WP3: Goals, results and plans

L. Thomas van Binsbergen[1], Lu-Chi Liu[2], Tom van Engers[23]

[1]Centrum Wiskunde & Informatica
l.t.van.binsbergen@cwi.nl

[2]University of Amsterdam

[3]Leibniz Institute, University of Amsterdam / TNO

September 2020

UNIVERSITEIT VAN AMSTERDAM

CWI

Section 1

Goals: norm-aware, distributed software systems

Data exchange systems governed by regulations, contracts and policies

as an application of

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

as an application of

Multi-agent systems in which agents form interpretations of norms and may decide to comply with the norms or to monitor and enforce the compliance of other agents

- NWO-funded projects:
  - DL4LD, Logistics (UvA, TNO, TKI Dinalog, ...)
  - SSPDDP, Data Processing (UvA, CWI, VU, ING, ABN AMRO, AirFrance KLM)
  - EPI, Personalized Interventions (UvA, CWI, VU, St. Antonius, UMC Utrecht, ...)

- Calculemus-FLINT project – Trust in the digital government
(UvA, TNO, ICTU, CWI, Ministry of Justice and Security, Ministry of Finance, Immigration and Naturalisation Service (IND), ...)

- NWO-funded projects:
  - DL4LD, Logistics (UvA, TNO, TKI Dinalog, ...)
  - SSPDDP, Data Processing (UvA, CWI, VU, ING, ABN AMRO, AirFrance KLM)
  - EPI, Personalized Interventions (UvA, CWI, VU, St. Antonius, UMC Utrecht, ...)

- Calculemus-FLINT project – Trust in the digital government
(UvA, TNO, ICTU, CWI, Ministry of Justice and Security, Ministry of Finance, Immigration and Naturalisation Service (IND), ...)

Contributions related to SSPDDP WP3

- DL4LD: Monitoring and auditing in distributed architectures
- EPI: Controlling access to assets (such as health records)
- SSPDDP: High-level norm specification and integration in lower-level architectures
- Calculemus-FLINT: Publication of specifications and natural language processing

Core properties of the envisioned architectures

- Distributed systems embed *regulatory services* that perform:
  - control • enforcement • monitoring • diagnosis
- Explicit *interpretations* of norms written as normative specifications in a high-level, domain-specific language
  - laws • regulations • policies • contracts • codes of conduct
- Multiple normative specifications can apply simultaneously, each having its own (set of) software component(s) implementing regulatory services
- The components can be dynamically updated to new versions of norms
- Simplified auditing through tracing and explainability

Core properties of normative specifications

- Formal *interpretation* of norms in terms of *duties* and *powers*
- Actions or events modify such *normative positions*
- Actors are in *normative relations* with each other:
  - power-liability • duty-claim
- Explicit *qualification* of system-level events as policy-level events

Section 2

Results: Language & Tools

Example – internal policy (1)

An employee can assign a risk to a client which is at least as high as the risk computed by the bank's risk assessment algorithm

```
Actor bank
Actor employee
Actor client

Fact risk              Identified by 0,1,2 // LOW, MEDIUM, HIGH
Fact computed-risk-of  Identified by bank * client * risk

Placeholder assigned-risk For risk

Act assign-risk
  Actor employee
  Recipient bank
  Related to client, assigned-risk
  Terminates perform-risk-assessment(employee',bank,client)
  Holds when computed-risk-of(bank,client,risk) && assigned-risk >= risk
```

Example – internal policy (2a)

An employee has to perform risk analysis within a certain time frame, when ...

```
Duty perform-risk-assessment
 Holder employee
 Claimant bank
 Related to client
 Violated when undue-delay-in-assessment()

Fact undue-delay-in-assessment Identified by employee * client

Event assessment-delay
  Related to client
  Creates undue-delay-in-assessment() When perform-risk-assessment()
  Holds when perform-risk-assessment()
```

Example – internal policy (2b)

