DPCL: a Language Template for Normative Specifications

16 January 2022, ProLaLa @ POPL 2022

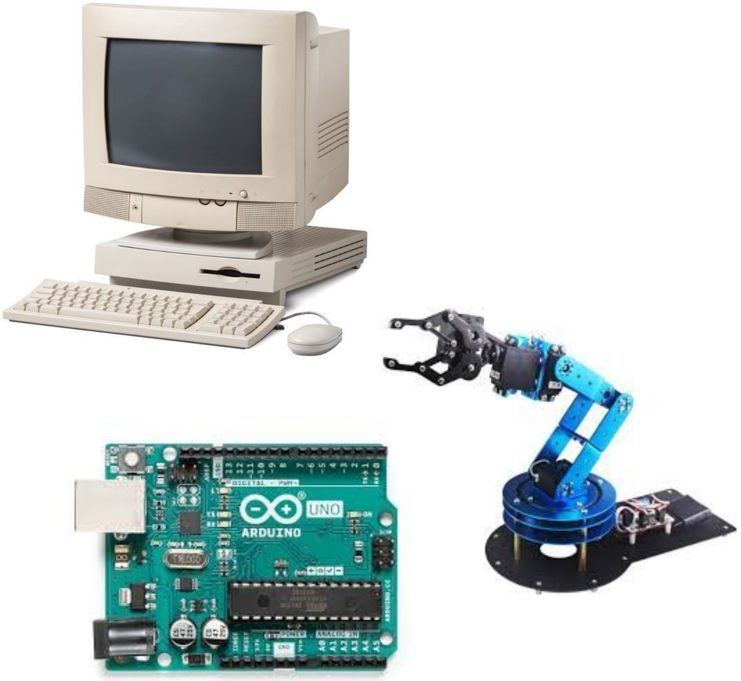
Giovanni Sileno g.sileno@uva.nl
Thomas van Binsbergen
Tom van Engers

University of Amsterdam

Matteo Pascucci

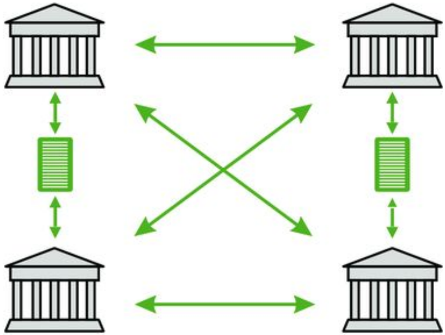
Slovak Academy of Sciences

from individual devices to digital social systems...



Social networks

Distributed Ledgers



Digital Markets

Internet of Things

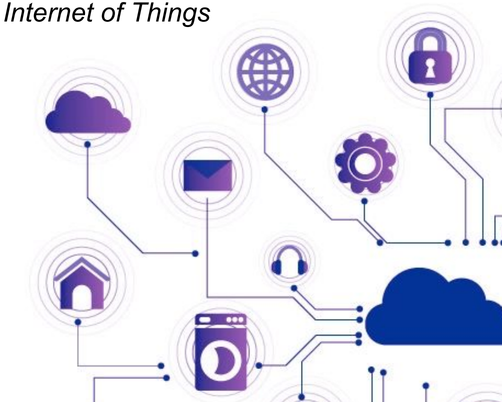
from “mechanical” to “institutional” approaches to computation...

not *instructions*, but ***contracts***, ***regulations***, ***laws***…

focus on
PERFOMANCE → **focus on**
COORDINATING EXPECTATIONS

Digital Markets

Internet of Things



ok, we need to represent normative directives, but how?

ok, we need to represent normative directives, but how?

1. do we need normative concepts?
2. if yes, which normative concepts do we need?
3. what do they “mean”?

