Policy reasoning in data exchange systems (with eFLINT)

L. Thomas van Binsbergen

Informatics Institute, University of Amsterdam
ltvanbinsbergen@acm.org

University of Amsterdam

EFRO-funded: AMDEX Fieldlab – neutral data-exchange infrastructure

Amsterdam Economic Board
SURF
amsix
Universiteit van Amsterdam
deXes
European Union
European Regional Development Fund
Investing in your future
Provincie Noord-Holland

Regulated data exchange:

data exchange systems governed by regulations, agreements and policies

as an instance of

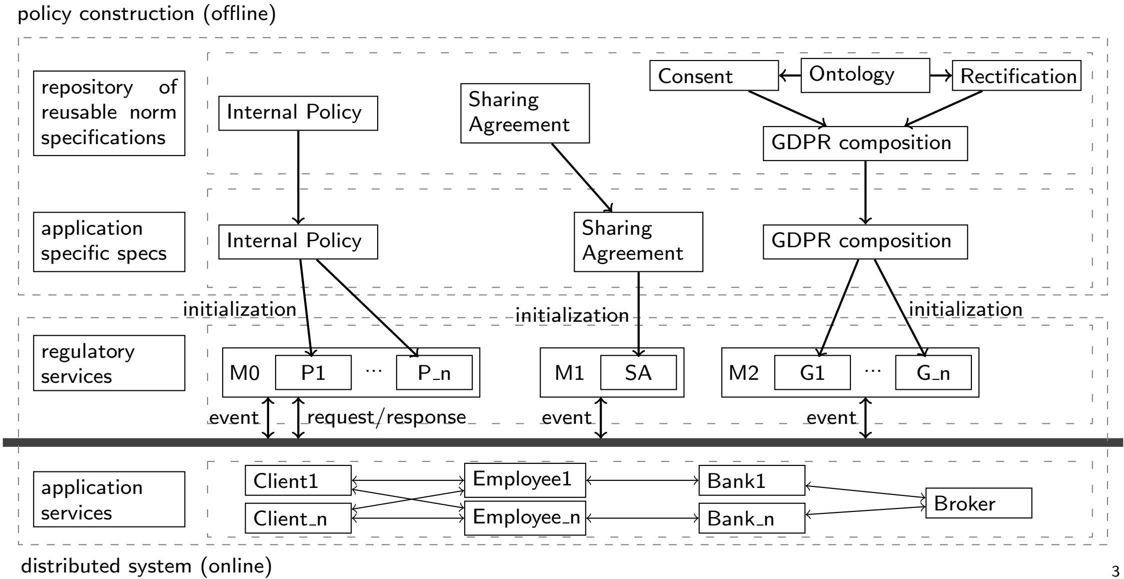
Regulated systems:

distributed software systems with embedded regulatory services derived from norm specifications that monitor and/or enforce compliance

NWO-funded: EPI – Enabling personalized interventions (Medical)

NWO-funded: SSPDDP – Secure and scalable, policy-driven data exchange (Financial)

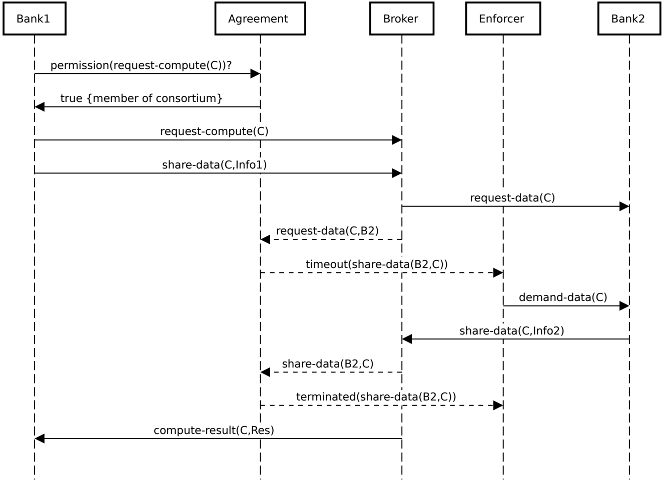
Experiment SSPDDP: Know Your Customer case study



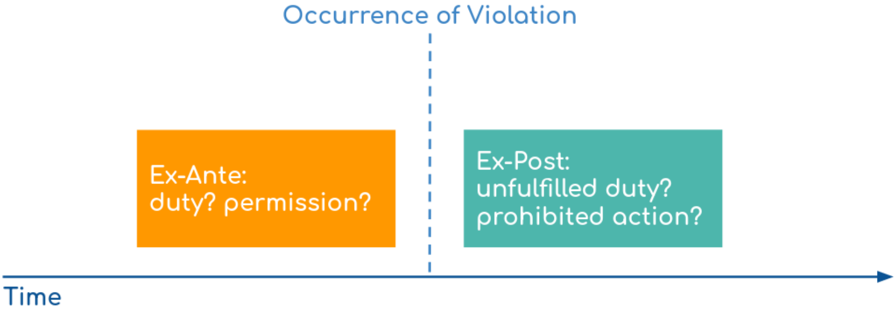
SSPDDP: Dynamic enforcement of sharing agreement

(Article 1) A member of the consortium has the right to request a risk assessment computation from the broker for any (potential) client

(Article 2) The data broker has the power to oblige members of the consortium to share information about any client the member does business with



Types of enforcement: ex-ante and ex-post



Occurrence of Violation

Ex-Ante:
duty? permission?

Ex-Post:
unfulfilled duty?
prohibited action?