An employee has to perform risk analysis within a certain time frame, when the client is new or when certain values of the client profile have been updated

```
Event new-client
  Related to client
  Creates perform-risk-assessment()

Event updated-profile
  Related to client
  Creates perform-risk-assessment()
  Terminates undue-delay-in-assessment()
```

Case assessment demo

scenario

```
new-client(Chloe).
assign-risk(employee=Ellis, client=Chloe, assigned-risk=1).

updated-profile(Chloe).
-computed-risk-of(client=Chloe, risk=1).
+computed-risk-of(client=Chloe, risk=2).

assessment-delay(Chloe).
//assign-risk(employee=Ellis, client=Chloe, assigned-risk=1).
assign-risk(employee=Ellis, client=Chloe, assigned-risk=2).
```

Run | Reset | Save | model name

response

```
* Duty violated at step 6
    ("Ellis":employee,"GNB":bank,"Chloe":client):perform-risk-assessment
```

eFLINT – foundations

- eFLINT is a high-level, domain-specific language for formalizing norms from a variety of sources
- (Institutional) facts, duties and powers are *fluents*, changing over time due to the effects of actions and events
- A specification is a sequence of type declarations.
- The type declarations give rise to a transition system
- Transitions are triggered by 'input events' and produce 'output events'
- There are only implicit references to time, and references are always to "now". The effects of actual time (in a running system) are triggered by input events

Applications

- Automatic **case assessment** and dispute resolution
  - Present: web interface and command-line tool for case assessment
  - Future: web interface with ‘diffs’ between interpretations and scenarios

Applications

- Automatic **case assessment** and dispute resolution
  - Present: web interface and command-line tool for case assessment
  - Future: web interface with 'diffs' between interpretations and scenarios
- Policy **design** through scenario exploration
  - Present: REPL command-line interface (with backtracking)
  - Future: tools for quering the set of scenarios that satisfy certain properties

Applications

- Automatic **case assessment** and dispute resolution
  - Present: web interface and command-line tool for case assessment
  - Future: web interface with 'diffs' between interpretations and scenarios
- Policy **design** through scenario exploration
  - Present: REPL command-line interface (with backtracking)
  - Future: tools for quering the set of scenarios that satisfy certain properties
- Policy **verification**
  - Present: runtime checking of invariants
  - Future: model checking safety and liveness properties

Applications

- Automatic **case assessment** and dispute resolution
  - Present: web interface and command-line tool for case assessment
  - Future: web interface with 'diffs' between interpretations and scenarios
- Policy **design** through scenario exploration
  - Present: REPL command-line interface (with backtracking)
  - Future: tools for quering the set of scenarios that satisfy certain properties
- Policy **verification**
  - Present: runtime checking of invariants
  - Future: model checking safety and liveness properties
- Agent-based modelling and **multi-agent** programming with NBDI agents
  - Present: NBDI design stage
  - Future: implementations and case studies (Scala Akka)

Applications

- Automatic **case assessment** and dispute resolution
  - Present: web interface and command-line tool for case assessment
  - Future: web interface with 'diffs' between interpretations and scenarios
- Policy **design** through scenario exploration
  - Present: REPL command-line interface (with backtracking)
  - Future: tools for quering the set of scenarios that satisfy certain properties
- Policy **verification**
  - Present: runtime checking of invariants
  - Future: model checking safety and liveness properties
- Agent-based modelling and **multi-agent** programming with NBDI agents
  - Present: NBDI design stage
  - Future: implementations and case studies (Scala Akka)
- Control, enforcement, monitoring and diagnosis in **distributed software systems**
  - Present: control and enforcement using regulatory actors (Scala Akka)
  - Future: monitoring and diagnosis