1. do we need normative concepts?

programs in themselves
are mandatory in nature

1. do we need normative concepts?

programs in themselves
are mandatory in nature

`a := 2 + 2`	system ***has*** to perform 2 + 2…
`?mother(maggie, bart)`	system ***has*** to prove that…
`animal :- dog.`	system ***has*** to make animal true if dog is true

1. do we need normative concepts?

programs in themselves
are mandatory in nature

PERFORMANCE
is expected

➡ ***the system does what we tell it to do***

1. do we need normative concepts?

programs in themselves are mandatory in nature

PERFORMANCE is expected

vs **FAILURE is expected**

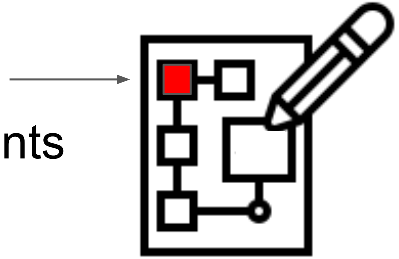
1. do we need normative concepts?

programs in themselves are mandatory in nature

PERFORMANCE is expected

vs FAILURE is expected

VIOLATION
certain components may not perform as required



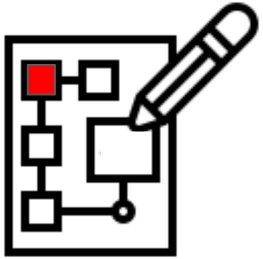
1. do we need normative concepts?

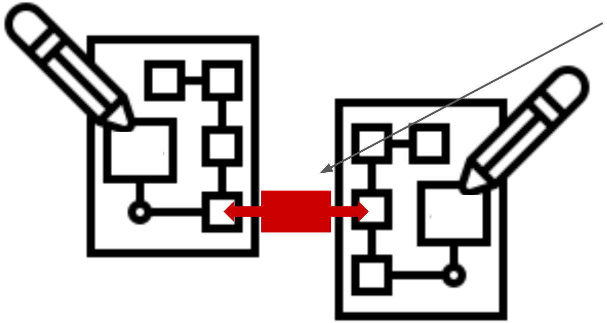
programs in themselves are mandatory in nature

PERFORMANCE is expected

vs FAILURE is expected

VIOLATION
certain components may not perform as required





CONFLICT
concurrent components may have incompatible requests

1. do we need normative concepts?

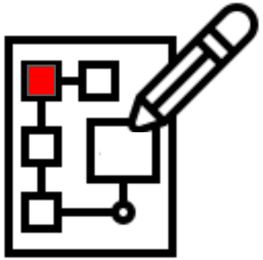
CENTRAL PROBLEM: who will declare that there is/was indeed a failure?

programs in themselves are mandatory in nature

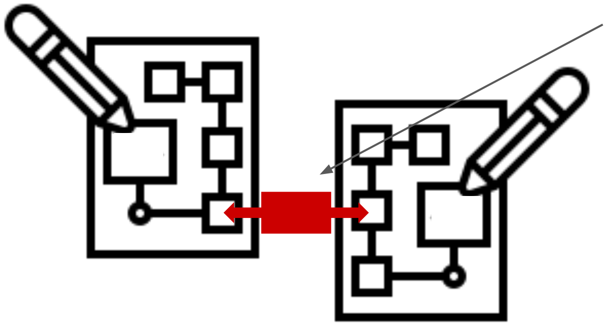
PERFORMANCE is expected

vs FAILURE is expected

VIOLATION
certain components may not perform as required

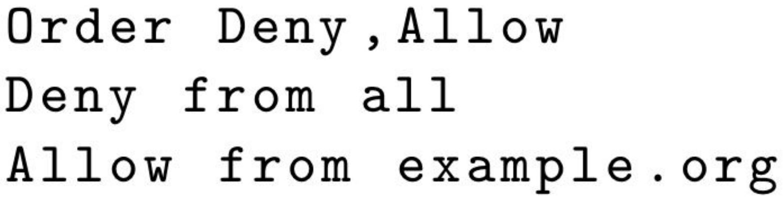


CONFLICT
concurrent components may have incompatible requests



2. which normative concepts do we need?

- Control models (e.g. access or usage control)

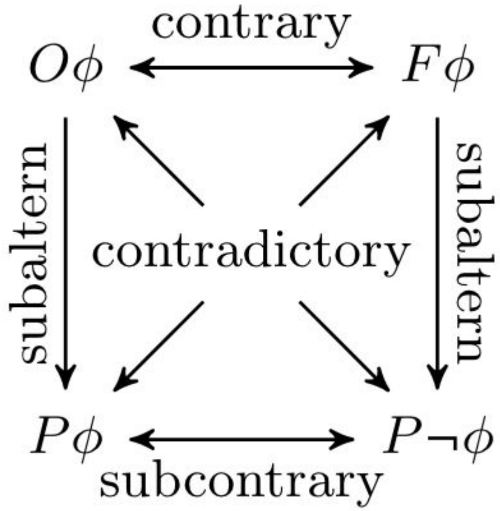


```
Order Deny,Allow
Deny from all
Allow from example.org
```

example from Apache webserver configuration

2. which normative concepts do we need?

