Prototyping regulated systems

L. Thomas van Binsbergen

Informatics Institute, University of Amsterdam
ltvanbinsbergen@acm.org

February 5, 2021
Monolithic programs
A history of software development, towards regulated systems

Monolithic programs $\xrightarrow{\text{distribution}}$ Service-oriented architectures
A history of software development, towards regulated systems

Monolithic programs $\xrightarrow{\text{distribution}}$ Service-oriented architectures

$\downarrow$

AI

Autonomous systems
A history of software development, towards regulated systems

Monolithic programs $\xrightarrow{\text{distribution}}$ Service-oriented architectures

\[\text{AI}\]

Autonomous systems $\xrightarrow{\text{distribution}}$ Social software systems

\[\text{AI}\]
A history of software development, towards regulated systems

Monolithic programs \(\xrightarrow{\text{distribution}}\) Service-oriented architectures

Autonomous systems \(\xrightarrow{\text{distribution}}\) Social software systems

Regulated (software) systems

Al

L. Thomas van Binsbergen
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
- 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
  e.g. auditing, conformance checking
  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 services
  
  *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

Application Services

Normative Services

Enforcement Services

Users

queries

monitors & notifies

penalizes, rewards & notifies

input/output

application services

regulatory services

L. Thomas van Binsbergen

Prototyping regulated systems 5 / 25
Our approach to prototyping

eFLINT – formalization of norms from a variety of sources
declarative reasoning about facts, actions and duties
reactive component for integration in software systems
including actor-based implementation

AgentScriptCC – specification of services as agents
reactive BDI agents,
cross-compiled to actor-based implementation

Actor-oriented programming in the Akka framework:
https://akka.io/
actor systems modelling social software systems
eFLINT actors

Changes in norms

Query (e.g. permission?)

Notification (e.g. of action)

Inference

Query (e.g. verification)

Notification (e.g. violation / new duty)
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
Our approach to prototyping

System specification

Executable actor-model

- A1
- A2
- A3
- A4
- A5

event log

logging

generation

compare

connect models

- alt. norms
- norms
- agents
- scenarios

- communication protocols
- event/message translation

L. Thomas van Binsbergen

Prototyping regulated systems
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
(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.
(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. [...]
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

- changes in norms
- query (e.g. permission?)
- notification (e.g. of action)
- notification (e.g. violation / new duty)
- query (e.g. verification)
eFLINT integration – overview

Consent → Ontology → Rectification

Composition

Reusable specification

Specialization

Specialized specification

\( M \quad I_1 \quad \ldots \quad I_n \)

Initialization

Online

Offline
## 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

Consent → Ontology → Rectification

Composition → Reusable specification

Specialization → Specialized specification

Initialization → $M \ I_1 \ \cdots \ I_n$

Offline → online
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(${client.path.name}).
  +bank(${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
Main component: ‘plan rules’ $E : C \Rightarrow A$

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

Example from client:

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

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

```plaintext
+!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);
  #achieve(Client,info(SBI,Country));
  #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)).
```

Bank

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

Client

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

Initialization

Client ClientAgent1 Registration to BankAgent2

Sharing

L. Thomas van Binsbergen
Prototyping regulated systems 21 / 25
We can produce executable models of regulated systems, by combining
- enforcement actors for dynamic ex-post enforcement
- normative actors derived from normative specifications (in eFLINT),
- queries sent to normative actors for dynamic ex-ante enforcement, and
- actor implementations derived from agent scripts (in AgentScriptCC)

this enables experiments with norms, enforcement mechanisms and system set-ups.
**Future work**

**Ongoing**
- DSL development and analysis for behavior, norm and scenario specification
- Complete generation of executable-actor models from high-level specification
- Bring modelling to practice;
  - apply models by deriving (parts of) containerized applications for use cases in our projects on data exchange: SSPDDP, DL4LD, EPI, and soon AMDEX
  - explainable decision making in projects with governmental organizations

**Future**
- Static analysis of (combined) models, e.g. model checking norm specification, and consistency checking between between behavior, normative actors and scenarios
- Additional execution platforms:
  - Containerized applications, e.g. Docker and Kubernetes
  - High-performance cloud (HPC)
  - Blockchain
The complex-cyber infrastructure group of the University of Amsterdam is experimenting with regulated systems – in which norms from a variety of sources are enforced – by deriving executable models from high-level specifications.

Such systems require several kinds of enforcement mechanisms for norms, based on whether compliance can/should be/is checked before or after a violation occurs and before or after an application runs.
Prototyping regulated systems

L. Thomas van Binsbergen

Informatics Institute, University of Amsterdam
ltvanbinsbergen@acm.org

February 5, 2021