From eFLINT specifications to (Akka) actors

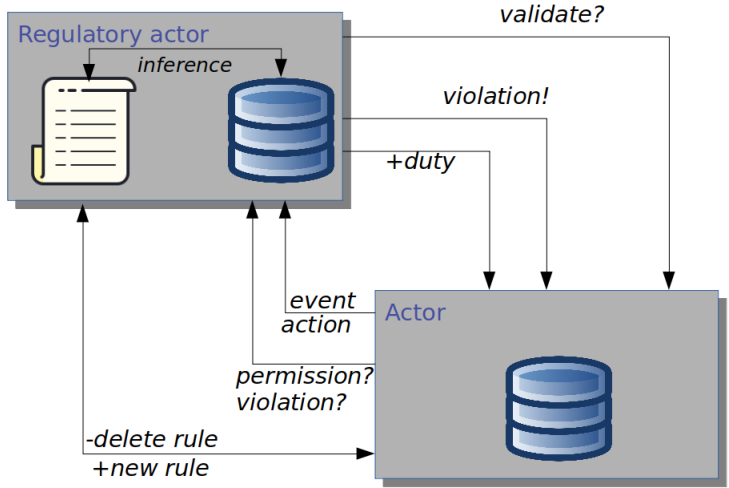
idea: let special 'regulator actors' execute eFLINT specifications

Incoming messages correspond to input events

- Creating/terminating facts and triggering actions and events
  - Dynamic scenario (case) construction with automated assessment
- Creating, modifying or removing fact-, act-, event- and duty-types
  - Dynamic policy construction
- Queries (checking for permissions, powers and (violated) duties)

Outgoing messages correspond to output events

- Notifying actors of (new) permissions and powers they possess or are recipient of
- Notifying performing and recipient actors of executed, prohibited actions
- Notifying actors of (new) duties they hold or are claimant of
- Notifying actors of violated duties they hold or are claimant of
- Querying an actor to determine or validate the truth of a fact

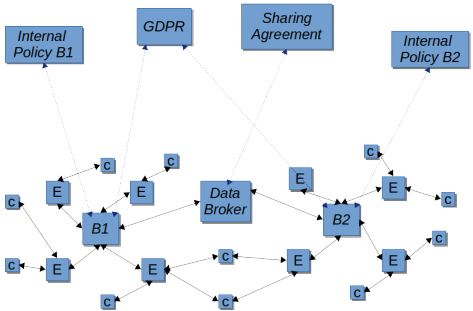

Regulatory actor
inference
validate?
violation!
+duty
event
action
Actor
permission?
violation?
-delete rule
+new rule

Section 3

KYC demo

actor-oriented programming:
Actor-role abstractions (types) are instantiated by actors. Actors have a private state and communicate through message-passing. Actors execute concurrently, always in response to an incoming message.

Akka is a toolkit for building highly concurrent, distributed, and resilient message-driven applications for Java and Scala – `https://akka.io`



Ex-ante enforcement for checking permission

Norm: GDPR (societal level)
The data controllers are prohibited from collecting personal data for a particular purpose without explicit consent to collect the data for that purpose

Actor: Is there permission to...? Regulatory actor: Yes/No + reasons
Employee: Can I record the client's data for KYC risk analysis? (i.e. collect personal data) Regulatory actor: No permission because there is no consent given for this purpose
Client: Can the bank record my data? Regulatory actor: Yes, but only for the purpose of KYC risk analysis

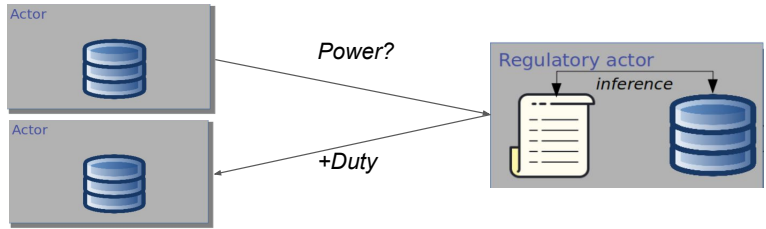
Ex-post enforcement of violations of prohibitions

Norm: Internal policy (bank-level)
An employee can assign a risk to a client which is at least as high as the risk computed by the bank's risk assessment algorithm
→ if employee gives lower risk value, it is a violation

Event: Risk analysis required for client C[SBI:Chemical, Country:NLD, risk:MEDIUM] Employee: Assigns HIGH risk value
Event: Risk analysis required for client C[SBI:Chemical, Country:NLD, risk:MEDIUM] Employee: Assigns LOW risk value Regulatory actor: Informs relevant actors (employee, bank) of the violation