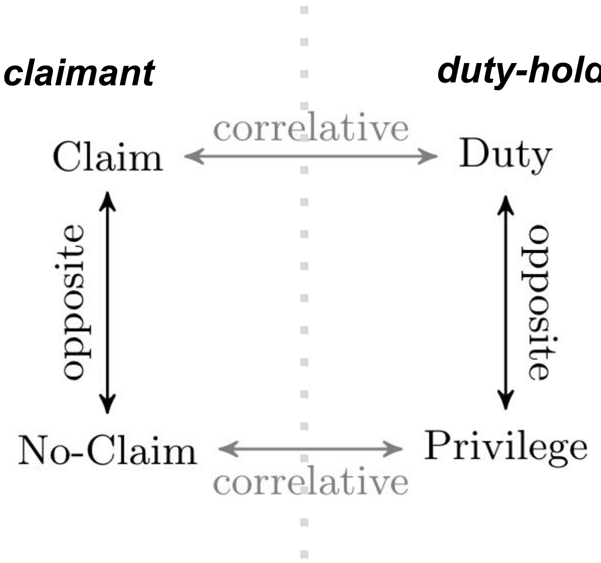
- Deontic logic(s)



2. which normative concepts do we need?

- Hohfeld’s (based on Salmond’s) normative relationships

claimant ***duty-holder***



Claim

correlative

Duty

opposite

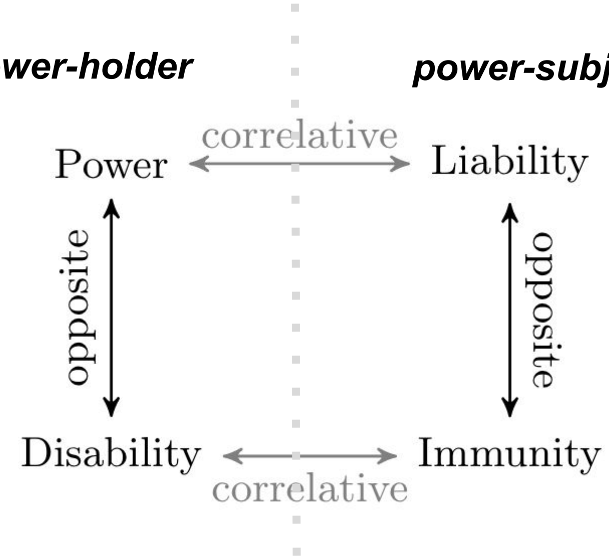
opposite

No-Claim

correlative

Privilege

power-holder ***power-subject***



Power

correlative

Liability

opposite

opposite

Disability

correlative

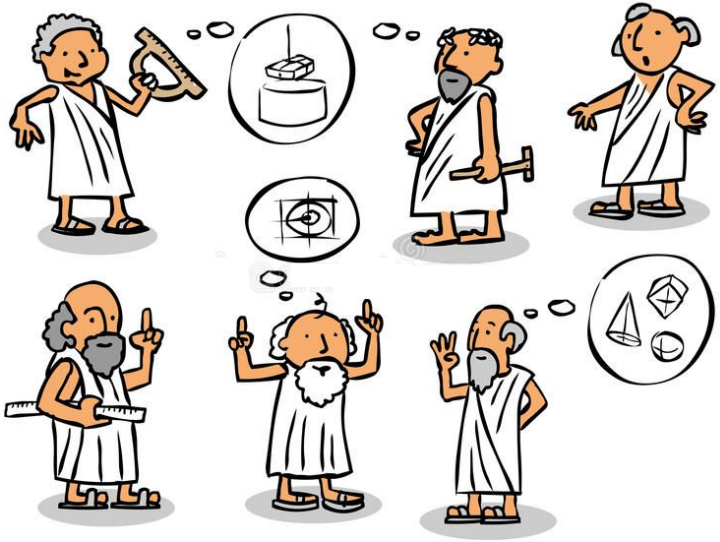
Immunity

2. which normative concepts do we need?