Time

Back to basics: Access control and XACML architecture

An access request typically consists of:

- An actor
- An action (e.g., read/write)
- A resource / asset
- Optionally: A context identifier

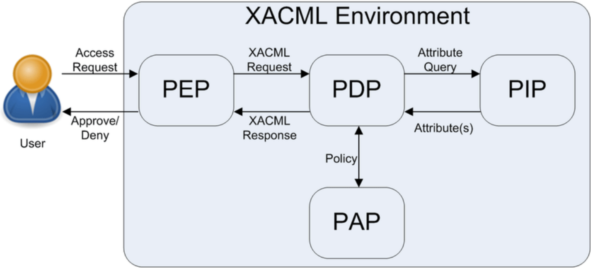


Figure: Simplified XACML architecture – M.S. Ferdous. "User-controlled identity management systems using mobile device". PhD thesis.

```
Fact actor
Fact asset

Act read  Actor actor Related to asset Syncs with access(actor,asset)
Act write Actor actor Related to asset Syncs with access(actor,asset)
```

What does eFLINT offer in addition to (standard) access control

- The language makes a connection between *legal primitives* and *computational primitives* (see upcoming slides),
- including *legal obligations*,
- ex-ante *and* ex-post enforcement of individual requests and entire scenarios (see various examples),
- as well as abstract scenarios and *properties* (experimental), and
- is designed such that specifications are *compositional* and *extensible* (see SSPDDP and DIPG case studies)

Foundational, normative & computational concepts

computational

state

$parent(A, B) = true$

$\ldots$

Foundational, normative & computational concepts

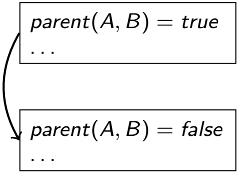
computational

state

$parent(A, B) = true$

$\ldots$

transitions



$parent(A, B) = true$

$\ldots$

$parent(A, B) = false$

$\ldots$

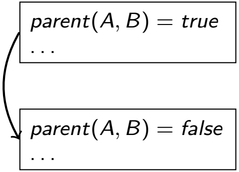
Foundational, normative & computational concepts

computational

state

$parent(A, B) = true$
$\ldots$

transitions



$parent(A, B) = true$
$\ldots$

$parent(A, B) = false$
$\ldots$

- Violations of state and transition

Foundational, normative & computational concepts

deontic

prohibitions

permissions

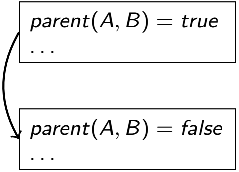
obligations

computational

state

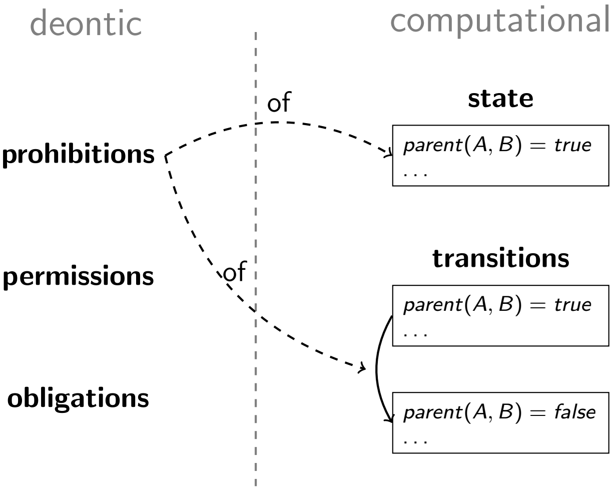
$parent(A, B) = true$
...

transitions



- Violations of state and transition

Foundational, normative & computational concepts



deontic

computational

prohibitions

permissions

obligations

of

of

state

$parent(A, B) = true$
. . .

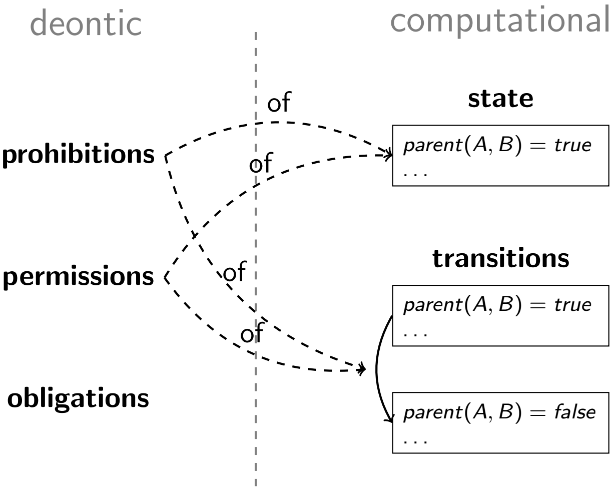
transitions

$parent(A, B) = true$
. . .

$parent(A, B) = false$
. . .

