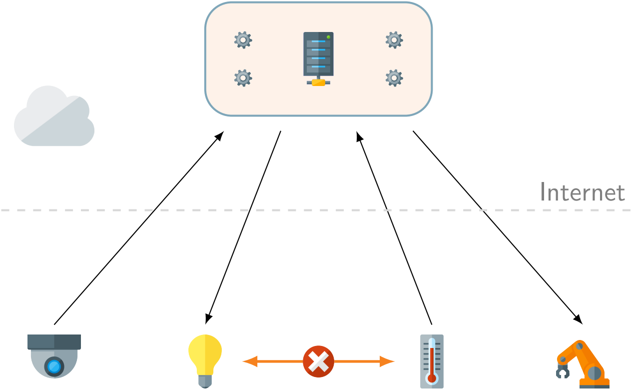
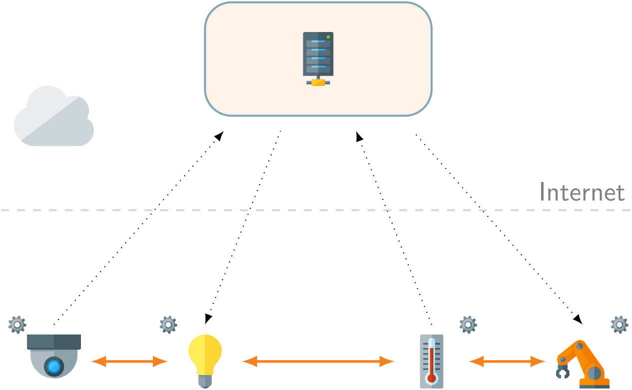
Actual smart (ECA) devices setting

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



Ideal smart (ECA) devices setting

- Fully distributed
- Communication between nodes
- Cloud-agnostic



WP3: From Models to Code

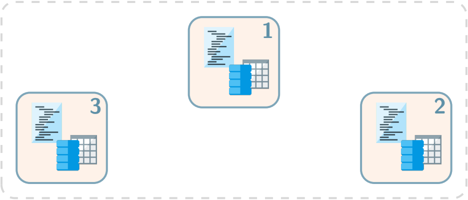
Excerpt from Modello A:

*[...] We will develop a **new programming model, integrating ECA, Attribute-based Programming**, and Aggregate Programming. This will require the definition of **a suitable formal model (e.g. a process algebra) for "ECA Attribute-based Programming", merging concepts from AbC** and the field calculus [...] On the basis of this programming model, we will provide techniques and tools for the static and dynamic verification of attribute-based programs [...] At the implementation level, we will embed our model within languages such as Java, Haskell, or Erlang.*

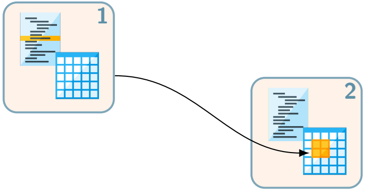
Attribute-based Memory Updates

Nodes behavior: ECA rules

Nodes state: local memory



Communication: remote updates



Attribute-based interaction: send (the value of) e to all nodes satisfying Π

AbC-AbU correspondence

	AbC	AbU
Communication	message-passing	memory updates
Output	$e \;@\; \Pi$	$@\; \Pi$
Input	$x \mid \Pi$	nodes invariant

In AbU there are no explicit input primitives, to filter incoming updates.
But we can specify *admissible* states by means of state invariants.

The AbU calculus

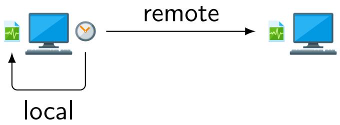
- An **AbU system** S is a **AbU node** $R, \iota\langle\Sigma, \Theta\rangle$ or the parallel of systems $S_1 \parallel S_2$
- Each node is equipped with a list R of **AbU rules**

Syntax

$$
\begin{array}{rcl}
\text{rule} & ::= & \text{evt} \gg \text{act}, \text{task} \\
\text{evt} & ::= & x \mid \text{evt evt} \\
\text{act} & ::= & \epsilon \mid x \leftarrow \varepsilon \text{ act} \mid \overline{x} \leftarrow \varepsilon \text{ act} \\
\text{task} & ::= & \text{cnd} : \text{act}
\end{array}
\qquad
\begin{array}{rcl}
\text{cnd} & ::= & \varphi \mid @\varphi \\
\varphi, \iota & ::= & \mathbf{F} \mid \mathbf{T} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varepsilon \bowtie \varepsilon \\
\varepsilon & ::= & v \mid x \mid \overline{x} \mid \varepsilon \otimes \varepsilon \\
 & & x \in \mathbb{X} \quad v \in \mathbb{V}
\end{array}
$$

assignments

- $x \leftarrow \varepsilon$ (local)
- $\overline{x} \leftarrow \varepsilon$ (remote)



