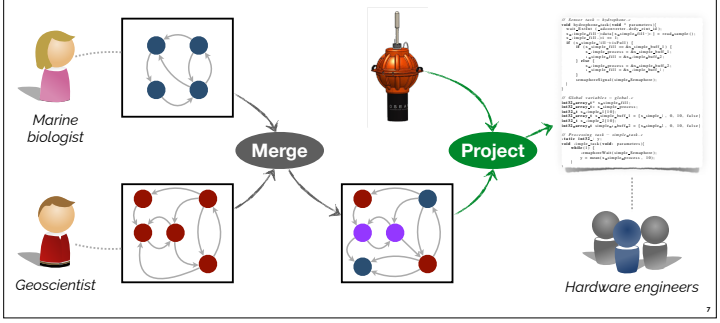
~~Running Software~~ Composing, 20,000 leagues under the seas

- ***So "composition" is all about hacking a buoy?***

- Scientific challenges:
  - **Conflict detection** among **requirements** expressed by **scientists**
    - **Static analysis** of the applications ***(what-if scenarios)***
  - **Modelling constraints** related to the "**hostile**" environment
    - **Empirical benchmark** and **simulation** for energy consumption

All these research challenges are related to composition!

Software Composition by example



Software Composition in the context of MERMAIDs

1. Model the MERMAID lifecycle & campaigns
   - ***Compose*** multiple campaigns together (*merge*, ≡)
   - ***Compose*** campaigns with the lifecycle (*project*)
   - **Open-source** domain-specific language for geoscientists (*MeLa*)
2. Simulate applications to validate the **composition**
   - **Benchmark** energy consumption
   - **Trade-off analysis:** ***Whales, Earthquakes, both?***

S. Bonnieux's PhD 2017-2020

[Sensors 2020] [Oceans 2019]

Funding: OSEAN-PACA + CNRS, ~140K$

Lessons learned: Software Composition matters! (2010-...)

- ***Divide*** to **conquer**, but ***compose*** to **vanquish**.
- Like *Mr Jourdain*, **you're composing without knowing it**:
  - ***Modularizing*** *code (e.g., packages, functions)*
  - ***Configuring*** *the Linux kernel*
  - ***Weaving*** *persistance into a Spring application*
  - ***Pulling*** *code from a Git repository*
  - ***Deploying*** & ***invoking*** *micro-services*
  - ...

Research challenge: Theoretical foundations + Trade-offs

(unsolved) Research Challenge

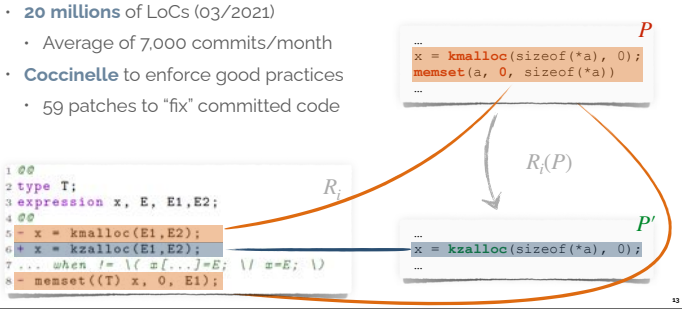
Can we make Software Composition Scalable?

Code rewriting at large-scale

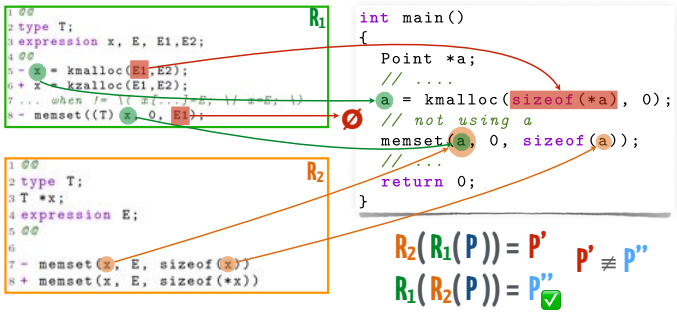
- Code rewriting **in a nutshell**:
  - $Rule : Program \rightarrow Program$
  - With ρ a rule and P a program, $P' = \rho(P)$ the rewritten one
- Code rewriting **from the trenches**:
  - **Image layering** for containers (Docker)
  - **Code quality** in the Linux Kernel (Coccinelle)
  - **Automated bug fix** in Android (Spoon)
- **Rules are composable** by definition:
  - $\rho_{12}(P) = (\rho_1 \bullet \rho_2)(P) = \rho_1(\rho_2(P))$

Rewriting the Linux Kernel

- **20 millions** of LoCs (03/2021)
  - Average of 7,000 commits/month
- **Coccinelle** to enforce good practices
  - 59 patches to "fix" committed code



Example of rule composition conflict



Scalability: Composing 59 rules!