- Violations of state and transition

Foundational, normative & computational concepts



deontic

computational

prohibitions

permissions

obligations

of

of

of

of

state

$parent(A, B) = true$
$\ldots$

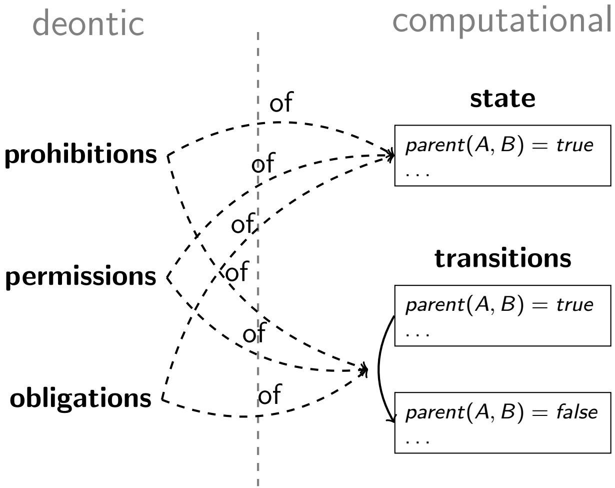
transitions

$parent(A, B) = true$
$\ldots$

$parent(A, B) = false$
$\ldots$

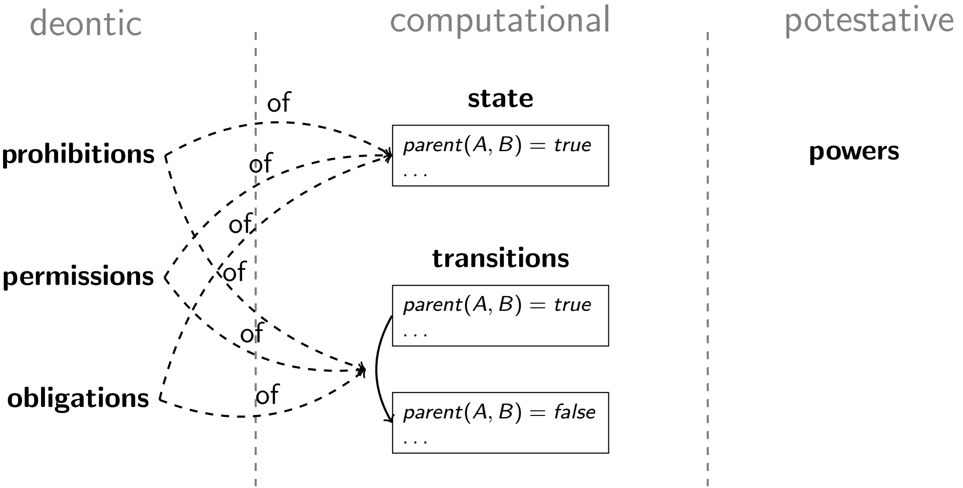
- Violations of state and transition

Foundational, normative & computational concepts



- Violations of state and transition

Foundational, normative & computational concepts



deontic

computational

potestative

prohibitions

permissions

obligations

state

$parent(A, B) = true$
. . .

transitions

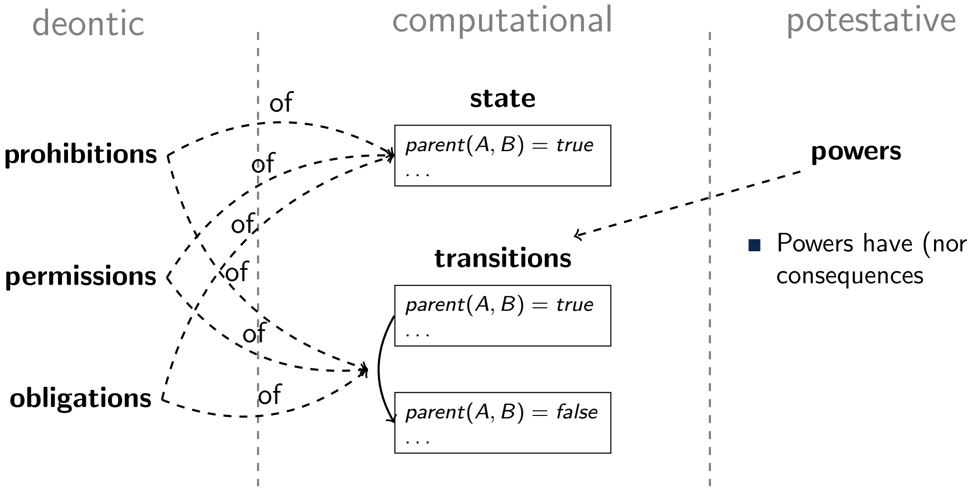
$parent(A, B) = true$
. . .

$parent(A, B) = false$
. . .

powers

- Violations of state and transition

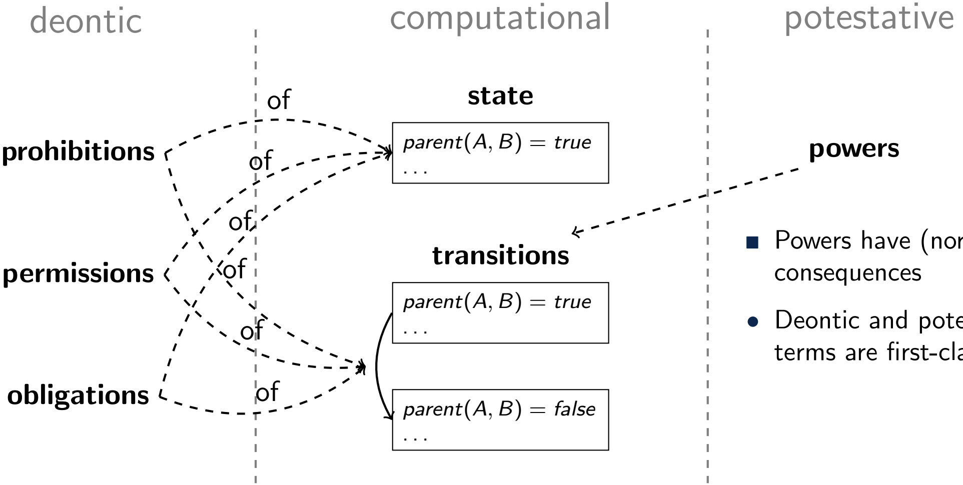
Foundational, normative & computational concepts



- Powers have (normative) consequences

- Violations of state and transition

Foundational, normative & computational concepts



- Powers have (normative) consequences
- Deontic and potestative terms are first-class

