Overview

1. Regulated systems
   Relating normative and computational concepts

2. The eFLINT language
   eFLINT 1.0
   eFLINT 2.0
   Goals for eFLINT 3.0

3. Reflections
Section 1

Regulated systems
Towards regulated systems

Monolithic programs
Towards regulated systems

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

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

\(\downarrow\)

AI

\(\downarrow\)

Autonomous systems
Towards regulated systems

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

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

AI

AI

distribution

distribution

Regulated (software) systems

norms + enforcement
Towards regulated systems

Monolithic programs → \textit{distribution} → Service-oriented architectures

Autonomous systems → \textit{distribution} → Social software systems

\textit{AI} → \textit{distribution} → Social software systems

\textit{AI} → \textit{distribution} → Regulated (software) systems

\textit{norms} + \textit{enforcement}
Regulated system $=\text{application services} + \text{regulatory services}$
(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.
Our approach to model-driven experimentation

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,
compiled to actor-based implementation

Actor-oriented programming in the Akka framework:
https://akka.io/
actor systems modelling social software systems
Subsection 1

Relating normative and computational concepts
“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
Foundational, normative & computational concepts

combinatorial

state

\[ \text{parent}(A, B) = true \]
\[ \ldots \]
Foundational, normative & computational concepts

computational

**state**

\[ \text{parent}(A, B) = \text{true} \]

... 

**transitions**

\[ \text{parent}(A, B) = \text{true} \]

...

\[ \text{parent}(A, B) = \text{false} \]

...
Foundational, normative & computational concepts

state

\[ \text{parent}(A, B) = true \]
\[ \ldots \]

transitions

\[ \text{parent}(A, B) = true \]
\[ \ldots \]

\[ \text{parent}(A, B) = false \]
\[ \ldots \]

■ Violations of state and transition
Foundational, normative & computational concepts

**deontic**

- prohibitions
- permissions
- obligations

**computational**

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

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

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

■ Violations of state and transition
Foundational, normative & computational concepts

- **Deontic**
  - Prohibitions
  - Permissions
  - Obligations

- **Computational**
  - **State**
    - \( \text{parent}(A, B) = \text{true} \)
  - **Transitions**
    - \( \text{parent}(A, B) = \text{true} \)
    - \( \text{parent}(A, B) = \text{false} \)

■ Violations of state and transition
Violations of state and transition
Foundational, normative & computational concepts

---

**Deontic**

- Prohibitions
- Permissions
- Obligations

---

**Computational**

- **State**
  - $\text{parent}(A, B) = true$
  - ...

- **Transitions**
  - $\text{parent}(A, B) = true$
  - ...
  - $\text{parent}(A, B) = false$
  - ...

---

- Violations of state and transition

---
Foundational, normative & computational concepts

- **Deontic**
  - Prohibitions
  - Permissions
  - Obligations

- **Computational**
  - State
    - $parent(A, B) = true$
    - ...
  - Transitions
    - $parent(A, B) = true$
    - ...
    - $parent(A, B) = false$
    - ...

- **Potestative**
  - Powers

■ Violations of state and transition
Foundational, normative & computational concepts

- **Deontic**
  - Prohibitions
  - Permissions
  - Obligations

- **Computational**
  - State
    - \( \text{parent}(A, B) = \text{true} \)
    - ...
  - Transitions
    - \( \text{parent}(A, B) = \text{true} \)
    - ...
  - \( \text{parent}(A, B) = \text{false} \)
    - ...

- **Potestative**
  - Powers
    - Powers have (normative) consequences

- Violations of state and transition
Foundational, normative & computational concepts

**Deontic**
- prohibitions
- permissions
- obligations

**Computational**
- state
  - $parent(A, B) = true$
  - ...
- transitions
  - $parent(A, B) = true$
  - ...
  - $parent(A, B) = false$
  - ...

