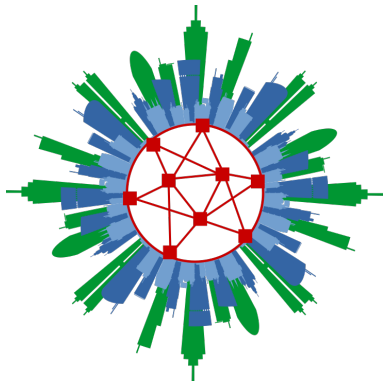
Towards Integrated IoT Languages

SuMoCoS Workshop, Ulaanbaatar, Mongolia

Muzaffar Artikov[1] Johannes Meier[2] Andreas Winter[2]

[1]Software Engineering Department, Urgench branch of Tashkent University of Information Technologies, Uzbekistan
[2]Software Engineering Group, Department of Computing Science, University of Oldenburg, Germany

26. September 2019



Context

- *How* do IoT systems *look like*? (Overview)
- *What* have to be done to *introduce* IoT systems? (Process)
- *What* should IoT systems *do*? (Runtime)

Context

- *How* do IoT systems *look like*? (Overview)
- *What* have to be done to *introduce* IoT systems? (Process)
- *What* should IoT systems *do*? (Runtime)

- *How* do we *develop* IoT systems? (Development)

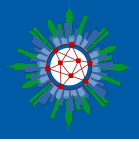
Context

- *How* do IoT systems *look like*? (Overview)
- *What* have to be done to *introduce* IoT systems? (Process)
- *What* should IoT systems *do*? (Runtime)

- *How* do we *develop* IoT systems? (Development)

- additional stakeholder: software developer, hardware developer, architects, maintenance people, project manager, . . .

Context




- *How* do IoT systems *look like*? (Overview)
- *What* have to be done to *introduce* IoT systems? (Process)
- *What* should IoT systems *do*? (Runtime)

- *How* do we *develop* IoT systems? (Development)

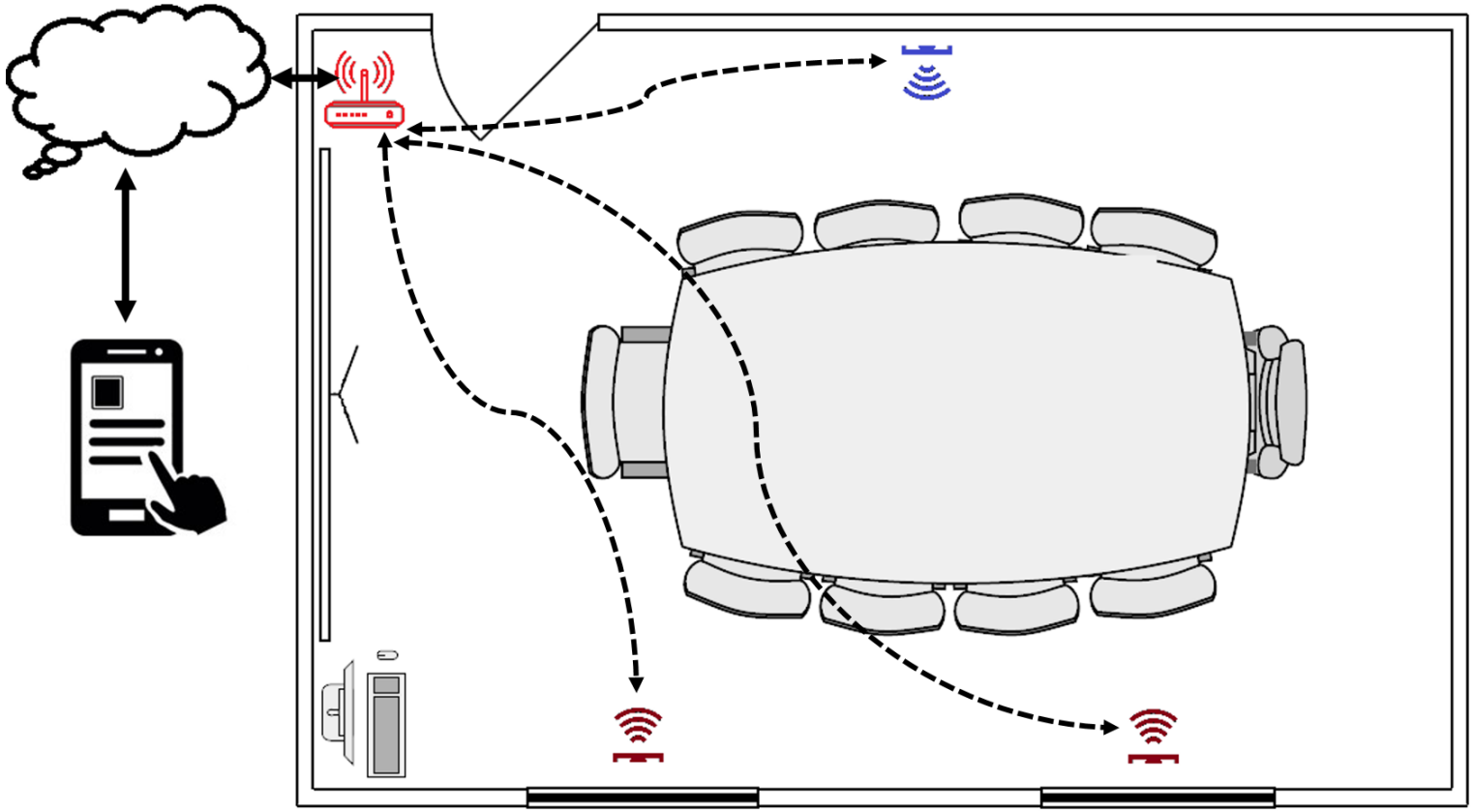
- additional stakeholder: software developer, hardware developer, architects, maintenance people, project manager, . . .

- Integration is required

IoT Example: SmartOffice

ongoing IoT example: very simple SmartOffice with two modes:

1. Presentation: close blinds, switch off bulbs
2. Discussion: open blinds, switch on bulbs if not sufficient light



IoT Languages

Required IoT Languages:

- *Things* in the SmartOffice with Sensors and Actuators
- *Rules* controlling of Actuators depending on Sensors
- *Communication* of Things with each other

Integration Challenges:

- select Things to fulfill the scenario
- specify Rules for the scenario using states of Things
- create Network depending on Rules and Things

Everything has to fit to each other!

Motivation in general

- different parts of IoT systems:
  - Hardware / Things: sensors, actuators, gateways, ...
  - Software: operating systems, drivers, libraries, application software including rules, ...
  - Network: protocols, topologies, ...
- Initial development and ongoing maintenance / evolution
- different stakeholder, development tools, data, ...

Everything has to fit to each other!

Problem in general

There are lots of IoT Projects to develop and evolve:

- various Data / Artifacts
- various Consistency issues
- various Traceability information

General Problem:

- high Heterogeneity requires different IoT languages
- IoT Artifacts are separated by tools, but interrelated contentwise
- IoT project-specific Consistency rules have to be fulfilled

Problems of interrelated Artifacts

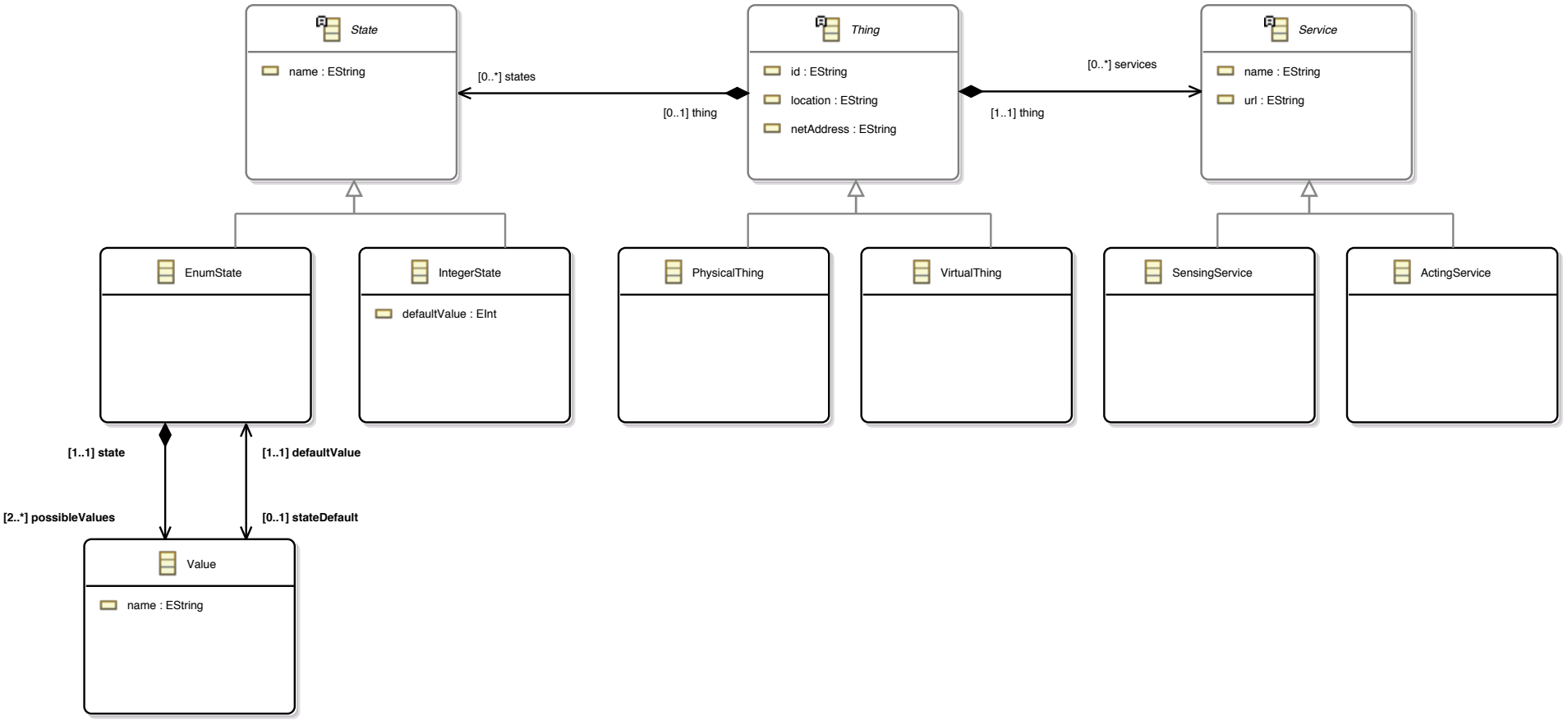
- Interrelations introduce Redundancies and Dependencies, which lead to Inconsistencies
- manual or unstructured Consistency Preservation requires high effort and is error-prone

Solution Ideas

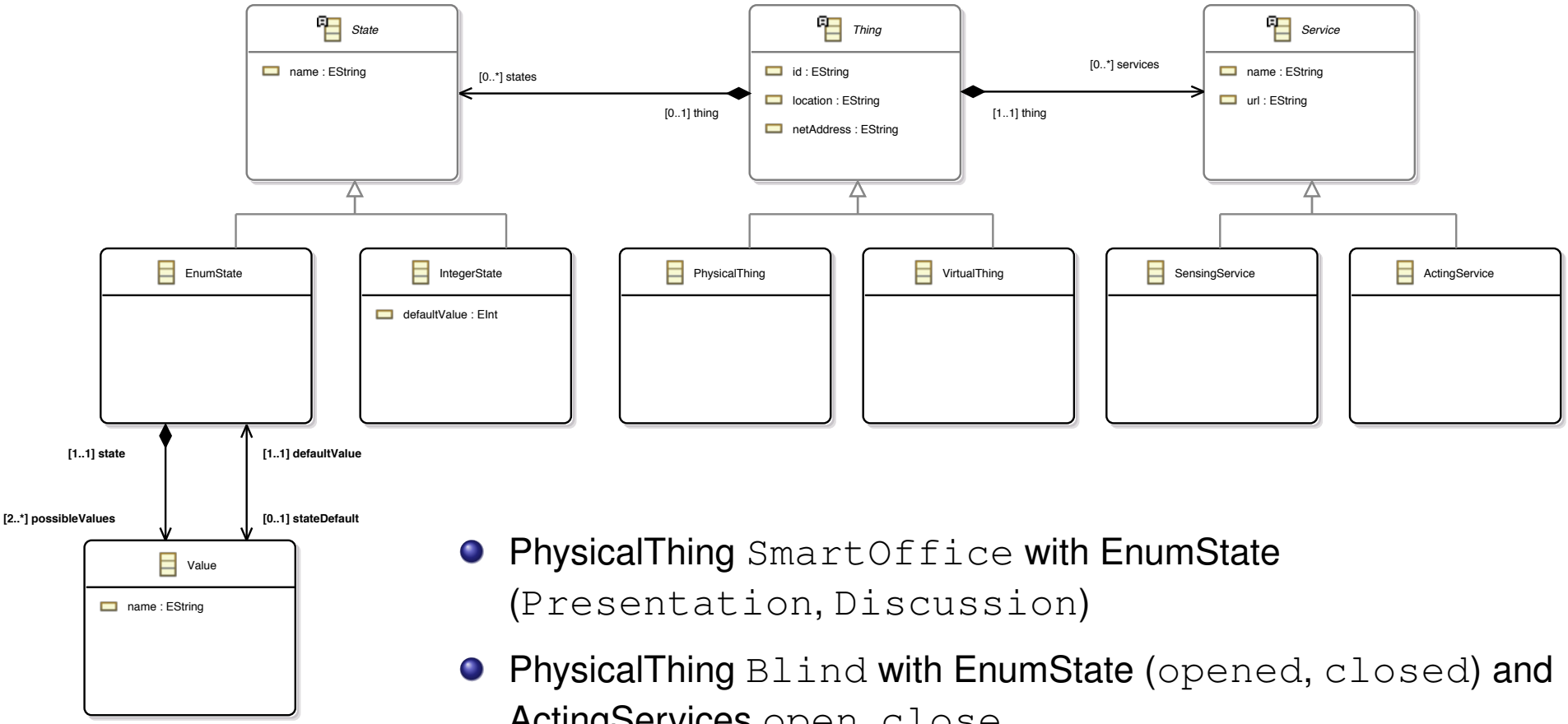
- Use Model-Driven Engineering (MDE) to ...
  - deal with complexity by using multiple Views
  - develop IoT languages
  - generate code and configurations for development and deployment
- Describe all Data as Views conforming to Viewpoints (structural formalization [Chechik, Nejati, and Sabetzadeh, 2012])
- Reuse approach for Consistency in Multi-View-Environments [Meier, Klare, Tunjic, Atkinson, Burger, Reussner, and Winter, 2019]