	Control models	Deontic Logic(s)	Hohfeld's framework
permission	X	X	X (as liberty)
prohibition	X	X	X (as duty not)
obligation		**X**	X (as duty)
power/ability			**X**
	1 party	1 party	**2 parties**
focus on	**actions**	**situations**	**actions**

3. what normative concepts “mean”?

- long-standing debate
- no shared agreement
- new semantics continuously released



ok, we need to represent normative directives, but how?

expecting performance vs expecting failures (violations and conflicts)

1. **do we need normative concepts?**
2. **if yes, which normative concepts do we need?**

control models vs deontic logics
vs hohfeldian relationships

3. **what do they "mean"?**

…long-standing debate. no shared agreement.

ok, we need to represent normative directives, but how?

expecting performance vs expecting failures (violations and conflicts)

1. do we need normative concepts?
2. if yes, which normative concepts do we need?

control models vs deontic logics
vs hohfeldian relationships

3. what do they "mean"?

…long-standing debate. no shared agreement.

4. how to specify normative directives?

ok, we need to represent normative directives, but how?

expecting performance vs expecting failures (violations and conflicts)

1. do we need normative concepts?
2. if yes, which normative concepts do we need?

control models vs deontic logics
vs hohfeldian relationships

3. what do they "mean"?

…long-standing debate. no shared agreement.

4. how to **specify** normative directives?

programmability, readability, (cognitive) accessibility, …?

Success story: ODRL (Open Digital Rights Language)

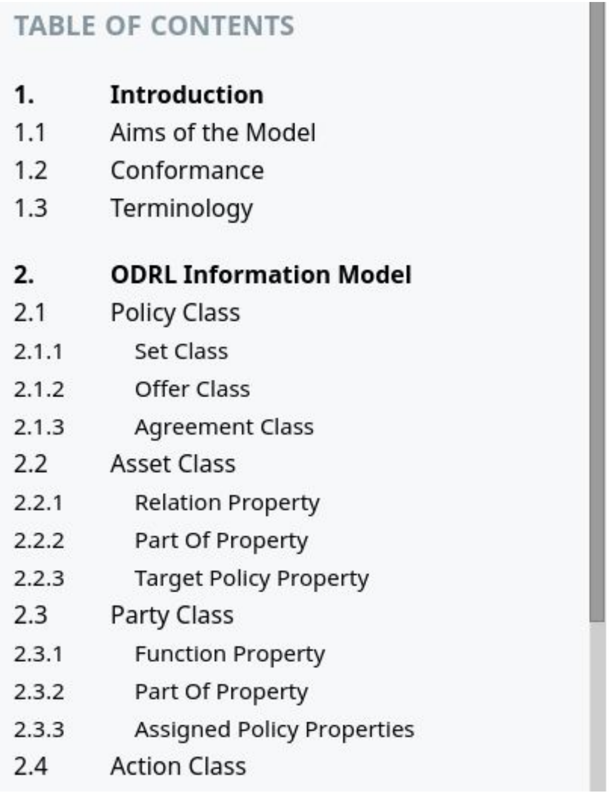
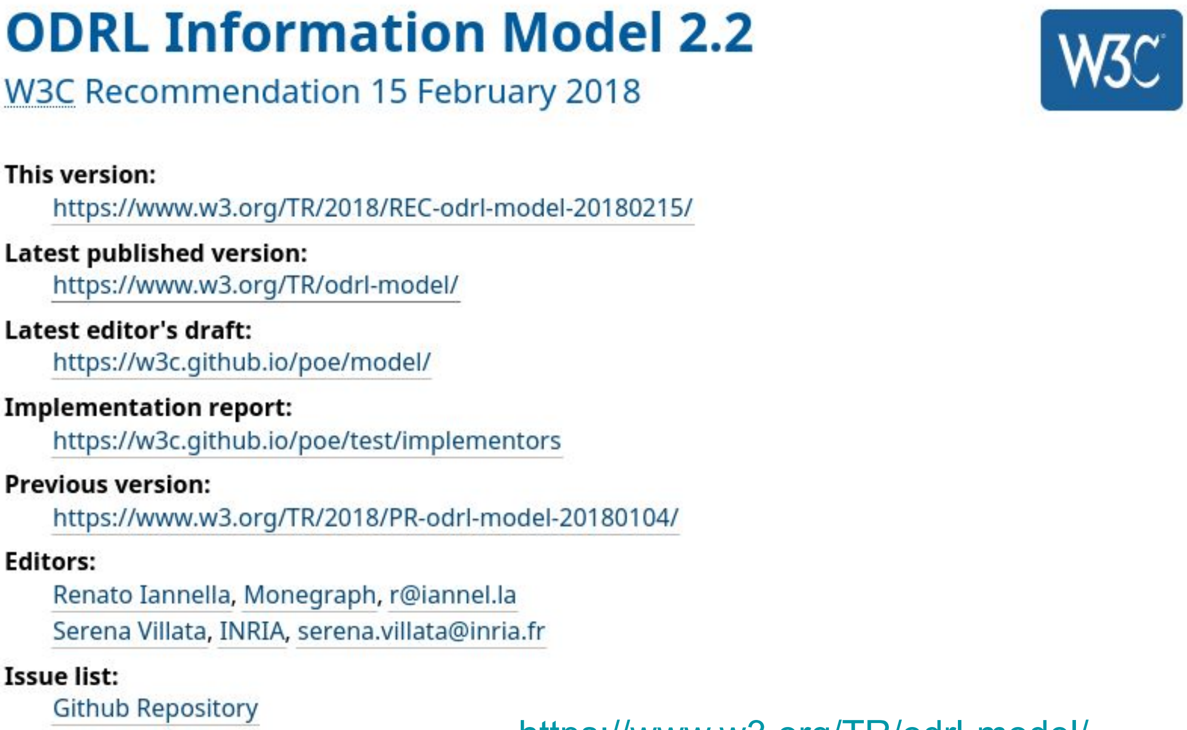


