Finanziato dall'Unione europea NextGenerationEU

Ministero dell'Università e della Ricerca

Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA

SERICS SECURITY AND RIGHTS IN THE CYBERSPACE

Attribute-based Communication over Pub/Sub: Transactional Coordination for Smart Systems

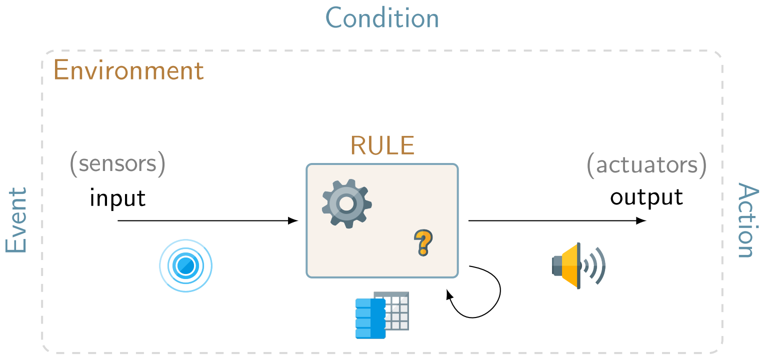
SWOPS WP1 - Core Programming Languages

Venice, 26/06/2025

M. Comini, L. Gemolotto and M. Miculan marino.miculan@uniud.it

Second Software and Platform Security Workshop

Event-driven programming of smart systems

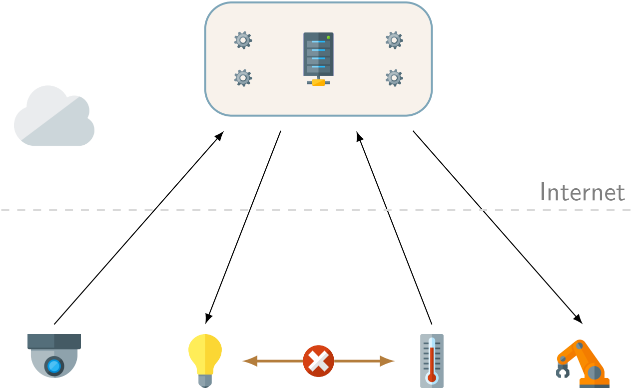


State-based ECA rules: "**on** movement **if** alarm = "active" **then** siren ← on"

variables can be internal, or connected to sensors or to actuators

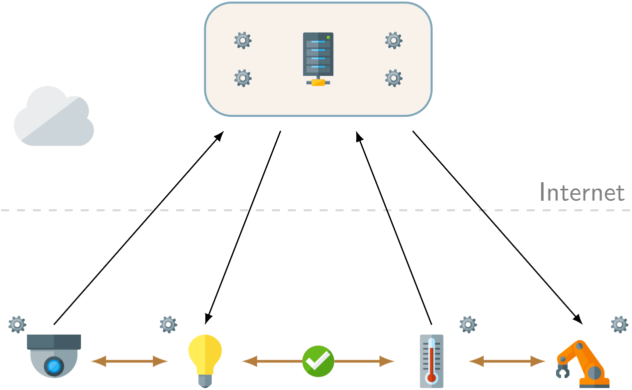
Actual smart (ECA) devices setting

- Centralized
- No intra-nodes communication
- Cloud-dependent
- Big security concerns
- Very popular as Trigger-Action Platforms (TAP):



Next (ECA) IoT architecture: *edge computing*

- Fully distributed
- Communication between nodes
- Cloud-agnostic
- Identity decoupled, for scalability
- *Collective Adaptive Systems*



Programming model for edge CAS?

We need programming abstractions and models for edge computing with:

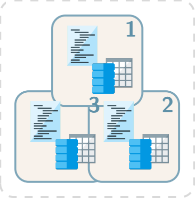
- peer-to-peer, decentralised control
- identity decoupling, for scalability (no point-to-point communication)
- open and flexible (nodes can join and leave dynamically)
- which integrate neatly within the ECA paradigm

Attribute-based Updates (AbU):
ECA rules + Attribute-based Communication

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

AbU syntax

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

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

for all: $@(x < \bar{x}) : \bar{x} \leftarrow x, \bar{y} \leftarrow \bar{y} + 1$

"update the (remote) x with the current (local) x, and increment remote y"

AbU operational semantics

[M. Pasqua, MM, TCS 2023]

LTS semantics, with judgments:

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

A label α can be:

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

AbU operational semantics: rules

$$(\textsc{Exec})\ \frac{\begin{array}{c}\mathsf{upd} \in \Theta \quad \mathsf{upd} = (\mathrm{x}_1, v_1)\dots(\mathrm{x}_k, v_k) \quad \Sigma' = \Sigma[v_1/\mathrm{x}_1 \dots v_k/\mathrm{x}_k] \\ \Sigma' \models \iota \quad X = \{\mathrm{x}_i \mid i \in [1..k] \land \Sigma(\mathrm{x}_i) \neq \Sigma'(\mathrm{x}_i)\} \\ \Theta' = (\Theta \setminus \{\mathsf{upd}\}) \cup \mathsf{LocalUpds}(R, X, \Sigma') \quad T = \mathsf{ExtTasks}(R, X, \Sigma')\end{array}}{R, \iota\langle \Sigma, \Theta\rangle \xrightarrow{\triangleright T} R, \iota\langle \Sigma', \Theta'\rangle}$$

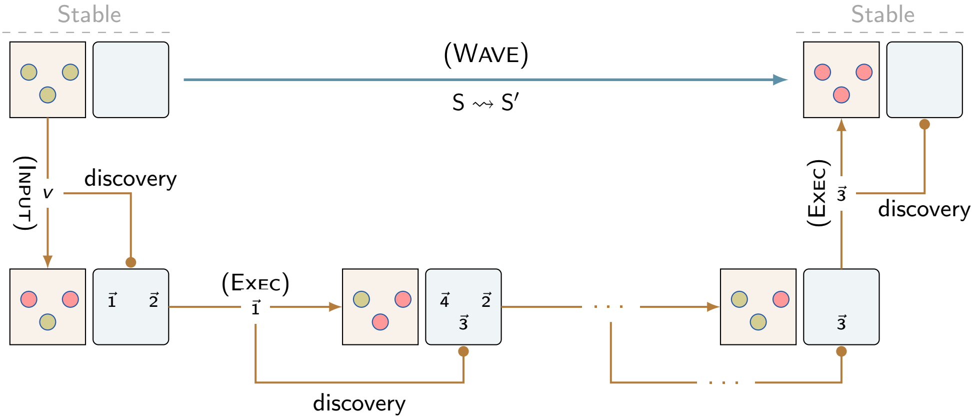
$$(\textsc{Exec-F})\ \frac{\mathsf{upd} \in \Theta \quad \mathsf{upd} = (\mathrm{x}_1, v_1)\dots(\mathrm{x}_k, v_k) \quad \Sigma[v_1/\mathrm{x}_1 \dots v_k/\mathrm{x}_k] \not\models \iota \quad \Theta' = \Theta \setminus \{\mathsf{upd}\}}{R, \iota\langle \Sigma, \Theta\rangle \xrightarrow{\triangleright \epsilon} R, \iota\langle \Sigma, \Theta'\rangle}$$

$$(\textsc{Input})\ \frac{\begin{array}{c}v_1, \dots, v_k \in \mathbb{V} \quad \Sigma' = \Sigma[v_1/\mathrm{x}_1 \dots v_k/\mathrm{x}_k] \quad X = \{\mathrm{x}_1, \dots, \mathrm{x}_k\} \\ \Theta' = \Theta \cup \mathsf{LocalUpds}(R, X, \Sigma') \quad T = \mathsf{ExtTasks}(R, X, \Sigma')\end{array}}{R, \iota\langle \Sigma, \Theta\rangle \xrightarrow{\blacktriangleright T} R, \iota\langle \Sigma', \Theta'\rangle}$$

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

$$(\textsc{StepL})\ \frac{\mathsf{S}_1 \xrightarrow{\alpha} \mathsf{S}_1' \quad \mathsf{S}_2 \xrightarrow{T} \mathsf{S}_2'}{\mathsf{S}_1 \parallel \mathsf{S}_2 \xrightarrow{\alpha} \mathsf{S}_1' \parallel \mathsf{S}_2'}\ \alpha \in \{\triangleright T, \blacktriangleright T\} \qquad (\textsc{StepR})\ \frac{\mathsf{S}_1 \xrightarrow{T} \mathsf{S}_1' \quad \mathsf{S}_2 \xrightarrow{\alpha} \mathsf{S}_2'}{\mathsf{S}_1 \parallel \mathsf{S}_2 \xrightarrow{\alpha} \mathsf{S}_1' \parallel \mathsf{S}_2'}\ \alpha \in \{\triangleright T, \blacktriangleright T\}$$

