Combining Flows and Rules in a Low-Code Platform for Smart Water Management

Jens Nicolay*, Bjarno Oeyen*✉, Samuel Ngugi Ndung'u*, Thierry Renaux*, Maxime Démarest**, Boud Verbeiren**, Wolfgang De Meuter*

✉: bjarno.oeyen@vub.be

*:
VUB SOFTWARE LANGUAGES LAB
Brussels, Belgium

**:
HYDRIA
Brussels, Belgium

Funded by:
innoviris .brussels we fund your future
Brussels, Belgium

IARIA

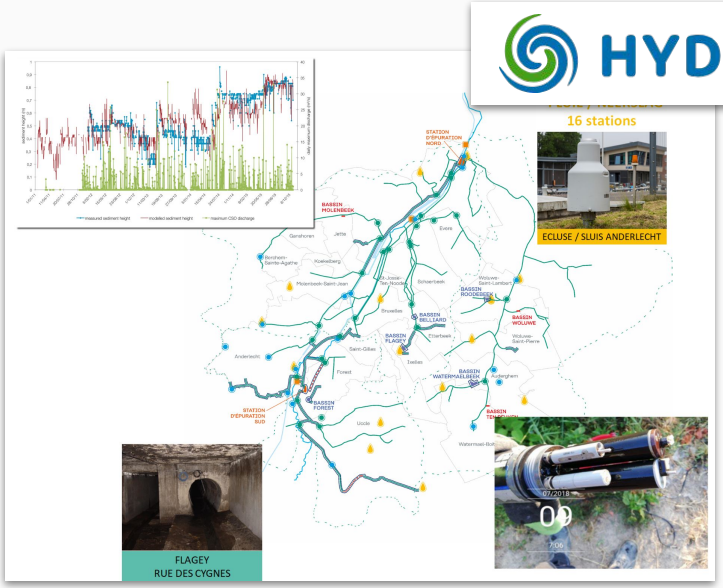
Bjarno Oeyen

PhD Student @ Vrije Universiteit Brussel, Belgium

- Reactive Programming & Reactive Streams
- Programming Language Design
- Virtual Machines

Software Languages Lab

Research lab active in the **design**, **implementation** and **application** of better **languages** to support the software engineering process.



Hydria

Hydria ensures the **public sanitation** of urban waste water in the Brussels-Capital Region, the **collection** of waste water and the **regulation** of its flow in the collectors, and **monitors** rainfall and runoff in collectors and streams.

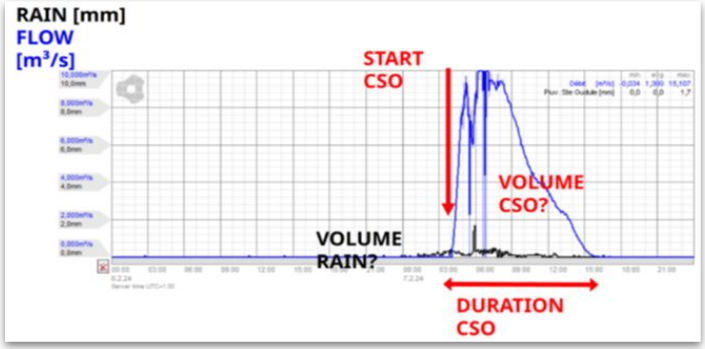
HYDRIA = Domain Experts

Environmental Impact

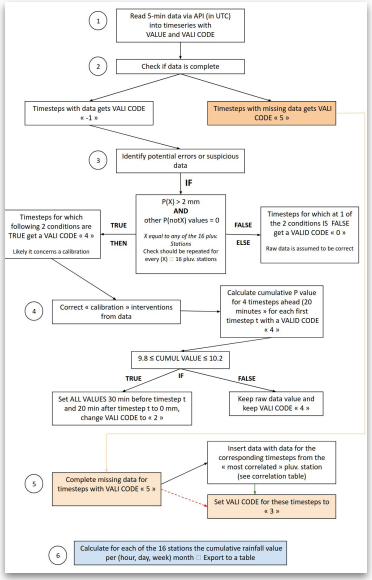
Costly Preventive and Curative Maintenance

Costly Semi-manual Operation

Combined Sewer Overflow



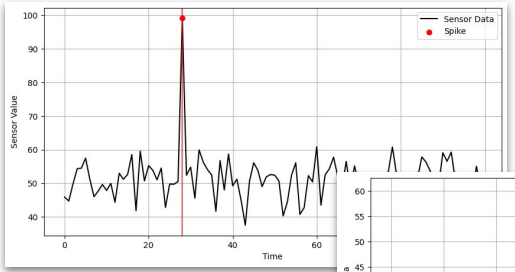
UC1: Pre-Validation of Rainfall Measurements



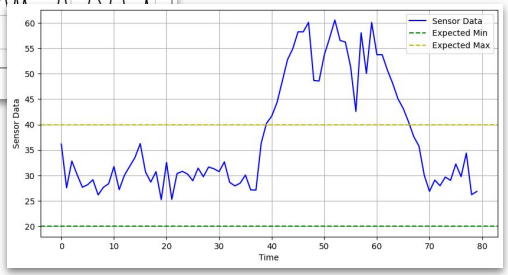
Intricate logic to detect **manual** calibration events