The AbU calculus

- An **AbU system** S is a **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**

Syntax

$$\begin{aligned}
\text{rule} &::= \text{evt} \gg \text{act}, \text{task} & \text{cnd} &::= \varphi \mid @\varphi \\
\text{evt} &::= x \mid \text{evt evt} & \varphi, \iota &::= \mathtt{F} \mid \mathtt{T} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varepsilon \bowtie \varepsilon \\
\text{act} &::= \epsilon \mid x \leftarrow \varepsilon \text{ act} \mid \overline{x} \leftarrow \varepsilon \text{ act} & \varepsilon &::= v \mid x \mid \overline{x} \mid \varepsilon \otimes \varepsilon \\
\text{task} &::= \text{cnd} : \text{act} & & x \in \mathbb{X} \quad v \in \mathbb{V}
\end{aligned}$$

“send to all nodes with (remote) x greater than the sender (local) x”

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

“assign the receiver (remote) x with the sender (local) x”

The AbU calculus (Cont'd)

AbU node $R, \iota\langle \Sigma, \Theta \rangle$

Execution **state**: map from resources to values

Execution **pool**: set of updates to execute

Invariant

atomically update $(x_1, v_1) \ldots (x_n, v_n)$

resources x_i will take values v_i after the update execution

Invariants are predicates over states which must not be violated, i.e., updates which would violated ι will be not executed.

Given a set X of resources that have been changed

- Active rules: rules $\mathsf{evt} \gg \mathsf{act}, \mathsf{task}$ in R such that $\mathsf{evt} \cap X \neq \varnothing$
- External tasks: tasks $@\varphi : \mathsf{act}$ of active rules
  - Pre-evaluation: $\{\!|@(x < \overline{x}) : \overline{y} \leftarrow x + \overline{y}|\!\}[x \mapsto 1 \; y \mapsto 0] = (1 \leq x) : y \leftarrow 1 + y$

The AbU calculus (Cont'd)

Semantics

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

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

- Similar (INPUT) rule for sensors change, with labels of the form: $\mathsf{upd} \blacktriangleright T$
- The discovery phase is used for synchronization:

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

The AbU calculus (Cont'd)

Semantics

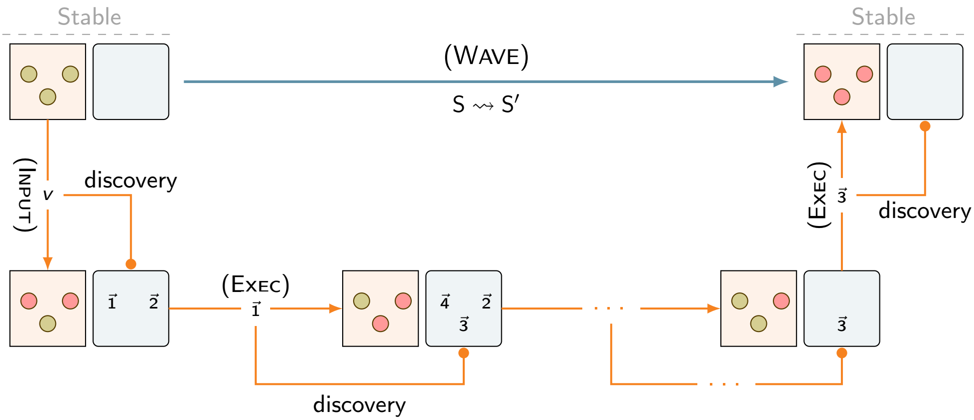
$$\text{(EXEC-FAIL)}\ \frac{\mathsf{upd} \in \Theta \quad \mathsf{upd} = (x_1, v_1)\dots(x_k, v_k) \quad \Sigma' = \Sigma[v_1/x_1 \dots v_k/x_k] \quad \Sigma' \not\models \iota \quad \Theta' = \Theta \setminus \{\mathsf{upd}\}}{R, \iota\langle\Sigma, \Theta\rangle \xrightarrow{\mathsf{upd} \triangleright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

