Formal Analysis of Policies in Wireless Sensor Network Applications

Marco Patrignani Nelson Matthys José Proença Danny Hughes Dave Clarke

K.U.Leuven

May 31, 2012

Outline

Outline

Wireless Sensor Network Applications with CaPI

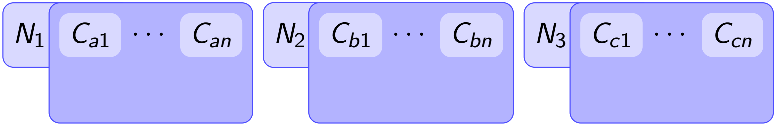
- Application: high level of abstraction (no routing, no low level communication).

Wireless Sensor Network Applications with CaPI

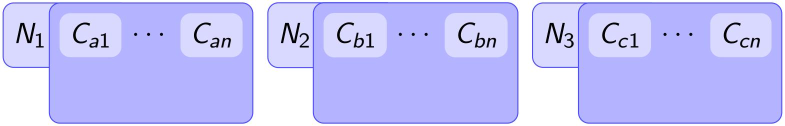
- Application: high level of abstraction (no routing, no low level communication).
- Key elements of CaPI middleware:
  - Components
  - Wires
  - Policies

Graphical Representation of Components

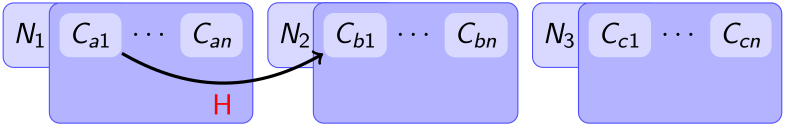
Graphical Representation of Components



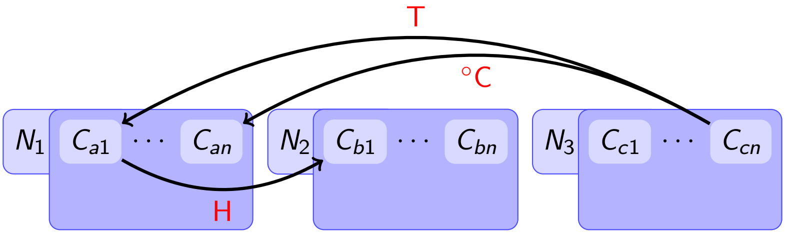
Graphical Representation of Wires



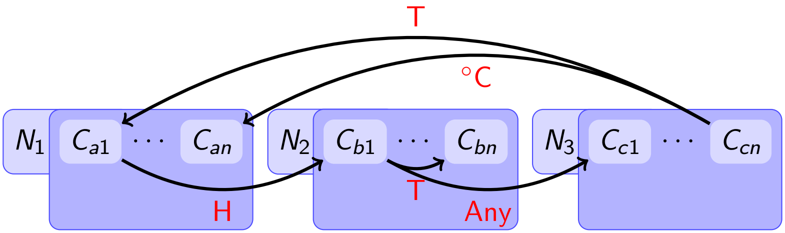
Graphical Representation of Wires



Graphical Representation of Wires



Graphical Representation of Wires



Policies

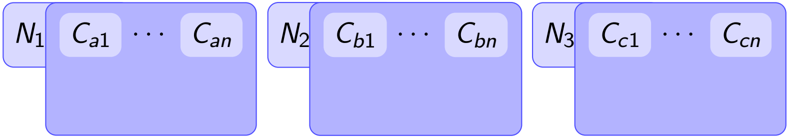
- Take care of non functional requirements via domain-specific actions.

Policies

- Take care of non functional requirements via domain-specific actions.
- Triggered by messages when messages leave and enter a node.

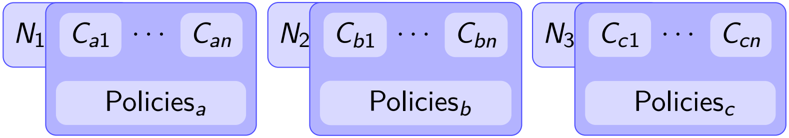
Policies

- Take care of non functional requirements via domain-specific actions.
- Triggered by messages when messages leave and enter a node.
- Executed at 4 entry points: from components, to components, from network, to network.