Positive duties

Norm: GDPR (societal level)
The data subject has the right to demand the rectification of personal data, producing a duty on controllers and by derivation on processors

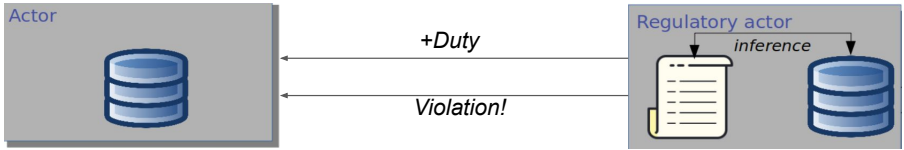


Event: The client's business operations change
Client: Can I demand rectification of my personal data?
Regulatory actor: Yes, the bank holds inaccurate data
Client: Informs employee of the bank of their new operations (SBI) and demands rectification
Regulatory actor: Notifies the bank of their duty to rectify the SBI code
Employee: Updates client profile
Regulatory actor: Terminates the duty (no violation)

Positive duties

Norm: Sharing agreement (consortium-level)

- A bank can request "compute" from the broker by sending a risk analysis algorithm.
- Other banks have to send client information to the broker within a certain time frame (configurable) after broker's request



Bank A asks the broker to compute risk for client C (exercises their right)
Broker requests data from all other banks
Regulatory actor informs the other banks of their duty to share (ex-ante enforcement)
Bank B decides not to comply because some information is too sensitive
Event: The time window for complying with the duty closes
Regulatory actor informs relevant actors (holder,claimant,enforcer) of B's violation (ex-post)

KYC demo code fragment

An employee has to perform risk analysis within a certain time frame, when ...

```
case NewDuty(h,c,duty) ⇒ {
  if ( c == self && duty.fact_type.equals("perform-risk-assessment")) {
    Behaviors.withTimers { timers ⇒
      val client = ...
      timers.startSingleTimer(DelayedRiskAnalysis(client), 1s)
      listen(policy_ref, ...)
    }
  }
  else ...
```

Bank actor responding to new duty message

```
case m:DelayedRiskAnalysis ⇒ {
  policy_ref ! trigger_assessment_delay(m.client)
  listen(policy_ref, ...)
}
```

Bank actor responding to delayed risk analysis message

Akka demo

```
[info] 11:59:01.442 [BankingKYCexperiments-akka.actor.default-dispatcher-8] INFO banking.Client - Sanchez applied to BankA
[info] 11:59:01.443 [BankingKYCexperiments-akka.actor.default-dispatcher-9] INFO banking.Client - George applied to BankA
[info] 11:59:01.444 [BankingKYCexperiments-akka.actor.default-dispatcher-8] INFO banking.Client - Sanchez applied to BankB
[info] 11:59:01.444 [BankingKYCexperiments-akka.actor.default-dispatcher-9] INFO banking.Client - George applied to BankB
[info] 11:59:01.470 [BankingKYCexperiments-akka.actor.default-dispatcher-9] INFO banking.Bank - bank BankB received risk LOW from BankB-Employee
[info] 11:59:01.470 [BankingKYCexperiments-akka.actor.default-dispatcher-8] INFO banking.Bank - bank BankA received risk LOW from BankA-Employee
[info] 11:59:01.489 [BankingKYCexperiments-akka.actor.default-dispatcher-9] INFO banking.Bank - bank BankB received risk LOW from BankB-Employee
[info] 11:59:01.489 [BankingKYCexperiments-akka.actor.default-dispatcher-8] INFO banking.Bank - bank BankA received risk LOW from BankA-Employee
[info] eFLINT instance for "src/eflint/policy.eflint" ready at port 8998 with PID 10321
[info] eFLINT instance for "src/eflint/policy.eflint" ready at port 8999 with PID 10320
[info] eFLINT instance for "src/eflint/gdpr.eflint" ready at port 9000 with PID 10319
[info] 11:59:02.401 [BankingKYCexperiments-akka.actor.default-dispatcher-5] INFO banking.Bank - OOPS! employee of BankB assigned risk LOW which is too low
[info] 11:59:02.465 [BankingKYCexperiments-akka.actor.default-dispatcher-7] INFO banking.Bank - OOPS! employee of BankA assigned risk LOW which is too low
[info] 11:59:03.324 [BankingKYCexperiments-akka.actor.default-dispatcher-5] INFO banking.Bank - OOPS! employee of BankB failed to assess the risk of client(akka://
BankingKYCexperiments/user/ClientGeorge#-1609507467)
[info] 11:59:03.456 [BankingKYCexperiments-akka.actor.default-dispatcher-8] INFO banking.Bank - OOPS! employee of BankA failed to assess the risk of client(akka://
BankingKYCexperiments/user/ClientGeorge#-1609507467)
```

