Programming in Relation Algebra

Stef Joosten

March 9th, 2021

Contact: stef.joosten@ou.nl

Open Universiteit
www.ou.nl

Who is Stef Joosten?

- Since 1999 with the OU, before that at Univ. Twente.
  Since 1997 with Ordina: Management consultant, software engineer, auditor.
- I love to design information systems;
- I love programming languages (e.g. Haskell).
- I love to apply formal methods in practice;
- I love to professionalize IT.
- I love to collaborate and learn from other professionals.

Open Universiteit
www.ou.nl

Why on earth use Relation Algebra for programming?

- Cool language properties: Calculate with programs, declarative semantics, 0-thread, reactive, domain-driven, strongly typed.
- Better information systems: Generate software, correctness-by-design, generate documentation, produce distributed systems.
- Important applications: Law-driven design, data migration projects, domain-specific process support.
- Academic pride: Formal methods in practice, holistic thinking, tackling complex problems...

outline

Preliminaries

Relations

We specify the storage of an information system by a triple

$$\langle \mathcal{C}, \mathcal{R}, \mathcal{H} \rangle$$

in which $\mathcal{C}$ is a finite set of concepts, $\mathcal{R}$ is a finite set of relations, and $\mathcal{H}$ is a finite set of rules.

Notation:
Italic capitals A, B, ... represent concepts, so $A \in \mathcal{C}$, $B \in \mathcal{C}$, ...
Italic letters r, s, ... represent relations, so $r \in \mathcal{R}$, $s \in \mathcal{R}$, ...

Examples of concepts: "Person", "Trip", "Destination".
Examples of relations: $\text{booked}_{\langle \text{Person}, \text{Trip} \rangle}$, $\text{destination}_{\langle \text{Trip}, \text{City} \rangle}$.

Relations

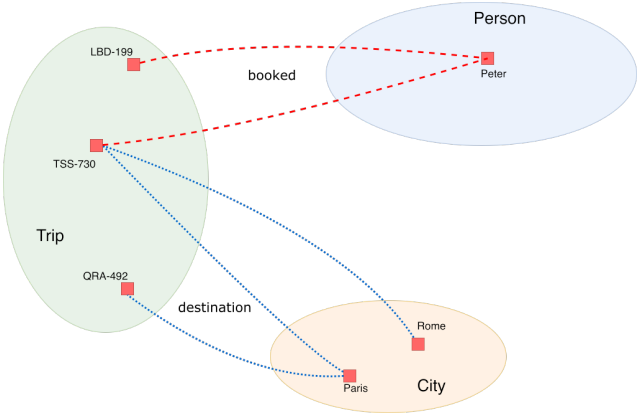


Figure: Relations “booked” and “destination”

Relations

Interpretation of primitive terms

We define terms that consist of operators and relations. Each term has an interpretation $\mathfrak{I}(\_)$. For every $A, B \in \mathcal{C}$ and $r \in \mathcal{T}$

$$\mathfrak{I}(r) = \{\langle a, b\rangle |\ a\ r\ b\} \quad (1)$$

$$\mathfrak{I}(I_A) = \{\langle a, a\rangle |\ a \in A\} \quad (2)$$

$$\mathfrak{I}(V_{\langle A,B\rangle}) = \{\langle a, b\rangle |\ a \in A, b \in B\} \quad (3)$$

See also: `https://ampersandtarski.gitbook.io/documentation/the-language-ampersand/terms`

Relations

Interpretation of terms with boolean operators

For every $A, B \in \mathcal{C}$ and $r, s \in \mathcal{T}$

$$\mathfrak{I}(r \cup s) = \{\langle a, b\rangle |\ \langle a, b\rangle \in \mathfrak{I}(r) \ \vee \ \langle a, b\rangle \in \mathfrak{I}(s)\} \quad (4)$$

$$\mathfrak{I}(r - s) = \{\langle a, b\rangle |\ \langle a, b\rangle \in \mathfrak{I}(r) \ \wedge \ \langle a, b\rangle \notin \mathfrak{I}(s)\} \quad (5)$$

$$\mathfrak{I}(r \cap s) = \{\langle a, b\rangle |\ \langle a, b\rangle \in \mathfrak{I}(r) \ \wedge \ \langle a, b\rangle \in \mathfrak{I}(s)\} \quad (6)$$

$$\mathfrak{I}(\overline{r_{\langle A,B\rangle}}) = V_{\langle A,B\rangle} - \mathfrak{I}(r) \quad (7)$$

