# BURPing

# Through RML Test Cases

Dylan Van Assche and Christophe Debruyne

2024-05-27, KGC Workshop @ ESWC


# Context (I)

In 2023, we worked on RML -- the start of a soon-to-be 🤞 new specification.

> Ana Iglesias-Molina, Dylan Van Assche, Julián Arenas-Guerrero, Ben De Meester, Christophe Debruyne, Samaneh Jozashoori, Pano Maria, Franck Michel, David Chaves-Fraga, Anastasia Dimou: The RML Ontology: A Community-Driven Modular Redesign After a Decade of Experience in Mapping Heterogeneous Data to RDF. ISWC 2023: 152-175

We need compliant RML Processors, and we need test cases (for the challenge)

# Context (II)

- No RML Processor supported RML-CC → “Pressure”
  - This drove the paper we present.

# Problems (I)

- Existing RML Processors result from different initiatives and focus on different aspects (parallel computing, distributed computing, optimization, …)
  - Distributed computing requires commutative monoids, for example.
- Existing RML Processors are a ✨lovely✨ mess of patches and branches
  - RMLmapper supports R2RML, RML.io, and the new specification.

# Problems (II)

- Things became arguably a bit messy once the ISWC`23 paper was accepted
- RML is “modular” → Core, IO, FNML, CC, and STAR.
  - Coverage?
  - Inconsistencies?
  - Propagation of decisions?
  - …
- Problem: the community (read Dylan) struggled setting up track 1.
  - Remember: we could not rely on RMLmapper.

# Starting from a clean slate
# The Basic and Unassuming RML Processor -- BURP

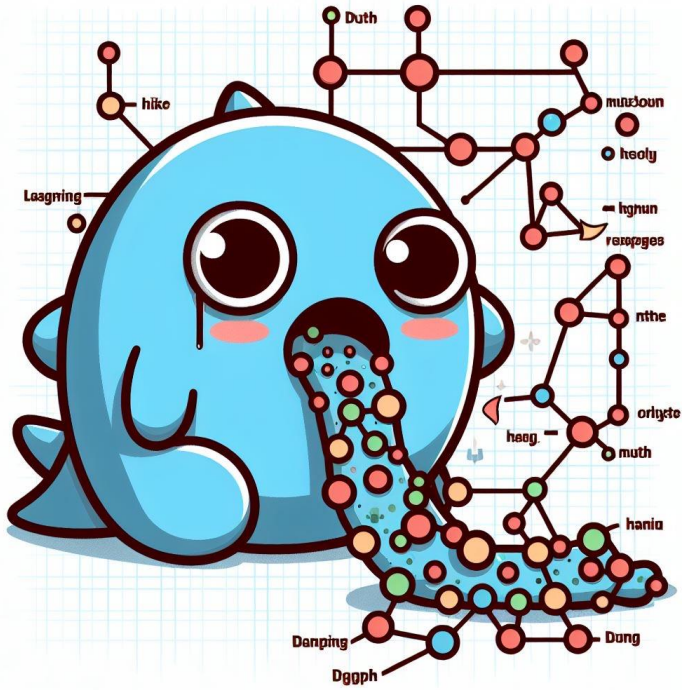
Quid?

- Simple data structures
- Nested loops galore
- No parallel and concurrent processes
- No elegant exception handling, no attempt to recovery

Motivation?

- To KISS (Keep It Simple, Stupid)
- A reference algorithm/implementation à la R2RML
  - From scratch!
- Easy to extend for prototyping purposes sandbox

MIT License, available at https://github.com/kg-construct/BURP




Generated with AI.