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

- Similar (INPUT) rule for sensors change, with labels of the form: $\mathsf{upd} \blacktriangleright T$
- The discovery phase is used for synchronization:

$$\text{(STEP)}\ \frac{S_1 \xrightarrow{\alpha} S_1' \quad S_2 \xrightarrow{T} S_2'}{S_1 \parallel S_2 \xrightarrow{\alpha} S_1' \parallel S_2'}\ \alpha \in \{\mathsf{upd} \triangleright T, \mathsf{upd} \blacktriangleright T\}$$

AbU execution model



Encoding AbC into AbU

node $R_1\langle\Sigma_1, \varnothing\rangle$ (resources given by the attributes in Γ_1)

$\ddots$

node $R_n\langle\Sigma_n, \varnothing\rangle$ (resources given by the attributes in Γ_n)

$$\mathcal{T}(\Gamma_1 : P_1 \parallel \cdots \parallel \Gamma_n : P_n)$$

rules $R_1 = \mathcal{T}(P_1) \dots R_n = \mathcal{T}(P_n)$ inductively generated from each P_i

- Reconstruct the sequential execution flow of AbC components, by means of token-passing mechanism to activate rules (program counter)
- Encode attribute-based communication with attribute-based memory updates

Theorem (AbC to AbU correctness)

Consider an AbC component C and its corresponding AbU encoding $\mathsf{S} = \mathcal{T}(C)$. Then, for all C' such that $C \rightarrowtriangle^* C'$ there exists S' such that $\mathsf{S} \rightarrowtriangle^* \mathsf{S}'$ and $\mathsf{S}' \succeq C'$.

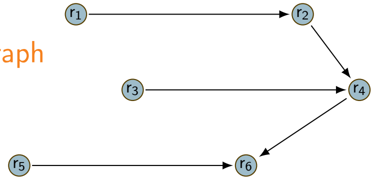
WP3: From Models to Code

Excerpt from Modello A:
[...] We will develop a ***new programming model, integrating ECA, Attribute-based Programming****, and Aggregate Programming. This will require the definition of a suitable formal model (e.g. a process algebra) for "ECA Attribute-based Programming", merging concepts from AbC and the field calculus [...] On the basis of this programming model,* ***we will provide techniques and tools for the static and dynamic verification of attribute-based programs*** *[...] At the implementation level, we will embed our model within languages* such as Java, Haskell, or Erlang.

Stabilization

The wave semantics may exhibit internal divergence, namely $S \xrightarrow{\alpha_0} S^0 \xrightarrow{\alpha_1} \ldots$

ECA
dependency graph



rule A $r_4\ (\square) : r_6 \leftarrow \square$

rule B $r_3\, r_2\ (\square) : r_4 \leftarrow \square$

rule C $r_5\ (\square) : r_6 \leftarrow \square$

rule D $r_1\ (\square) : r_2 \leftarrow \square$

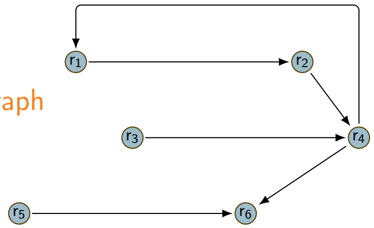
Theorem (AbU stabilization)

If the ECA dependency graph of an AbU system S is acyclic, then S is stabilizing.

Stabilization

The wave semantics may exhibit internal divergence, namely $S \xrightarrow{\alpha_0} S^0 \xrightarrow{\alpha_1} \ldots$

ECA
dependency graph



rule A $r_4\ (\square) : r_6 \leftarrow \square\ r_1 \leftarrow \square$

rule B $r_3\ r_2\ (\square) : r_4 \leftarrow \square$

rule C $r_5\ (\square) : r_6 \leftarrow \square$

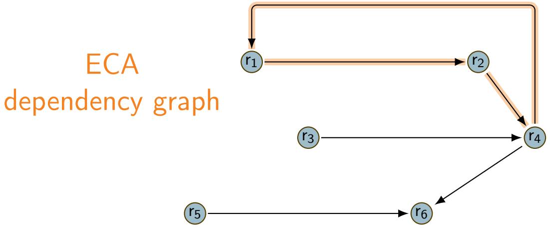
rule D $r_1\ (\square) : r_2 \leftarrow \square$

Theorem (AbU stabilization)

If the ECA dependency graph of an AbU system S is acyclic, then S is stabilizing.