Relations

Boolean operators defined algebraically

$$r \cup (s \cup t) = (r \cup s) \cup t \quad \text{(associative)} \tag{8}$$

$$r \cup s = s \cup r \quad \text{(commutative)} \tag{9}$$

$$r \cup r = r \quad \text{(idempotent)} \tag{10}$$

$$r_{\langle A,B \rangle} \cup \overline{V_{\langle A,B \rangle}} = r_{\langle A,B \rangle} \quad (\overline{V_{\langle A,B \rangle}} \text{ is an identity element}) \tag{11}$$

$$r - s = t \Leftrightarrow \begin{array}{l} r \subseteq s \cup t \quad \text{and} \\ \forall t' : s \cup t' = r \rightarrow t \cup t' = t' \end{array} \tag{12}$$

$$r \cap s = r - (r - s) \tag{13}$$

$$\overline{r_{\langle A,B \rangle}} = V_{\langle A,B \rangle} - r \tag{14}$$

Relations

Interpretation of terms with relational operators

For every $A, B \in \mathcal{C}$ and $r, s \in \mathcal{T}$

$$\mathfrak{I}(r;s) = \{\langle a,c\rangle |\ \exists b,\ \langle a,b\rangle \in \mathfrak{I}(r)\ \wedge\ \langle b,c\rangle \in \mathfrak{I}(s)\} \tag{15}$$

$$\mathfrak{I}(r^{\smile}) = \{\langle b,a\rangle |\ \langle a,b\rangle \in \mathfrak{I}(r)\} \tag{16}$$

Relations

Relational operators defined algebraically

$$r;(s;t) = (r;s);t \quad \text{(associative)} \tag{17}$$
$$I_A; r_{\langle A,B\rangle} = r_{\langle A,B\rangle} \quad \text{(left identity)} \tag{18}$$
$$r_{\langle A,B\rangle}; I_B = r_{\langle A,B\rangle} \quad \text{(right identity)} \tag{19}$$
$$r^{\smile\smile} = r \tag{20}$$
$$r^{\smile}; s^{\smile} = s; r^{\smile} \tag{21}$$

Rules

A rule may have three forms:

$$\text{RULE } < \texttt{term} > \subseteq < \texttt{term} > \tag{22}$$
$$\text{RULE } < \texttt{term} > = < \texttt{term} > \tag{23}$$
$$\text{RULE } < \texttt{term} > \tag{24}$$

Semantics:

$$\text{RULE } r \subseteq s \Leftrightarrow \mathfrak{I}(r - s) = \emptyset \tag{25}$$
$$\text{RULE } r = s \Leftrightarrow \begin{array}{l}\text{RULE } r \subseteq s \quad \text{and} \\ \text{RULE } s \subseteq r\end{array} \tag{26}$$
$$\text{RULE } r \Leftrightarrow \text{RULE } V_{\langle A,B\rangle} \subseteq r \tag{27}$$

Information systems

General structure of an information system

layers:

1. communication: event streams, components
2. storage: concepts, relations, rules
3. services: roles, interfaces, frameworks
4. user experience: shapes, colors, etc.

Ampersand focuses on storage and services. Hopefully we can add communication in the future.

Example: School trips

Once in their school career, students get to go on a field trip abroad. However, the organizer of the field trip may require that you have completed specific courses. This application gives overviews of students and the trips for which they qualify.

