UNIVERSITÀ DEGLI STUDI DI UDINE

hic sunt futura

Behavioral Equivalences for AbU: Verifying Security and Safety in Distributed IoT Systems

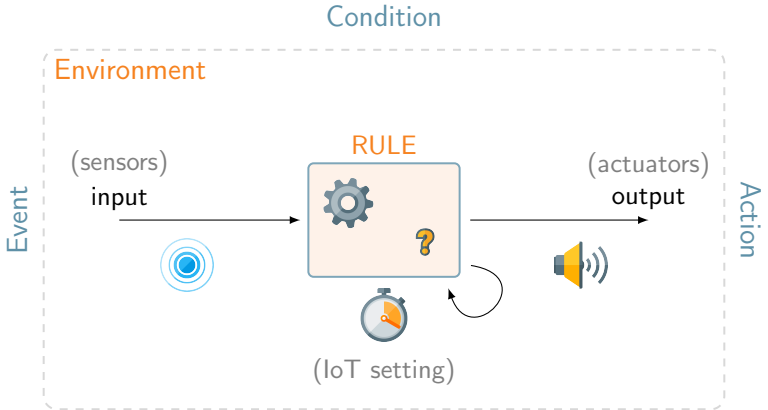
joint work with *Michele Pasqua* from *University of Verona*

May 24, 2023

Marino Miculan marino.miculan@uniud.it

IMT Alti Studi - Lucca

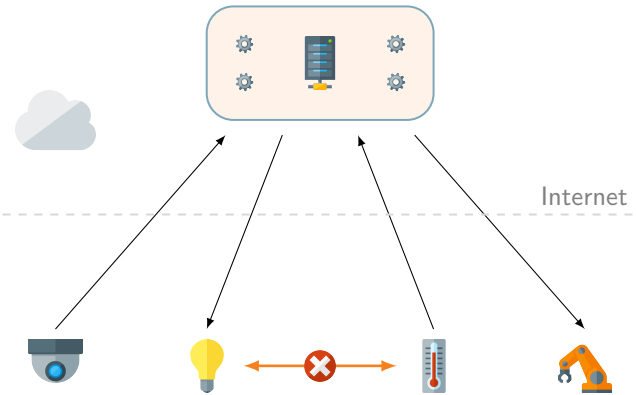
The Event-Condition-Action model



Memory-based rules: "**when** movement **if** is-night **then** ring-alarm"

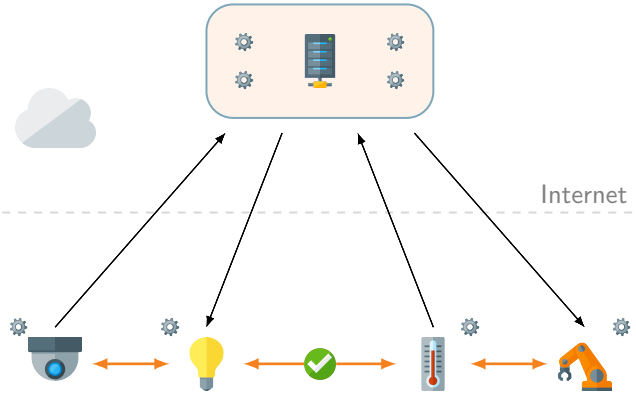
Actual smart (ECA) devices setting

- Centralized
- No intra-nodes communication
- Cloud-dependent
- Very popular:



Next smart (ECA) devices setting: *edge computing*

- Fully distributed
- Communication between nodes
- Cloud-agnostic
- Identity decoupled, for scalability

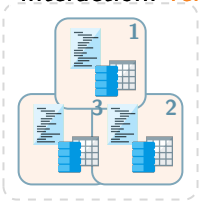


Attribute-based Memory Updates

Nodes behavior: defined by ECA rules like "on z for all $\Pi : x \leftarrow e$"

Nodes state: local memory

Interaction: remote updates



Attribute-based interaction: on all nodes satisfying Π, update the remote x with e

(Inspired by Attribute-based communication, by Alrahman, De Nicola, Loreti)