AbU execution model



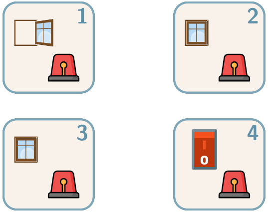
Example: Smart Building Alarm

Four nodes: $S = W\langle\Sigma_1, \varnothing\rangle \parallel W\langle\Sigma_2, \varnothing\rangle \parallel W\langle\Sigma_3, \varnothing\rangle \parallel P\langle\Sigma_4, \varnothing\rangle$

Nodes state:

Triggered rules:

Rule set:



$\mathsf{alarmSwitch} \gtrdot @(!\mathsf{alarmSwitch}) : \overline{\mathsf{siren}} \leftarrow \mathit{false}, \overline{\mathsf{alarmOn}} \leftarrow \mathit{false}$

$\mathsf{doorOpen} \gtrdot @(\mathsf{alarmOn} \wedge \mathsf{doorOpen}) : \overline{\mathsf{siren}} \leftarrow \mathit{true}$

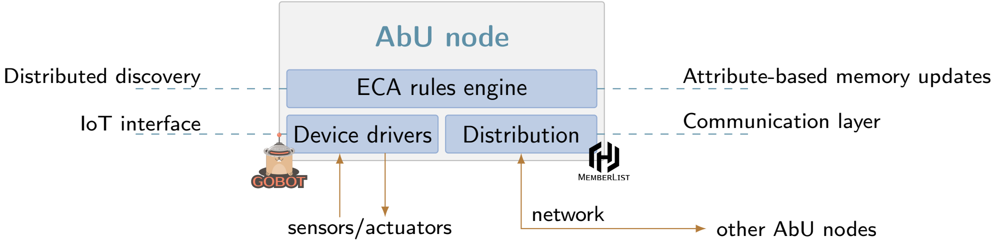
$\mathsf{doorOpen} \gtrdot (\mathsf{alarmOn} \wedge \mathsf{doorOpen}) : \mathsf{siren} \leftarrow \mathit{true}$

$\mathsf{doorOpen} \gtrdot (\mathsf{alarmOn} \wedge \mathsf{doorOpen}) : \mathsf{siren} \leftarrow \mathit{true}$

$\mathsf{alarmSwitch} \gtrdot @(\mathsf{alarmSwitch}) : \overline{\mathsf{alarmOn}} \leftarrow \mathit{true}$

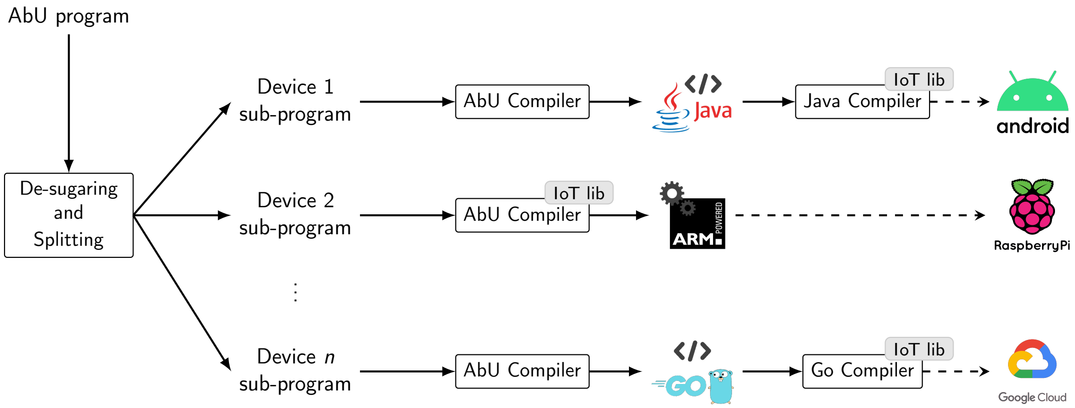
$\mathsf{alarmSwitch} \gtrdot @(!\mathsf{alarmSwitch}) : \overline{\mathsf{siren}} \leftarrow \mathit{false}, \overline{\mathsf{alarmOn}} \leftarrow \mathit{false}$

A (modular) distributed implementation



- ECA rules engine module: AbU semantics
- Device drivers module (GOBOT-based): abstraction of physical resources
- Distribution module (Memberlist-based): abstraction of send/receive and cluster nodes join/leave
- Transactional communication (three-phase commit protocol)

AbU-lang Programs Compilation Cycle



How to port AbU to more IoT systems?