$\rightarrow$ Integration of IoT Languages

IoT Languages: Things

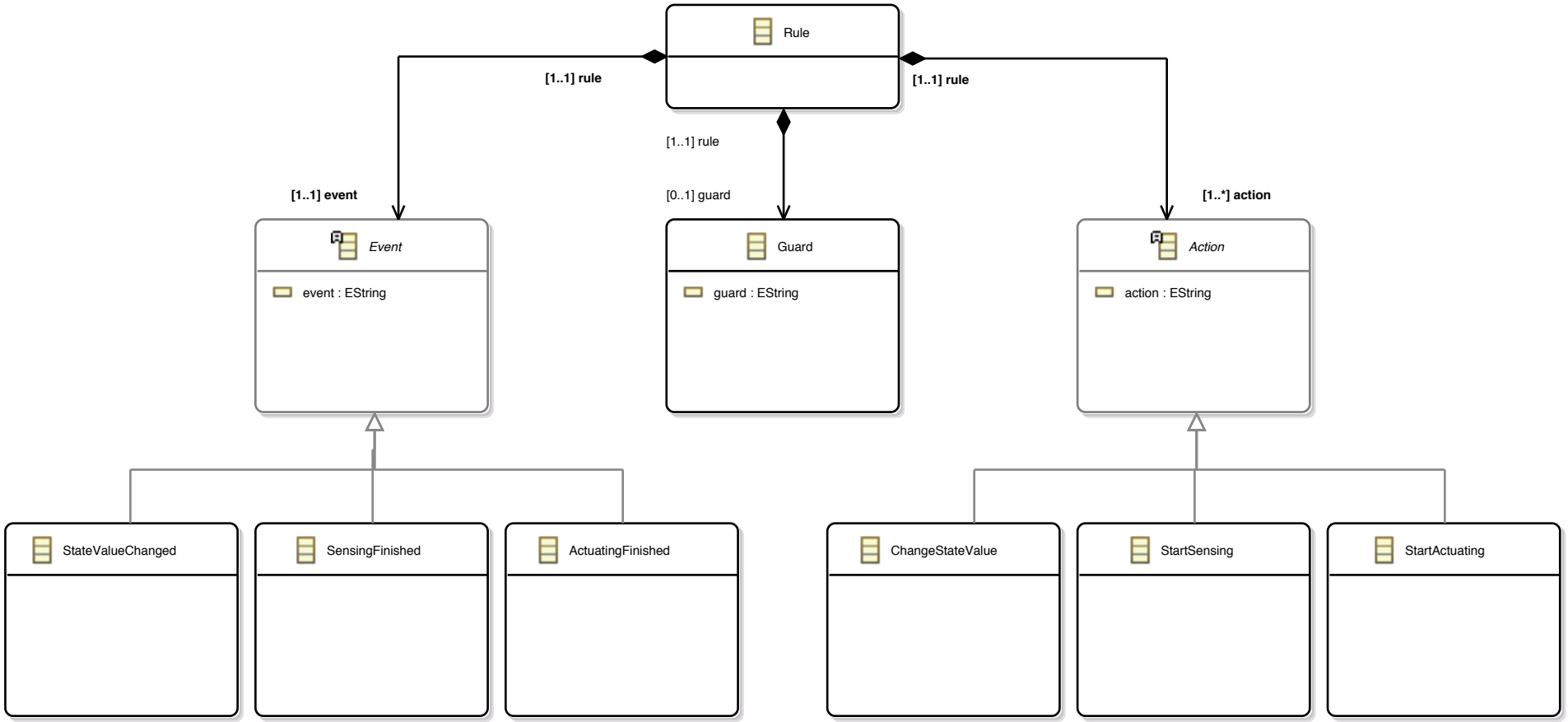


IoT Languages: Things

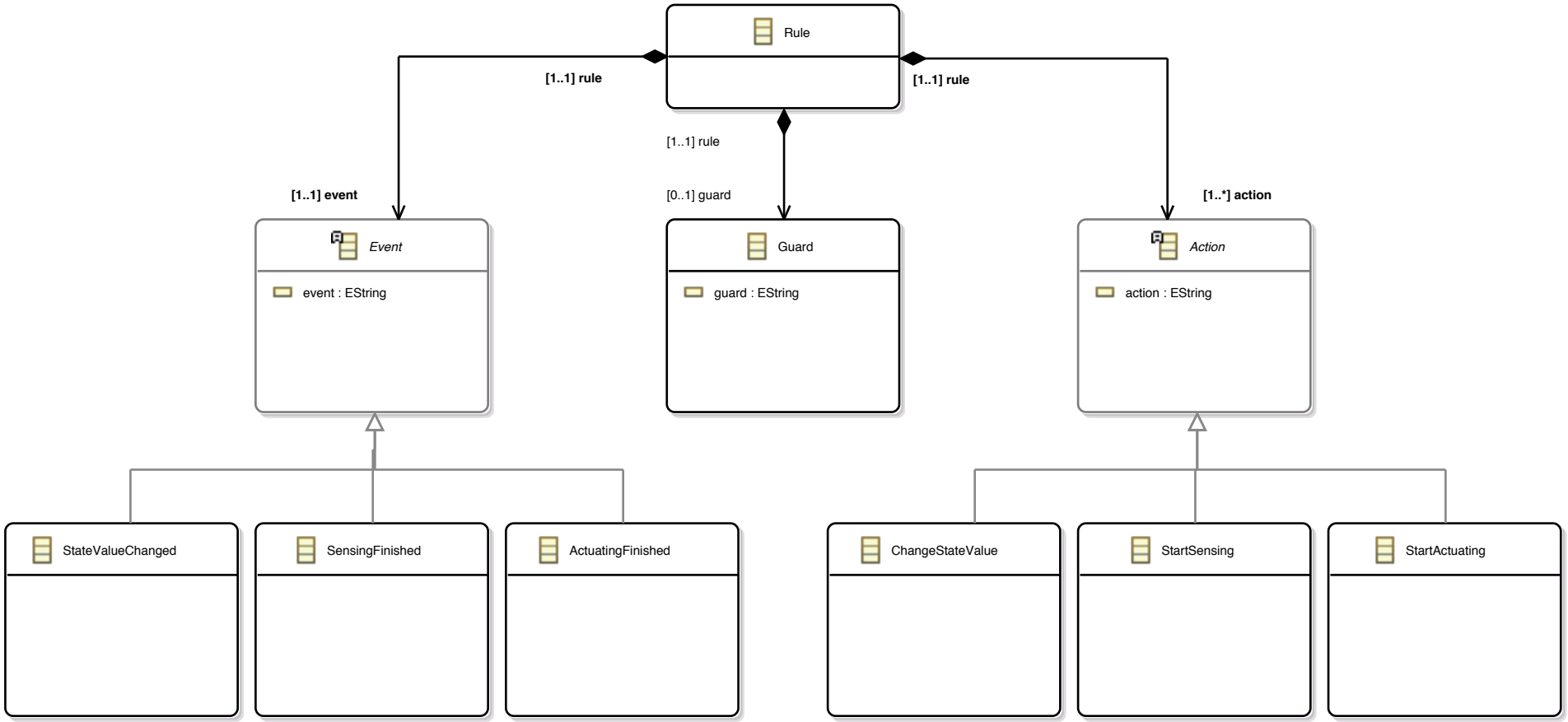


- PhysicalThing `SmartOffice` with EnumState (`Presentation`, `Discussion`)
- PhysicalThing `Blind` with EnumState (`opened`, `closed`) and ActingServices `open`, `close`
- PhysicalThing `Bulb` with EnumState (`on`, `off`) and ActingServices `switchOn`, `switchOff`
- PhysicalThing `LightSensor` with IntegerState `light` and with SensingService `measureLight`