- 3 relations: passed, required, registered
- 1 rule: A student can only attend a trip, if the student passed all the required courses for that destination.
- 1 service, no roles

Example: Relations with population

```
RELATION required[Subject*Destination]

POPULATION required CONTAINS
   [ ("Surfing", "Hawaii")
   ; ("Latin", "Rome")
   ; ("World Religions", "Rome")
   ]

RELATION passed[Subject*Student]

POPULATION passed CONTAINS
   [ ("Surfing", "Brown")
   ; ("Surfing", "Conway")
   ...
   ]
```

Example: Rule

```
PURPOSE RULE guardPrerequisites
{+ This rule prevents students from registering for a trip
   without having passed the required courses.
+}

RULE guardPrerequisites : registered~ |- required\passed
MEANING "A student can only attend a trip, if the student passed all the required
MESSAGE "Attempt to register student(s) for a trip without the proper qualificati
VIOLATION (TXT "Student ", TGT I, TXT " cannot go to ", SRC I, TXT ".")
```

Note: "`RULE required;registered~ |- passed`" is equivalent w.r.t its truth value, but not w.r.t violations.

Example: Service

```
INTERFACE Overview : "_SESSION"                          cRud
BOX <TABS>
     [ Students     : V[SESSION*Student]                 cRuD
       BOX <TABLE>
            [ "Student" : I[Student]                     cRud
            , "passed" : passed~                         CRUD
            , "Qualify for" : passed~/required~          cRud
            , "registered" : registered                  CRUd
            ]
     , Subjects     : V[SESSION*Subject]                 cRuD
       BOX <TABLE>
            [ "Subject" : I                              cRud
            , "required for trip" : required             CRUD
            , "students that passed" : passed            CRUD
            ]
     , Destinations : V[SESSION*Destination]             cRuD
       BOX <TABLE>
            [ "Destination" : I                          cRud
            , "required subject" : required~             CRUD
            , "Qualifying students" : required\passed    cRud
            , "registered" : registered~                 cRUd
            ]
     ]
```

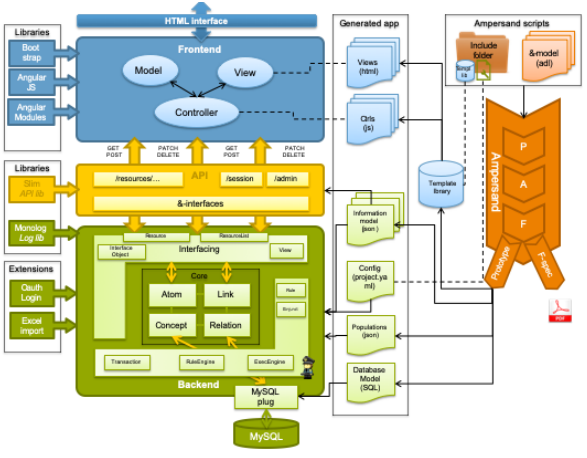
Demo

This demo uses RAP (`https://rap.cs.ou.nl`) as platform for running Ampersand prototypes.