Pre-processing used to be performed “by hand” with

UC2: Real-Time Monitoring of Surface Water Qualities



Spikes



Threshold Violation

Send out an alert (e.g., by ✉️) for *unexpected* sensor data

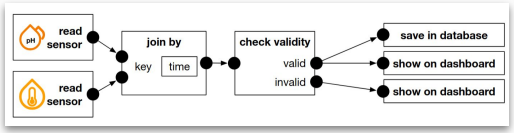
Spreadsheets are one of the most widely used tools for low-code programming [*]

= “low code”

= tedious
= error-prone
= not performed on live data

Can we do better with *real* code?

[*] M. Burnett, C. Cook, and G. Rothermel, “End-user software engineering,” Commun. ACM, vol. 47, no. 9, pp. 53–58, Sep. 2004.



Flow-Based Programming

- Easy to understand (visual)
- Often used for data-processing pipelines

Rule-Based Programming

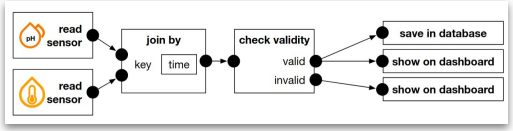
- Better at expression *correlation* between events
- Based on *declarative* if-then rules

```
rainfallAtStationOtherThan(T, S_ID) :=
  rainfall(T, MM, S_ID),
  rainfall(T, MM_OTHER, S_ID_OTHER),
  (S_ID ≠ S_ID_OTHER),
  (MM_OTHER ≠ 0).

suspiciousRainfall(T, MM, S_ID) :=
  rainfall(T, MM, S_ID),
  (MM > 2),
  not rainfallAtStationOtherThan(T, S_ID).

unsuspiciousRainfall(T, MM, S_ID) :=
  rainfall(T, MM, S_ID),
  (MM > 2),
  not suspiciousRainfall(T, MM, S_ID).

unsuspiciousRainfall(T, MM, S_ID) :=
  rainfall(T, MM, S_ID),
  (MM ⩽ 2).
```

Datalog

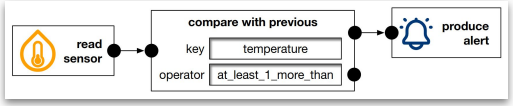
Flow-Based Programming

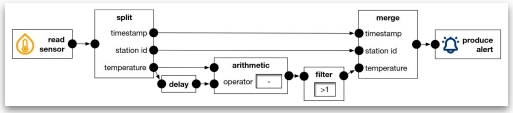
- **Poor Abstraction of Sub-tasks**



Complexity of a “box” varies
High-level and low-level logic within the same visual language

- **Poor Visualisation of Correlation**





Either “correlation” is built-in into a component, or a special operator. But with which semantics?

Rule-Based Programming

```
rainfallAtStationOtherThan(T, S_ID) :=
  rainfall(T, MM, S_ID),
  rainfall(T, MM_OTHER, S_ID_OTHER),
  (S_ID ≠ S_ID_OTHER),
  (MM_OTHER ≠ 0).

suspiciousRainfall(T, MM, S_ID) :=
  rainfall(T, MM, S_ID),
  (MM > 2),
  not rainfallAtStationOtherThan(T, S_ID).

unsuspiciousRainfall(T, MM, S_ID) :=
  rainfall(T, MM, S_ID),
  (MM > 2),
  not suspiciousRainfall(T, MM, S_ID).

unsuspiciousRainfall(T, MM, S_ID) :=
  rainfall(T, MM, S_ID),
  (MM ⩽ 2).
```

Datalog

- **Complexity of State Management**
  Running out-of-memory by the continuous aggregation of facts in the fact base
- **Poor Fit for Imperative Actions**
  Not declarative, break the rule-based model
- **Poor Mobility of Rules**
  Most systems employ a single (shared) fact base