**Potestative**
- powers
  - Powers have (normative) consequences
  - Deontic and potestative terms are first-class
  - Powers modify truth-assignments to variables

■ Violations of state and transition
Violations of state and transition

- Powers have (normative) consequences
- Deontic and potestative terms are first-class
- Powers modify truth-assignments to variables
- Explicit permissions cause conflicts
Normative relations between actors

• A deontic value is associated with several actors:
  – The **holder** of the prohibition, obligation or prohibition
  – Zero or more **claimants** to the prohibition or obligation
  – The actor who **assigned** the prohibition, obligation or permission

• A potestative value is associated with several actors
  – The **performing** actor
  – One or more **recipients** being affected by the power
  – The actor who **assigned** the power
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
  enables decisions (behavioral, normative) that depend on input

- Embedded ex-post enforcement: specified responses to violations
  enables (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
  enables (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
don’t get it right falling 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 or certificates

• 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
Regulated systems for Know Your Customer case study

Repository of reusable norm specifications

Application specific specs

Regulatory services

Policy construction (offline)

Internal Policy

Sharing Agreement

Consent

Ontology

Rectification

GDPR composition

Internal Policy

Sharing Agreement

GDPR composition

SA

G1

G2

... G_n

M0

P1

P_n

M1

SA

M2

G1

G_n

Event

Request/response

Event

Event

Application services

Client1

Client_n

Employee1

Employee_n

Bank1

Bank_n

Broker

Distributed system (online)
Section 2

The eFLINT language
Subsection 1

eFLINT 1.0
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)
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)
Example – scenario

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). // Alice is Bob’s legal parent
+homework-due(Bob). // homework deadline passed
?Violated(help-with-homework(Alice,Bob)). // query duty violation
help(Alice,Bob). // duty terminated

Listing 3: Scenario
eFLINT online!

frames

- 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)
  \[\| \text{natural-parent(parent,child)}\]
- Act ask-for-help
  - Actor child
  - Recipient parent
  - Creates help-with-homework(parent,child)
  - Holds when legal-parent(parent,child)
- Fact homework-due Identified by child
- Duty help-with-homework
  - Holder parent
  - Claimant child
  - Violated when homework-due(child)
- Act help
  - Actor parent
  - Recipient child
  - Terminates help-with-homework(parent,child)
  - Holds when help-with-homework(parent,child)

domains

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

initial state

- natural-parent(Alice, Bob).
- adoptive-parent(Cloe, David).

Examples

Knowledge representation: Vehicles | Departments | Count Votes | Cast Votes
GPCE2020 paper examples: Help with homework | GDPR
Various: Buyer/Seller (v1) | Buyer/Seller (v2) | Buyer/Seller (v3) | Permit Applications | Permit Applications (v2) | Multiple taxpayers | Voting
Load file: Browse | No file selected.

scenario

- ask-for-help(Bob, Alice).
  - homework-due(Bob) // homework deadline passed
  - violating help-with-homework(Alice,Bob).
  - help(Alice,Bob).

response

- Duty violated at step 2
  - ("Alice":person,"Bob":person):help-with-homework

output

Step 0: initial state

Step 1: ("Bob":person,"Alice":person):ask-for-help
  - Alice(person,"Bob":person):help-with-homework

Step 2: ("Bob":person):homework-due
  - "Bob":person):homework-due

Step 3: query

Step 4: ("Alice":person,"Bob":person):help
  - Alice(person,"Bob":person):help-with-homework
Subsection 2

eFLINT 2.0
From DSL Specification to Interactive Computer Programming Environment

Pierre Jeanjean  
Inria, Univ Rennes, CNRS, IRISA  
Rennes, France  
pierre.jeanjean@inria.fr

Benoit Combemale  
University of Toulouse  
Toulouse, France  
benoit.combemale@irit.fr

Olivier Barais  
Univ Rennes, Inria, CNRS, IRISA  
Rennes, France  
olivier.barais@irisa.fr

Figure: Software Language Engineering 2019

Bacatá: Notebooks for DSLs, Almost for Free

Mauricio Verano Merino\textsuperscript{a,d}, Jurgen Vinju\textsuperscript{a,b}, and Tijs van der Storm\textsuperscript{b,c}

\begin{itemize}
  \item[a] Eindhoven University of Technology, The Netherlands
  \item[b] Centrum Wiskunde & Informatica, The Netherlands
  \item[c] University of Groningen, The Netherlands
  \item[d] Océ Technologies B.V., The Netherlands
\end{itemize}

Figure: Art, Science, and Engineering of Programming 2020
Deriving REPL/Notebook – commonalities

- **READ**: Identify entry points, i.e. the alternatives in syntactic root
- **EVAL**: Connect entry points with evaluation function in DSL interpreter
- **PRINT**: Specify function to convert evaluation result to string
- **LOOP**:

---

**Figure 8.** Overall Execution Flow for Logo
Deriving REPL/Notebook – commonalities

- **READ:** Identify entry points, i.e. the alternatives in syntactic root
- **EVAL:** Connect entry points with evaluation function in DSL interpreter
- **PRINT:** Specify function to convert evaluation result to string
- **LOOP:**

\[\text{How does one execution affect the next?}\]

**Figure 8.** Overall Execution Flow for Logo
Distinguish between REPL language and base language (e.g. JShell vs Java)

A Principled Approach to REPL Interpreters

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

Mauricio Verano Merino
Eindhoven University of Technology
Eindhoven, The Netherlands
m.verano.merino@tue.nl

Pierre Jeanjean
Inria, University of Rennes, CRNS,
IRISA
Rennes, France
pierre.jeanjean@inria.fr

Tijs van der Storm
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
University of Groningen
Groningen, The Netherlands
storm@cwi.nl

Benoit Combemale
University of Rennes, Inria, CNRS,
IRISA
Rennes, France
benoit.combemale@irit.fr

Olivier Barais
University of Rennes, Inria, CNRS,
IRISA
Rennes, France
olivier.barais@irisa.fr

Figure: Onward!2020
Observation..!

REPLs with incremental execution implement a language with the following property:
REPLs with incremental execution implement a language with the following property:

A **sequential language** is a language in which $p_1 \otimes p_2$ is a (syntactically) valid program iff $p_1$ and $p_2$ are valid programs and iff $p_1 \otimes p_2$ is equivalent to ‘doing’ $p_1$ and then $p_2$

$$\llbracket p_1 \otimes p_2 \rrbracket = \llbracket p_2 \rrbracket \circ \llbracket p_1 \rrbracket$$
Observation..!

REPLs with incremental execution implement a language with the following property:

A **sequential language** is a language in which $p_1 \otimes p_2$ is a (syntactically) valid program iff $p_1$ and $p_2$ are valid programs and iff $p_1 \otimes p_2$ is equivalent to ‘doing’ $p_1$ and then $p_2$

$$\llbracket p_1 \otimes p_2 \rrbracket = \llbracket p_2 \rrbracket \circ \llbracket p_1 \rrbracket$$

A **REPL** is a monoid homomorphism between programs and their effects
Replization is: extending a base language to a sequential variant
Replization is: extending a base language to a sequential variant

1. Define the syntax of the extended language (phrases/entry points)
Replization is: extending a base language to a sequential variant

1. Define the syntax of the extended language (phrases/entry points)
2. Extend interpreter by linking phrases to functions in base interpreter
Replization is: extending a base language to a sequential variant

1. Define the syntax of the extended language (phrases/entry points)
2. Extend interpreter by linking phrases to functions in base interpreter
3. Add phrase composition operator to the language (it is now sequential by definition)

\[ [p_1 \otimes p_2] = [p_2] \circ [p_1] \]
Replization is: extending a base language to a sequential variant

1. Define the syntax of the extended language (phrases/entry points)
2. Extend interpreter by linking phrases to functions in base interpreter
3. Add phrase composition operator to the language (it is now sequential by definition)
   \[
   \llbracket p_1 \otimes p_2 \rrbracket = \llbracket p_2 \rrbracket \circ \llbracket p_1 \rrbracket
   \]

- The effect of one phrase on the next is determined by (2)
Onward!2020 (MiniJava case study)

```java
Config eval((Phrase)`<Expression e> ;`, Config c)  
    = catchExceptions(collectBindings(  
        setOutput(createBinding(eval(c, e))));

Config eval((Phrase)`<Statement s>`, Config c)  
    = catchExceptions(collectBindings(  
        setOutput(exec(s, c))));

Config eval((Phrase)`<ClassDecl cd>`, Config c)  
    = catchExceptions(collectBindings(  
        declareClass(cd, c)));

Config eval((Phrase)`<VarDecl vd>`, Config c)  
    = catchExceptions(collectBindings(  
        declareVariables(vd, c)));

Config eval((Phrase)`<MethodDecl md>`, Config c)  
    = catchExceptions(collectBindings(  
        declareGlobalMethod(md, c)));

Config eval((Phrase)`<Phrase p1> <Phrase p2>`, Config c)  
    = eval(p2, eval(p1, c));
```
1. eFLINT 2.0: REPLization applied to eFLINT using eFLINT 1.0 interpreter