TABLE OF CONTENTS



ODRL Information Model 2.2

W3C Recommendation 15 February 2018

This version:
https://www.w3.org/TR/2018/REC-odrl-model-20180215/

Latest published version:
https://www.w3.org/TR/odrl-model/

Latest editor's draft:
https://w3c.github.io/poe/model/

Implementation report:
https://w3c.github.io/poe/test/implementors

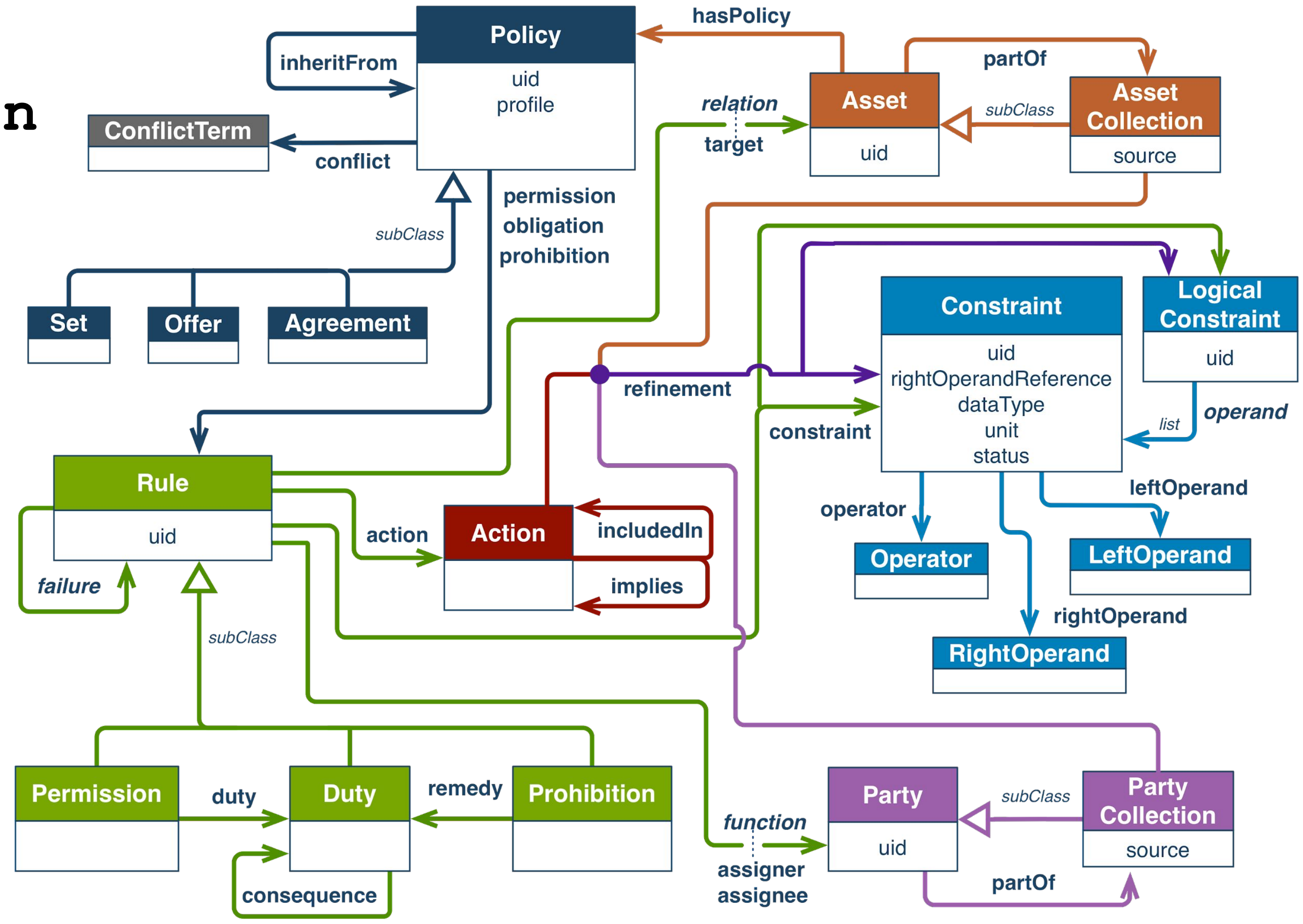
Previous version:
https://www.w3.org/TR/2018/PR-odrl-model-20180104/

Editors:
Renato Iannella, Monegraph, r@iannel.la
Serena Villata, INRIA, serena.villata@inria.fr

Issue list:
Github Repository

https://www.w3.org/TR/odrl-model/

ODRL Information Model



primacy
to deontic
categories

ODRL example

```
{
  "@context": "http://www.w3.org/ns/odrl.jsonld",
  "@type": "Offer",
  "uid": "http://example.com/policy:4444",
  "profile": "http://example.com/odrl:profile:11",
  "permission": [{
    "assigner": "http://example.com/org88",
    "target": {
      "@type": "AssetCollection",
      "source":  "http://example.com/media-catalogue",
      "refinement": [{
        "leftOperand": "runningTime",
        "operator": "lt",
        "rightOperand": { "@value": "60", "@type": "xsd:integer" },
        "unit": "http://qudt.org/vocab/unit/MinuteTime"
      }]
    },
    "action": "play"
  }]
}
```

json
data
structure

roughly: permission to org88 to play assets in collection with running length < 60 min

ODRL example

```
{
  "@context": "http://www.w3.org/ns/odrl.jsonld",
  "@type": "Offer",
  "uid": "http://example.com/policy:4444",
  "profile": "http://example.com/odrl:profile:11",
  "permission": [{
    "assigner": "http://example.com/org88",
    "target": {
      "@type": "AssetCollection",
      "source":  "http://example.com/media-catalogue",
      "refinement": [{
        "leftOperand": "runningTime",
        "operator": "lt",
        "rightOperand": { "@value": "60", "@type": "xsd:integer" },
        "unit": "http://qudt.org/vocab/unit/MinuteTime"
      }]
    }

  }]
}
```

json data structure

almost any IT practitionner is able to read through it

roughly: permission to org88 to play assets in collection with running length < 60 min

DPCL: in a nutshell

- **JSON-like syntax**
- with foundational ontological intuitions expressed in
  - LKIF-core and cognitive linguistics: **objects** vs **events**
  - LPS: **transformational rules** vs **reactive rules**
- finer representational granularity given by **Hohfeld’s framework**,
- expressed in **frames** as in FLINT/eFLINT, but with more & simpler frames
- bottom-line informational model rather than a full-fledged formal semantics

