Languages for prototyping regulated systems: A case study

L. Thomas van Binsbergen Lu-Chi Liu Mostafa Mohajeri Parizi

Informatics Institute, University of Amsterdam
ltvanbinsbergen@acm.org

December 9, 2020

Section 1

Regulated systems

Software development

Monolithic programs

Software development

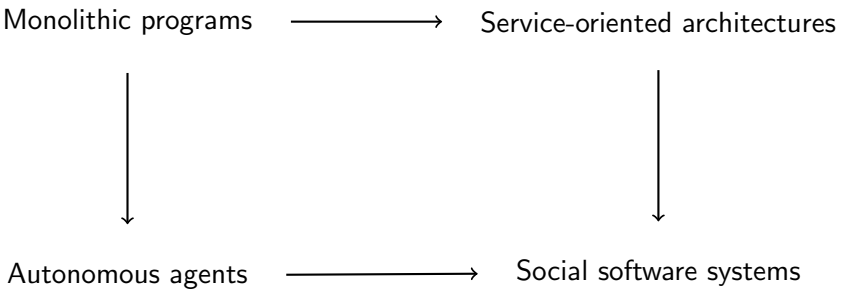
Monolithic programs $\longrightarrow$ Service-oriented architectures

Software development

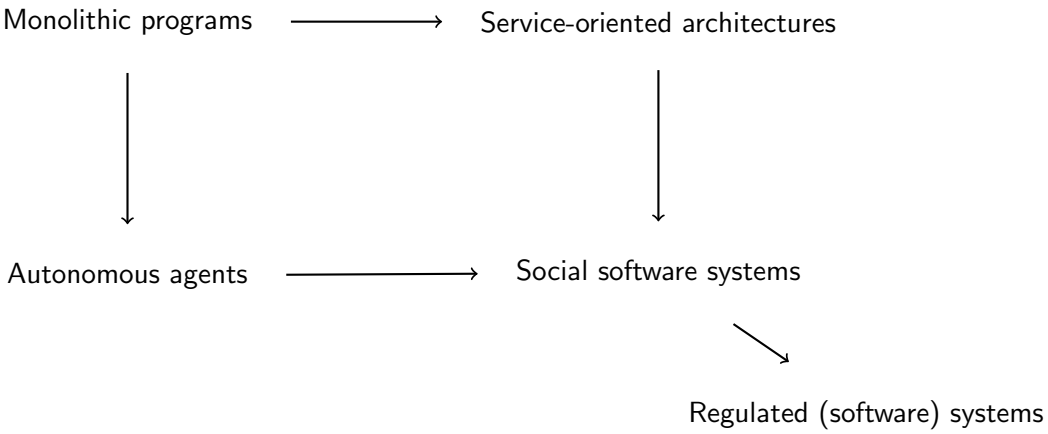
Monolithic programs ⟶ Service-oriented architectures

Autonomous agents

Software development



Software development



Regulated systems – points to address

Formalization of applicable norms: reusable, modular and dynamically updateable

Different methods of embedding and enforcing norms:

- Static ex-ante: verify and apply norms during software production
  e.g. correct-by-construction arguments, model checking, conformance checking
- Dynamic ex-ante: apply rules at run-time, guaranteeing compliance
  permits decisions (behavioral, normative) that depend on input
- Embedded ex-post enforcement: specified responses to violations
  permits (regulated) non-compliant behavior, e.g. based on risk assessment by agent
- External ex-post enforcement: external responses to violations
  permits (human-)intervention in running system

Production of diagnostic reports and/or audit trails to enable evaluation and reflection

Regulated *systems* – points to address

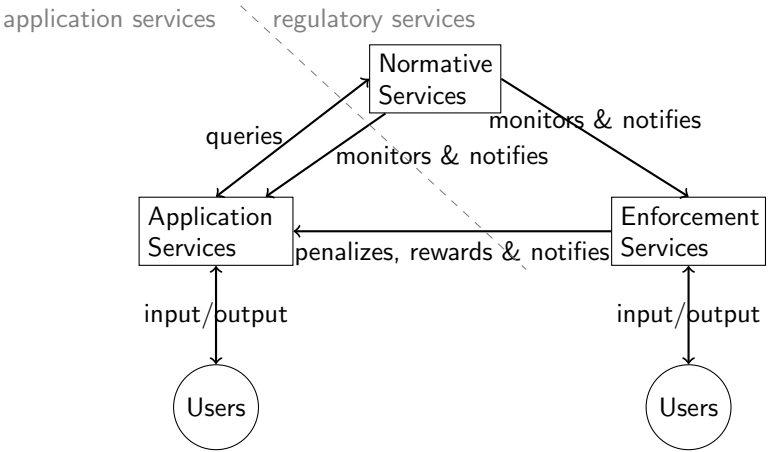
Derivation of regulatory services from formalization of norms