- Violations of state and transition

Normative reasoning – scenarios

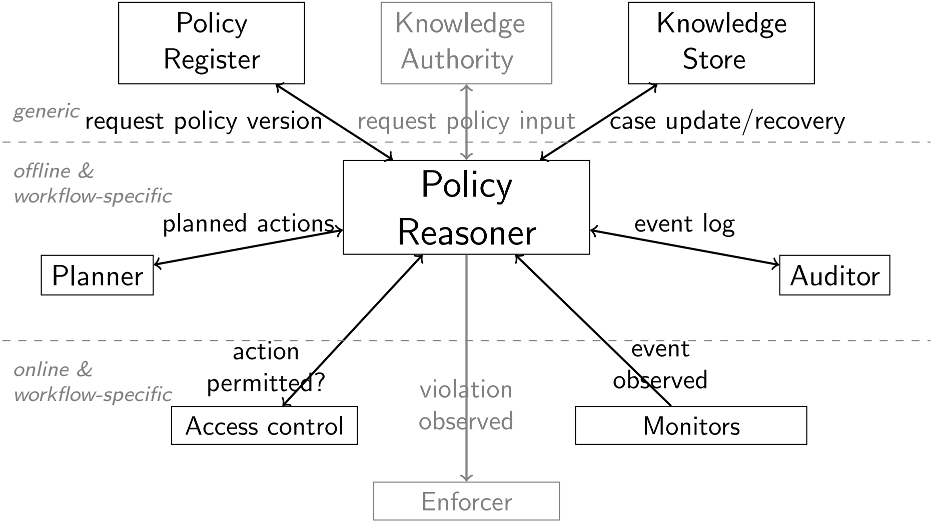
$$\begin{tikzcd}
s_0 \arrow[r, dashed, bend left] & s_i \arrow[r, bend left] & s_{i+1} \arrow[r, dashed, bend left] & s_{i+k}
\end{tikzcd}$$

Types of reasoning

A concrete **scenario** describes a single trace in the transition system

- Static ex-ante/ex-post assessment, of a given scenario {eFLINT 1.0}
- Dynamic assessment, of a given action (sequence) {eFLINT 2.0 }
  - ex-post: *execute* action(s) and report on findings
  - ex-ante: *try* the action(s) and decide based on report whether to perform the action
- Reasoning with abstract scenarios (e.g. planning / property checking) {eFLINT-CHECK}

Policy reasoner integration



Section 1

The eFLINT language

Toy example – knowledge representation

(Toy Article 1) a natural person is a legal parent of another natural person if:

- *they are a natural parent, or*
- *they are an adoptive parent*

```
Fact person Identified by String
Placeholder parent    For person
Placeholder child     For person

Fact natural-parent   Identified by parent * child
Fact adoptive-parent  Identified by parent * child

Fact legal-parent     Identified by parent * child
  Holds when adoptive-parent(parent,child)
          || natural-parent(parent,child)
```

Toy example – powers and duties

> *(Toy Article 2) a child has the power to ask a legal parent for help with their homework, resulting in a duty for the parent to help.*

```
Act ask-for-help
  Actor       child
  Recipient   parent
  Creates     help-with-homework(parent,child)
  Holds when  legal-parent(parent,child)

Duty help-with-homework
  Holder         parent
  Claimant       child
  Violated when  homework-due(child)

Fact homework-due Identified by child

Act help
  Actor       parent
  Recipient   child
  Terminates  help-with-homework(parent,child)
  Holds when  help-with-homework(parent,child)
```

Toy example – scenario / case

```
Fact person Identified by Alice, Bob, Chloe, David
```

Listing 1: Domain specification

```
+natural-parent(Alice, Bob).
+adoptive-parent(Chloe, David).
```

Listing 2: Initial state

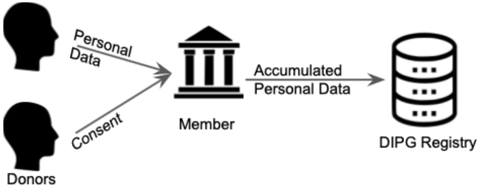
```
ask-for-help(Bob, Alice).                 // permitted: Alice is Bob's legal parent
+homework-due(Bob).                       // homework deadline passed
?Violated(help-with-homework(Alice,Bob)). // query confirms duty is violated
help(Alice,Bob).                          // duty terminated
```

Listing 3: Scenario

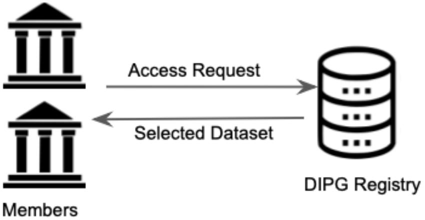
The DIPG case – Compliance questions

According to the GDPR (1) and the DIPG regulatory document (2):

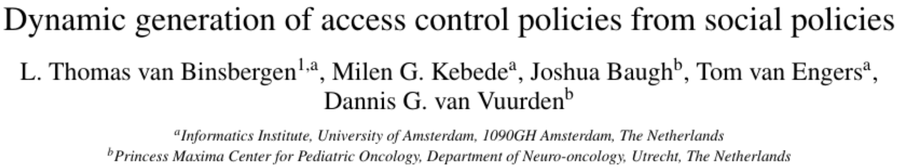
1. What conditions need to be fulfilled by a member before making data available?