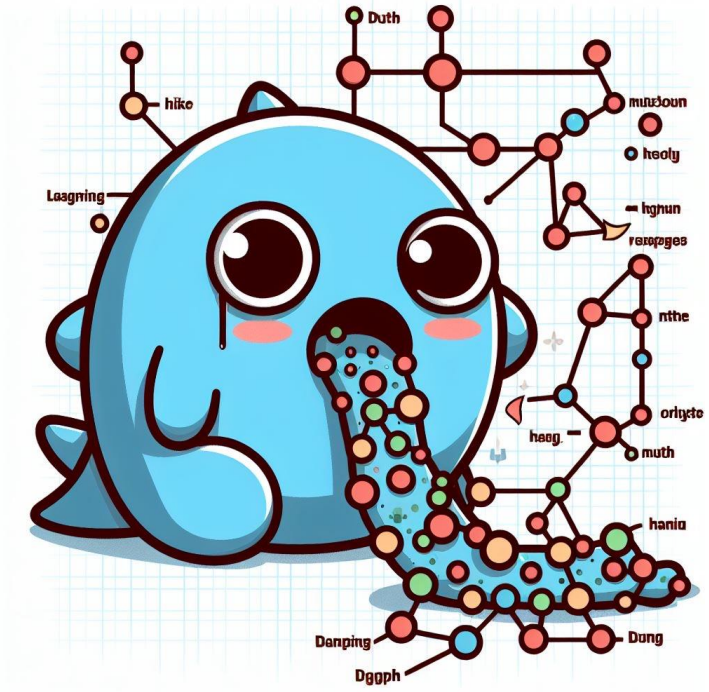
# Starting from a clean slate
# The Basic and Unassuming RML Processor -- BURP

BURP…

1. Loads RML SHACL shapes to validate input
   *Spoiler alert → not everything is covered*
2. Only rewrites RML shortcuts in the mapping
3. Executes the mapping
4. Returns non-zero integers upon failure.

Supports: Core, CC, FNML, IO-source

**BURP helped us (read Dylan) identify problems within and across test cases?**




Generated with AI.

# RML-Core

- Compatibility with other standards
  - Shout out to Pano Maria
- Ill-formed language tags were well-formed
- Inconsistent test cases for invalid mappings
- Incomplete coverage
  - E.g., data type conversions, still TBD
- Base IRI configuration (assumed to be in the mapping)
- Inconsistent shapes across and within modules

# RML-IO

- Improper use of standards
  - E.g., use of certain properties
- Relative file paths
- Datatype inference
- Ambiguities
  - E.g., interpretation of rml:encoding

# RML-CC, RML-FNML, and RML-Star

- Relative file paths
  - Because other modules evolved
- SHACL validation errors (and inconsistencies)
  - Because specifications evolved, but shapes did not

- CC and FNML should cover more corner cases
- FNML: Mappings should be deterministic

# Lessons learned

- Developing an RML processor from scratch
  - Was not only a useful exercise, but
  - Improved the RML specification and raised additional issues

- RML module specifications need to co-evolve with test cases and shapes
- We need better coordination across modules

- Challenge: while the modules are not “intertwined,” the shapes are
  - *Can we automate the shapes for subsets of modules?*


# A Fresh Start: Implementing an RML Processor from Scratch to Validate RML Specifications and Test Cases

Christophe Debruyne and Dylan Van Assche

2024-05-27, KGC Workshop @ ESWC


# Starting from a clean slate
# The Basic and Unassuming RML Processor -- BURP

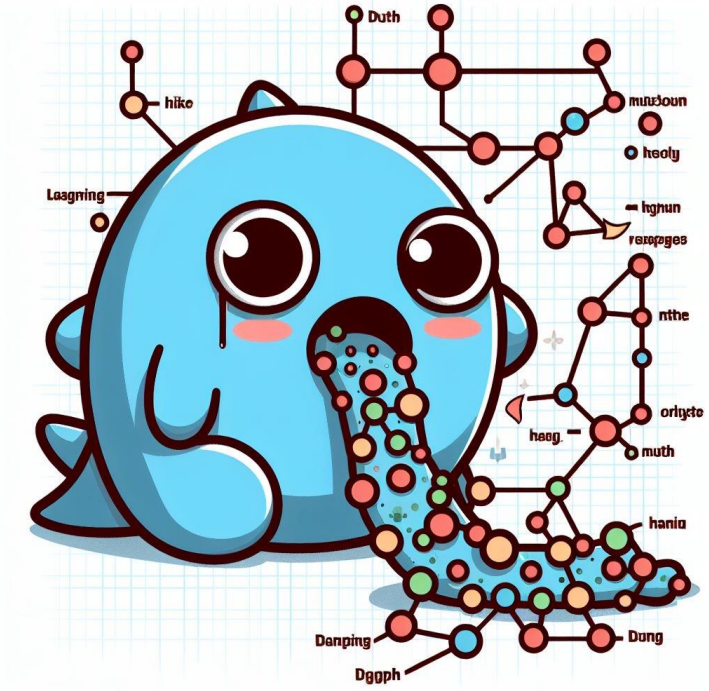
Quid?