Flow-Based Programming

+

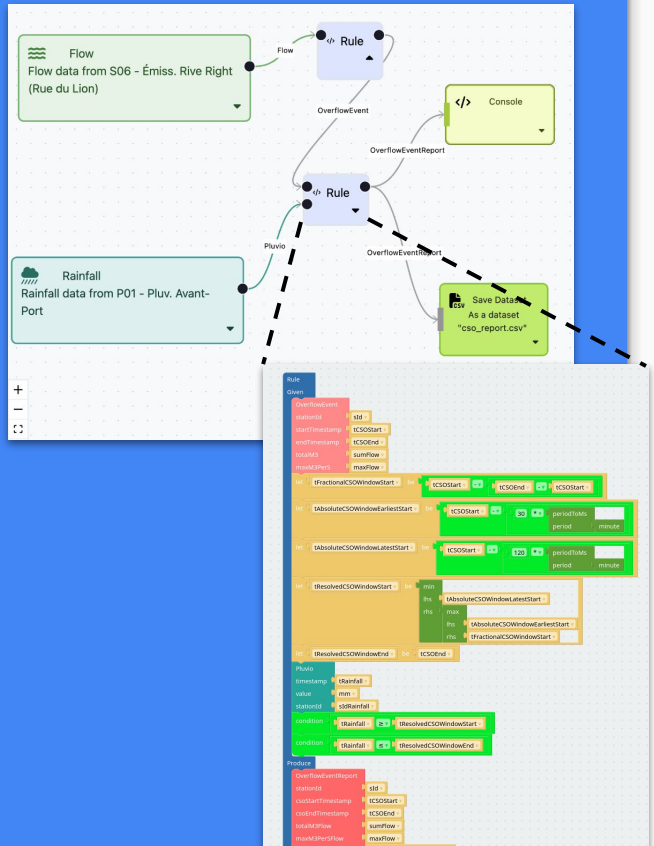
Rule-Based Programming

+

Visual Programming Language

SWAMP

Smart **WA**ter **M**anagement **P**latform

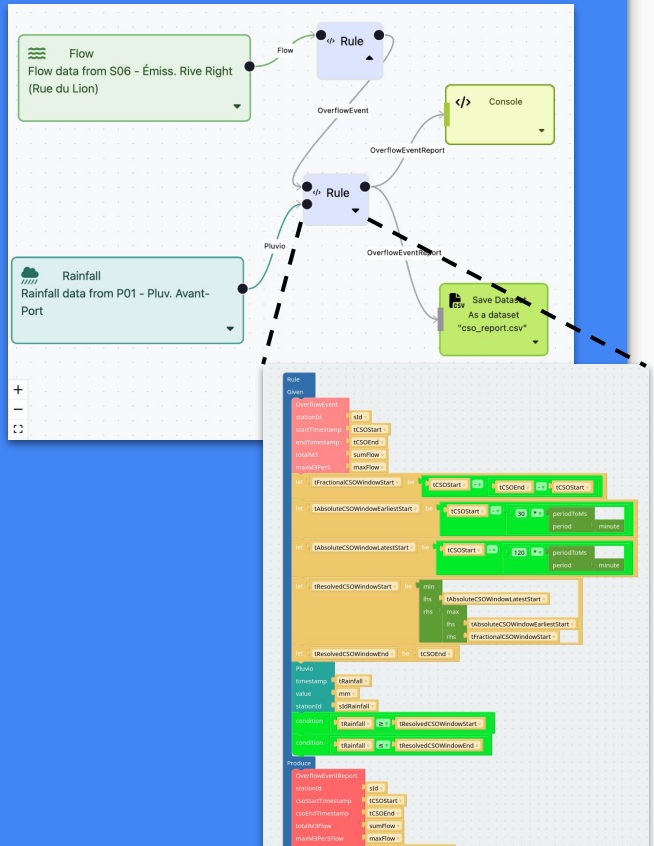


SWAMP Platform combines the power of Flow-Based Programming and Rule-Based Programming

with a user-friendly *visual language* for **domain experts**.

SWAMP

Smart **WA**ter **M**anagement **P**latform



Approach for SWAMP

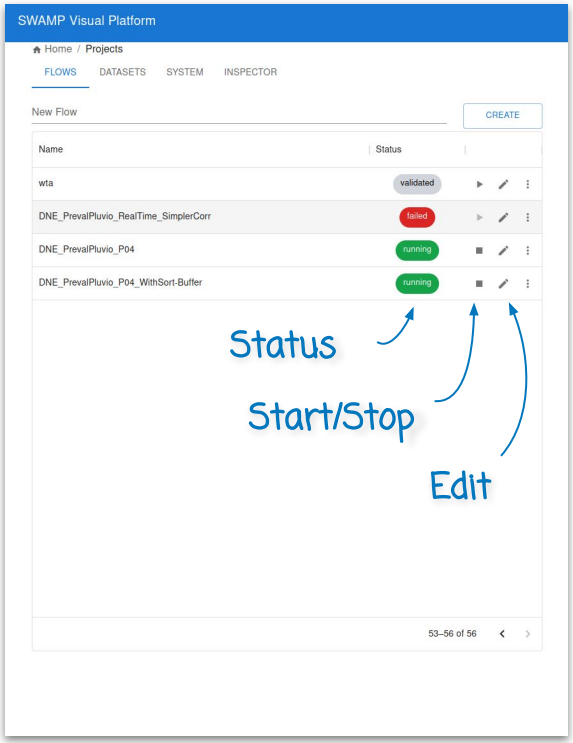
- **Rule-based Specification of Sub-tasks**
  Rules components allow for describing “low-level” reasoning