2. What conditions need to be fulfilled when accessing (3) data from the registry?



Modular GDPR specification



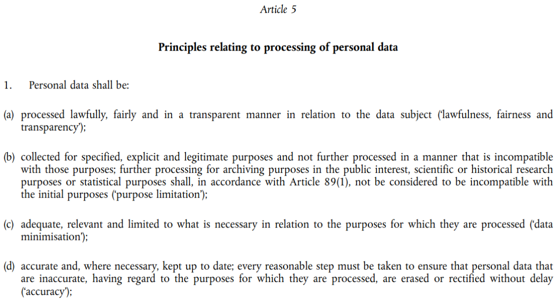
Dynamic generation of access control policies from social policies

L. Thomas van Binsbergen[1,a], Milen G. Kebede[a], Joshua Baugh[b], Tom van Engers[a], Dannis G. van Vuurden[b]

[a] Informatics Institute, University of Amsterdam, 1090GH Amsterdam, The Netherlands
[b] Princess Maxima Center for Pediatric Oncology, Department of Neuro-oncology, Utrecht, The Netherlands

```
Act collect-personal-data
  Actor controller
  Recipient subject
  Related to data, processor, purpose
    Where subject-of(subject, data)
  Creates processes(processor, data, controller, purpose)
```

Article 5 – processing conditions



Article 5

Principles relating to processing of personal data

1. Personal data shall be:

(a) processed lawfully, fairly and in a transparent manner in relation to the data subject ('lawfulness, fairness and transparency');

(b) collected for specified, explicit and legitimate purposes and not further processed in a manner that is incompatible with those purposes; further processing for archiving purposes in the public interest, scientific or historical research purposes or statistical purposes shall, in accordance with Article 89(1), not be considered to be incompatible with the initial purposes ('purpose limitation');

(c) adequate, relevant and limited to what is necessary in relation to the purposes for which they are processed ('data minimisation');

(d) accurate and, where necessary, kept up to date; every reasonable step must be taken to ensure that personal data that are inaccurate, having regard to the purposes for which they are processed, are erased or rectified without delay ('accuracy');

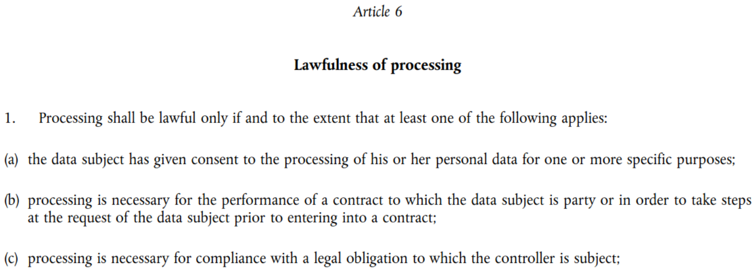
```
Fact minimal-for-purpose Identified by data * purpose
Extend Act collect-personal-data Conditioned by minimal-for-purpose(data, purpose)
```

Listing 4: Member (1c)

```
Fact accurate-for-purpose Identified by data * purpose
Extend Act collect-personal-data Conditioned by accurate-for-purpose(data, purpose)
```

Listing 5: Member (1d)

Article 6 – legal processing



Article 6

Lawfulness of processing

1. Processing shall be lawful only if and to the extent that at least one of the following applies:

(a) the data subject has given consent to the processing of his or her personal data for one or more specific purposes;

(b) processing is necessary for the performance of a contract to which the data subject is party or in order to take steps at the request of the data subject prior to entering into a contract;

(c) processing is necessary for compliance with a legal obligation to which the controller is subject;

```
Fact consent Identified by subject * controller * purpose * data
Extend Act collect-personal-data
  Holds when consent(subject, controller, purpose, data)
```

Listing 6: Member (1a)

```
Fact has-legal-obligation Identified by controller * purpose
Extend Act collect-personal-data
  Holds when has-legal-obligation(controller, purpose)
```

Listing 7: Member (1c)

Compliance Question 1

DIPG Regulatory document – Article 4(2):

> *Members should transfer data to the DIPG registry in a coded form only*

```
Fact coded Identified by dataset

Act make-data-available
  Actor institution
  Related to dataset
  Conditioned by coded(dataset)
  Holds when member(institution)
```

Compliance Question 1

```
Extend Act make-data-available Syncs with (Foreach donor:
  collect-personal-data(controller = institution
                       ,subject    = donor
                       ,data       = dataset
                       ,processor  = "DCOG"
                       ,purpose    = "DIPGResearch")
    When subject-of(donor, dataset))
```

An institution can make a dataset available when (for each donor (subject) in the dataset):

- The institution is a member (DIPG Regulatory Document – Article 4(2))
- Data is coded (DIPG Regulatory Document – Article 4(2))
- Consent is given by *each* donor for data processing by the DCOG for the purpose of DIPGResearch (GDPR – Article 6)
- Data should be accurate for the purpose DIPGResearch (GDPR – Article 5)

Compliance Question 2

```
Extend Act read Holds when (Exists project, institution:
  approved(project,institution) &&
  selected(asset,project) &&
  affiliated(actor, institution))
```

An actor can *read* an asset when (there exists a project and an institution for which):

- The project is approved for the institution
- The asset is selected for the project
- The actor is affiliated with the institution

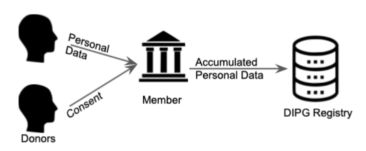
Section 2

Policy reasoning in data exchange systems

eFLINT reasoner as Policy Decision Point

Question 1

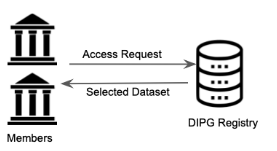
What conditions need to be fulfilled before making data available?



```
?Enabled(write(<X>,<Y>))
```

Question 2

What conditions need to be fulfilled when accessing data from the registry?



```
?Enabled(read(<X>,<Y>))
```

eFLINT reasoner as Policy Administration Point

```
Fact action        Identified by READ, WRITE, DELETE
Fact decision      Identified by PERMIT, DENY
Fact policy-rule   Identified by actor * asset * action * decision
  Derived from policy-rule(read.actor, read.asset, READ, PERMIT) When Enabled(read)
  Derived from policy-rule(write.actor,write.asset,WRITE,PERMIT) When Enabled(write)
```

Listing 8: Deriving policy rules from write and read permissions

```
?--policy-rule(asset = "DIPG.PMC.0001", action=READ)
```

Listing 9: Instance query requesting all READ policies on a given asset

```
policy-rule(SintAntionius, "DIPG.PMC.0001", action=READ, decision=PERMIT)
policy-rule(PMC, "DIPG.PMC.0001", action=READ, decision=PERMIT)
policy-rule(AmsUMC, "DIPG.PMC.0001", action=READ, decision=DENY)
```

Figure: Output produced by the reasoner

Zooming out: what types of reasoning do we ambition?