IoT Languages: Rules

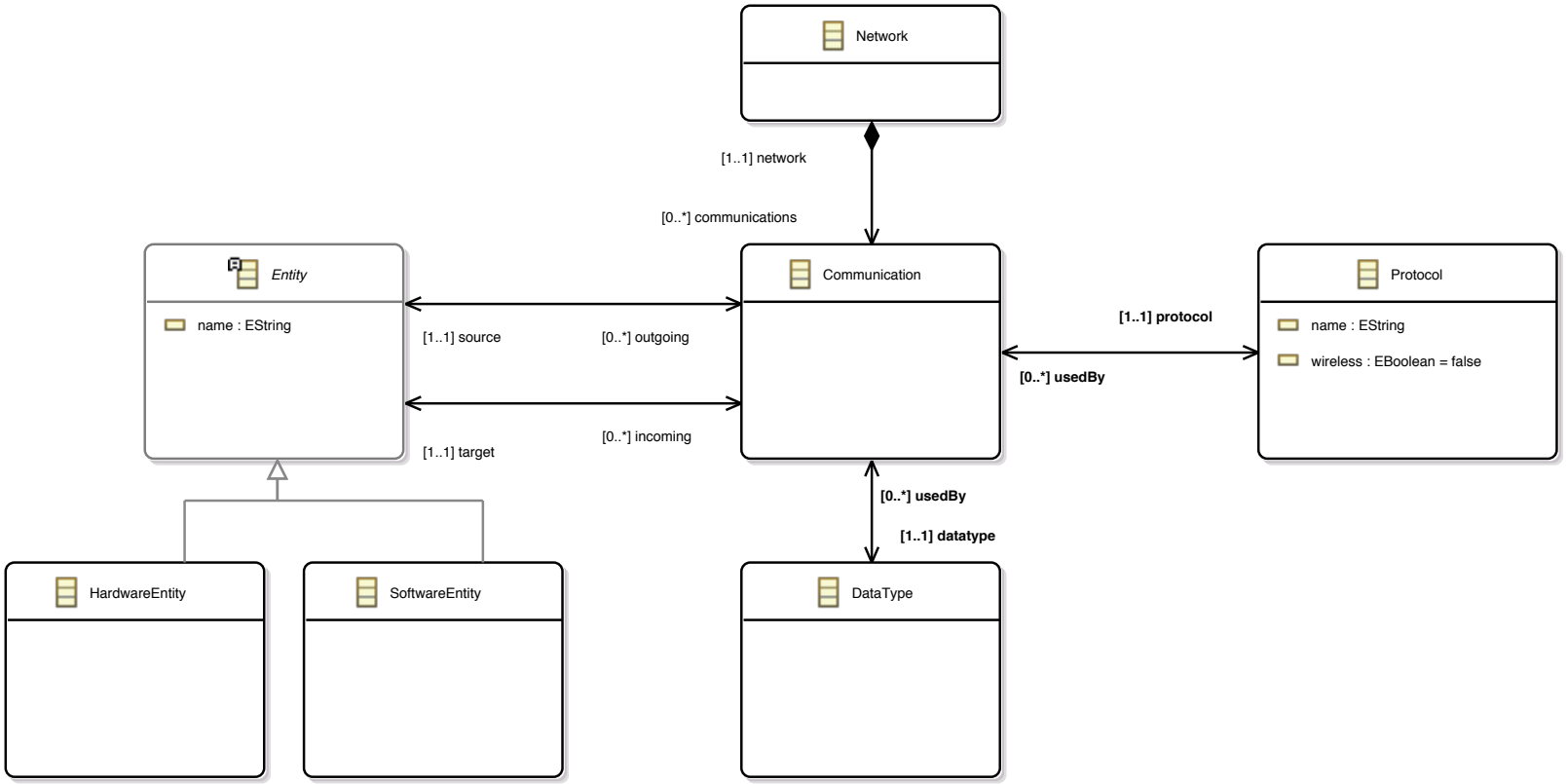


IoT Languages: Rules

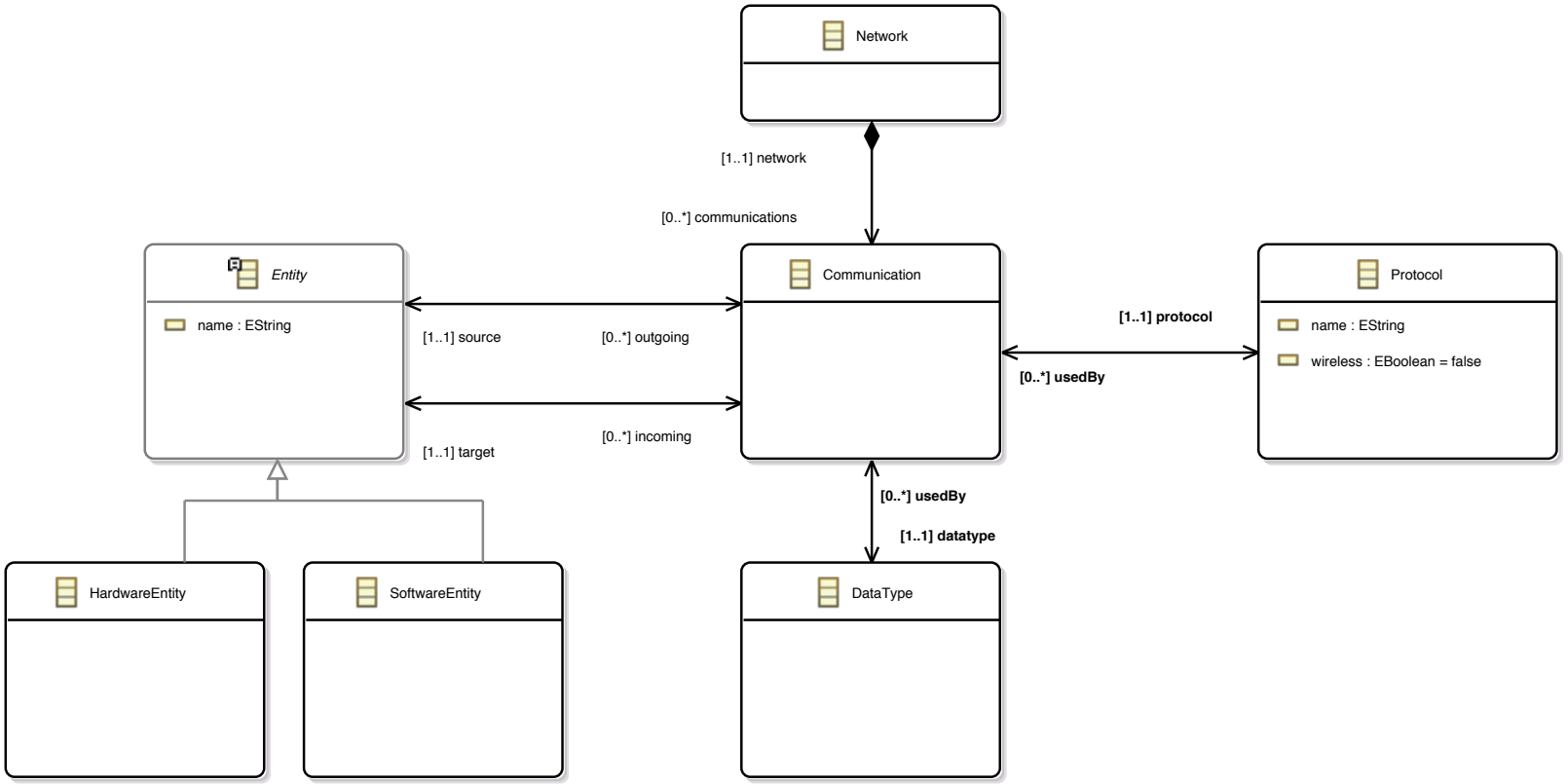


- **When** value of `SmartOffice` changed to `Presentation` **than** execute `Blind.close` **and** execute `Bulb.switchOff`
- **When** value of `SmartOffice` changed to `Discussion` **than** execute `Blind.open`
- **When** value of `SmartOffice` changed to `Discussion` **if** `LightSensor.light < 42` **than** execute `Bulb.swithOn`

IoT Languages: Communication



IoT Languages: Communication



Rules require (indirect) Communication between . . .

- `SmartOffice` ⟶ `Blind`
- `SmartOffice` ⟶ `Bulb`
- `SmartOffice` ⟶ `LightSensor`
- ? `LightSensor` ⟶ `Bulb` ?

Need for Integration

- Model *one* IoT system, not several *independent* artifacts
- Ensure contentwise *consistency* between Things, Rules and Communication
- Enable different stakeholder to *work together*
- Enable analyses *across* different artifacts

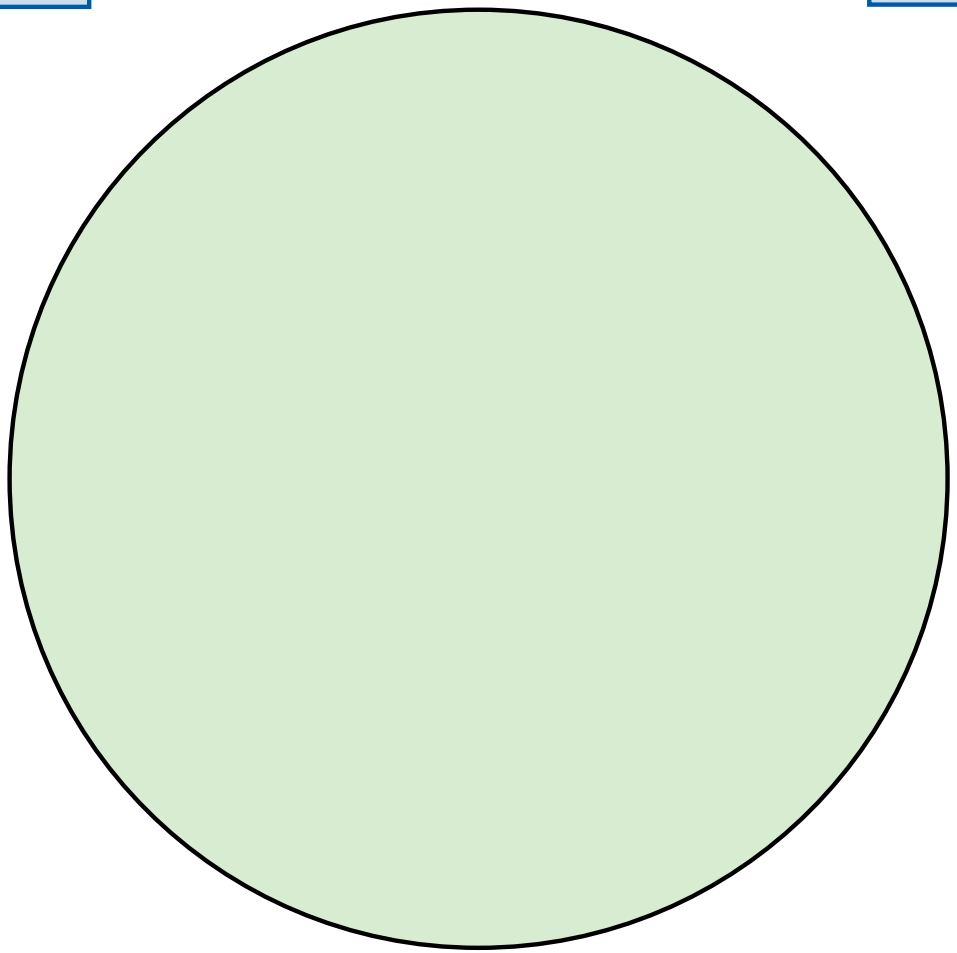
Requirements for Integration

1. Ensure Consistency between Views automatically
2. Support initial View(point)s
3. Integrated (Meta)Model

Integration: Related work

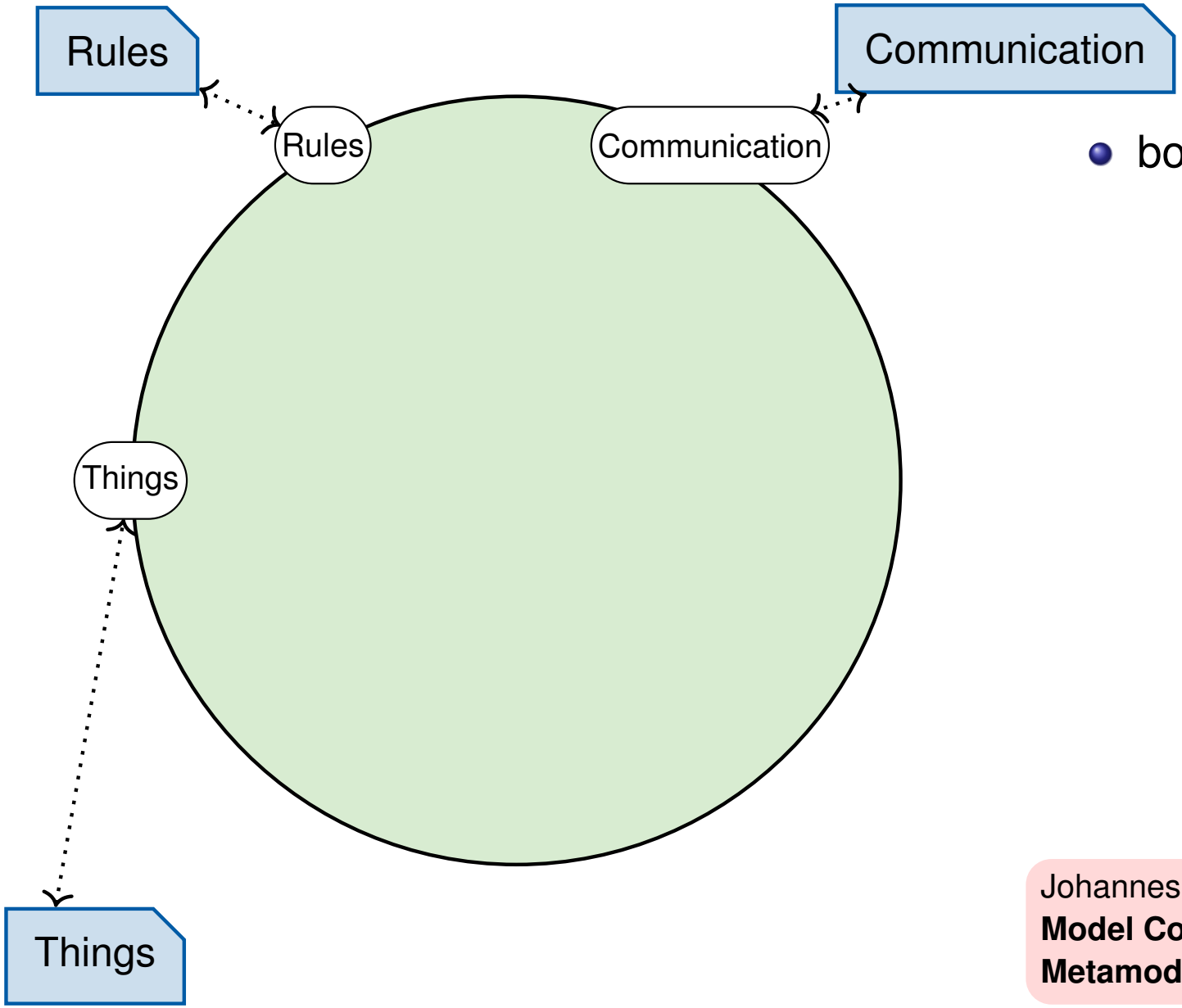
1. Idea: treat IoT languages as projectional viewpoints onto the whole system
2. Single Underlying (Meta)Model, SUM(M) [Atkinson, Stoll, and Bostan, 2009]
3. Reuse approach for ensuring Consistency in Multi-View-Environments [Meier, Klare, Tunjic, Atkinson, Burger, Reussner, and Winter, 2019]
    - Requirements: reuse existing artifacts, explicit SUM(M)
    - $\rightarrow$ MoConseMI

MoConseMI at a glance [Meier and Winter, 2018]



Johannes Meier and Andreas Winter: **Model Consistency ensured by Metamodel Integration** (2018)

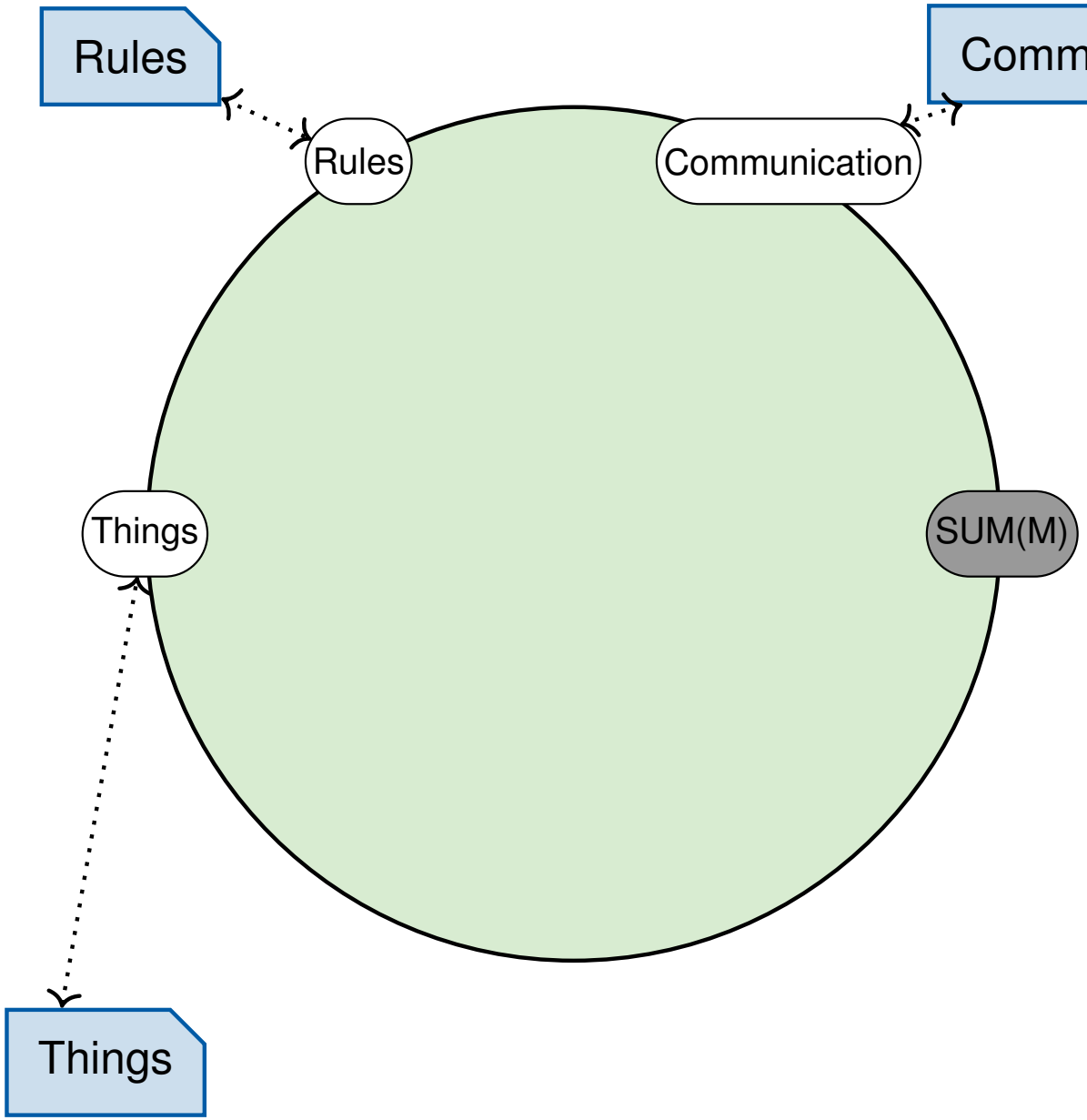
MoConseMI at a glance [Meier and Winter, 2018]