DPCL: in a nutshell

- **JSON-like syntax**
- with foundational ontological intuitions expressed in
  - LKIF-core and cognitive linguistics: **objects** vs **events**
  - LPS: **transformational rules** vs **reactive rules**
- finer representational granularity given by **Hohfeld's framework**,
- expressed in **frames** as in FLINT/eFLINT, but with more & simpler frames
- bottom-line informational model rather than a full-fledged formal semantics

- yet, semantics can be partially defined by ***rewriting rules***
- we are exploring an alternative standpoint to the usual types/instances extensional semantics, but more in line to ***qualification*** acts
- we are integrating a conditional ***preferential ordering*** to manage conflicts

DPCL: entities

We follow the common-sensical distinction:

- states: `condition, object, agent`
- (transition) events:
    - primitive events: `#action`
    - production/removal events: `+object`, `-object`
    - qualification/disqualification events: `object in group`, ...

DPCL: parameters and refinements

Any entity can be refined via some parameter, eg. in the case of actions:

```
#give {
    agent: john
    item: apple
    recipient: paul
}

#eat {
    agent: paul
    item: apple
}
```

DPCL: power frame

```
power {
    holder: priest
    action: #marry { patient: [john, paul] }
    consequence: +married(john, paul)
}
```

DPCL: power frame

```
power {
    holder: priest
    action: #marry { patient: [john, paul] }
    consequence: +married(john, paul)
}
```

a power reifies an
(institutional) causal mechanism

- conditioned by **qualification** of agent
- conditioned by **procedure** of action
- affecting a limited **domain of competence**

DPCL: duty frame

```
duty {
   holder: john
   counterparty: university
   action: #teach { recipient: student }
   violation: john.online is False
}
```

DPCL: duty frame

a duty reifies an expectation (of “good”) for the counterparty

```
duty {
    holder: john
    counterparty: university
    action: #teach { recipient: student }
    violation: john.online is False
}
```

DPCL: duty frame

a duty reifies an expectation (of “good”) for the counterparty

```
duty {
    holder: john
    counterparty: university
    action: #teach { recipient: student }
    violation: john.online is False
}
```

sometimes violations may be defined independently of the content of the duty

DPCL: prohibition frame

```
prohibition {
    holder: john
    action: #go { destination: swimming }
    termination: ~winter
}
```

DPCL: prohibition frame

*another example of “**semantic neutrality**”**:** not all logics consider the “prohibition to do A” the same as the “obligation of not doing A”*

```
prohibition {
    holder: john
    action: #go { destination: swimming }
    termination: ~winter
}
```

sometimes normative directives have terminating conditions independent of performance

DPCL: conditioning rules

- Transformational rules (as long as the premise is true, the conclusion is true):

```
raining -> wet
bike -> vehicle
```

- Reactive rules (when the antecedent occurs, the consequent occurs):

```
#rain => +wet
#raise_hand => +bet
```

DPCL: conditioning rules

- Transformational rules (as long as the premise is true, the conclusion is true):

```
raining -> wet
bike -> vehicle
```

- Reactive rules (when the antecedent occurs, the consequent occurs):

```
#rain => +wet
#raise_hand => +bet
```

- Contexts are generally involved in transformational rules:

```
auction -> { #raise_hand => +bet }
```

DPCL, example: library regulation

student or staff can register as member of the library by using their id card.

```
power {
   holder: student | staff
   action: #register { instrument: holder.id_card }
   consequence: holder in member
}
```

DPCL, example: library regulation

any member can borrow a book for a certain time (e.g. 1 month).

```
power {
    holder: member
    action: #borrow { item: book }
    consequence: +borrowing {
        lender: library
        borrower: member
        item: book
        timeout: now() + 1m
    }
}
```

reference to compound, parametrized institutional object

DPCL, example: library regulation

by borrowing, the borrower can be requested in any moment to return the item.

```
borrowing(lender, borrower, item, timeout) {

    power {
        holder: lender
        action: #request_return { item: item }
        consequence: +duty {
            holder: borrower
            counterparty: lender
            action: #return { item: item }
        }
    }
```

compound, parametrized institutional object (other examples: ownership)