1. Access control based on laws, regulations and agreements
   Status: Moving from lab to practice in AMdEX fieldlab
   Reasoning: Straightforward scenario/action compliance
   Challenge: Nothing major, practical matters

2. Usage control based on laws, regulations and agreements
   Status: To be investigated in DMI with Surf and service providers (?)
   Reasoning: Straightforward scenario/action compliance
   Challenge: Monitoring, sensitive meta-data

3. Auditing and accountability
   Status: To be investigated in DMI with KPMG
   Reasoning: Straightforward scenario/action compliance
   Challenge: Monitoring, sensitive meta-data, differences in interpretation (disputes)

Zooming out: what types of reasoning do we ambition?

4 Finding and resolving conflicts in laws, regulations and agreements
Status: Ongoing experiments
Reasoning: Abstract scenarios, properties and ‘conformance’
Challenge: Computationally more expensive and complex reasoning

5 Planning of processing activities (e.g., workflows)
Status: To be investigated in DMI
Reasoning: Abstract scenarios, properties and ‘search’
Challenge: Computationally more expensive and complex reasoning

Policy reasoning in data exchange systems (with eFLINT)

L. Thomas van Binsbergen

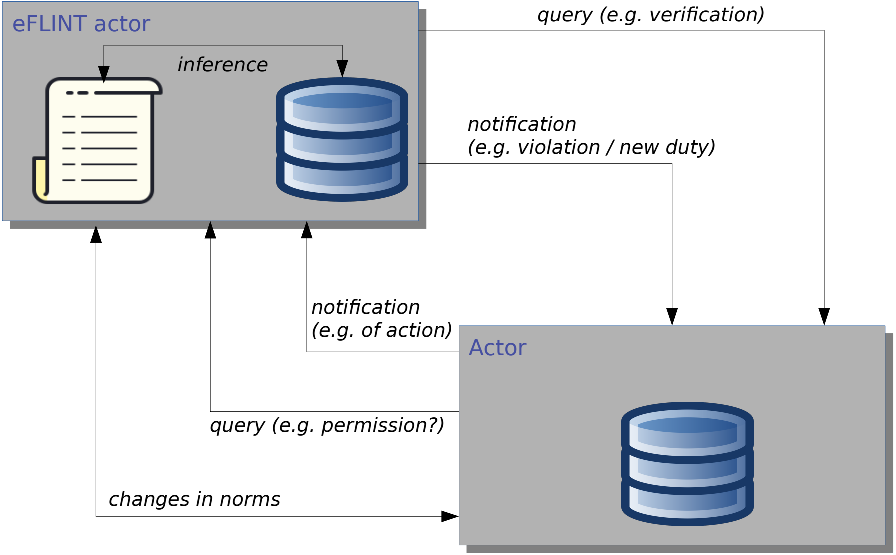
Informatics Institute, University of Amsterdam
ltvanbinsbergen@acm.org

University of Amsterdam

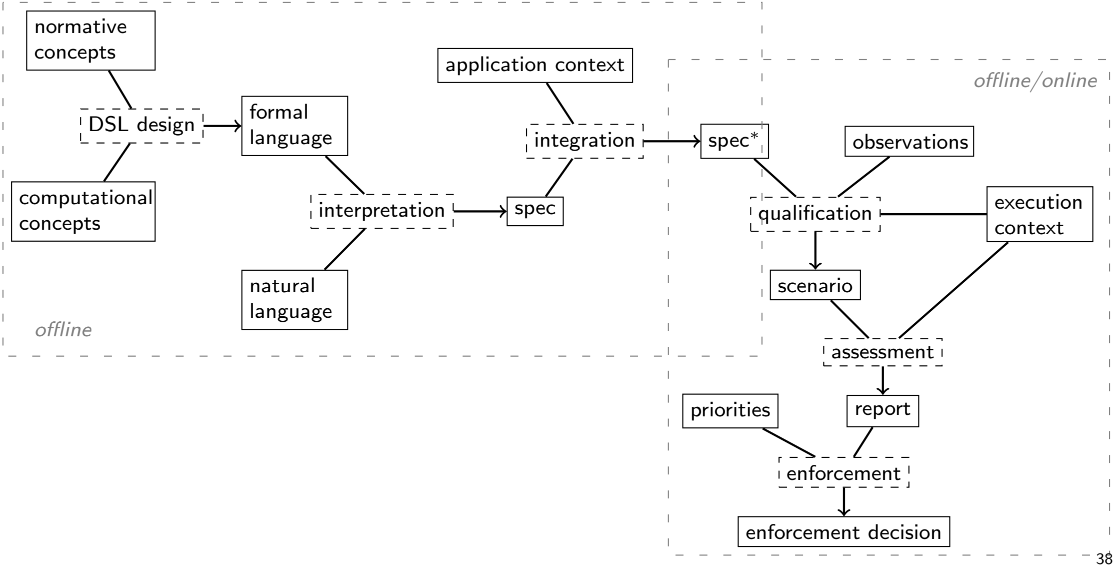
EFRO-funded: AMDEX Fieldlab – neutral data-exchange infrastructure