- Often enough, IoT systems do not use RPC/REST or similar technologies
- Nodes might not even be aware of other nodes
- Applications like robotics or smart building often rely on **pub/sub middlewares**, such as ROS, MQTT or

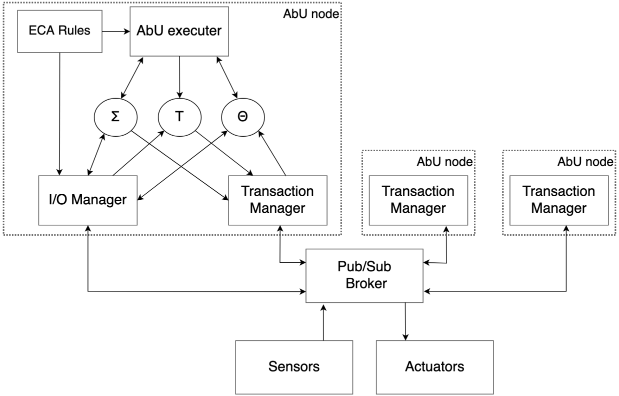
This work [Comini, Gemolotto, M., FORTE 2025]:

A new architecture and implementation of AbU over pub/sub systems.

Node Architecture

Three threads in parallel for each node

- Executer for rule processing
- I/O Manager for handling sensors and actuators
- Transaction manager for global rule handling



Transaction Manager Protocol: Desired Properties

Eventual Transaction Termination: every transaction will eventually be committed.

Absence of Deadlocks: the Executer thread will always be released from its wait on T.

Absence of Race Conditions: at any point, there cannot be two nodes reaching the commit phase at the same time, on different transactions.

Assumptions on the system

Reliability: the middleware provides mechanism to ensure that messages are received.

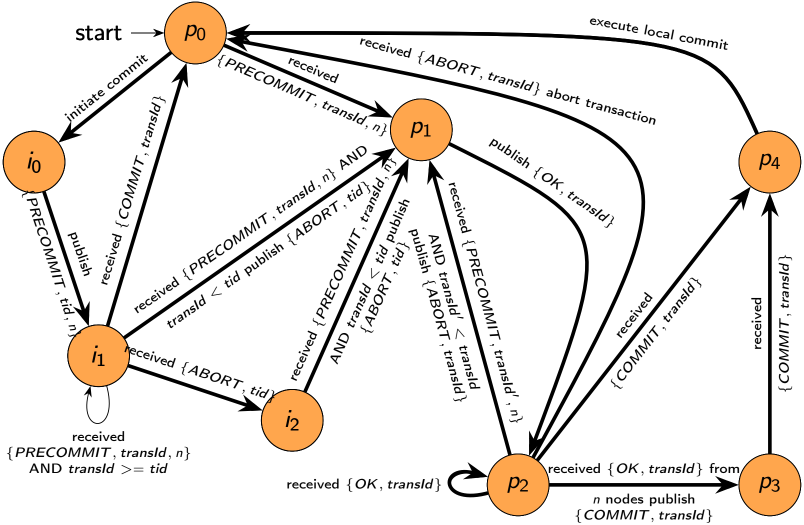
Scheduling Fairness: each node implements a fair scheduler such that no thread can be infinitely often enabled and never executed.

FIFO ordering: given two messages m_1 and m_2 sent in this order by the same node, each node will receive them in the same order.

Uniqueness of message identifiers: transaction identifiers are generated locally on each node by combining a local counter t with the node's unique identifier id, denoted as Id(t,id).

Many pub/sub platforms, such as the DDS implementations in use by ROS, are able to guarantee these.

Protocol Automata



Proving Properties: Reachability of the commit state

Proposition (Reachability of the commit state)

For a given transaction t among n participants, if no faults occur, at least one node will eventually be able to count n "OK" messages.

Theorem (Eventual commit)

Regardless of the conflicts, any transaction will eventually commit.

Proving Properties: Deadlocks and Race Conditions

Corollary

The Executer thread will never indefinitely wait for T to become empty (i.e., loop indefinitely in lines 2-3 of algorithm 1), thus deadlocks are avoided.

Corollary

At any time, all automata which are in state p_4 (local commit) have the same transId.