Section 4

Plans and Future Work

An overview of architecture requirements

strategic layer:
construction and publication of policies,
meta-level rules specifying what policies are loaded,
how they are specialized within system environment, and
how policies are composed and any conflicts resolved

tactical layer:
interfaces between regulatory services and other system components,
the relations between events from different policies and
the effects of policy-level events such as (violations of) prohibitions, duties and powers

operational/implementation layer:
specification of nodes, credentials and system-level events,
the interpretation of system-level events as policy-level events, and
blockchain to record/publish events for auditing and diagnosis

Immediate targets

• An interface specification language that restricts the input and output messages of regulator actors, based on cryptographic keys, credentials and other node attributes.
Inspiration: the KeyNote trust management system (Matt Blaze et al.)

• Monitoring actors that interpret system-level events as policy-level events by sending action and event triggers to regulatory actors

• Translating eFLINT specifications to regulatory actors that run on the blockchain.
Possible targets: Solidity/EVM (or IELE VM), Plutus Core (Cardano), DAML or Ergo

WP3: Goals, results and plans

L. Thomas van Binsbergen[1], Lu-Chi Liu[2], Tom van Engers[23]

[1]Centrum Wiskunde & Informatica
l.t.van.binsbergen@cwi.nl

[2]University of Amsterdam

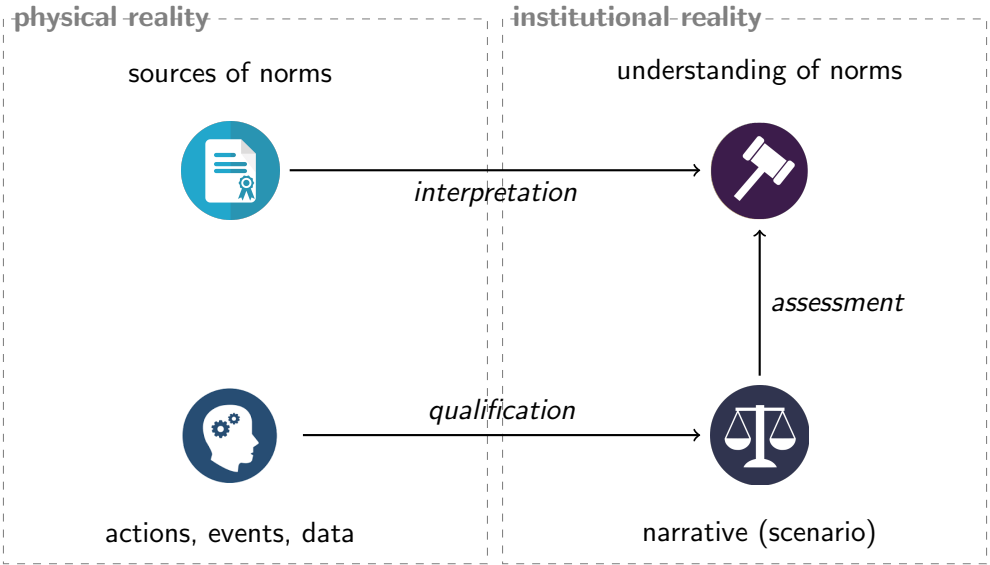
[3]Leibniz Institute, University of Amsterdam / TNO

September 2020

UNIVERSITEIT VAN AMSTERDAM

CWI

Realities


physical reality
institutional reality
sources of norms
understanding of norms
interpretation
assessment
qualification
actions, events, data
narrative (scenario)

Producing normative actors

Legal analyst / policy expert

Produces a semi-formal interpretation of relevant sources (e.g. using the FLINT language) in terms of (Hohfeldian) power-liability and duty-claim relations between actor roles, possibly aided by natural language processing and/or editorial software.