Stabilization

The wave semantics may exhibit internal divergence, namely $S \xrightarrow{\alpha_0} S^0 \xrightarrow{\alpha_1} \dots$



rule A $r_4\ (\square) : r_6 \leftarrow \square\ r_1 \leftarrow \square$

rule B $r_3\ r_2\ (\square) : r_4 \leftarrow \square$

rule C $r_5\ (\square) : r_6 \leftarrow \square$

rule D $r_1\ (\square) : r_2 \leftarrow \square$

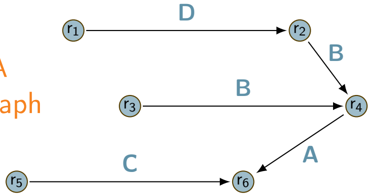
Theorem (AbU stabilization)

If the ECA dependency graph of an AbU system S is acyclic, then S is stabilizing.

Confluence

The AbU scheduler should not influence the AbU semantics: for all S_1 and S_2 such that $S \rightharpoondown^* S_1$ and $S \rightharpoondown^* S_2$, there exists S' such that $S_1 \rightharpoondown^* S'$ and $S_2 \rightharpoondown^* S'$

labeled ECA dependency graph



rule A $r_4\ (\square) : r_6 \leftarrow \square$

rule B $r_3\ r_2\ (\square) : r_4 \leftarrow \square$

rule C $r_5\ (\square) : r_6 \leftarrow \square$

rule D $r_1\ (\square) : r_2 \leftarrow \square$

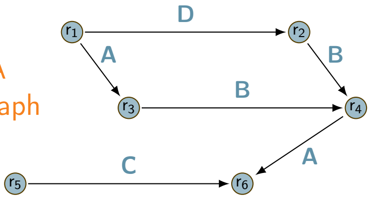
Theorem (AbU confluence)

If for each pair (x, y) of nodes in the labeled ECA dependency graph of an AbU system S we have that $|\mathrm{walks}(x, y)| \leq 1$, then S is confluent.

Confluence

The AbU scheduler should not influence the AbU semantics: for all S_1 and S_2 such that $S \rightarrowtriangle^* S_1$ and $S \rightarrowtriangle^* S_2$, there exists S' such that $S_1 \rightarrowtriangle^* S'$ and $S_2 \rightarrowtriangle^* S'$

labeled ECA dependency graph



rule A $r_4\ (\square) : r_6 \leftarrow \square\ r_3 \leftarrow \square$

rule B $r_3\ r_2\ (\square) : r_4 \leftarrow \square$

rule C $r_5\ (\square) : r_6 \leftarrow \square$

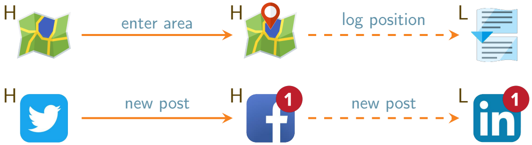
rule D $r_1\ (\square) : r_2 \leftarrow \square$

Theorem (AbU confluence)

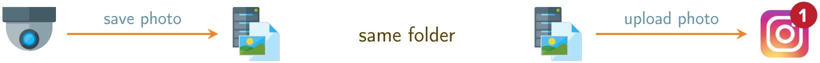
If for each pair (x, y) of nodes in the labeled ECA dependency graph of an AbU system S we have that $|\mathrm{walks}(x, y)| \leq 1$, then S is confluent.