Amsterdam Economic Board
SURF
amsix
Universiteit van Amsterdam
deXes
European Union
European Regional Development Fund
Investing in your future
Provincie Noord-Holland

eFLINT actors



Normative reasoning – information flow



State of development

1. `haskell-implementation`
   - Reference implementation of eFLINT DSL
   - `eflint-repl`: interpreter (debugging, running scenarios and tests)
   - `eflint-server`: TCP server (dynamic assessment)
   - Formal syntax / semi-formal operational semantics

State of development

1. `haskell-implementation`
   - Reference implementation of eFLINT DSL
   - `eflint-repl`: interpreter (debugging, running scenarios and tests)
   - `eflint-server`: TCP server (dynamic assessment)
   - Formal syntax / semi-formal operational semantics
2. `java-implementation`
   - TCP client
   - HTTP server
   - rudimentary EDSL for accessing `eflint-server`

State of development

1. `haskell-implementation`
   - Reference implementation of eFLINT DSL
   - `eflint-repl`: interpreter (debugging, running scenarios and tests)
   - `eflint-server`: TCP server (dynamic assessment)
   - Formal syntax / semi-formal operational semantics
2. `java-implementation`
   - TCP client
   - HTTP server
   - rudimentary EDSL for accessing `eflint-server`
3. `scala-implementation`
   - eFLINT actors in the actor-oriented Akka framework

State of development

1. `haskell-implementation`
   - Reference implementation of eFLINT DSL
   - `eflint-repl`: interpreter (debugging, running scenarios and tests)
   - `eflint-server`: TCP server (dynamic assessment)
   - Formal syntax / semi-formal operational semantics
2. `java-implementation`
   - TCP client
   - HTTP server
   - rudimentary EDSL for accessing `eflint-server`
3. `scala-implementation`
   - eFLINT actors in the actor-oriented Akka framework
4. Development environments
   - Jupyter notebooks
   - Various experimental web-applications
   - FLINT editor

Goals for eFLINT 3.0

Language design

- Clear separation between:
  - Computational concepts: actions, events, synchronisation
  - Normative concepts: prohibition, obligation, permission, power
- (eFLINT 2.0 can serve as a core/inner language to eFLINT 3.0)
- A module system, introducing namespaces and a versioning mechanism
- Modular, rule-based specification as the default through implicit extensions

Language engineering

- Additional static analyses to detect inconsistencies and possible errors
- Detailed reports as part of reasoning output to improve explainability
- User-friendly programming environment for writing and testing specifications
- Interoperability, e.g. with linked data / semantic web

FAQ

> *eFLINT is just defining a transition system with some extra conditions lying on top*

– PL expert

> *Law is subject to interpretation and has (deliberate) open terms*

– Legal expert

> *eFLINT is still too difficult to use*

– Legal expert

Takeaway messages

At the University of Amsterdam, we are experimenting with approaches to enforcing laws, regulations, agreements and contracts in (distributed) systems

Takeaway messages

At the University of Amsterdam, we are experimenting with approaches to enforcing laws, regulations, agreements and contracts in (distributed) systems

The eFLINT DSL serves as a tool to demonstrate and experiment with various aspects of our approach, with a focus on runtime enforcement using 'regulatory services'

Takeaway messages

At the University of Amsterdam, we are experimenting with approaches to enforcing laws, regulations, agreements and contracts in (distributed) systems

The eFLINT DSL serves as a tool to demonstrate and experiment with various aspects of our approach, with a focus on runtime enforcement using 'regulatory services'

We are currently working on a prototype to demonstrate our approach in data exchange systems such as the Amsterdam Data Exchange (AMdEX)

Takeaway messages

At the University of Amsterdam, we are experimenting with approaches to enforcing laws, regulations, agreements and contracts in (distributed) systems

The eFLINT DSL serves as a tool to demonstrate and experiment with various aspects of our approach, with a focus on runtime enforcement using ‘regulatory services’

We are currently working on a prototype to demonstrate our approach in data exchange systems such as the Amsterdam Data Exchange (AMdEX)

These experiments highlight the importance of software engineering concepts such as modularity, reuse, version control, overriding/overloading mechanisms and inheritance

Takeaway messages

At the University of Amsterdam, we are experimenting with approaches to enforcing laws, regulations, agreements and contracts in (distributed) systems

The eFLINT DSL serves as a tool to demonstrate and experiment with various aspects of our approach, with a focus on runtime enforcement using 'regulatory services'

We are currently working on a prototype to demonstrate our approach in data exchange systems such as the Amsterdam Data Exchange (AMdEX)

These experiments highlight the importance of software engineering concepts such as modularity, reuse, version control, overriding/overloading mechanisms and inheritance

The next phase is to improve the practicality and usability of eFLINT through higher-level abstractions, (domain-specific) editors, static analyses, and explainability

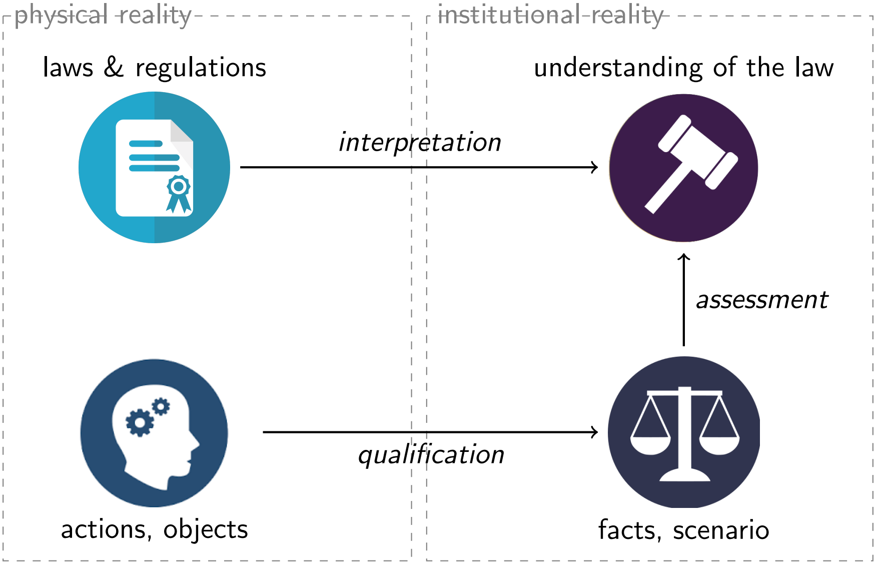
Policy reasoning in data exchange systems (with eFLINT)