- Simple data structures
- Nested loops galore
- No parallel and concurrent processes
- No elegant exception handling, no attempt to recovery

Motivation?

- To KISS (Keep It Simple, Stupid)
- A reference algorithm/implementation à la R2RML
  - From scratch!
- Easy to extend for prototyping purposes sandbox

MIT License, available on the KGC repo




Generated with AI.

# Starting from a clean slate
# The Basic and Unassuming RML Processor -- BURP

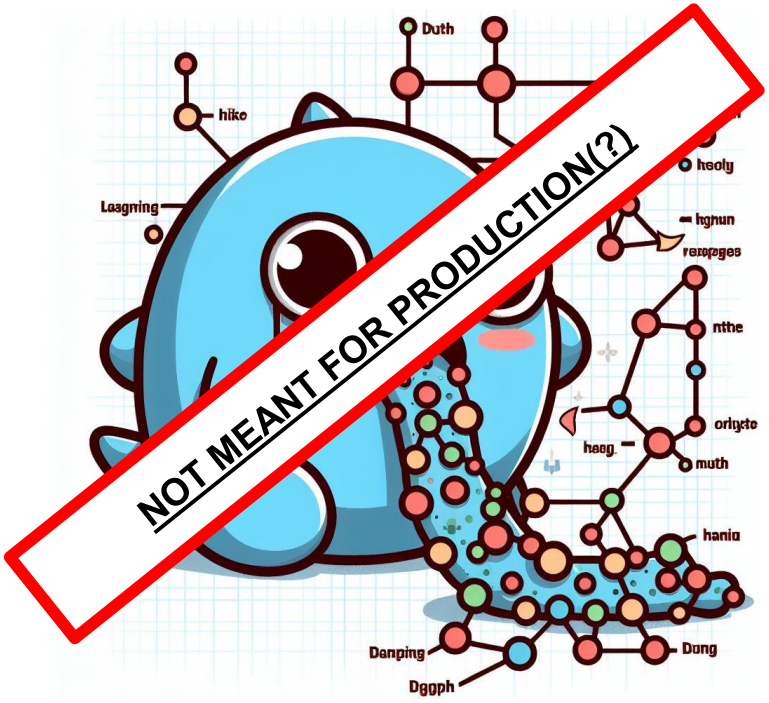
Quid?

- Simple data structures
- Nested loops galore
- No parallel and concurrent processes
- No elegant exception handling, no attempt to recovery

Motivation?

- To KISS (Keep It Simple, Stupid)
- A reference algorithm/implementation à la R2RML
  - From scratch!
- Easy to extend for prototyping purposes sandbox

MIT License, available on the KGC repo




Generated with AI.

# Compliance

First, during the challenge, some tests suddenly failed 🤔 😱

But this was due to

- The wrong URL in the script
- Some test cases in the ZIP were outdated

# Compliance

**BURP passes 100% of the RML-Core test cases.** 🥳

**BURP passes 100% of the RML-CC test cases.** 🥳
*(it would be sad if it did not)*

**BURP passes 92% of the RML-FNML test cases.** 😥
RMLFNOTC0000-CSV relies on generating a (constant) UUID 😤
We refuse to hardcode UUIDs 😇💅

**BURP passes 78% of the RML-IO source test cases.**
(some dialects are not yet implemented)

**BURP passes 2% of the RML-IO target test cases.**
No effort has been spent on the RML-IO target, but we will take it. 🙏