Interfacing between application and regulatory services:

- Monitoring (communicated and silent) behavior of application services/agents
  difficulties: fallible and subject to manipulation
- Regulatory services responding to queries about normative positions
  e.g. do I have permission to...? or the obligation to... ?
- Application services verifying facts on behalf of regulatory services
  e.g. verifying credentials
- Regulatory services communicating changes in normative positions
  e.g. gaining/losing powers, holding/satisfying obligations, violations

Challenges: different interpretations of norms and different qualifications of situations

Our approach to regulated systems



Our approach to prototyping

- Actor-oriented programming in the Akka framework: `https://akka.io/`
  actors as an abstraction capturing users and software components/services
- Digital Enforceable Contracts: Making Smart Contracts Smarter
  actor-oriented view on smart contracts and enforcement
  published at Jurix (presented tomorrow @ 17:40 18:30)
- eFLINT – formalization of norms from a variety of sources
  declarative reasoning about facts, actions and duties
  reactive component for integration in software systems
  published at GPCE (part of SPLASH 2020)
- AgentScriptCC – specification of application components and enforcement actors
  scripts describe BDI agents, cross-compiled to actor-based implementation
  published at Agere (part of SPLASH 2020)

Ongoing work

eFLINT: A Domain-Specific Language for Executable Norm Specifications

L. Thomas van Binsbergen
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
ltvanbinsbergen@acm.org

Lu-Chi Liu
University of Ams
Amsterdam, The Ne
l.liu@uva.n

t van Doesburg
niversity of Amsterdam / TNO
am, The Netherlands
ndoesburg@uva.nl

Tom van En
Leibniz Institute, University of
Amsterdam, The Ne
vanengers@uv

Digital Enforceable Contracts (DEC): Making Smart Contracts Smarter

Lu-Chi LIU [a,1], Giovanni SILENO [a] and Tom VAN ENGERS [b,a]

Run, Agent, Run

Architecture and Benchmarking of Actor-based Agents

Mostafa Mohajeri Parizi
m.mohajeriparizi@uva.nl
Informatics Institute, University of Amsterdam
Amsterdam, the Netherlands

Giovanni Sileno
g.sileno@uva.nl
Informatics Institute, University of Amsterdam
Amsterdam, the Netherlands

Tom van Engers
vanengers@uva.nl
Informatics Institute, University of Amsterdam
Amsterdam, the Netherlands

Sander Klous
s.klous@uva.nl
Informatics Institute, University of Amsterdam
Amsterdam, the Netherlands

Languages for prototyping regulated systems: A case study

L. Thomas van Binsbergen, Mostafa Mohajeri Parizi[1], Lu-Chi Liu[1], Giovanni Sileno[1], and Tom van Engers[2]

Section 2

The KYC case study

The KYC case study

Case study around the Know Your Customer principle adopted by financial institutions to meet international regulations by assessing client profiles to compute risk

Involves three types of "normative documents":

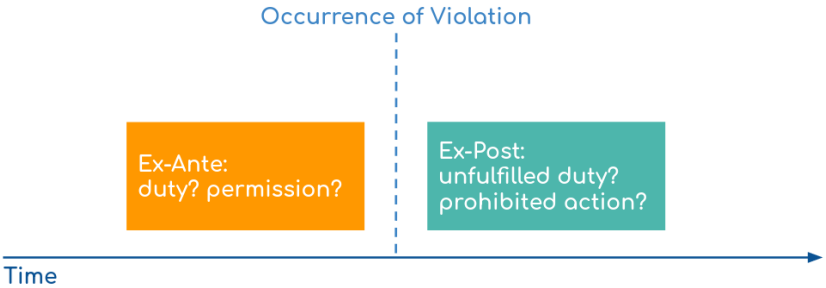
1. Sharing agreement – a contract between banks of a consortium
2. Internal policy – a sort of contract between bank and employee
3. GDPR – a sort of contract between bank and client

For each document we can describe its norms, the behavior of relevant actors (clients, banks, employees and broker) and how the norms are enforced