DPCL, example: library regulation

the borrower has the duty to return the item within the given date.

```
duty d1 {
    holder: borrower
    counterparty: lender
    action: #return { item: item }
    violation: now() > timeout % illustrative
}
```

DPCL, example: library regulation

if the borrower does not return it, (s)he may be fined.

```
    +d1.violation => +power {
        holder: lender
        action: #fine
        consequence: +fine(borrower, lender)
    }

}
```

reactive conditional

“Lingua franca”, and rewriting

- As the informational model of DPCL covers most common constructs and concepts observable in normative languages, one could in principle:
    - re-specify existing normative directives almost literally
    - utilize **rewriting rules** to re-encode certain constructs into others
    - cross-compile the transformed model into a target “policy” tool (interpreting it according to its own semantics), eg. BGP policies for routing, a deontic reasoner, etc.

Rewriting example: all is about power!

- All conditions (e.g. preconditions, violation, termination) implicitly refers to a power that may (should?) be assigned to someone.
- This is an actual step in **policy operationalization** in administrative settings.

Rewriting example: all is about power!

- Unfolding a violation construct to the power to declare that violation…

```
prohibition p {
    action: #smoke
}
```

```
p -> {
    #smoke => +power {
        holder: *
        action: #declare_violation { item: p }
        consequence: p.violated
    }
}
```

Rewriting example: all is about power!

- More in general any duty comes with two powers: one to declare fulfilment, another one to declare violation.

```
duty d {
    holder: john
    counterparty: paul
    action: #pay
    violation: timeout
}
```

Rewriting example: all is about power!

- More in general any duty comes with two powers: one to declare fulfilment, another one to declare violation.

```
duty d {
    holder: john
    counterparty: paul
    action: #pay
    violation: timeout
}
```

⇨

```
d -> {
     #pay => +power {
          holder: paul
          action: #declare_fulfillment { item: d }
          consequence: d.fulfilled
     }
     timeout => +power {
          holder: paul
          action: #declare_violation { item: d }
          consequence: d.violated
     }
}
```

here we assign these powers to the counterparty, the claimant

Rewriting example: rules as duties & powers

- Transformational rules can be seen not only as “epistemic” duties (about producing knowledge), but also as powers!

```
bike -> vehicle
```

↓ *Logic rules as goals*

```
vehicle :- bike.
```

system ***has*** to make vehicle true if bike is true

Rewriting example: rules as duties & powers

- Transformational rules can be seen not only as “epistemic” duties (about producing knowledge), but also as powers!

```
bike -> vehicle
```

```
bike -> {
    duty {
        holder: *
        action: +vehicle
    }
    power {
        holder: *
        action: #state { item: vehicle }
        consequence: +vehicle
    }
}
```

mandatory view

ability view

Rewriting example: rules as duties & powers

- Transformational rules can be seen not only as “epistemic” duties (about producing knowledge), but also as powers!

```
bike -> vehicle
```

```
bike -> {
    duty {
        holder: *
        action: +vehicle
    }
    power {
        holder: *
        action: #state { item: vehicle }
        consequence: +vehicle
    }
}
```

mandatory view

LESS IMPORTANT IN A SOCIAL COORDINATION SETTING!

ability view

Rewriting example: maintenance duties

- Unfolding maintenance duties (about states of affairs) in terms of duties of actions

maintenance duty

```
duty d1 {
    target: g1
}
```

⇒

achievement duty

```
d1 -> {
    ~g1 -> duty { action: +g1 }
    g1 -> prohibition { action: -g1 }
}
```

avoidance duty

Perspectives

- Working on languages for computational regulatory functions is a highly relevant and urgent topic.
- Very dispersed literature, opinions, standpoints. In the years, new generations of researchers and practitioners often restarted from scratch to try to solve old, partially resolved problems.
- Ideally, as a community, we should start by creating grounds and infrastructures to compare and organize all these experiences.

Perspectives

- Practical standpoint of modelling practitioners (generally not logicians, nor expert programmers) is generally not taken into account.
- Besides, normative systems have characteristics that make them very different from standard computer engineering/science perspectives.
- DPCL started as an experiment in the design of a programming language motivated by these alternative practical requirements. So far, lots of ideas!
- First prototype of interpreter in course of development.