- **Opt-in Statefulness**
  Specialised rule components that do not persist data

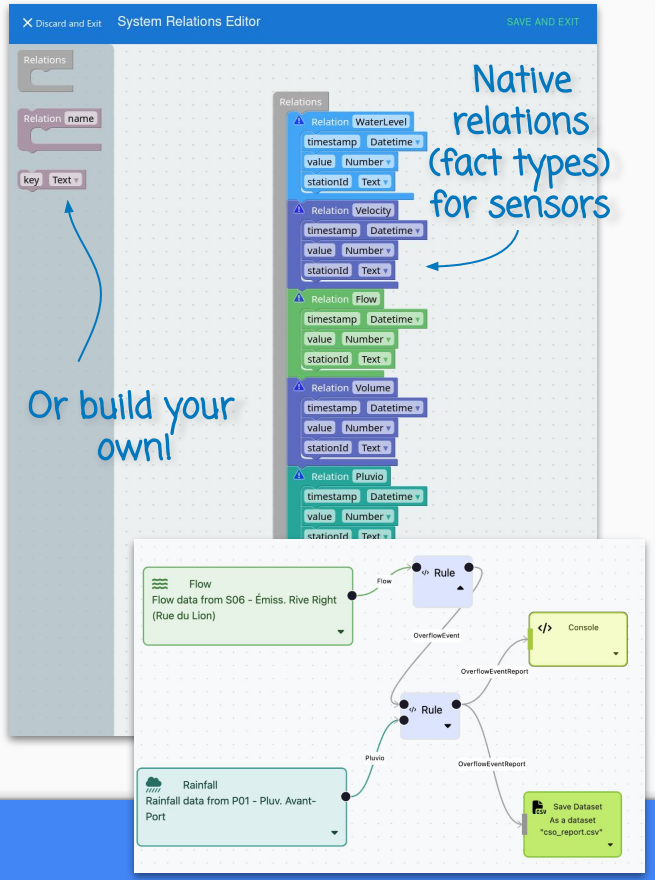
- **Restricted Imperative Actions**
  Imperative actions (I/O) with source and sink components