L. Thomas van Binsbergen

Informatics Institute, University of Amsterdam
ltvanbinsbergen@acm.org

University of Amsterdam

University of Amsterdam



physical reality

laws & regulations

actions, objects

institutional reality

understanding of the law

facts, scenario

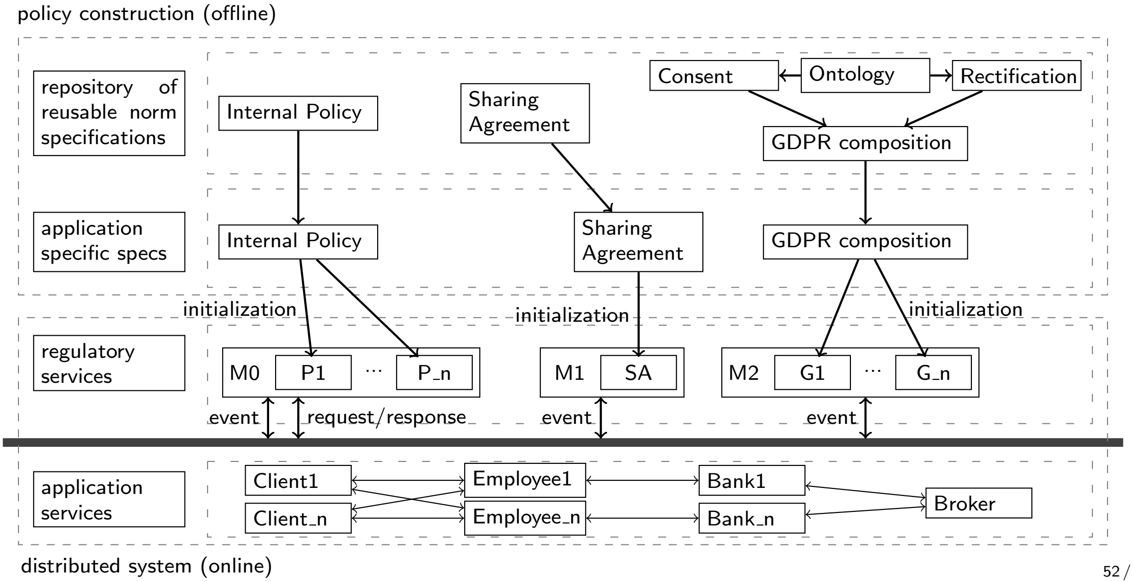
interpretation

qualification

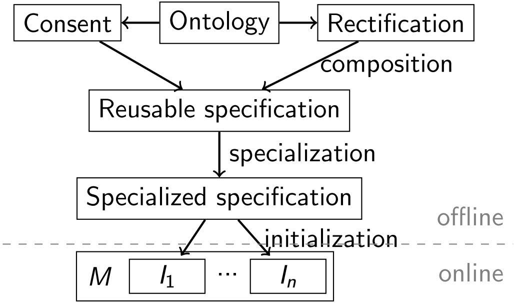
assessment

"If the facts are against you, argue the law. If the law is against you, argue the facts. If the law and the facts are against you, pound the table ..." -Carl Sandburg

Experiment SSPDDP: Know Your Customer case study



eFLINT integration – overview (GDPR example)



eFLINT integration – example

Reusable GDPR concepts

```
Fact controller
Fact subject

Fact data
Fact subject-of
 Identified by subject * data
```

Specialisation to application

```
Fact bank   //exactly one
Fact client //exactly one

Fact controller
 Derived from bank
Fact subject
 Derived from client

Fact data
 Identified by Int

Event data-change
 Terminates data
 Creates data(data + 1)

Fact subject-of
 Derived from
  subject-of(client,processed)
 ,subject-of(client,data)

Fact processed
 ...
```

Instantiation at run-time

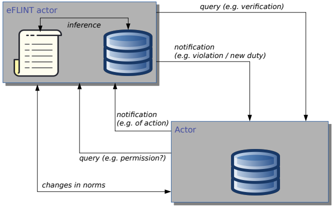
```
+bank(GNB).
+client(Alice).
+data(0).
```

Derived after instantiation

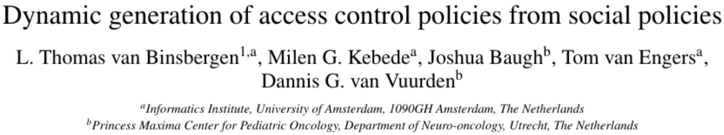
```
+controller(GNB).
+subject(Alice).
+subject-of(Alice,0).
```

Two approaches to enforcing norms

Embedding eFLINT specifications as eFLINT actors, akin to ‘policy decision point’:



Generating system-level policies, akin to ‘policy administration point’



Dynamic generation of access control policies from social policies

L. Thomas van Binsbergen[1,a], Milen G. Kebede[a], Joshua Baugh[b], Tom van Engers[a], Dannis G. van Vuurden[b]

[a] Informatics Institute, University of Amsterdam, 1090GH Amsterdam, The Netherlands
[b] Princess Maxima Center for Pediatric Oncology, Department of Neuro-oncology, Utrecht, The Netherlands