The AbU calculus

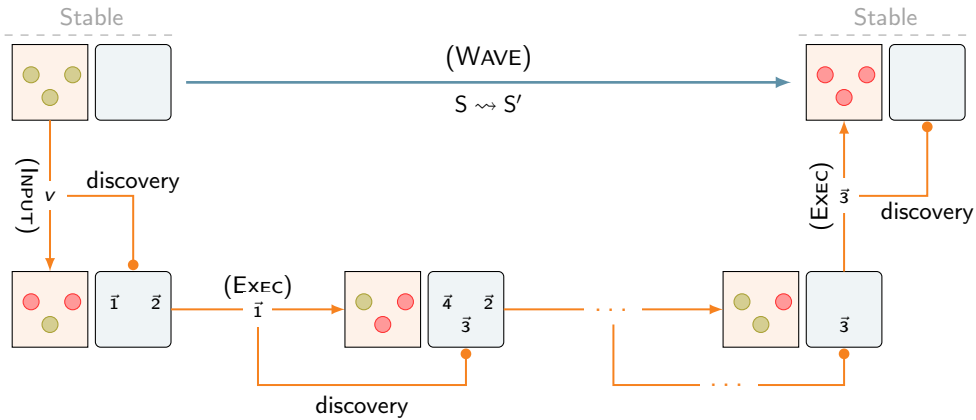
- An **AbU system** S is an **AbU node** $R, \iota\langle\Sigma, \Theta\rangle$ or the parallel of systems $S_1 \parallel S_2$
- Each node is equipped with a list R of **AbU rules** and an **invariant** ι

for all: $@(x < \overline{x}) \gg \overline{x} \leftarrow x$

"select all nodes with (remote) x greater than the current (local) x"

"assign the selected nodes (remote) x with the current (local) x"

AbU execution model



AbU operational semantics

LTS semantics, with judgments:

$$R, \iota\langle\Sigma, \Theta\rangle \xrightarrow{\alpha}_{\triangleright} R, \iota\langle\Sigma', \Theta'\rangle$$

A label α can be:

- an **input** label, upd $\blacktriangleright$ T
- an **execution** label, upd $\triangleright$ T
- a **discovery** label, T

(See paper on TCS 2023, DOI 10.1016/j.tcs.2023.113841)

AbU operational semantics: rules

$$(\textsc{Exec})\ \frac{\mathsf{upd} \in \Theta \quad \mathsf{upd} = (x_1, v_1) \dots (x_k, v_k) \quad \Sigma' = \Sigma[v_1/x_1 \dots v_k/x_k] \quad \Sigma' \models \iota \quad \Theta'' = \Theta \setminus \{\mathsf{upd}\} \quad X = \{x_i \mid i \in [1..k] \wedge \Sigma(x_i) \neq \Sigma'(x_i)\} \quad \Theta' = \Theta'' \cup \mathsf{DefUpds}(R, X, \Sigma') \cup \mathsf{LocalUpds}(R, X, \Sigma') \quad T = \mathsf{ExtTasks}(R, X, \Sigma')}{R, \iota\langle \Sigma, \Theta\rangle \xrightarrow{\mathsf{upd} \triangleright T} R, \iota\langle \Sigma', \Theta'\rangle}$$

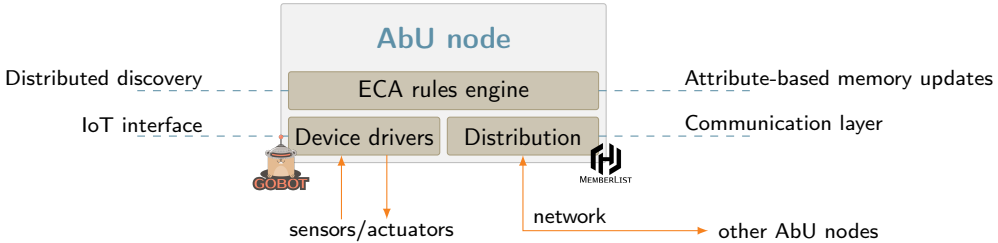
$$(\textsc{Exec-Fail})\ \frac{\mathsf{upd} \in \Theta \quad \mathsf{upd} = (x_1, v_1) \dots (x_k, v_k) \quad \Sigma' = \Sigma[v_1/x_1 \dots v_k/x_k] \quad \Sigma' \not\models \iota \quad \Theta' = \Theta \setminus \{\mathsf{upd}\}}{R, \iota\langle \Sigma, \Theta\rangle \xrightarrow{\mathsf{upd} \triangleright T} R, \iota\langle \Sigma, \Theta'\rangle}$$

$$(\textsc{Input})\ \frac{v_1, \dots, v_k \in \mathbb{V} \quad \Sigma' = \Sigma[v_1/x_1 \dots v_k/x_k] \quad X = \{x_1, \dots, x_k\} \quad \Theta' = \Theta \cup \mathsf{DefUpds}(R, X, \Sigma') \cup \mathsf{LocalUpds}(R, X, \Sigma') \quad T = \mathsf{ExtTasks}(R, X, \Sigma')}{R, \iota\langle \Sigma, \Theta\rangle \xrightarrow{(x_1, v_1) \dots (x_k, v_k) \blacktriangleright T} R, \iota\langle \Sigma', \Theta'\rangle}$$