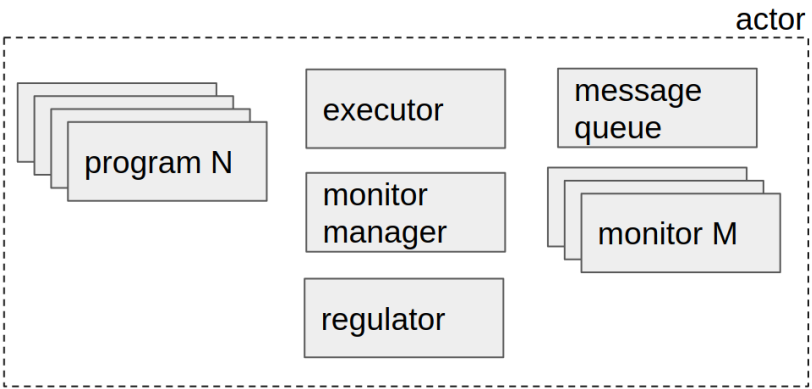
Subsection 1

Enforcement in digital contracts

Types of enforcement



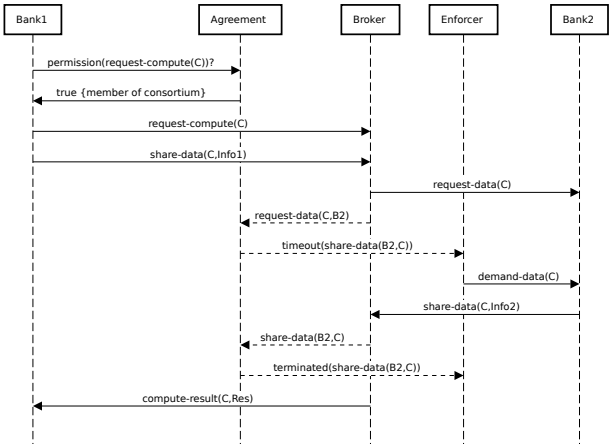
An actor-oriented enforcement architecture



Enforcement examples for 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



Subsection 2

eFLINT: Executable norm specifications

eFLINT example – GDPR

(Article 16) The data subject shall have the right to obtain from the controller without undue delay the rectification of inaccurate personal data concerning him or her. [...]

```
Act demand-rectification
 Actor subject
 Recipient controller
 Related to purpose
 Creates rectification-duty()
 Holds when (Exists data, processor:
  subject-of() && processes() && !accurate-for-purpose())

Duty rectification-duty
  Holder controller
  Claimant subject
  Related to purpose
  Violated when undue-rectification-delay()

Fact undue-rectification-delay
  Identified by controller * purpose * subject
```

From eFLINT specifications to eFLINT actors

idea: let 'eFLINT actors' administer eFLINT specifications

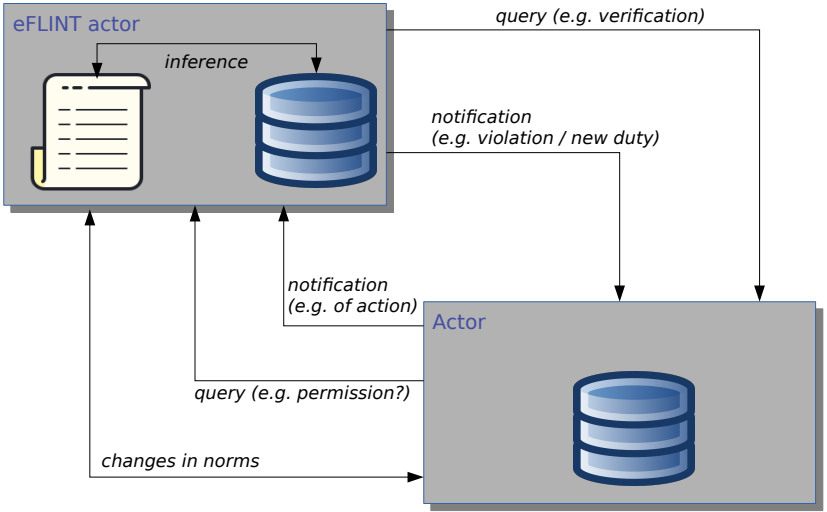
Incoming messages trigger input events

- Creating/terminating facts and triggering actions and events (statements)
  - Dynamic scenario (case) construction with automated assessment
- Creating, modifying or removing fact-, act-, event- and duty-types (declarations)
  - Dynamic policy construction
- Queries, e.g. to check whether actions are permitted or duties are violated