- bottom-up

Johannes Meier and Andreas Winter: **Model Consistency ensured by Metamodel Integration** (2018)

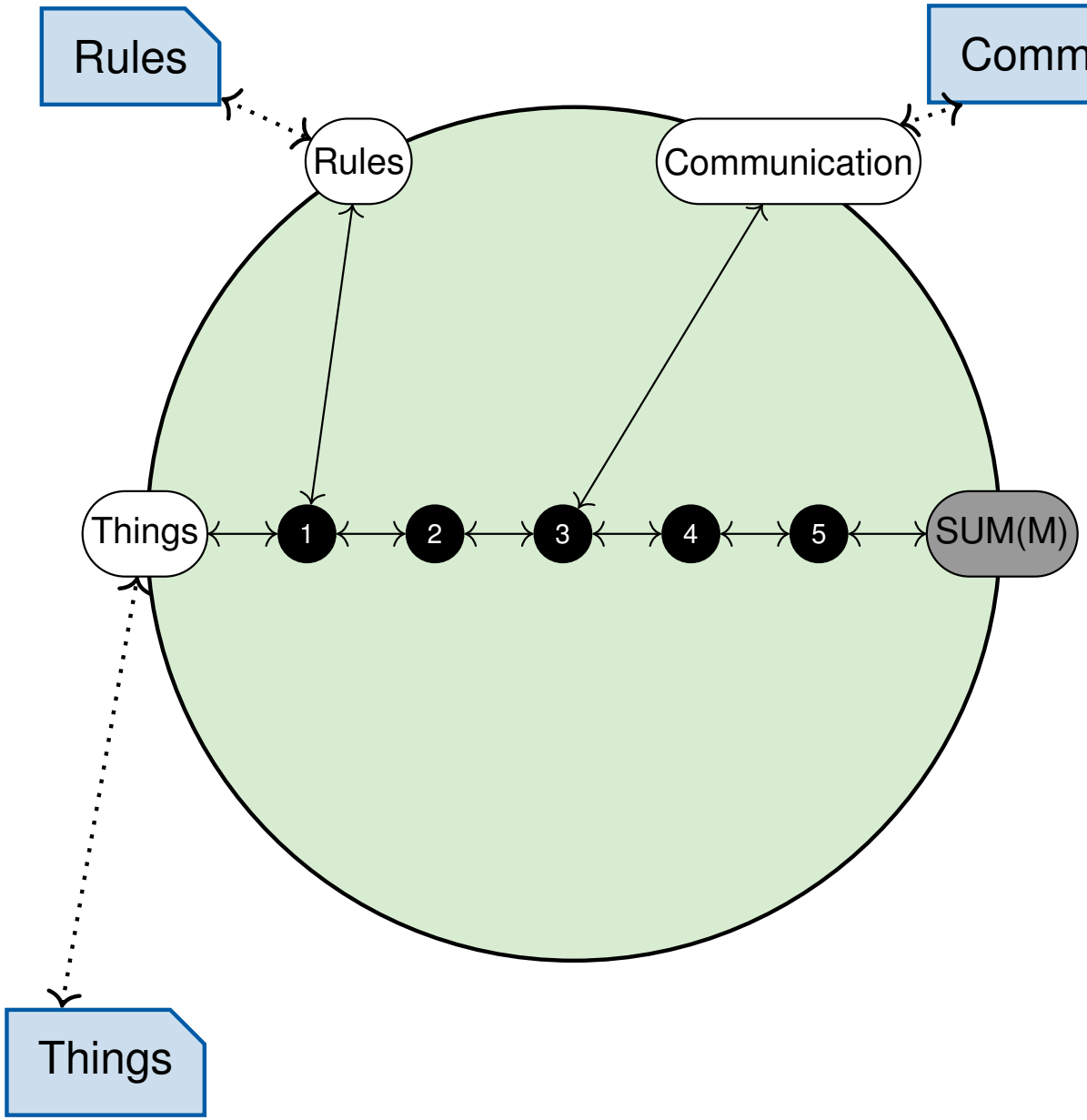
MoConseMI at a glance [Meier and Winter, 2018]



- bottom-up
- explicit SUM(M)

Johannes Meier and Andreas Winter: **Model Consistency ensured by Metamodel Integration** (2018)

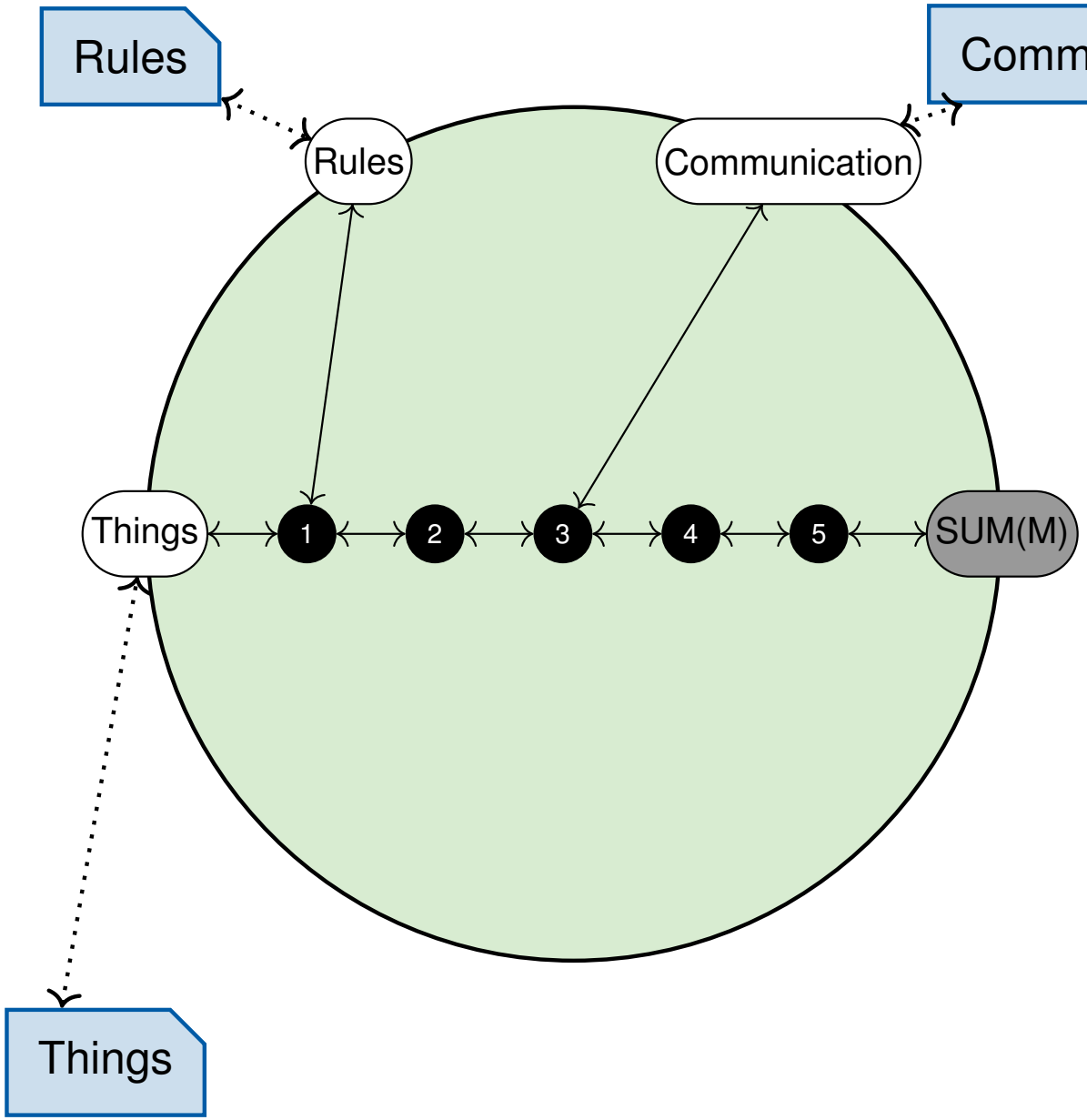
MoConseMI at a glance [Meier and Winter, 2018]



- bottom-up
- explicit SUM(M)
- Operator-based

Johannes Meier and Andreas Winter: **Model Consistency ensured by Metamodel Integration** (2018)

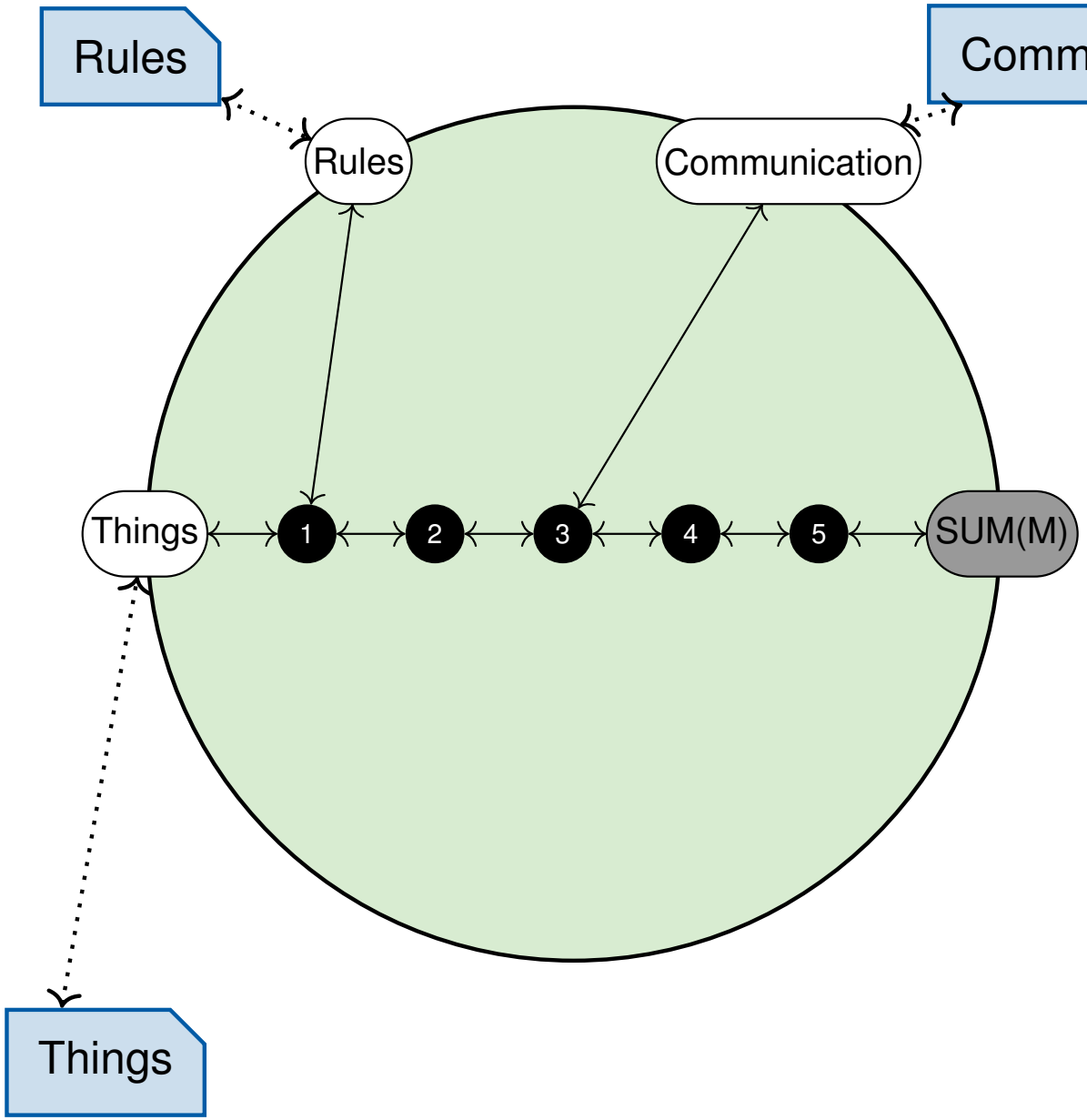
MoConseMI at a glance [Meier and Winter, 2018]



- bottom-up
- explicit SUM(M)
- Operator-based
  - → provided by MoConseMI

Johannes Meier and Andreas Winter: **Model Consistency ensured by Metamodel Integration** (2018)