- valid phrases: type-declarations, initialization, triggering action/events, queries
- enables backtracking for manual exploration
- enables implementation of ‘eFLINT actors’
- type-declarations as phrases enable dynamic policy construction
REPLization of eFLINT

1. eFLINT 2.0: REPLization applied to eFLINT using eFLINT 1.0 interpreter
   - valid **phrases**: type-declarations, initialization, triggering action/events, queries
   - enables backtracking for manual exploration
   - enables implementation of ‘eFLINT actors’
   - type-declarations as phrases enable dynamic policy construction

2. Tools based on the same REPLized interpreter
   - eflint-repl: command line tool for manual exploration and debugging
   - eflint-server: server that listens on a port for incoming phrases
eFLINT actors

changes in norms

notification (e.g. violation / new duty)

notification (e.g. of action)

query (e.g. permission?)

query (e.g. verification)

inference
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

...
Modular GDPR specification

Dynamic generation of access control policies from social policies

L. Thomas van Binsbergen\textsuperscript{1,a}, Milen G. Kebede\textsuperscript{a}, Joshua Baugh\textsuperscript{b}, Tom van Engers\textsuperscript{a}, Dannis G. van Vuurden\textsuperscript{b}

\textsuperscript{a}Informatics Institute, University of Amsterdam, 1090ZH Amsterdam, The Netherlands
\textsuperscript{b}Princess Maxima Center for Pediatric Oncology, Department of Neuro-oncology, Utrecht, The Netherlands

\textbf{Figure: ICTH2021}

\texttt{Act collect-personal-data}
\texttt{Actor controller}
\texttt{Recipient subject}
\texttt{Related to data, processor, purpose}
\texttt{Where subject-of(subject, data)}
\texttt{Creates processes(processor, data, controller, purpose)}
Article 5 – processing conditions

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 processes
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

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 processes
Extend Act collect-personal-data
   Holds when has-legal-obligation(controller, purpose)

Listing 7: Member (1c)
Compliance questions

According to the GDPR and the DIPG regulatory document:

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

2. What conditions need to be fulfilled when accessing data from the registry?
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
  Recipient dcog
  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 the donor for the processing of their personal data by the DCOG for the purpose of DIPGResearch (GDPR – Article 6)
- Data should be accurate for the purpose DIPGResearch (GDPR – Article 5)
Subsection 3

Goals for eFLINT 3.0
## Goals for eFLINT 3.0

### Language design

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

### 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
Section 3

Reflections
Bounded vs open-ended domains

Static analyses
- eFLINT 1.0 enabled automated scenarios assessment in finite domain
- Future work: applying model checking, and/or property-based testing

Dynamic enforcement
- eFLINT 2.0 enabled dynamic interpretation, qualification and assessment
- Domain established at runtime, based on the contents of the knowledge base

Design decision:

When enumerating instances, first check domain of type, then knowledge base

// opt1: Fact person Identified by Alice, Bob, Chloe, David

?(Forall person: !homework-due(person))
Two approaches to enforcing social policies

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\textsuperscript{1,a}, Milen G. Kebede\textsuperscript{a}, Joshua Baugh\textsuperscript{b}, Tom van Engers\textsuperscript{a}, Dannis G. van Vuurden\textsuperscript{b}

\textsuperscript{a}Informatics Institute, University of Amsterdam, 1090GH Amsterdam, The Netherlands
\textsuperscript{b}Princess Maxima Center for Pediatric Oncology, Department of Neuro-oncology, Utrecht, The Netherlands
Takeaway messages

At the Complex Cyber Infrastructure group, we are experimenting with approaches to enforcing laws, regulations, agreements and contracts in (distributed) systems.
Takeaway messages

At the Complex Cyber Infrastructure group, 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’.
At the Complex Cyber Infrastructure group, 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’.

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

At the Complex Cyber Infrastructure group, 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’

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

The next phase is to improve the practicality and usability of eFLINT and to demonstrate our approach in data exchange systems such as the Amsterdam Data Exchange (AMdEX)
Towards a DSL for formalising laws and regulations
intermediate findings and results

L. Thomas van Binsbergen
Informatics Institute, University of Amsterdam
ltvanbinsbergen@acm.org

September 9, 2021
Strumenta, Virtual Meetup