- **How to identify conflicting rules?**
  - **Ordering** challenge
    - ***59!*** (10^{80}) ***combinations*** *to handle*
  - **Performance** challenge
    - *Patches are* ***time consuming*** *(~190 minutes)*
  - **Maintenance/safety** challenge
    - *How to* ***reduce the search space****?*
  - **Engineering** challenge
    - *Linux and Coccinelle are* ***non-negotiable***

https://github.com/torvalds/linux/tree/master/scripts/coccinelle

Abstracting rules composition

- Looking for **fixed points** at this scale is "***out of the question***"
- **Trade-off**: Let $\rho = (\varphi, \chi) \in (\Phi \times X) = P$, where
  - $\varphi : Program \rightarrow Program$ is an idempotent rewriting function
  - $\chi : Program \times Program \rightarrow \mathbb{B}$ is a postcondition checker ✅
  - **Assumption**: we can manipulate programs
  - $\ominus : Program \times Program \rightarrow \Delta$ (e.g., the `diff` command)
  - $\oplus : Program \times \Delta \rightarrow Program$ (e.g., the `patch` command)
  - $\odot : \Delta \times \Delta \rightarrow \Delta$ (e.g., a patch concatenation algorithm)

Modelling rules composition operators

- **Extending** the state-of-practice with **isolated composition**
  - Let p a program, $(\varphi_1, \varphi_2) \in \Phi^2$ two rewriting functions
    - $p_1 = \varphi_1(p), \Delta_1 = p_1 \ominus p,$ and $p_2 = \varphi_2(p), \Delta_2 = p_2 \ominus p$
    - $p' = p \oplus (\Delta_1 \odot \Delta_2)$
- **Generalizing** & Including **postconditions verification**

$$iso : Program \times P^n \rightarrow Program$$
$$p, \{\rho_1, \ldots, \rho_n\} \mapsto p_{iso} = p \oplus (\odot_{i=1}^{n} (\varphi_i(p) \ominus p)), \quad \wedge_{i=1}^{n} \chi_i(p, p_{iso})$$

B. Benni's PhD 2016-2019

[JSEP 2019]

Empirical validation

- **Dataset** considered for evaluation:
  - **Linux**: Randomly sampled **19 versions** (1/month)
  - **Coccinelle**: rules triggered when running "`make coccicheck`" (35)
- **Results**:
  - **Ordering**: Discovered **2 conflicting rules**
  - **Performance**: **Quicker** than sequential
  - **Maintenance**: ***2!*** instead of ***35!*** (10^{40}) to check
  - **Engineering**: **iso-functional** tooling

2 conflicts, 36% cases

Commit.id	Rewriter #1	Rewriter #2
38651683aa98	alloc_cast	memdup
4efe37f4c4ef	alloc_cast	memdup
b134bd90286d	alloc_cast	memdup
25a3ba610609	alloc_cast	memdup
bce1a65172d1	alloc_cast	memdup
2551a53053de	alloc_cast	memdup
bfd40eaff5ab	alloc_cast	memdup

https://github.com/torvalds/linux/blob/master/Documentation/dev-tools/coccinelle.rst

Can we leverage this composition model ? (Horizontal scalability)

- **Challenge** the abstraction to other **application domains**
  - ***Energy-efficiency*** *for smartphone applications*
  - ***Container-based deployment*** *of service architectures*

android

- **Scalability** dimensions: **Reuse & Ensure**
  - **Empirical** validation (22 android app, 11k images)
  - **Android**: identify ***overlapping fixes***
  - **Docker**: detect ***faulty containers*** *(ignored elsewhere)*

docker

[JSEP 2019] [ICSR 2018] [Amadeus Industrial GlobalTech Keynote 2018]

Lessons learned: Scalability matters (2018-2020)

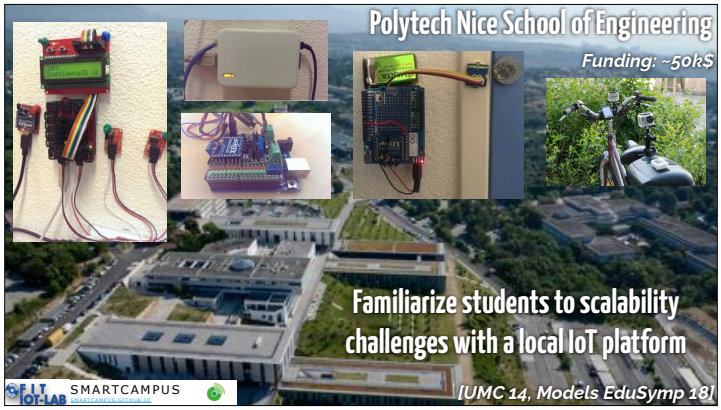
- Software engineering is **all about trade-offs**
  - From **optimal** to **usable** **(but trade-offs ≠ hacking/mundane)**
  - **Scalability as an afterthought is a mistake**

[JOT 2020]
[JSEP 2019]
[ICSR 2018]
[SAC 2018]

- **Research** on **composition models**:
  - Need to address **scalability issues**, at ≠ **dimensions**
    - *e.g., performance, maintenance, quality assurance, and reusability*
  - Will make the paradigm **usable at large-scale**
  - Requires **theoretical foundations** & **engineering** skill

Invited lecturer ENS Lyon (2018, 2019)

Funding*: NSERC Discovery (145k$),* CNRS (~20k$), UCA (~150k$)



Scalability: Formal properties do matter!