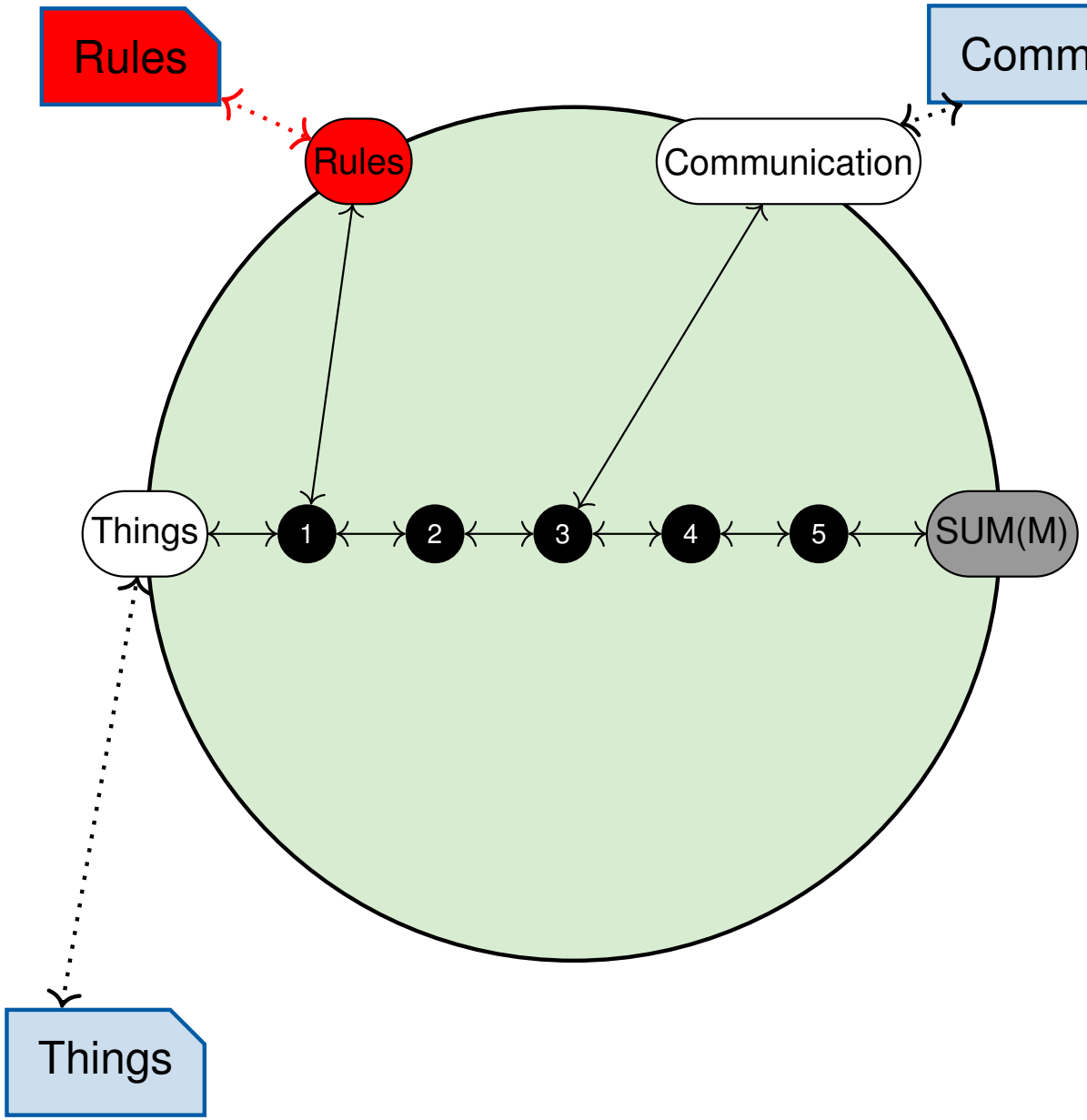
MoConseMI at a glance [Meier and Winter, 2018]



- bottom-up
- explicit SUM(M)
- Operator-based
  - → provided by MoConseMI
  - → configured by the IoT Expert

Johannes Meier and Andreas Winter: **Model Consistency ensured by Metamodel Integration** (2018)

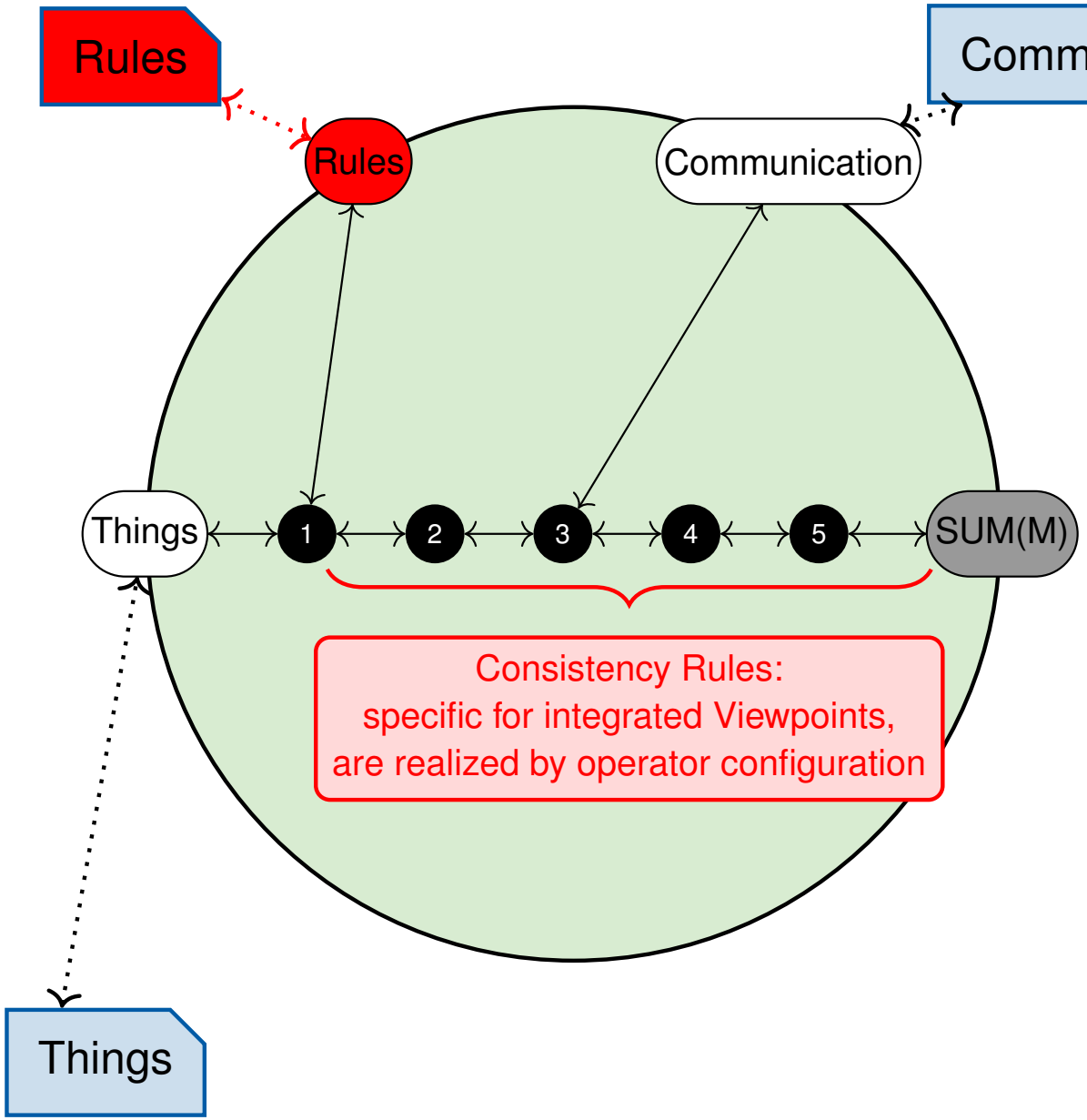
MoConseMI at a glance [Meier and Winter, 2018]



- bottom-up
- explicit SUM(M)
- Operator-based
  - → provided by MoConseMI
  - → configured by the IoT Expert
- IoT Developer changes Views

Johannes Meier and Andreas Winter: **Model Consistency ensured by Metamodel Integration** (2018)

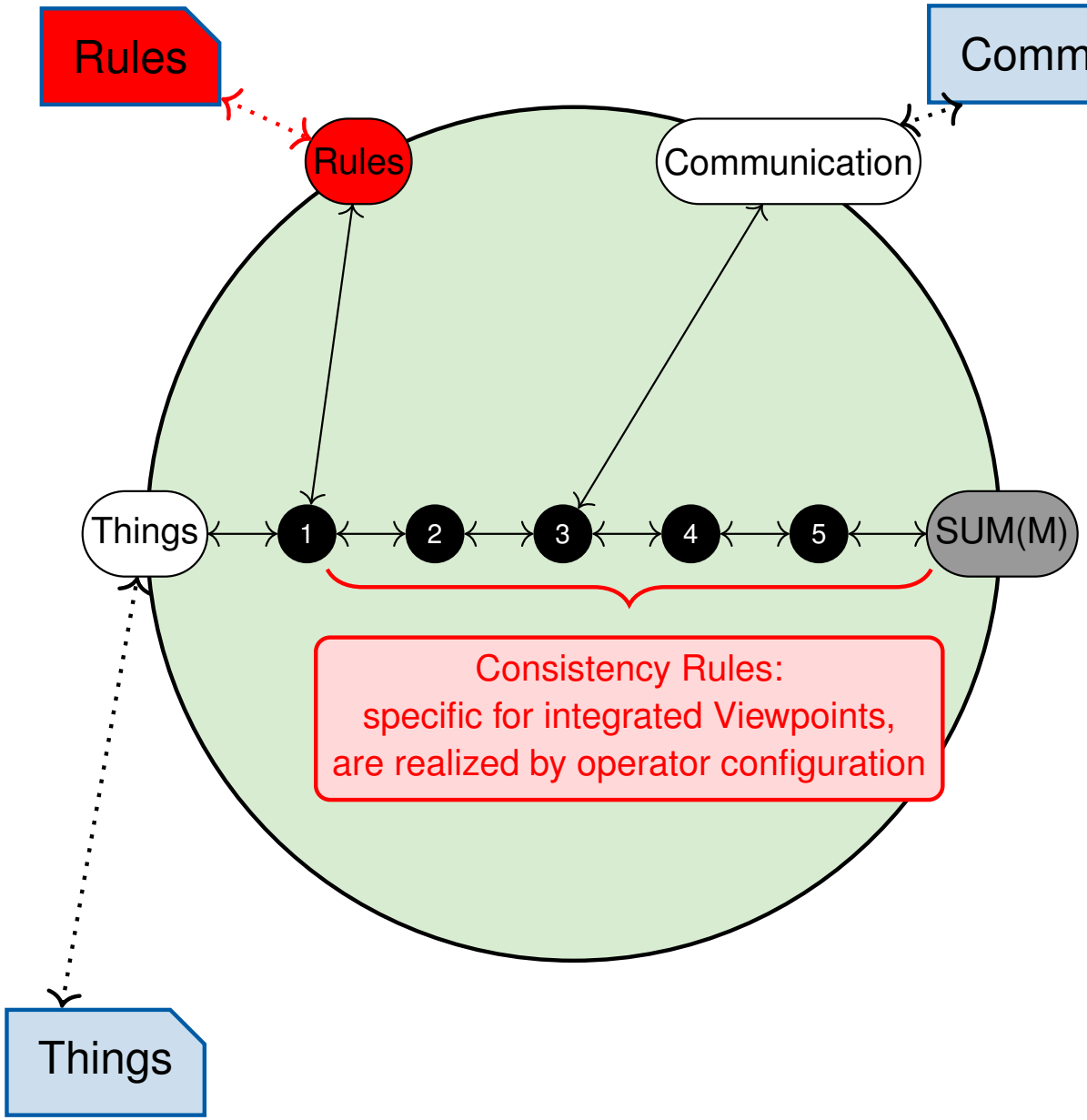
MoConseMI at a glance [Meier and Winter, 2018]



- bottom-up
- explicit SUM(M)
- Operator-based
  - → provided by MoConseMI
  - → configured by the IoT Expert
- IoT Developer changes Views

Johannes Meier and Andreas Winter: **Model Consistency ensured by Metamodel Integration** (2018)

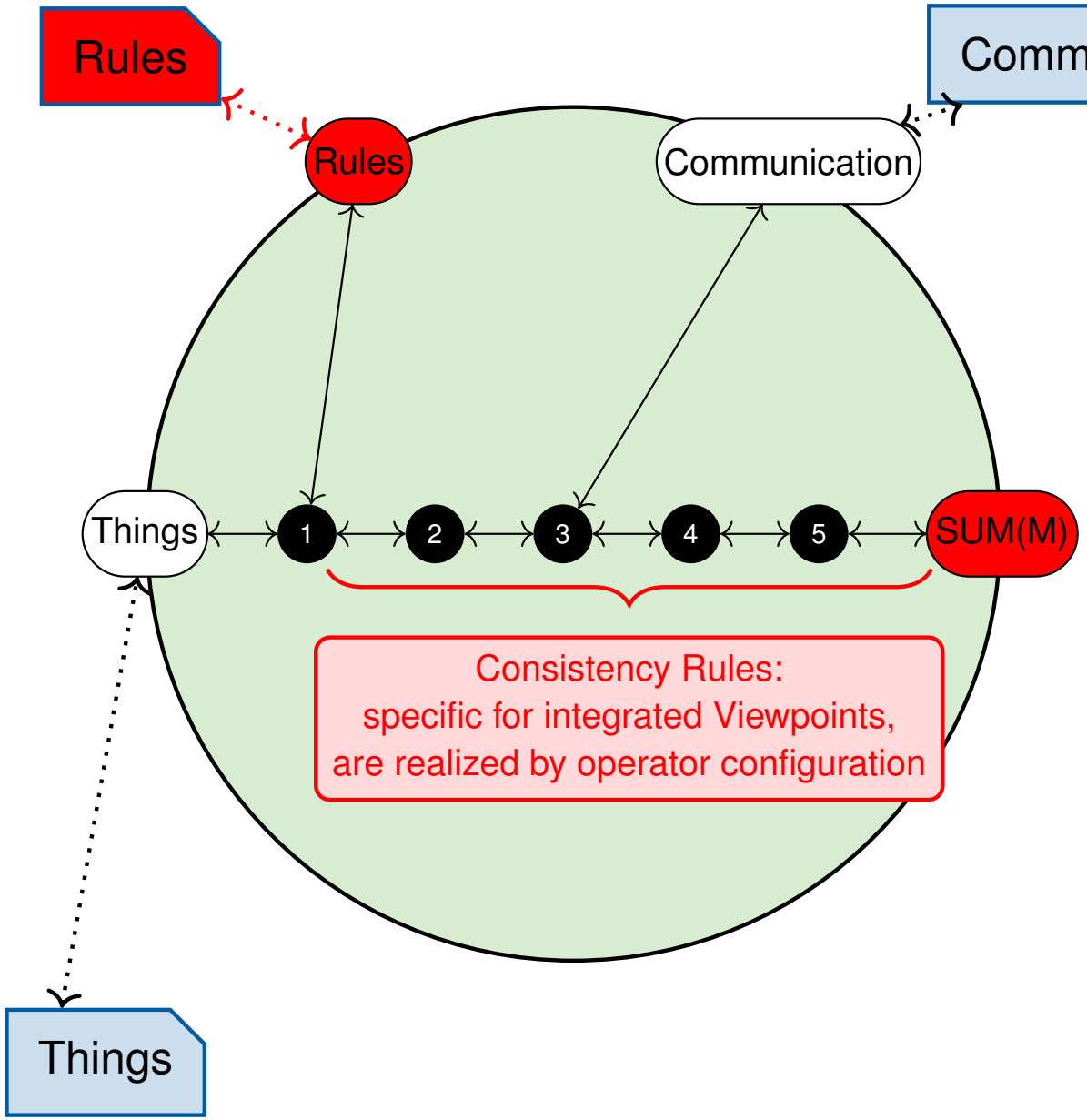
MoConseMI at a glance [Meier and Winter, 2018]