Security and Safety

Security

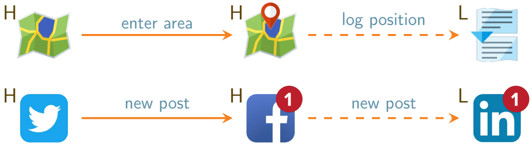


Safety



Security and Safety

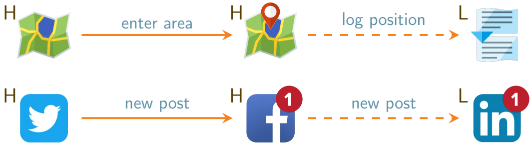
Security



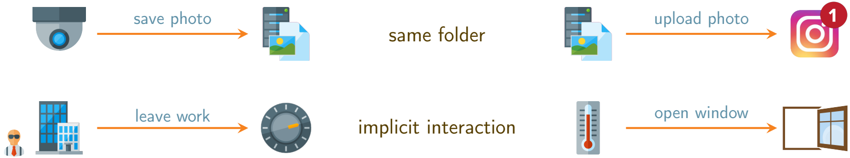
Safety

Security and Safety

Security



Safety

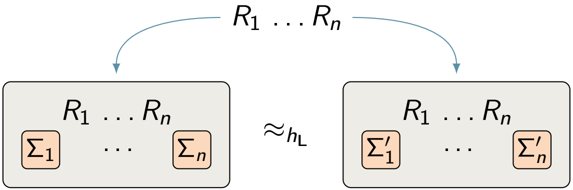


UNIVERSITÀ DEGLI STUDI DI UDINE

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

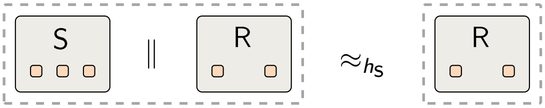


for all L-equivalent states $\Sigma_1 \equiv_L \Sigma'_1 \ldots \Sigma_n \equiv_L \Sigma'_n$

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 $h(\alpha) = \diamond$

$$\text{if } h(\alpha_2) \neq \diamond \qquad \xrightarrow{\alpha_1}\;\xrightarrow{\alpha_2}\;\xrightarrow{\alpha_3} \quad \approx_h \quad \xrightarrow{\alpha_1}\;\cdots\;\xrightarrow{\alpha_2}\;\cdots\;\xrightarrow{\alpha_3}$$

$$\text{if } h(\alpha_2) = \diamond \qquad \xrightarrow{\alpha_1}\;\xrightarrow{\alpha_2}\;\xrightarrow{\alpha_3} \quad \approx_h \quad \xrightarrow{\alpha_1}\;\cdots\;\xrightarrow{\alpha'_2}\;\cdots\;\xrightarrow{\alpha_3} \qquad \text{with } h(\alpha'_2) = \diamond$$

Security h_L hides:

- discovery labels
- execution labels with H resources

Safety h_S hides:

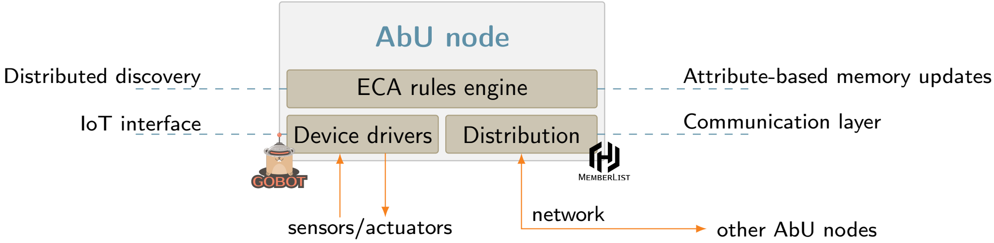
- discovery labels
- execution labels produced by S

WP3: From Models to Code

Excerpt from Modello A:

*[...] We will develop a **new programming model, integrating ECA, Attribute-based Programming**, and Aggregate Programming. This will require the definition of a suitable formal model (e.g. a process algebra) for "ECA Attribute-based Programming", merging concepts from AbC and the field calculus [...] On the basis of this programming model, we will provide techniques and tools for the static and dynamic verification of attribute-based programs [...] **At the implementation level, we will embed our model within languages** such as Java, Haskell, or Erlang.*