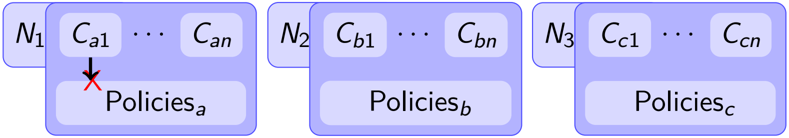
Policies

- Take care of non functional requirements via domain-specific actions.
- Triggered by messages when messages leave and enter a node.
- Executed at 4 entry points: from components, to components, from network, to network.



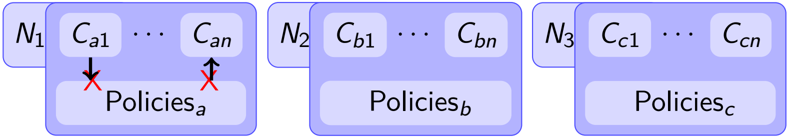
Policies

- Take care of non functional requirements via domain-specific actions.
- Triggered by messages when messages leave and enter a node.
- Executed at 4 entry points: from components, to components, from network, to network.



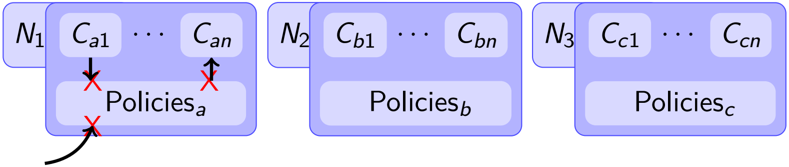
Policies

- Take care of non functional requirements via domain-specific actions.
- Triggered by messages when messages leave and enter a node.
- Executed at 4 entry points: from components, to components, from network, to network.



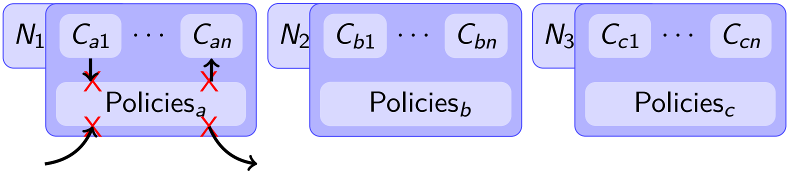
Policies

- Take care of non functional requirements via domain-specific actions.
- Triggered by messages when messages leave and enter a node.
- Executed at 4 entry points: from components, to components, from network, to network.



Policies

- Take care of non functional requirements via domain-specific actions.
- Triggered by messages when messages leave and enter a node.
- Executed at 4 entry points: from components, to components, from network, to network.



Problems Related to Policies

Policies generate problems like:

Problems Related to Policies

Policies generate problems like:

- security holes;

Problems Related to Policies

Policies generate problems like:

- security holes;
- wastage of resources;

Problems Related to Policies

Policies generate problems like:

- security holes;
- wastage of resources;
- other (logging encrypted data).

Problems Related to Policies

Policies generate problems like:

- security holes;
- wastage of resources;
- other (logging encrypted data).

because:

- they are deployed over time;

Problems Related to Policies

Policies generate problems like:

- security holes;
- wastage of resources;
- other (logging encrypted data).

because:

- they are deployed over time;
- they change over time;

Problems Related to Policies

Policies generate problems like:

- security holes;
- wastage of resources;
- other (logging encrypted data).

because:

- they are deployed over time;
- they change over time;
- their semantics can conflict.

Outline

Policies Example

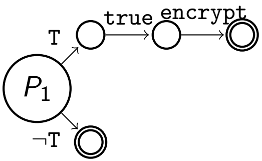
P_1@N_1

`on event (T) as e`
`if (true) then`
`encrypt e;`

Policies Example

$P_1 @ N_1$

```
on event (T) as e
  if (true) then
    encrypt e;
```



Policies Composition Example

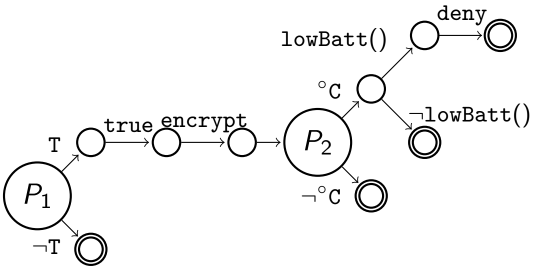
P_1@N_1	P_2@N_1

`on event (T) as e`
`  if (true) then`
`    encrypt e;`
	`on event (°C) as e`
	`  if (lowBatt()) then`
	`    deny e;`