- **Modular Fact Bases for Rules**
  Each rule component is standalone

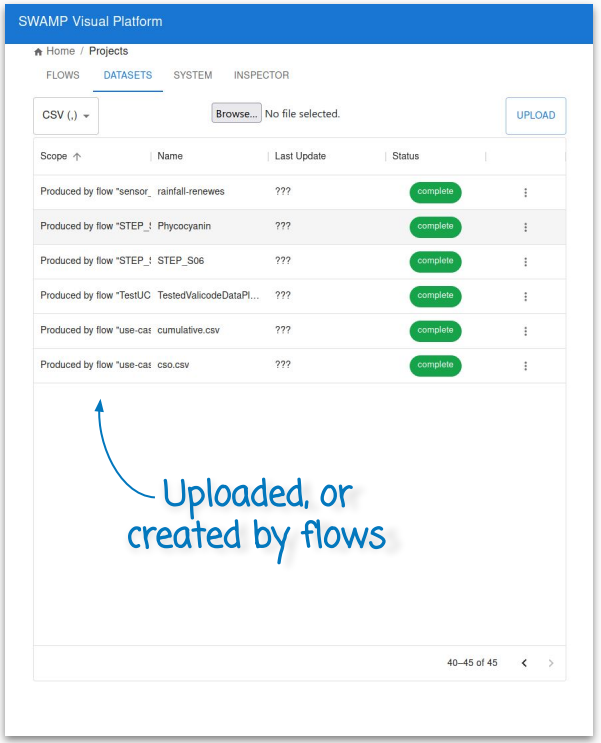
Platform hosts flows



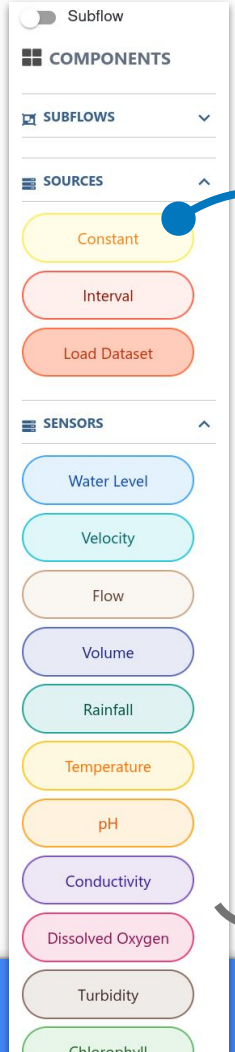
Facts in flows are typed



Datasets



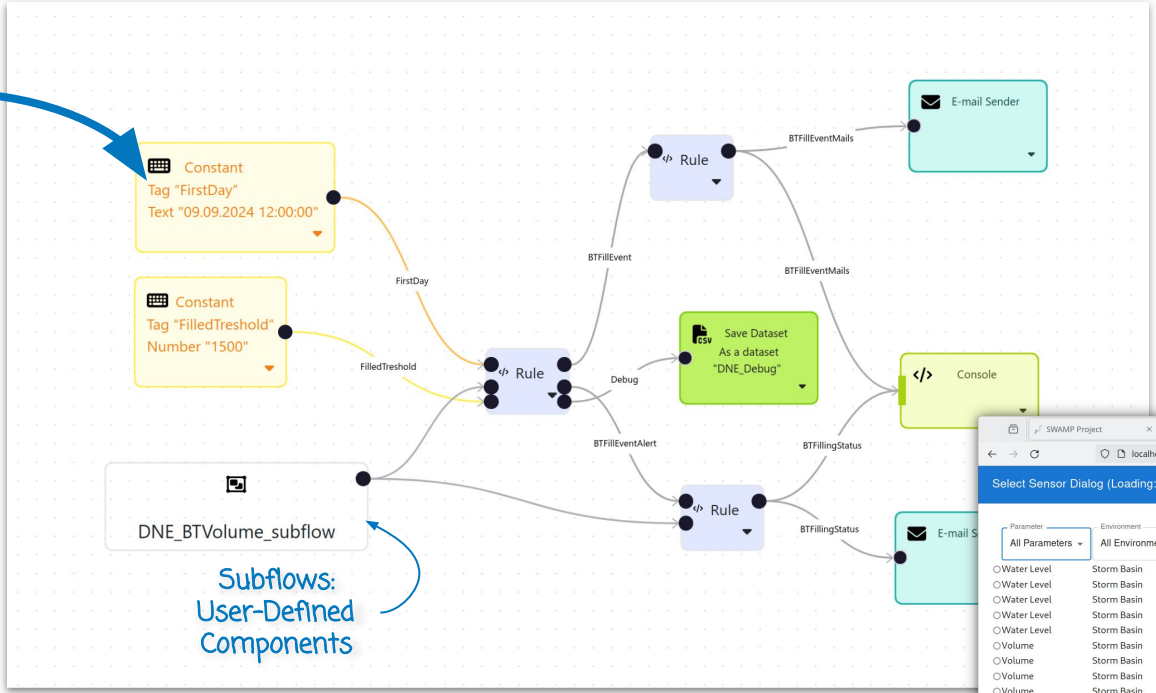
Flows, Relations, Datasets



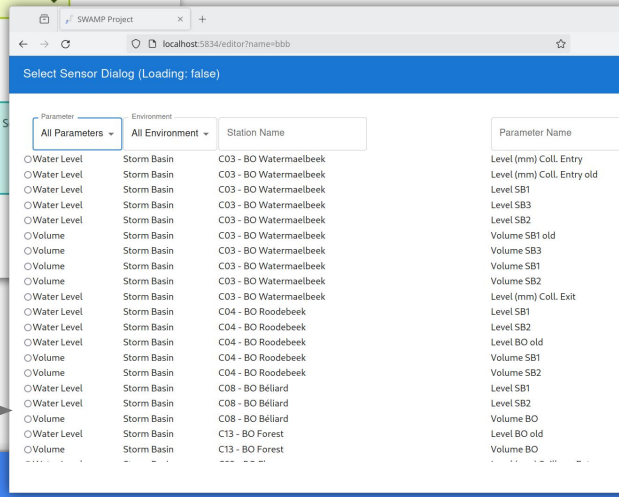
Sources:
Produce data

Operators:
Transform data

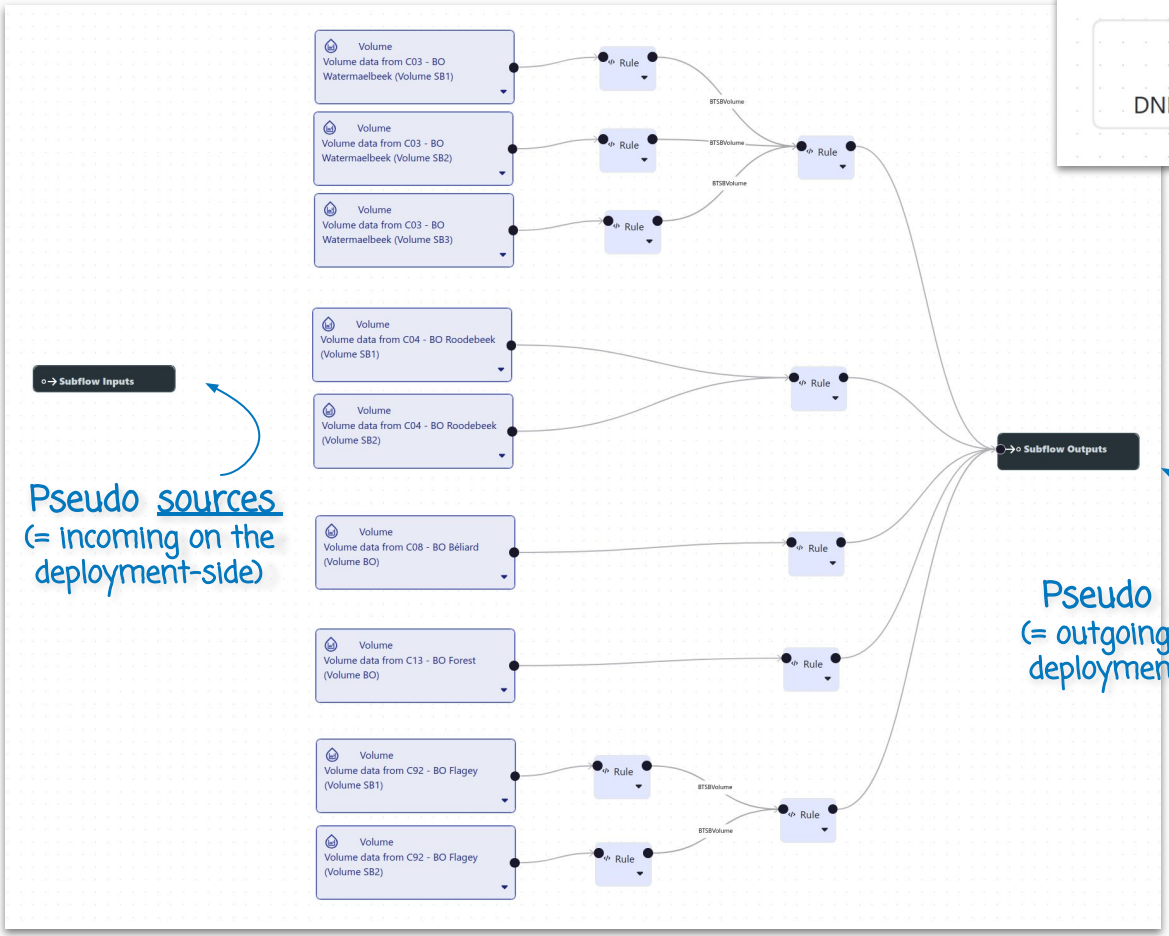
Sinks:
Consume data



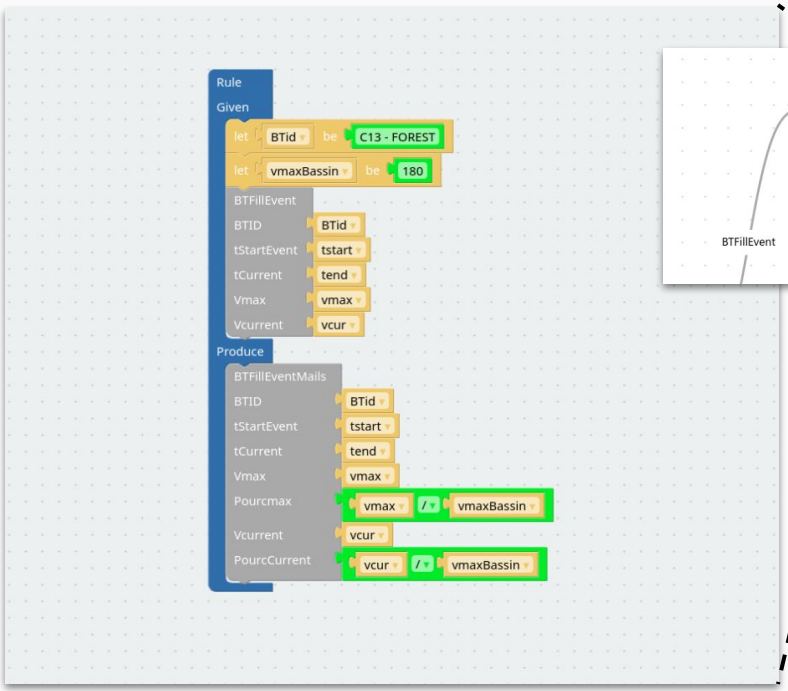
Subflows:
User-Defined
Components

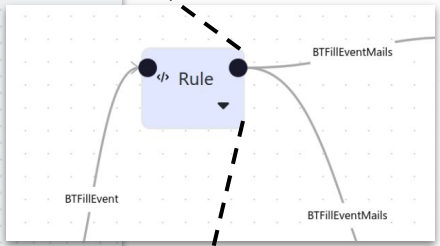


Flow-Based Programming in RuleFlow


Volume
Volume data from C03 - BO Watermaelbeek (Volume SB1)
Volume
Volume data from C03 - BO Watermaelbeek (Volume SB2)
Volume
Volume data from C03 - BO Watermaelbeek (Volume SB3)
Rule
Rule
Rule
BTSBVolume
BTSBVolume
BTSBVolume
Rule
Volume
Volume data from C04 - BO Roodebeek (Volume SB1)
Volume
Volume data from C04 - BO Roodebeek (Volume SB2)
Rule
Subflow Inputs
Pseudo sources
(= incoming on the deployment-side)
Volume
Volume data from C08 - BO Béliard (Volume BO)
Rule
Volume
Volume data from C13 - BO Forest (Volume BO)
Rule
Subflow Outputs
Pseudo sinks
(= outgoing on the deployment-side)
Volume
Volume data from C92 - BO Flagey (Volume SB1)
Volume
Volume data from C92 - BO Flagey (Volume SB2)
Rule
Rule
Rule
BTSBVolume
BTSBVolume