- bottom-up
- explicit SUM(M)
- Operator-based
  - → provided by MoConseMI
  - → configured by the IoT Expert
- IoT Developer changes Views

Johannes Meier and Andreas Winter: **Model Consistency ensured by Metamodel Integration** (2018)

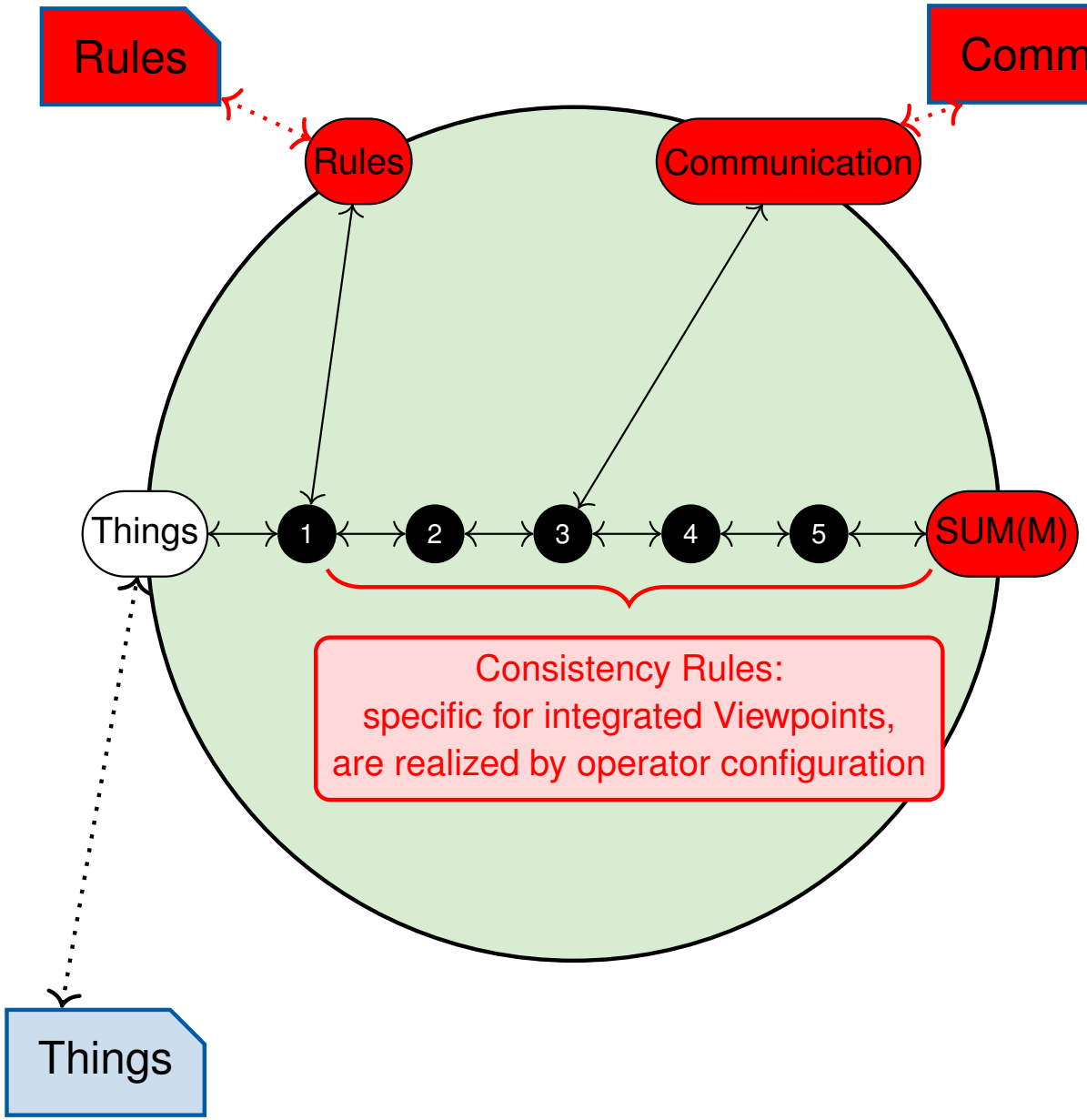
MoConseMI at a glance [Meier and Winter, 2018]



- bottom-up
- explicit SUM(M)
- Operator-based
  - → provided by MoConseMI
  - → configured by the IoT Expert
- IoT Developer changes Views

Johannes Meier and Andreas Winter: **Model Consistency ensured by Metamodel Integration** (2018)

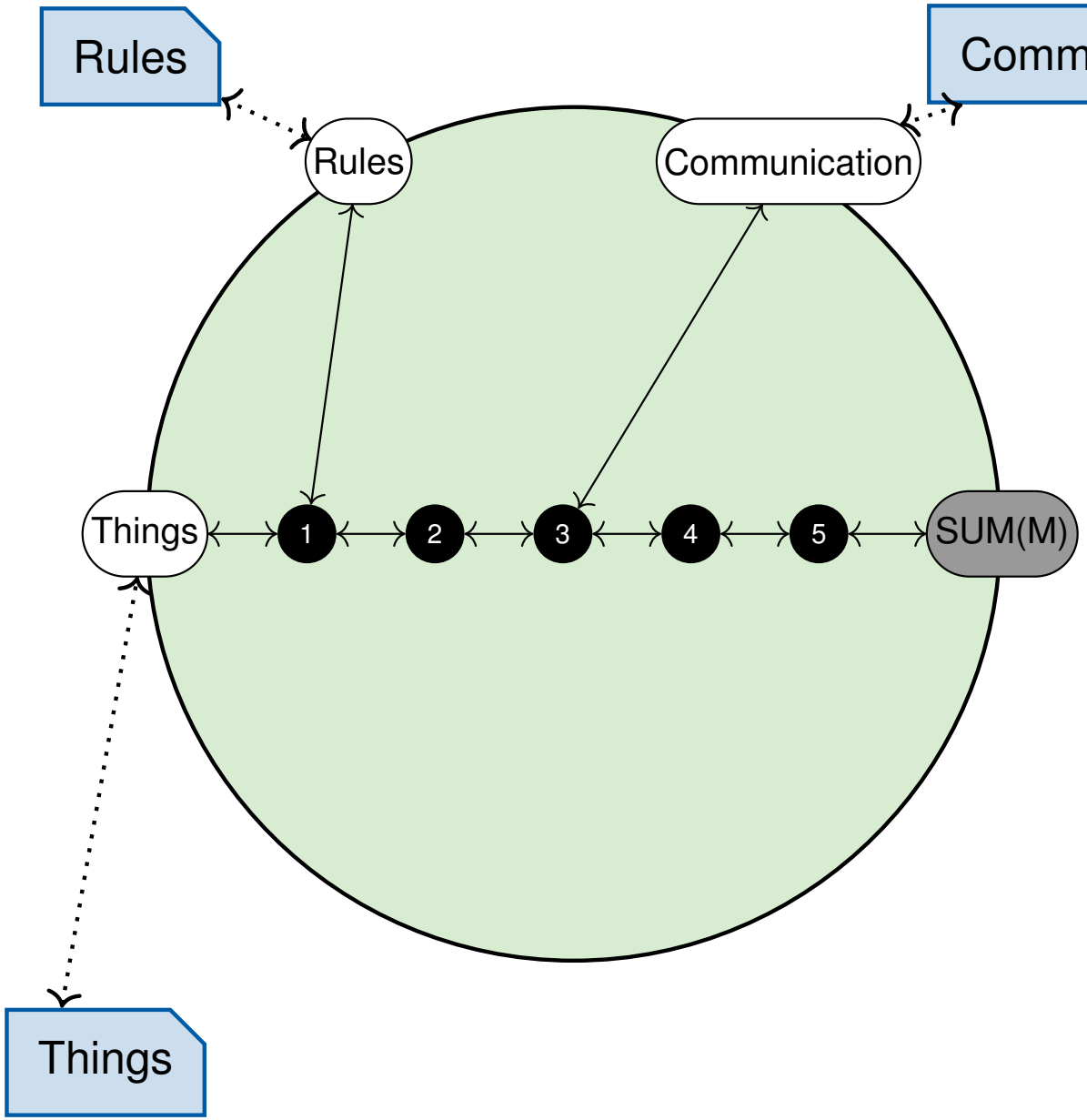
MoConseMI at a glance [Meier and Winter, 2018]



- bottom-up
- explicit SUM(M)
- Operator-based
  - → provided by MoConseMI
  - → configured by the IoT Expert
- IoT Developer changes Views

Johannes Meier and Andreas Winter: **Model Consistency ensured by Metamodel Integration** (2018)

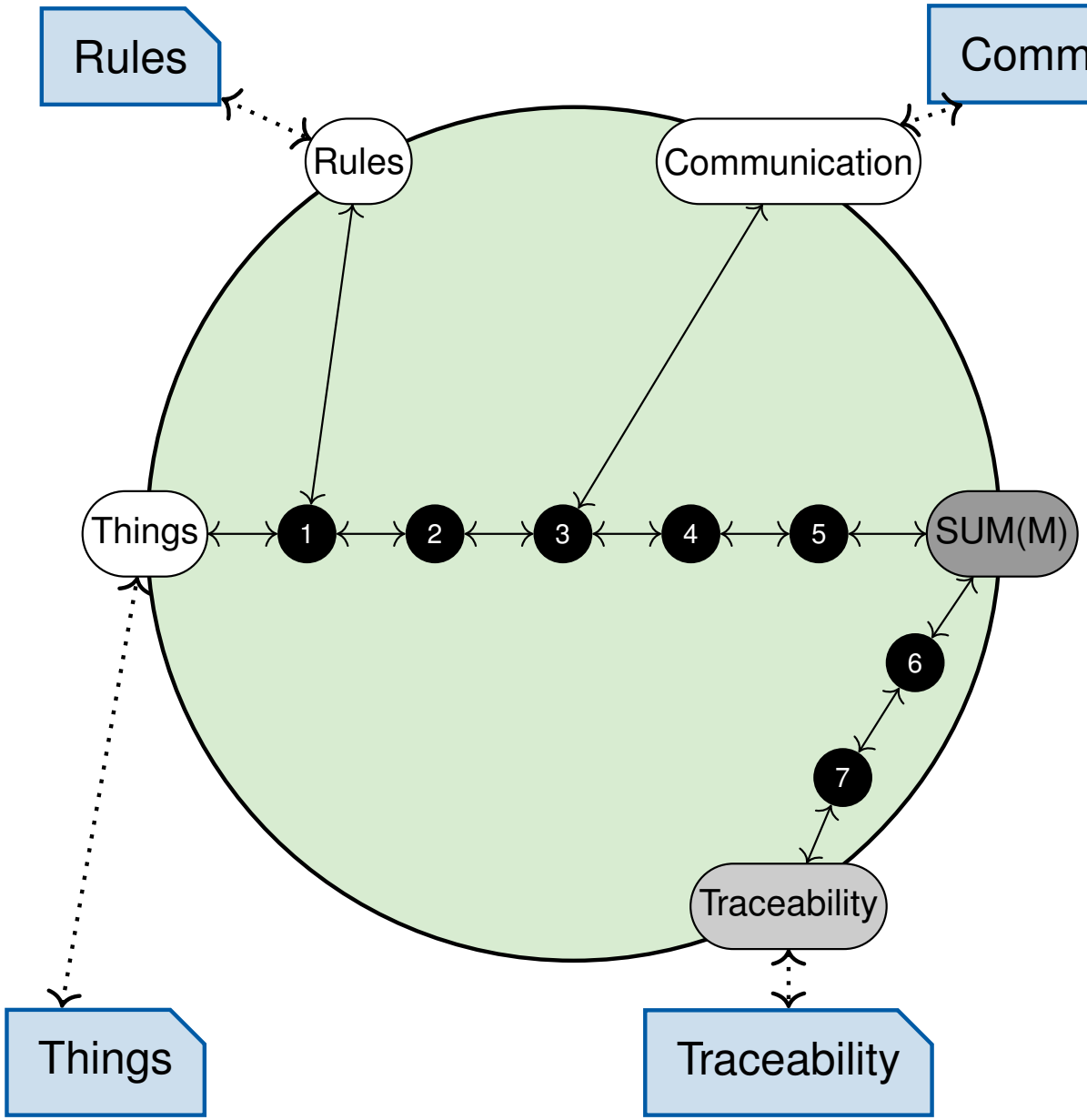
MoConseMI at a glance [Meier and Winter, 2018]