Producing normative actors

Legal analyst / policy expert

Produces a semi-formal interpretation of relevant sources (e.g. using the FLINT language) in terms of (Hohfeldian) power-liability and duty-claim relations between actor roles, possibly aided by natural language processing and/or editorial software.

Software engineer

Formalizes the semi-formal interpretation produced by the legal analyst in a high-level, domain-specific language (e.g. using the eFLINT language). The resulting interpretation can be analyzed with formal verification techniques (e.g. consistency and safety checks) and can be used to assess and compare concrete scenarios.

Producing normative actors

Legal analyst / policy expert

Produces a semi-formal interpretation of relevant sources (e.g. using the FLINT language) in terms of (Hohfeldian) power-liability and duty-claim relations between actor roles, possibly aided by natural language processing and/or editorial software.

Software engineer

Formalizes the semi-formal interpretation produced by the legal analyst in a high-level, domain-specific language (e.g. using the eFLINT language). The resulting interpretation can be analyzed with formal verification techniques (e.g. consistency and safety checks) and can be used to assess and compare concrete scenarios.

All interpretations are stored modularly, with references to sources, and under version control.

Producing normative actors

Legal analyst / policy expert

Produces a semi-formal interpretation of relevant sources (e.g. using the FLINT language) in terms of (Hohfeldian) power-liability and duty-claim relations between actor roles, possibly aided by natural language processing and/or editorial software.

Software engineer

Formalizes the semi-formal interpretation produced by the legal analyst in a high-level, domain-specific language (e.g. using the eFLINT language). The resulting interpretation can be analyzed with formal verification techniques (e.g. consistency and safety checks) and can be used to assess and compare concrete scenarios.

All interpretations are stored modularly, with references to sources, and under version control.

Application as normative actors

A specific version of a formal interpretation is concretized based on configuration options.
The concrete interpretation is compiled to the source code of a normative actor.
The normative actor is dynamic in that it can receive policy updates.

Actor-role abstraction

object-oriented programming:
Class abstractions (types) are instantiated to objects. Objects have a private state and communicate information through method calls. An object relinquishes execution control when calling a method of another object.

actor-oriented programming:
Actor-role abstractions (types) are instantiated by actors. Actors have a private state and communicate through message-passing. Actors execute concurrently, always in response to an incoming message.

Actor-role abstraction

object-oriented programming:
Class abstractions (types) are instantiated to objects. Objects have a private state and communicate information through method calls. An object relinquishes execution control when calling a method of another object.

actor-oriented programming:
Actor-role abstractions (types) are instantiated by actors. Actors have a private state and communicate through message-passing. Actors execute concurrently, always in response to an incoming message.

Akka is a toolkit for building highly concurrent, distributed, and resilient message-driven applications for Java and Scala – `https://akka.io`

Actor-role abstraction

object-oriented programming:
Class abstractions (types) are instantiated to objects. Objects have a private state and communicate information through method calls. An object relinquishes execution control when calling a method of another object.

actor-oriented programming:
Actor-role abstractions (types) are instantiated by actors. Actors have a private state and communicate through message-passing. Actors execute concurrently, always in response to an incoming message.

Akka is a toolkit for building highly concurrent, distributed, and resilient message-driven applications for Java and Scala – `https://akka.io`

agent-oriented programming:
Actor-oriented programming in which the actors (called agents) have mental qualities, such as beliefs, desires and intentions, and in which only certain kinds of messages are used, such as requests, offers, declines and promises