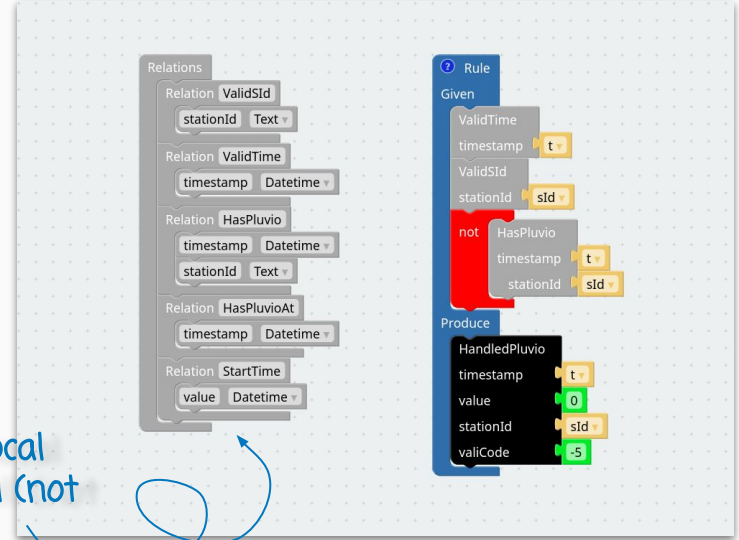

DNE_BTVolume_subflow



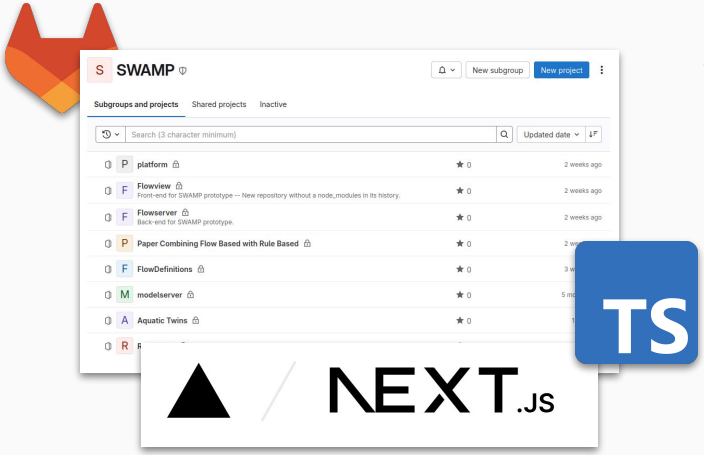


Logic programming with visual blocks (**rules** + **blocks** = **rocks**)

Expresses low-level logic

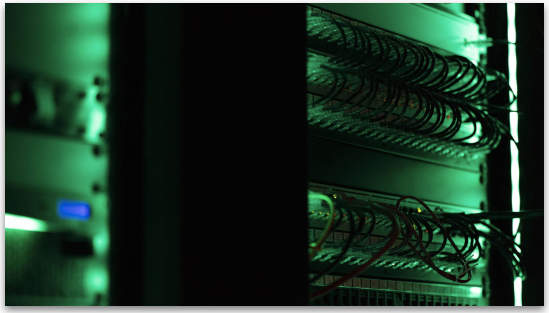


Support for "local relations" as well (not exported)



- Prototypical implementation built in **TypeScript**
  - **Deno** (back-end)
  - **Next.js** (front-end)
    - → ReactFlow for Flow Canvas
    - → Blockly for Rocks Editor

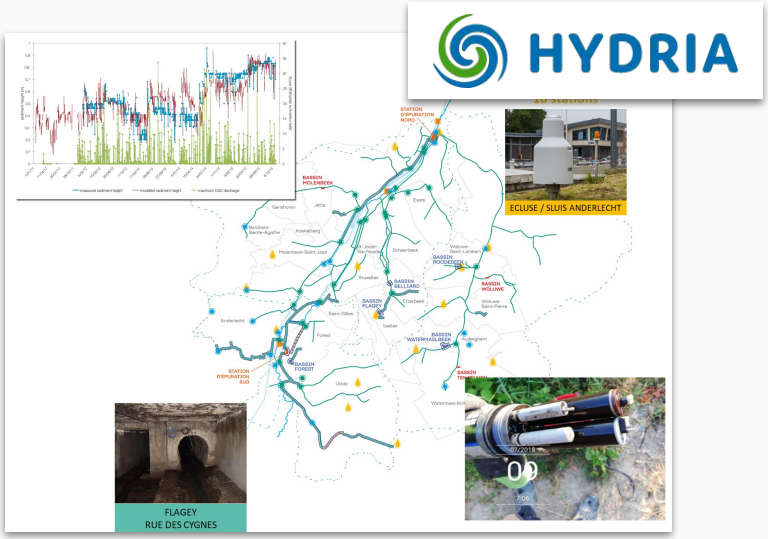
- **SaaS Cloud Platform** hosted on *our* (Software Languages Lab) infrastructure
- Plans to expand / transition



- Hydria experimented with designing surface water monitoring flows on the platform (flows for flows).