- **Challenge**: focus on **usage by developers**
  - *How to **develop** operators **faster** and **safer**?*
  - *How to **improve support** for operator's users?*

- **Modelling properties** associated to operators
  - ***Equivalent elements:*** *idempotency*
  - ***Operator folding:*** *associativity*
  - ***Avoid useless calls:*** *regularity, identity and absorption*
  - ***Order independence:*** *commutativity*
  - ***Composing composition operators:*** *distributivity*

ace

No silver-bullet!

Funding*: NSERC Discovery (145k$), 2020-2025*

Application domain: Source code management

Inria

- Team effort to **bind together Compilers & Software Engineering**
  - "***Équipe associée***" (start: 50k$) with ENS Lyon & Inria

UNIVERSITÉ DE LYON

- Explore **two dimensions of composition:**
  - **Charting** a large-scale **compiler infrastructure** (LLVM, ~10MLoC, Clang)
  - **Improving source code merging** at the abstract level (56M+ GitHub users)

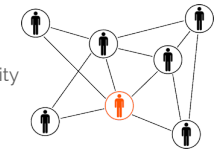
- **Skills**: *graph theory (Reinharz), compilers (Gonnord, Privat), and soft. eng. (Mosser)*

Ambition: Make `git-merge` & LLVM great again!

Application domain: Micro-services architectures

- **Micro-services** are a *de facto* **architectural style** in 2021
  - ***Starting a show on Netflix composes up to 700 services***

- Composition-related challenges: **No silver-bullet, no afterthoughts**
  - **Agile requirements** composition for **traceability**
    - *Natural Language Processing (Meurs), Certification (Polacsek, Blay)*
  - **Large-scale** deployment & **configuration** variability
    - *Polyglot development & Software product lines (Klewerton, Kruger) [ICSOC, SPLC2020]*



Ambition: Composition in heterogenous systems of systems!

1 Diving into Software Composition

Composing Rewriting Rules 2 *(2018-2020)*

3 Modelling & Scaling Composition *(2020-...)*

Conclusions & Take-aways 4

Software Composition is everywhere!

- Research driven by **industrial & practical collaborations**
  - **Geosciences** *(OSEAN-PACA+CNRS, ~140k$)*
  - **Embedded pipelines** *(Visteon, ~115k$)*
  - **Cyber-physical systems** *(Datathings technological transfer, ~225k$)*
  - Software **deployment** *(UCA, ~150k$)* & **visualization** *(UNS, ~150k$)*

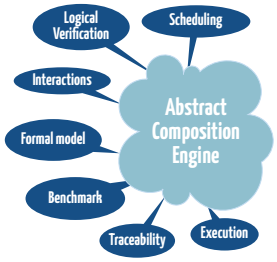
- **Research challenges (2020-...)**
  - ***Scalable*** composition models are **hard to elaborate**
  - ***Formal*** properties **impact software developers** daily life

Fundamentals of Composition Scalability

- Stop **reinventing a squared wheel!**

- **Abstract** the composition **expertise**
  - *Formal definition & tooling*
  - *Conflicts & Interactions handling*
  - *Verification & proofs*
  - *...*

- **Customize** for each **domain**



Time to consider (scalable) composition as a first class citizen

Recent composition results (2014-2020)

- **Technological transfer**
  - ***Visteon***: Pipeline composition *[ICSE 2019, SAC' 2018]*
  - ***DataThing***: time series composition (x 20) *[FGS 2019, SAC 2018]*
  - ***PulseTotem***†: Spin-off startup company *[SEAA 2014]*

- **Open-source software**
  - Incremental **exploration of software** *[SPLC 2020, ICSOC 2020]*
  - **Abstract Composition Engine** *[JOT 2020]*
  - **SmartCampus reference architecture** *[UMC 2014]*

https://ace-design.github.io/

Joining McMaster's Department of Computing and Software

- Faculty of **Engineering**
  - Strong **industrial ecosystem, multidisciplinary collaborations**
  - **Renowned** CS and SE **programs**
- Departmental **expertise**
  - **Digital & Smart Systems** research cluster
  - **Software Quality** (& theory of computation) areas of specialization
    - *E.g., "safety" of composed systems (emergent behaviours)*
  - **Teaching**: Software engineering
- **Hamilton**, Ontario
  - *Campus! Waterfalls! Great Lakes! Biking! Hiking! Canoeing!*

This work is a team effort


Jinx


Dumpling


Serge Dogny
PhD, 09/2021-...


Alexandra Lapointe
PhD, 01/2021-...


Corinne Pulgar
MSc, 09/2021-...


J.-P. Caissy
MSc, 01/2020-...


Sébastien Bonnieux
PhD (2020)


Sami Lazreg
PhD (2020)


Benjamin Benni
PhD (2019), MSc (2016)


Günter Jungbluth
MSc (2018)


Cyril Cecchinel
PhD (2017), MSc (2014)


Ivan Logre
PhD (2017), MSc (2013)


Alexandre Feugas
PhD (2014)


Eirik Brandtzæg
MSc (2011)

And all co-authors & colleagues...