# Questions

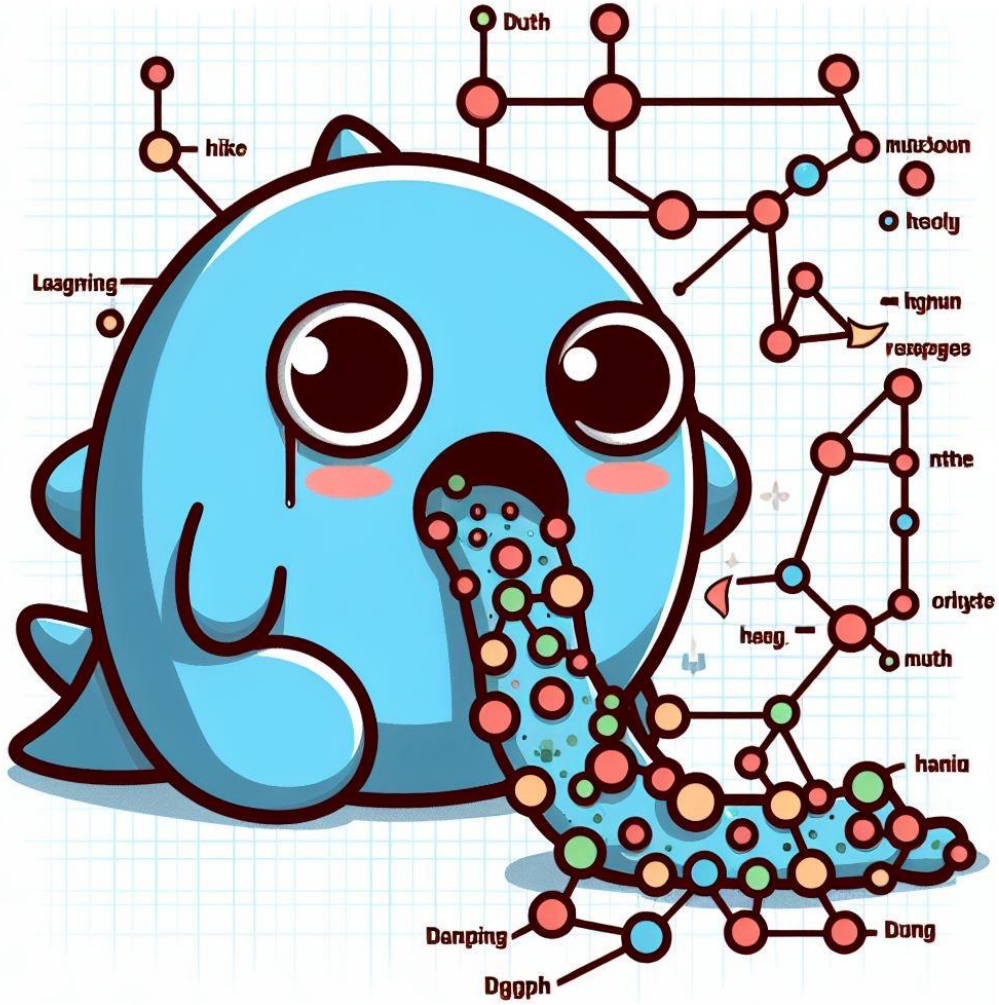
How can we ensure we have covered most combination modules?

1) RML-FNO uses rml:inputValueMap to link an input with a Term Map. Some Term Maps have a Graph Map (e.g., Subject Maps), how does that impact the Predicate Object Maps with Graph Maps?
2) Quoted triples can be included in RDF Containers and Collections, but what is the expected behavior when RML Quoted Triples Maps are also used as a Gather Map?

# To conclude

- BURP is an arguably simple implementation of RML
- BURP was developed from scratch
- BURP is naïve for a reason (e.g., commutative monoids)
- It is hoped to become a reference implementation

- Future work
  - Implement the other modules (hopefully during the 2nd half of 2024)
  - [ On a less serious note ] participate in Track 2? 😇




Generated with AI.