- bottom-up
- explicit SUM(M)
- Operator-based
  - → provided by MoConseMI
  - → configured by the IoT Expert
- IoT Developer changes Views

Johannes Meier and Andreas Winter: **Model Consistency ensured by Metamodel Integration** (2018)

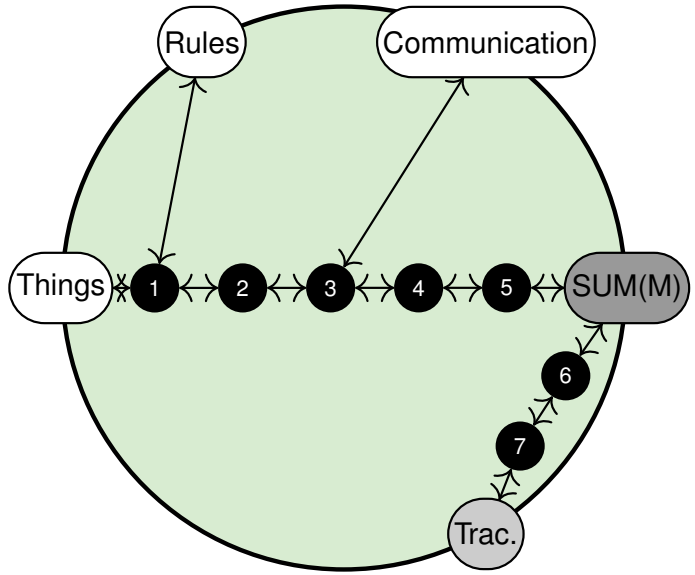
MoConseMI at a glance [Meier and Winter, 2018]



- bottom-up
- explicit SUM(M)
- Operator-based
  - → provided by MoConseMI
  - → configured by the IoT Expert
- IoT Developer changes Views
- new View(Point)s

Johannes Meier and Andreas Winter: **Model Consistency ensured by Metamodel Integration** (2018)

Transformation Challenges



Challenges for Transformations:

1. transform whole Models and Metamodels
2. support both Directions
3. propagate Changes to all Models
4. provide Changes inside the same Model
5. prevent Information Loss
6. generic / flexible: arbitrary (Meta)Models and Consistency Rules
7. reusable in different IoT Projects $\rightarrow$ Granularity

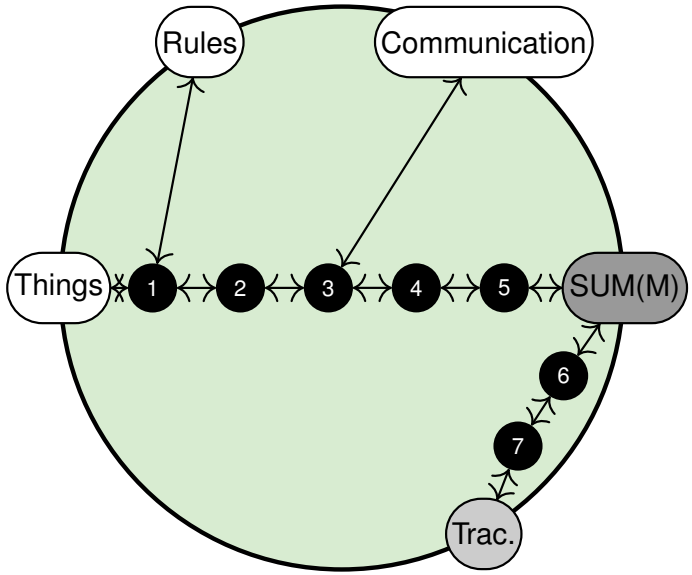
Design Decision “Operators as Transformations”:

- reusable Units for the IoT Expert
- created once and provided as Library by MoConseMI
- understandable: intermediate steps for debugging, visualizations

Consistency between IoT Languages

Consistency Rules . . .

- specify the impact of User Changes
- fulfilled by configured chain of operators



Things

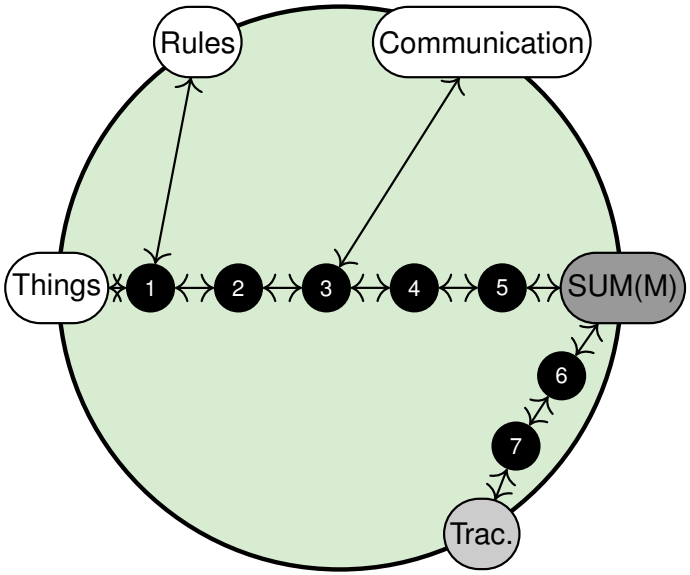
Rules

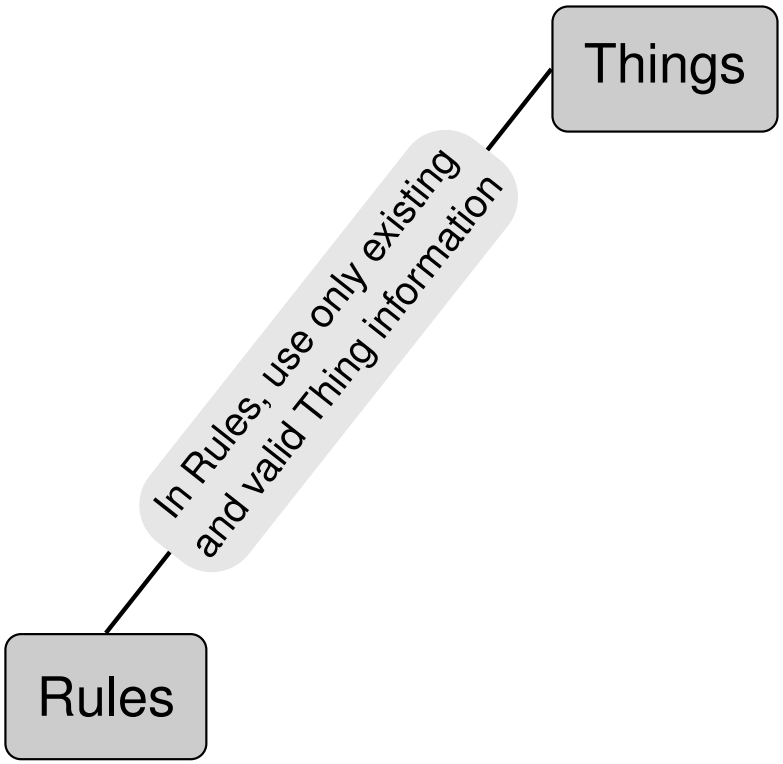
Communication

Consistency between IoT Languages

Consistency Rules . . .

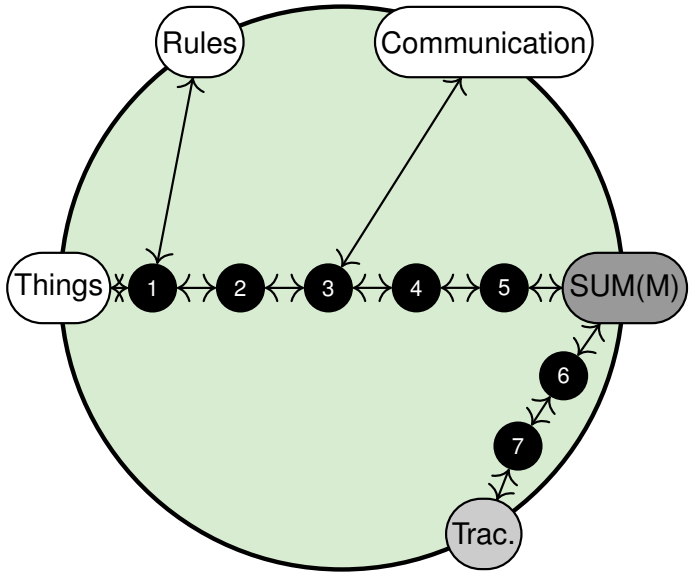
- specify the impact of User Changes
- fulfilled by configured chain of operators





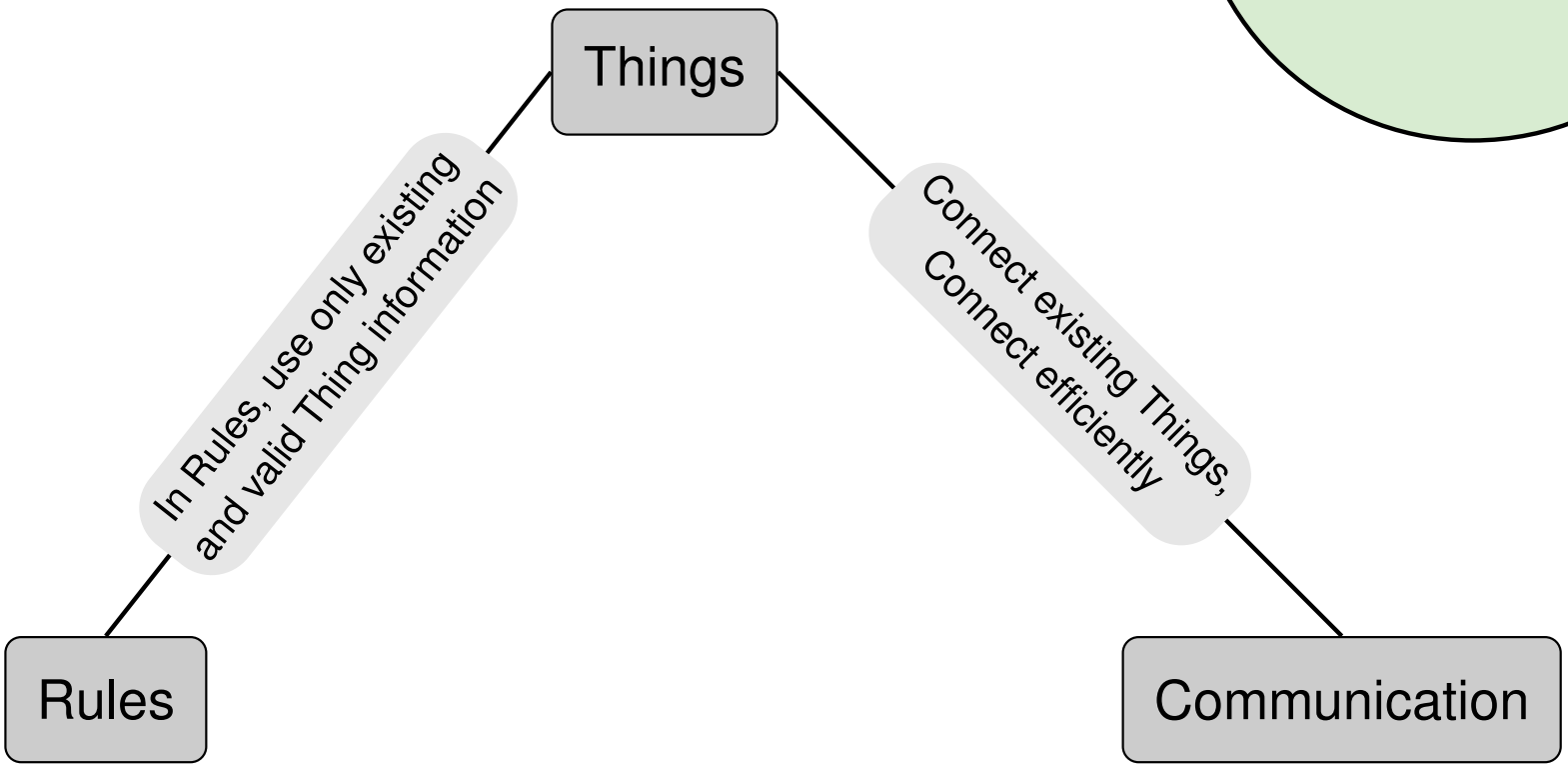
Communication

Consistency between IoT Languages



Consistency Rules . . .