Ampersand Architecture

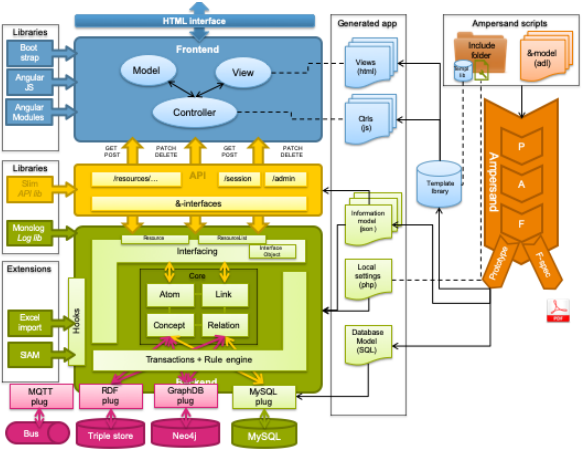
Open Universiteit
www.ou.nl

Architecture

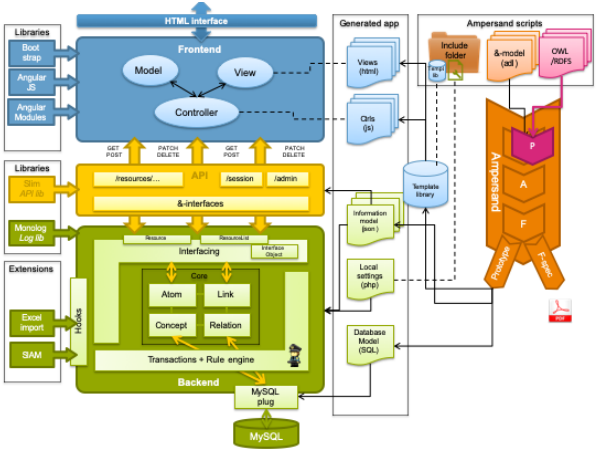


Research Challenges

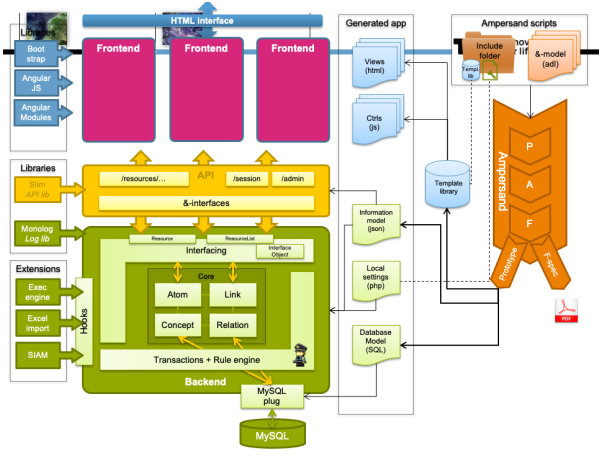
How do NoSQL databases perform?



How to generate from OWL+RDFS?

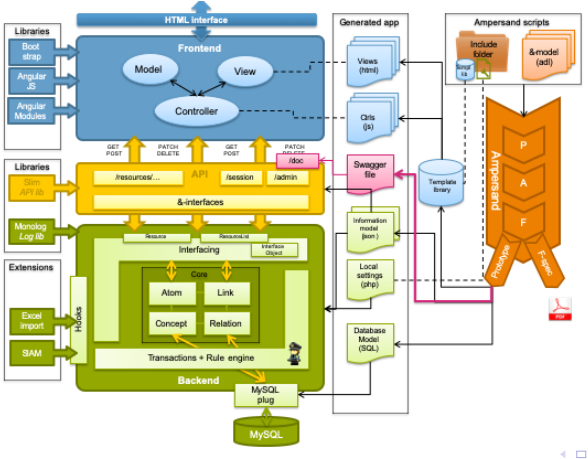


Can we formally specify microservices executably?



Can we enhance with software development improvements?

E.g. generate API documentation in Swagger files



Other Research Challenges

- How to enrich the language to comprise computations?
- How to optimize code further?
  - RA $\rightarrow$ RelAlg $\rightarrow$ SQL
  - RA $\rightarrow$ Graph Algebra $\rightarrow$ ...
- How much security can we generate into a prototype?
- How much testing software can we generate specifically for one generated application?
- How much genericity and automation can we use to deploy?

Questions

Links

- Documentation: `https://ampersandtarski.gitbook.io/documentation/`
- Tool: `https://rap.cs.ou.nl`
- Joosten, *Relation Algebra as programming language using the Ampersand compiler*, J. Logic and Algebraic Methods in Programming, vol 100, pp 113-129, 2018.
- Joosten and Joosten, *Type Checking by Domain Analysis in Ampersand* Proceedings RAMiCS 2015, Braga, Portugal, 2011.
- Michels et.al., *Ampersand: Applying relation algebra in practice* Proceedings RAMiCS 2011, Rotterdam, 2011.