Pseudocode

```
Function executer (T, Θ, Σ):
  while true do
    while T ≠ NIL do
      ; // wait for potentially
        initiated transaction to
        end
    U ← selectUpdate(Θ);
     // select next update from
     Θ; blocks if Θ = ∅
    lock(T); lock(Θ);
    Θ ← Θ \ {U}; // remove it
     from pool
    (X, Σ′) ← applyUpdate(U, Σ);
    if Σ′ ⊨ ι then
      Σ ← Σ′;
      Θ ←
        Θ ∪ localUpdates(R, X, Σ);
      T ←
        externalUpdates(R, X, Σ);
    unlock(Θ); unlock(T);
```

Algorithm 1: Pseudocode for the AbU executer.

```
Function inputManager(T, Θ, Σ):
  while true do
    while T ≠ NIL do
      ; // wait for
        potentially initiated
        transaction to end
    U ← receiveSensors();
    lock(T); lock(Θ);
    (X, Σ) ← applyUpdate(U, Σ);
    Θ ←
     Θ ∪ localUpdates(R, X, Σ);
    T ←
     externalUpdates(R, X, Σ);
    unlock(Θ); unlock(T);
```

Algorithm 2: Pseudocode for the AbU input manager.

Pseudocode (cont.)

Algorithm 1 Pseudocode for the AbU Transaction Manager

```
function TransactionManager(T, Θ, Σ, nodeId)
    isInitiator ← false
    tid ← NIL; U ← NIL; lTid ← 0
    while true do
        if T ≠ NIL and tid = NIL then
            tid ← GetTransId(nodeId, lTid)
            nParticipants ← GetParticipants
            Publish(PRECOMMIT, tid, nParticipants, T)
            isInitiator ← true
        end if
        msg ← ReceiveFromTopic
        if msg = (PRECOMMIT, transId, n, T′) then
            if tid = NIL then
                tid ← transId
                counter ← n
                U ← selectValid(T′, Σ)
                Publish(OK, tid)
            else if transId < tid then
                Publish(ABORT, tid)
                U ← selectValid(T′, Σ)
                tid ← transId
                isInitiator ← false
                Publish(OK, tid)
            end if
        else if msg = (OK, transId) then
            if transId = tid and not isInitiator then
                counter ← counter − 1
                if counter = 0 then
                    Publish(COMMIT, tid)
                end if
            end if
        else if msg = (COMMIT, transId) then
            if transId = tid then
                if isInitiator then
                    T ← NIL
                    isInitiator ← false
                    lTid ← lTid + 1
                else
                    lock(Θ)
                    Θ ← Θ ∪ U
                    unlock(Θ)
                end if
                tid ← NIL
            end if
        else if msg = (ABORT, transId) then
            if transId = tid and not isInitiator then
                tid ← NIL
            end if
        end if
    end while
end function
```

Conclusion

New contributions

- A fully decentralized 2PC protocol based on broadcast primitives
- New implementation of AbU on ROS2
- ECA rule-based language for IoT and robotics applications

Future work

- Finalize implementation and testing
- Relaxing the assumptions (at the expense of the properties)
- Lowering latency / Adding priorities to remote tasks
- Application of the transactional protocol to other contexts

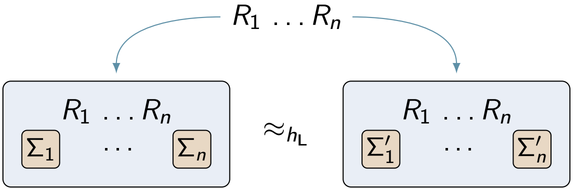
Thanks for the attention

- 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)
- M Comini, L Gemolotto, M Miculan, *Attribute-Based Communication over Pub/Sub: Transactional Coordination for Smart Systems*, Proc. FORTE 2025 - LNCS 15732
- `https://github.com/abu-lang`

Behavioral equivalences

Security: 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 L-level updates

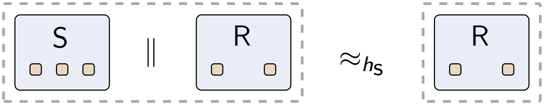


for all L-equivalent states $\Sigma_1 \equiv_L \Sigma'_1 \ldots \Sigma_n \equiv_L \Sigma'_n$

Behavioral equivalences (Cont'd)

Safety: 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 $\approx_{hs}$ that hides the updates of S

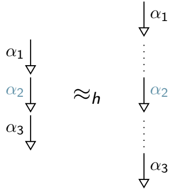


S does not interact with, or is transparent for, R

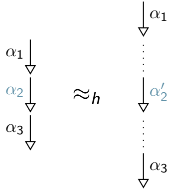
Hiding bisimulation

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

if $h(\alpha_2) \neq \diamond$



if $h(\alpha_2) = \diamond$



with $h(\alpha'_2) = \diamond$

Security h_L hides:

- discovery labels
- execution labels with H resources

Safety h_S hides:

- discovery labels
- execution labels produced by S