$$(\textsc{Disc})\ \frac{\Theta'' = \{[\![\mathsf{act}]\!]\Sigma \mid \exists i \in [1..n]\,.\,\mathsf{task}_i = \varphi : \mathsf{act} \wedge \Sigma \models \varphi\} \quad \Theta' = \Theta \cup \Theta''}{R, \iota\langle \Sigma, \Theta\rangle \xrightarrow{\mathsf{task}_1 \dots \mathsf{task}_n} R, \iota\langle \Sigma, \Theta'\rangle}$$

$$(\textsc{StepL})\ \frac{S_1 \xrightarrow{\alpha} S_1' \quad S_2 \xrightarrow{T} S_2'}{S_1 \parallel S_2 \xrightarrow{\alpha} S_1' \parallel S_2'}\ \alpha \in \{\mathsf{upd} \triangleright T, \mathsf{upd} \blacktriangleright T\} \qquad (\textsc{StepR})\ \frac{S_1 \xrightarrow{T} S_1' \quad S_2 \xrightarrow{\alpha} S_2'}{S_1 \parallel S_2 \xrightarrow{\alpha} S_1' \parallel S_2'}\ \alpha \in \{\mathsf{upd} \triangleright T, \mathsf{upd} \blacktriangleright T\}$$

A (modular) distributed implementation



- ECA rules engine module: AbU semantics
- Device drivers module: abstraction of physical resources
- Distribution module: abstraction of send/receive and cluster nodes join/leave
- Available at `https://github.com/abu-lang`

The AbU Language: a Domain Specific Language for the IoT

```
# AbU devices definition.

hvac : "An HVAC control system" {
   physical output boolean heating = false
   physical output boolean condit = false
   logical integer temp = 0
   logical integer humidity = 0
   physical input boolean airButton
   logical string node = "hvac"
      where not (condit and heating == true)
} has cool warm dry stopAir

tempSens : "A temperature sensor" {
   physical input integer temp
   logical string node = "tempSens"
} has notifyTemp

humSens : "A humidity sensor" {
   physical input integer humidity
   logical string node = "humSens"
} has notifyHum
```

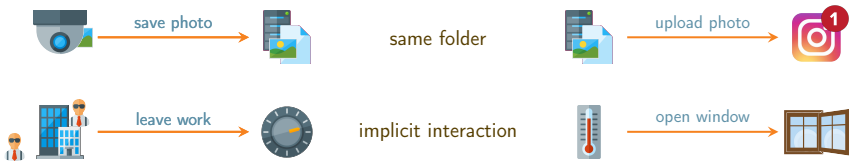
```
\%
   AbU (ECA) rules definition.
   Rules can be referenced by multiple devices.
%\

rule cool on temp
   for (this.temp < 18) do this.heating = true

rule warm on temp
   for (this.temp > 27) do this.heating = false

rule dry on humidity; temp
   for (this.temp * 0.14 < this.humidity)
      do this.condit = true

rule stopAir on airButton
   for (this.airButton) do this.condit = false

rule notifyTemp on temp
   for all (ext.node == "hvac")
      do ext.temp = this.temp
```

(See paper on IEEE Access 2022, DOI 10.1109/ACCESS.2022.3230287)

Security and Safety requirements

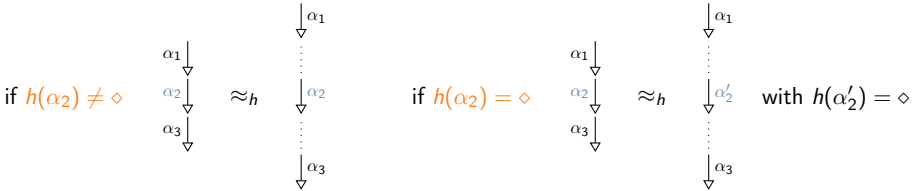
Security

Safety



Hiding bisimulation

- Weak bisimulation hiding labels not related to the requirements
- Parametric on a function h making non-observable labels α such that $h(\alpha) = \diamond$



Security h_L hides:

- discovery labels
- execution labels with H resources

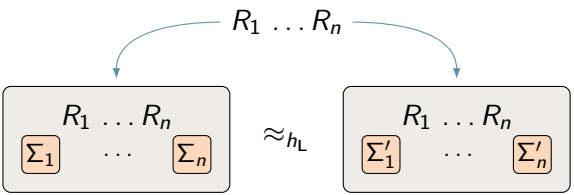
Safety h_S hides:

- discovery labels
- execution labels produced by S

Security: behavioral equivalence

Protection of confidential data (noninterference)

- Security policy: L (public) and H (confidential) resources
- No flows from H to L allowed
- Bisimulation $\approx_{h_L}$ that hides H-level updates
- $R_1 \ldots R_n$ is *interference-free* if it "behaves the same" for L-equivalent states



Hiding bisimulation:
execution labels with H resources

for all L-equivalent states $\Sigma_1 \equiv_L \Sigma'_1 \ldots \Sigma_n \equiv_L \Sigma'_n$

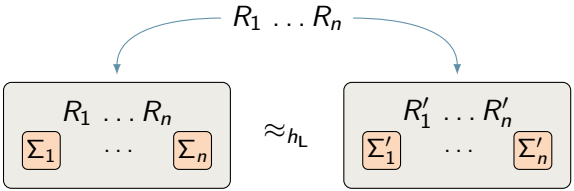
Security: behavioral equivalence

- This definition captures leaks due to internal resources modifications, but not leaks originated by external changes (i.e., inputs) on high-level variables. E.g.:

$$motion \gg (00{:}00 < time \wedge time < 06{:}00) : light \leftarrow \text{'on'}$$

(where *motion* is H and *light* is L) is interference-free as defined above, but it actually leaks confidential information.

- $R_1 \ldots R_n$ is *presence-sensitive interference-free* if it "behaves the same" for L-equivalent states *and under renaming of rule triggers of level H*



for all L-equivalent states $\Sigma_1 =_L \Sigma'_1 \ldots \Sigma_n =_L \Sigma'_n$

Security: verification algorithm

Algorithm `IFRules` for computing information flows:

- Compute a constancy analysis for conditions and expressions
- Check explicit flows for the default action
- Check explicit and implicit flows for the task action

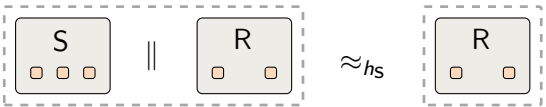
Theorem (Soundness for Security)

If `IFRules`$(R) =$ *false then* R *is noninterferent, namely* R *is secure.*

Safety: behavioral equivalence

Prevention of unintended interactions

- The systems S and R are known to be safe
- Is the ensemble of all nodes in S and R still safe?
- Bisimulation $\approx_{h_S}$ that hides the updates of S



Hiding bisimulation:

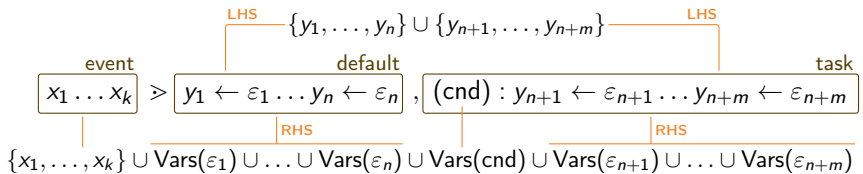
execution labels produced by S

S does not interact with, or is transparent for, R

Safety: verification algorithm

- Compute sinks: resources that rules may update
- Compute sources: resources that may influence rules behavior

Check that the sinks of S does not overlap with the sources of R



Theorem (Soundness for Safety)

If $\text{sinks}(S) \cap \text{sources}(R) = \varnothing$ *then* S *is transparent for* R.

Conclusion and Future Work

AbU: a new programming paradigm for smart (ECA) devices

- Distributed and decentralized
- Mitigate scalability, availability and privacy issues
- Preserve programming simplicity

In this talk

- Bisimulation-based Security and Safety requirements for AbU
- Sound verification mechanisms for the requirements

Future work

- Heuristics for implicit interactions
- Add a declassification mechanism
- Define correctness requirements (e.g., confluence)

Thanks for the attention!

- M Miculan, M Pasqua, *A Calculus for Attribute-based Memory Updates*, Proc. ICTAC 2021 - LNCS 12819;
- M Pasqua, M Miculan, *On the Security and Safety of AbU Systems*, International Conference on Software Engineering and Formal Methods, LNCS 13085, 2021.
- M Pasqua, M Miculan, *Distributed Programming of Smart Systems with Event-Condition-Action Rules*, ICTCS 2022: 201-206
- M Pasqua, M Comuzzo, M Miculan, *The AbU Language: IoT Distributed Programming Made Easy*, IEEE Access 10: 132763-132776 (2022)
- M Pasqua, M Miculan, *AbU: A calculus for distributed event-driven programming with attribute-based interaction*. TCS 958: 113841 (2023)
- `https://github.com/abu-lang`