Policies Composition Example

P_1@N_1	P_2@N_1

`on event (T) as e`
`if (true) then`
`encrypt e;`
	`on event (°C) as e`
	`if (lowBatt()) then`
	`deny e;`



Policies Network Composition Example

P_1@N_1	P_2@N_2

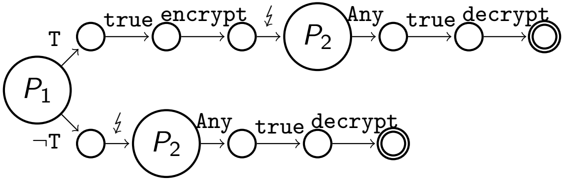
`on event (T) as e`
`if (true) then`
`encrypt e;`
	`on event (Any) as e`
	`if (true) then`
	`decrypt e;`

Policies Network Composition Example

P_1@N_1	P_2@N_2

`on event (T) as e`
`if (true) then`
`encrypt e;`
	`on event (Any) as e`
	`if (true) then`
	`decrypt e;`

Wires on Any messages.



Policy Programs

$$
\begin{array}{rcl@{\qquad}r}
\textit{Policy Program } P & ::= & \texttt{on}(t)\{P\} & \textit{Type } t \in \mathcal{T} \\
 & | & \texttt{if}(g)\{P\} & \textit{If-Guard } g \in \mathcal{L} \\
 & | & a & \textit{Action } a \in \mathcal{A} \\
 & | & P; P' &
\end{array}
$$

$\mathcal{A} ::= \{$`allow`, `deny`, `encrypt`, `decrypt`, `sign`, `verify`, `persist`, `delete`$\}$.

Semantics Function

The idea is to capture what was drawn before: all paths of a tree of execution.

$$\begin{aligned}
[\![\mathtt{on}(t)\{P\}]\!]_C =& \{\tau \mid \tau \in [\![P]\!]_{C\cap\{t\}}, C \frown t\} \cup \{(\epsilon; C \cap \{\neg t\}) \mid C \frown \neg t\} \\
[\![\mathtt{if}(g)\{P\}]\!]_C =& \{\tau \mid \tau \in [\![P]\!]_{C\cup\{g\}}, C \frown g\} \cup \{(\epsilon; C \cup \{\neg g\}) \mid C \frown \neg g\} \\
[\![a]\!]_C =& \{(a; C)\} \\
[\![P; P']\!]_C =& \{(\overline{a}\,\overline{a}'; C') \mid (\overline{a}; C'') \in [\![P]\!]_C, (\overline{a}'; C') \in [\![P']\!]_{C''}\}
\end{aligned}$$

Outline

Analysis

Specification of undesired/desired properties via modal logic:

```
[true*.encrypt.true*.↯.(!decrypt)*]<true*.decrypt> true;
                                                (dec-after-enc)
[true*.encrypt.true*.encrypt] false;            (enc-most-once)
[true*.encrypt]<true*.↯> true;                  (snd-after-enc)
```

Outline

Tool

We developed a tool in Scala that:

Tool

We developed a tool in Scala that:

- extracts the formalization of policies;

Tool

We developed a tool in Scala that:

- extracts the formalization of policies;
- calculates the semantics of policy programs;

Tool

We developed a tool in Scala that:

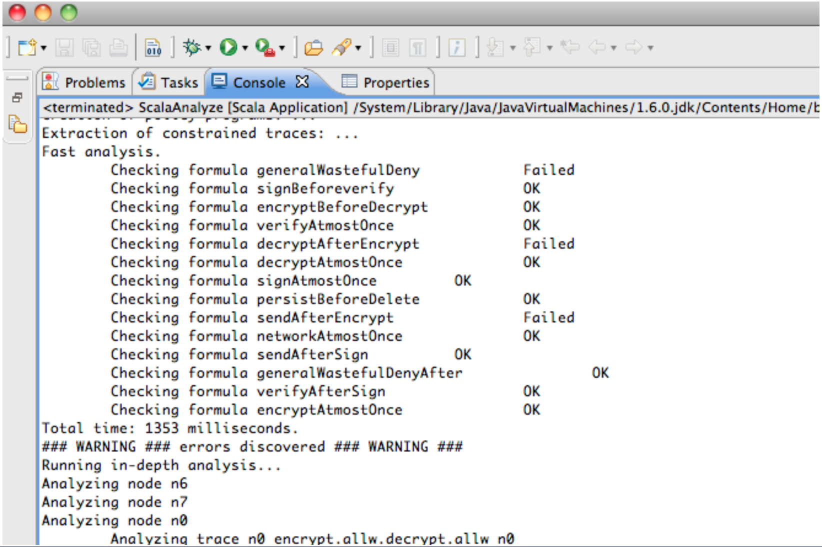
- extracts the formalization of policies;
- calculates the semantics of policy programs;
- analyzes the constrained trace semantics against some user-defined properties via `mCRL2`;

Tool

We developed a tool in Scala that:

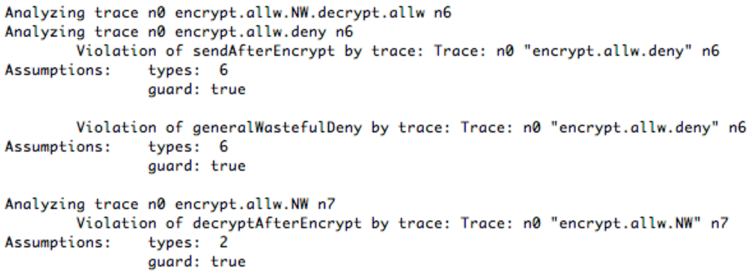
- extracts the formalization of policies;
- calculates the semantics of policy programs;
- analyzes the constrained trace semantics against some user-defined properties via `mCRL2`;
- scales linearly in terms of connections between nodes and number of policies (with similar logical conditions).

Tool- feedback



```
<terminated> ScalaAnalyze [Scala Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/b
Extraction of constrained traces: ...
Fast analysis.
        Checking formula generalWastefulDeny            Failed
        Checking formula signBeforeverify               OK
        Checking formula encryptBeforeDecrypt           OK
        Checking formula verifyAtmostOnce               OK
        Checking formula decryptAfterEncrypt            Failed
        Checking formula decryptAtmostOnce              OK
        Checking formula signAtmostOnce         OK
        Checking formula persistBeforeDelete            OK
        Checking formula sendAfterEncrypt               Failed
        Checking formula networkAtmostOnce              OK
        Checking formula sendAfterSign          OK
        Checking formula generalWastefulDenyAfter               OK
        Checking formula verifyAfterSign                OK
        Checking formula encryptAtmostOnce              OK
Total time: 1353 milliseconds.
### WARNING ### errors discovered ### WARNING ###
Running in-depth analysis...
Analyzing node n6
Analyzing node n7
Analyzing node n0
        Analyzing trace n0 encrypt.allw.decrypt.allw n0
```

Tool- feedback



```
Analyzing trace n0 encrypt.allw.NW.decrypt.allw n6
Analyzing trace n0 encrypt.allw.deny n6
        Violation of sendAfterEncrypt by trace: Trace: n0 "encrypt.allw.deny" n6
Assumptions:    types:  6
                guard: true

        Violation of generalWastefulDeny by trace: Trace: n0 "encrypt.allw.deny" n6
Assumptions:    types:  6
                guard: true

Analyzing trace n0 encrypt.allw.NW n7
        Violation of decryptAfterEncrypt by trace: Trace: n0 "encrypt.allw.NW" n7
Assumptions:    types:  2
                guard: true
```

Outline

Recap & Future Work

Recap

- Formalized policies and their semantics in CaPI-driven WSN applications;
- Created a prototype for the analysis of policies in real-world deployment.

Recap & Future Work

Recap

- Formalized policies and their semantics in CaPI-driven WSN applications;
- Created a prototype for the analysis of policies in real-world deployment.

Future Work

- Model dynamic actions of Policies;

Recap & Future Work

Recap

- Formalized policies and their semantics in CaPI-driven WSN applications;
- Created a prototype for the analysis of policies in real-world deployment.

Future Work

- Model dynamic actions of Policies;
- Provide a concurrency model for the modeled application.