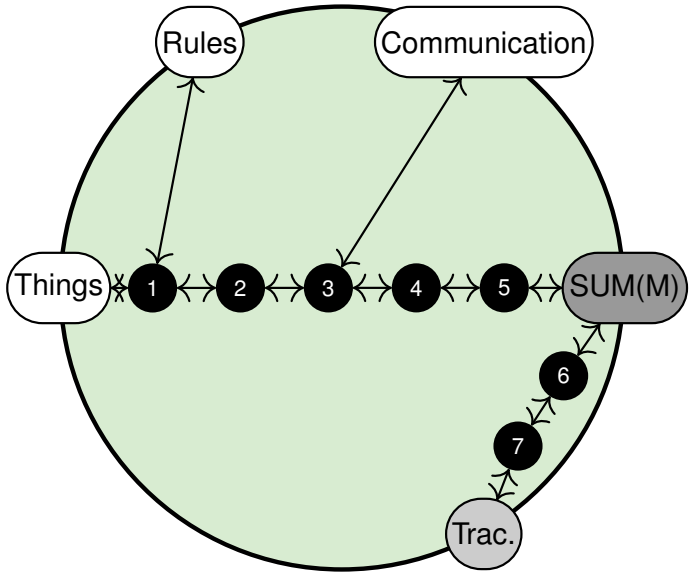
- specify the impact of User Changes
- fulfilled by configured chain of operators

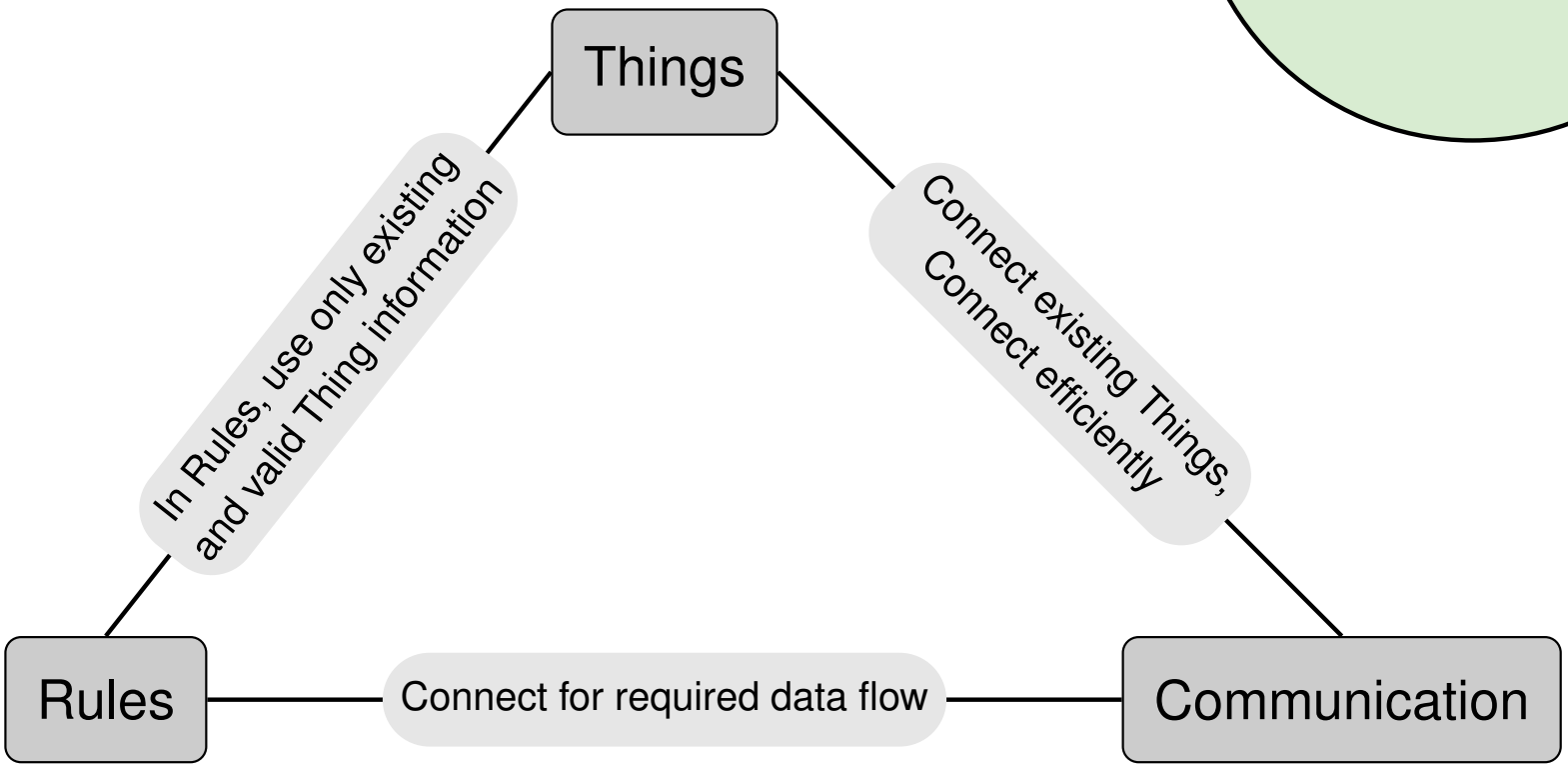


Consistency between IoT Languages

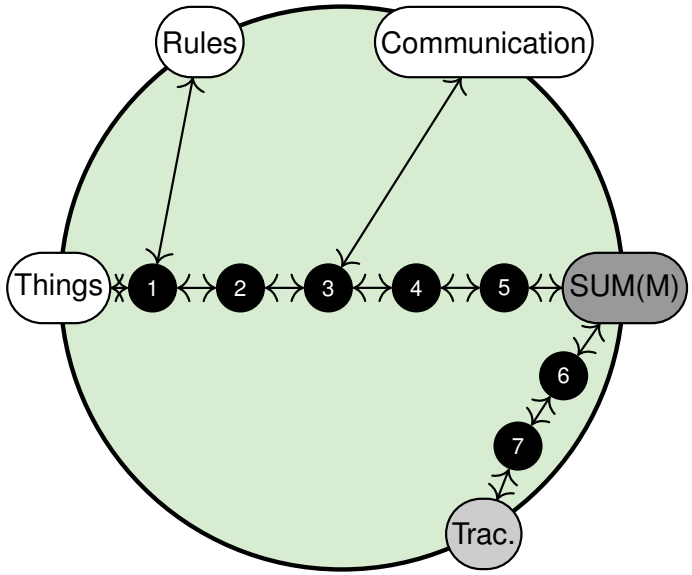
Consistency Rules . . .

- specify the impact of User Changes
- fulfilled by configured chain of operators



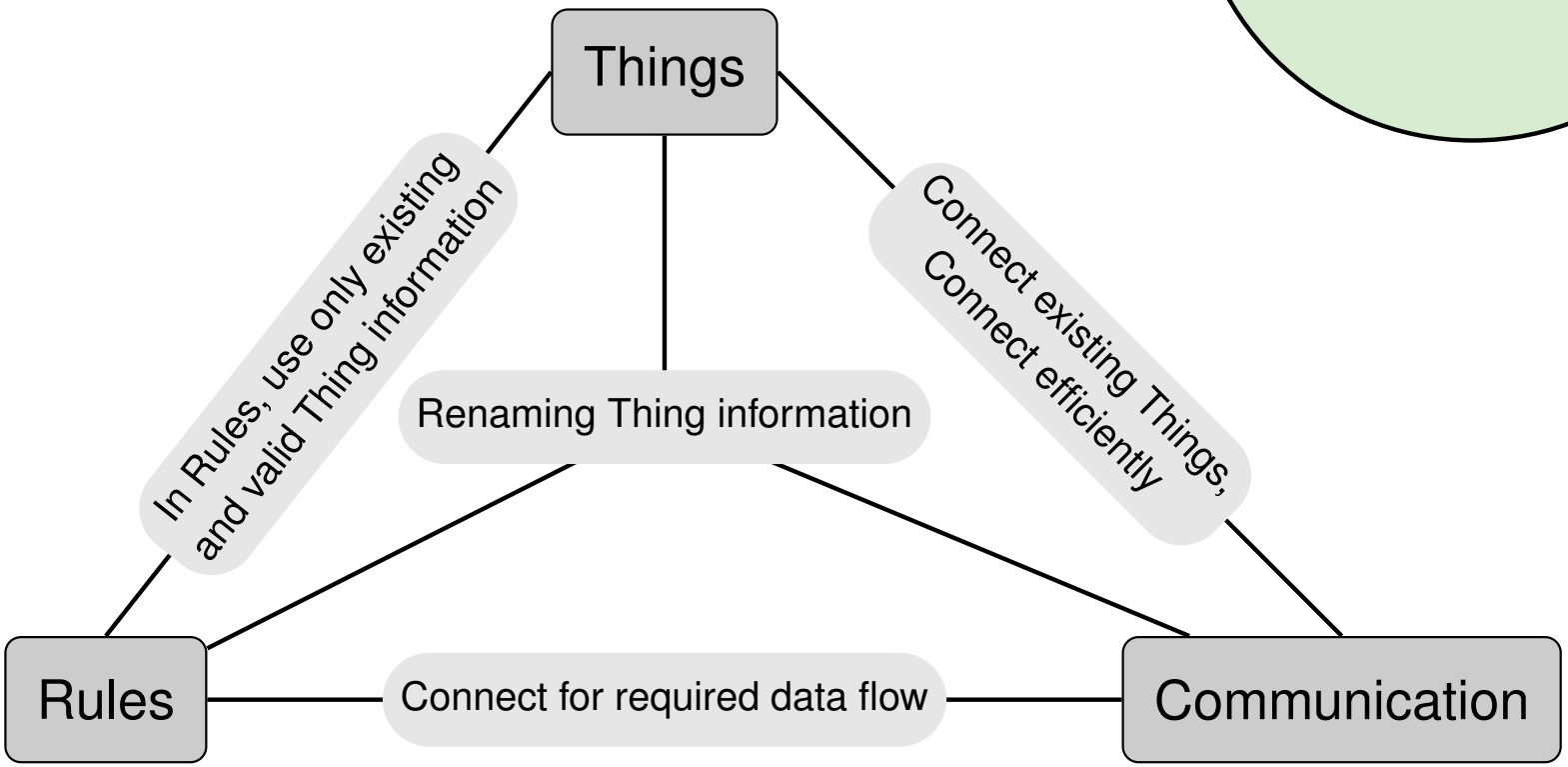


Consistency between IoT Languages



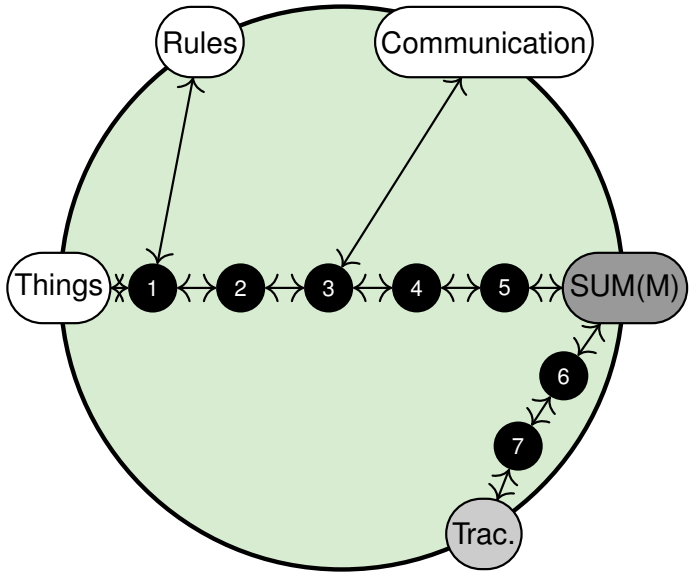
Consistency Rules . . .

- specify the impact of User Changes
- fulfilled by configured chain of operators

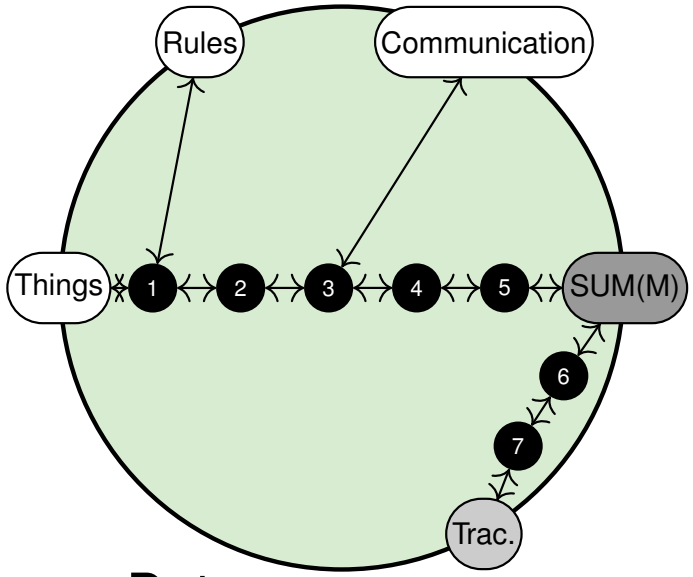


Summary: Contributions

- Use tailored IoT Languages to . . .
  - describe single aspects of IoT systems
  - to manage complexity
    by separation of concerns
- Integrate IoT Languages explicitly to . . .
  - ensure Consistency automatically
  - enable cross-analyses
- sketched examples:
  - SmartOffice example
  - 3 IoT Languages
  - Integration



Summary: Outlook



- Improve and extend the sketched IoT Languages, e.g. States, Guards
- Identify and develop more IoT Languages, e.g. Data
- Extend the integration of IoT Languages
- Apply the integrated IoT Languages for a big Application, e.g. SmartHome

Literature I

Colin Atkinson, Dietmar Stoll, and Philipp Bostan. Supporting View-Based Development through Orthographic Software Modeling. Evaluation of Novel Approaches to Software Engineering (ENASE), pages 71–86, 2009.

Marsha Chechik, Shiva Nejati, and Mehrdad Sabetzadeh. A Relationship-Based Approach to Model Integration. Innovations Syst Softw Eng, 8(123):3–18, 2012. doi: 10.1007/s11334-011-0155-2.

Johannes Meier and Andreas Winter. Model Consistency ensured by Metamodel Integration. 6th International Workshop on The Globalization of Modeling Languages, co-located with MODELS 2018, 2018.

Johannes Meier, Heiko Klare, Christian Tunjic, Colin Atkinson, Erik Burger, Ralf Reussner, and Andreas Winter. Single Underlying Models for Projectional, Multi-View Environments. In Slimane Hammoudi, Luis Ferreira Pires, and Bran Selic, editors, Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development, pages 119–130. SCITEPRESS - Science and Technology Publications, 2019. ISBN 978-989-758-358-2. doi: 10.5220/0007396401190130. URL http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0007396401190130.