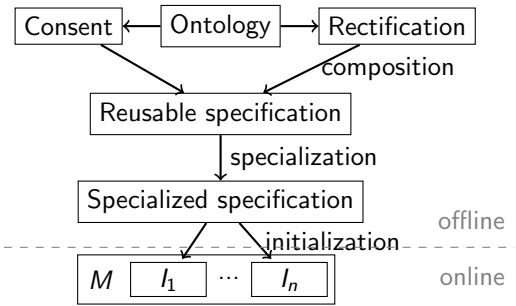
Output events trigger outgoing messages

- Notifications of newly permitted actions
- Notifications of executed actions and whether they were permitted
- Notifications of new duties and violations of duties
- Querying an actor to determine or verify the truth of a fact

eFLINT actors



eFLINT integration – overview



eFLINT integration – example

Reusable GDPR concepts

```
Fact controller
Fact subject

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

Specialization to application

```
Fact bank
Fact client

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
 ...
```

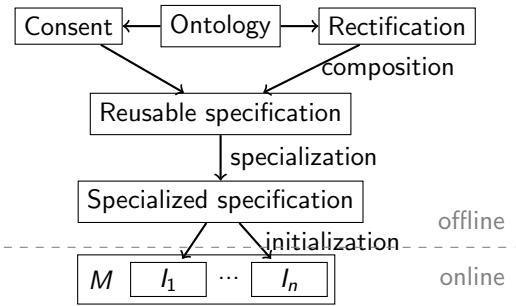
Instantiation at run-time

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

Derived after instantiation

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

eFLINT integration – overview



Monitoring GDPR compliance

```
WHEN
  Message(client:ClientRef,bank:BankRef,req:BankTypes.ApplicationRequest)
TRIGGER
  INIT gdpr(bank, client)                                  // instantiates GDPR actor

INIT gdpr                                                  // defines constructor
  WITH bank:BankRef, client:ClientRef                      // Scala class parameters
  IDENTIFIED BY (bank.path.name, client.path.name)         // pair of values as id
  FROM "gdpr_specialization.eflint"                        // eFLINT file to load
TRIGGER                                                    // eFLINT initialization
  +client(${client.path.name}).                            //        statements
  +bank(${bank.path.name}).
  +data(0).

WHEN
  Message(client:ClientRef,bank:BankRef,msg:BankTypes.CountryUpdate)
TRIGGER IN gdpr(bank.path.name, client.path.name)
  demand-rectification(purpose=KYC).                       // qualified as demand
```

Subsection 3

AgentScriptCC: BDI agent specification

AgentScriptCC DSL

Main component: 'plan rules' `E : C => A`

- when *event* `E` happens
- and if *condition* `C` holds,
- then do *action* `A`

Example from **client**:

- `E`: Agent adopted the goal `give_info`
- `C`: `B` is a bank to which client is applying or has successfully applied, `S` is SBI-code of client and `C` is country where client is based.
- `A`: send SBI-code and country to original sender of `give_info` message

```
+!give_info(B) :
 my_sbi(S) &&
 my_country(C) &&
 (applying_to(B) || client_of(B)) =>
  #achieve(#executionContext.sender.ref,info(S,C)).
```

AgentScriptCC - Internal policy example

(Rule 1) An employee has the duty to perform a risk analysis on the profile of a client within four weeks of the creation or modification of the profile

Employee

```
+!interview(Client) :
 bank(B) &&
 B == #executionContext.sender.name =>
  #achieve(Client,give_info(B)).

+!info(SBI,Country) :
 bank(B) =>
  Client = #executionContext.sender.name;
  Info = info(SBI,Country);
  +information(Client,Info);
  #achieve(B,interview_complete(Client,Info)).

+!do_risk_analysis(C,info(SBI,Country)) =>
  B = #executionContext.sender.name;
  R = #kyc.algorithms.risk(B,SBI,Country);
  #achieve(B,assign_risk(C,R)).
```

Client