A (modular) distributed implementation

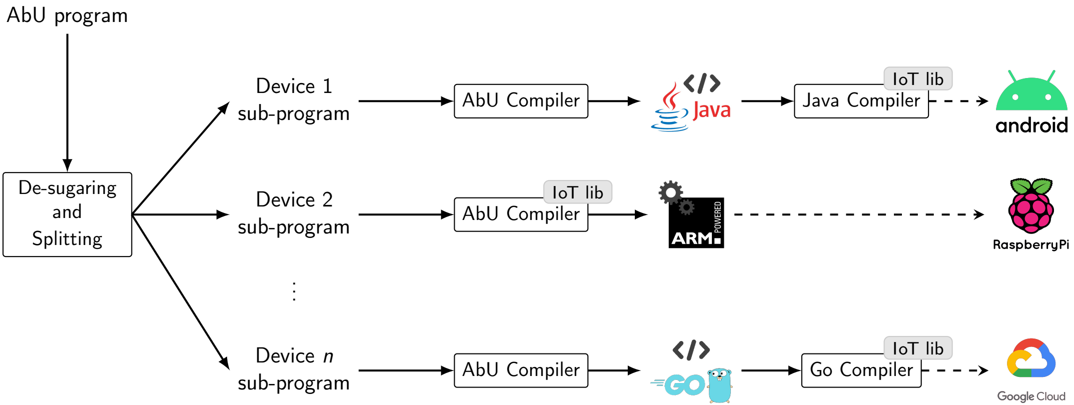


- ECA rules engine module: AbU semantics
- Device drivers module: abstraction of physical resources
- Distribution module: abstraction of send/receive and cluster nodes join/leave

AbU-lang: 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
```

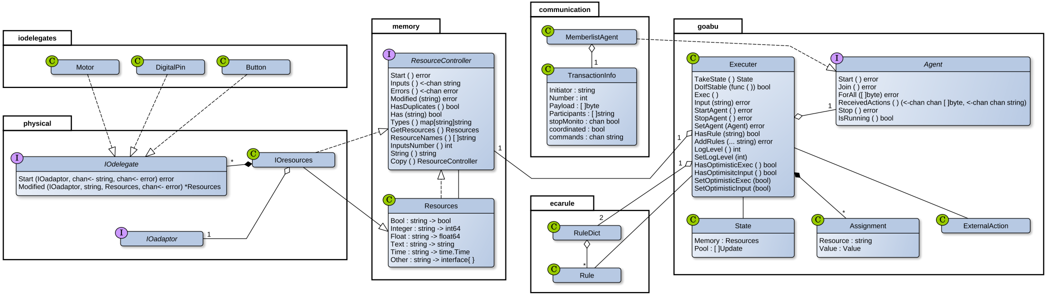
AbU-lang Programs Compilation Cycle



GoAbU Compiler

Prototype compiler from AbU-lang to Golang (available at `https://github.com/abu-lang`)

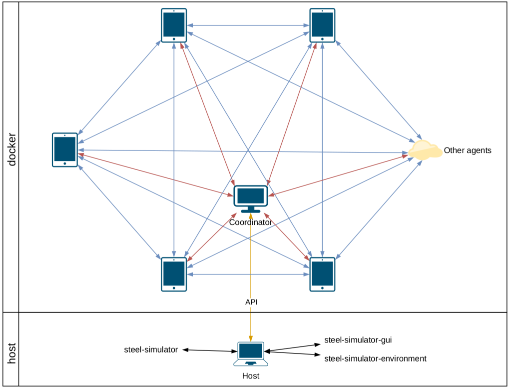
- Hashicorp Memberlist for cluster membership (gossip-based protocol)
- Transactional communication (three-phase commit protocol)
- GOBOT library for physical IoT input/output



AbU Simulator

Docker-based simulator for AbU nodes

- Quick multiple nodes definition via node prototypes
- Automatic deployment and interaction of nodes (one Docker image per node)
- Web-based inputs management and resources visualization



Conclusion

AbU

- Simple to use (ECA paradigm)
- Particularly suitable for the IoT domain
- Strongly distributed
- Local nodes coordination (Attribute-based memory updates)
- Supports several verification techniques

Future work

- Efficient distributed implementation of AbU
  - Distributed RETE algorithm (RPCs or message-passing)
  - More implementations (e.g., for Android, in Java)
- Porting (ECA) verification techniques from ECA-languages (e.g., [IE17]) to AbU
- Distributed runtime monitor for AbU
  - Runtime enforcing of correctness properties (e.g., [RV17])

[IE17] *Vannucchi C. et al.* (2017) vIRONy: A tool for analysis and verification of ECA rules in intelligent environments
[RV17] *Francalanza A. et al.* (2017) A Foundation for Runtime Monitoring

Excerpt from Modello A:

[...] We will develop a new programming model, integrating ECA, Attribute-based Programming, and Aggregate Programming. This will require the definition of a suitable formal model (e.g. a process algebra) for "ECA Attribute-based Programming", merging concepts from AbC and the field calculus [...] On the basis of this programming model, we will provide techniques and tools for the static and dynamic verification of attribute-based programs [...] At the implementation level, we will embed our model within languages such as Java, Haskell, or Erlang.

Almost there!

Thanks for the attention!