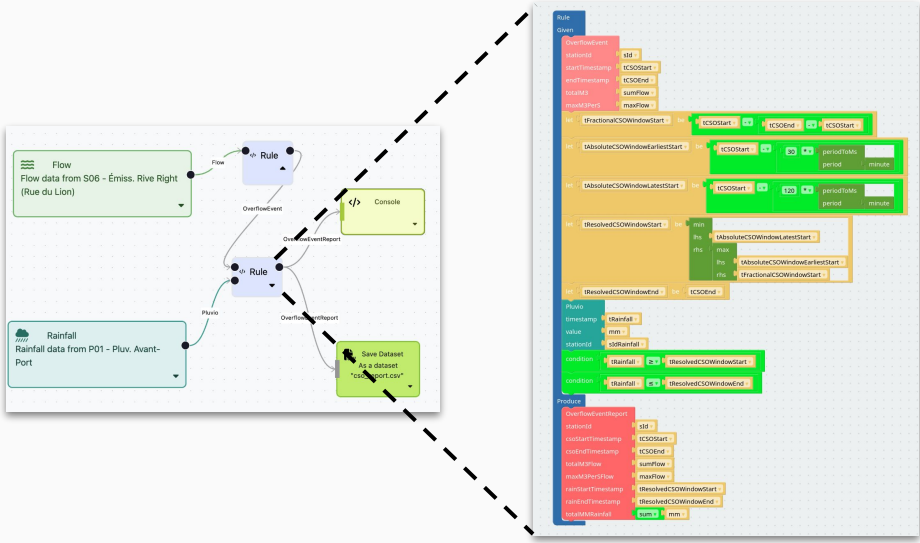
- +20 *real* flows defined
  Data Intake & **Real-Time Monitoring**, used as part of Hydria's decision-making process
  - UC1: Pre-Validation
  - UC2: Detecting Spikes & Threshold Violations

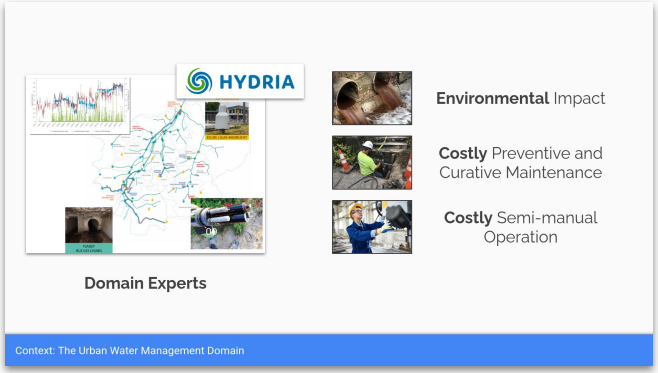
- Currently in a **"service contract"** for bug-fixing purposes

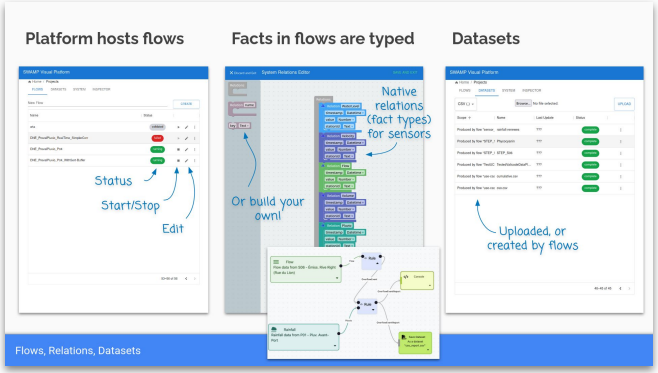


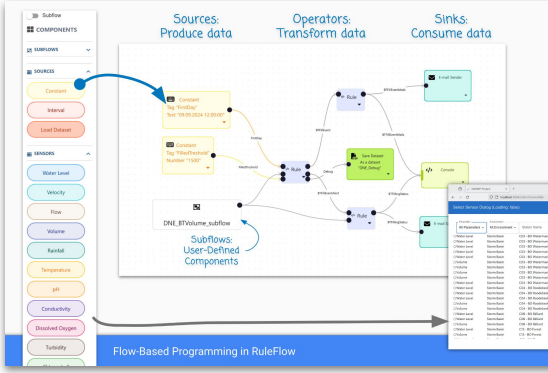
Flow-based and **rule-based** paradigms are complementary

When combined with a **visual language…** = SWAMP









Combining Flows and Rules in a Low-Code Platform for Smart Water Management

Jens Nicolay*, Bjarno Oeyen*✉, Samuel Ngugi Ndung'u*, Thierry Renaux*, Maxime Démarest**, Boud Verbeiren**, Wolfgang De Meuter*

✉: bjarno.oeyen@vub.be

*:
VUB SOFTWARE LANGUAGES LAB
Brussels, Belgium

**:
HYDRIA
Brussels, Belgium

Funded by:
innoviris .brussels we fund your future
Brussels, Belgium

IARIA

1. Linking Flow Definition and Use

Problem
Flows change over time. Subflows might break.

Solution
Warn “subflows” where they are being used to ensure compatibility.

2. Flow Versioning

Solution
Explain “why” subflows are (no longer) compatible. And if needed, allow to use a “pinned” version.

3. Hot-swapping of Stateful Components

Problem
Flow changes require re-starting. Resets state, replaying events (costly).

Solution
Integrate old state in the new flow deployment. But how?