```
+!give_info(B) :
 my_sbi(S) &&
 my_country(C) &&
 (applying_to(B) || client_of(B)) =>
  #achieve(#executionContext.sender.ref,info(S,C)).
```

Bank

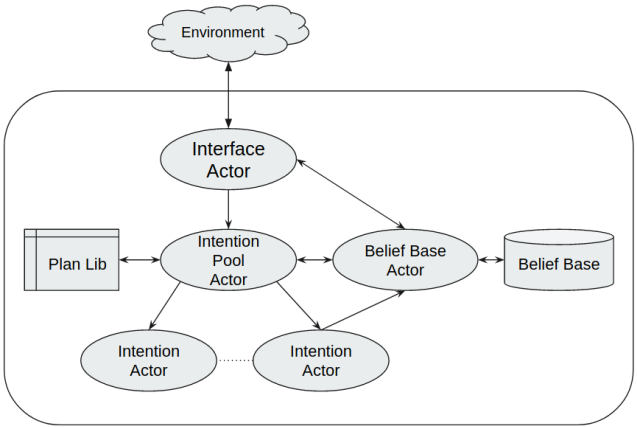
```
+!interview_complete(Client,Info):
 E = #executionContext.sender.name &&
 employee(E) &&
 not client(Client) =>
  #println("interview done for " + Client);
  +information(Client,Info);
  +client(Client);
  #achieve(E,do_risk_analysis(Client,Info)).
```

AgentScriptCC

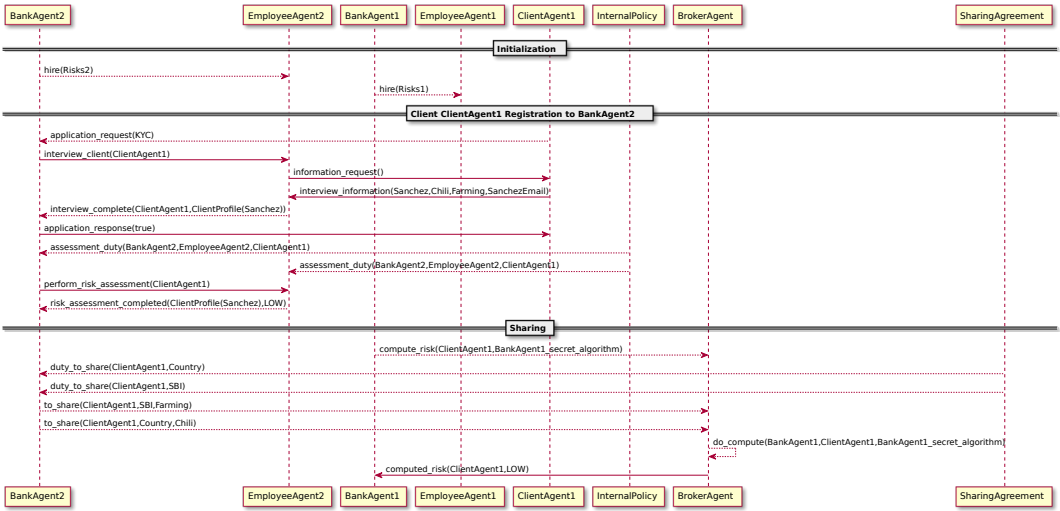
Agents are translated into actor-based micro-systems

Consisting of:

- Interface actor
- Intention pool actor
- $n \geq 1$ Intention actors
- Belief base actor
- Belief base
- Plan library



KYC case – example scenario



Ongoing work

eFLINT: A Domain-Specific Language for Executable Norm Specifications

L. Thomas van Binsbergen
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
ltvanbinsbergen@acm.org

Lu-Chi Liu
University of Ams
Amsterdam, The Ne
l.liu@uva.n

Digital Enforceable Contracts (DEC): Making Smart Contracts Smarter

t van Doesburg
niversity of Amsterdam / TNO
am, The Netherlands
mdoesburg@uva.nl

Tom van Eng
Leibniz Institute, University of
Amsterdam, The Ne
vanengers@uv

Lu-Chi LIU [a,1], Giovanni SILENO [a] and Tom VAN ENGERS [b,a]

Run, Agent, Run

Architecture and Benchmarking of Actor-based Agents

Mostafa Mohajeri Parizi
m.mohajeriparizi@uva.nl
Informatics Institute, University of Amsterdam
Amsterdam, the Netherlands

Giovanni Sileno
g.sileno@uva.nl
Informatics Institute, University of Amsterdam
Amsterdam, the Netherlands

Tom van Engers
vanengers@uva.nl
Informatics Institute, University of Amsterdam
Amsterdam, the Netherlands

Sander Klous
s.klous@uva.nl
Informatics Institute, University of Amsterdam
Amsterdam, the Netherlands

Languages for prototyping regulated systems: A case study

L. Thomas van Binsbergen, Mostafa Mohajeri Parizi[1], Lu-Chi Liu[1], Giovanni Sileno[1], and Tom van Engers[2]

Languages for prototyping regulated systems: A case study

L. Thomas van Binsbergen Lu-Chi Liu Mostafa Mohajeri Parizi

Informatics Institute, University of Amsterdam
ltvanbinsbergen@acm.org

December 9, 2020