Towards the Verification of Industry 4.0 Applications

Vivek Nigam & Carolyn Talcott

fortiss GmbH, Germany & SRI International, USA

Industry 4.0

New manufacturing paradigm **where devices (IoT devices, sensors) are highly interconnected, forming a cyber-physical system.**

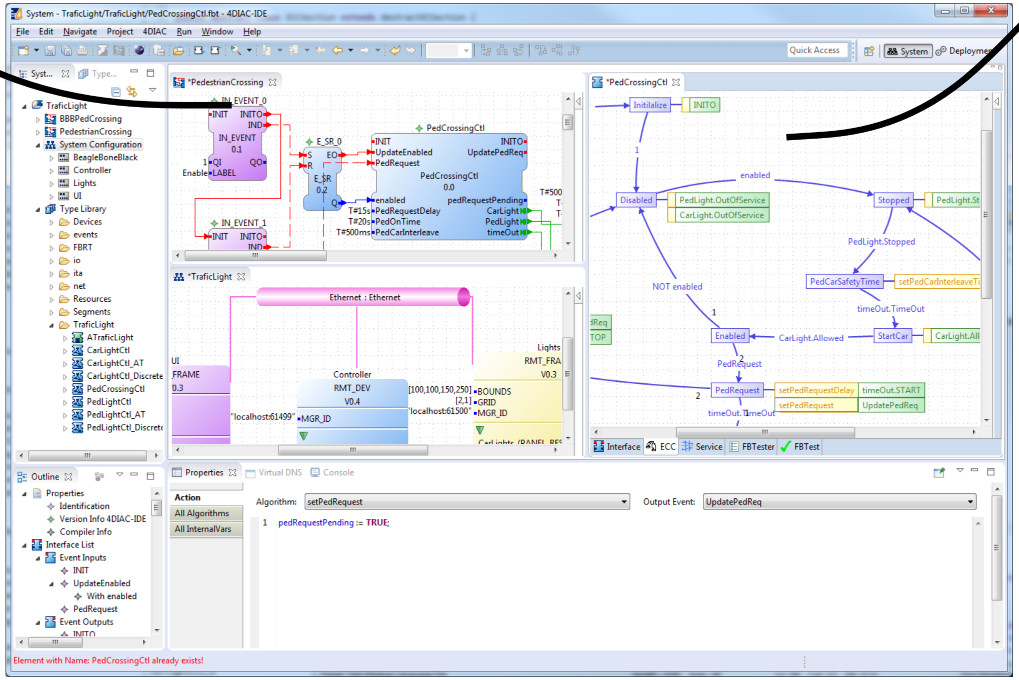
Smart Factory

- **Key features:** Interoperability, Information Transparency (raw data into high level data), Decentralized Decisions, Technical Assistance;
- **Examples:** Plug and Produce; using the cloud to optimize production, easy configuration of new production variants;
- **Fortiss Future Factory:** `https://www.youtube.com/watch?time_continue=202&v=Tkcv-mbhYqk`

4Diac

Programming Industry 4.0 Applications using Models: 4Diac framework.

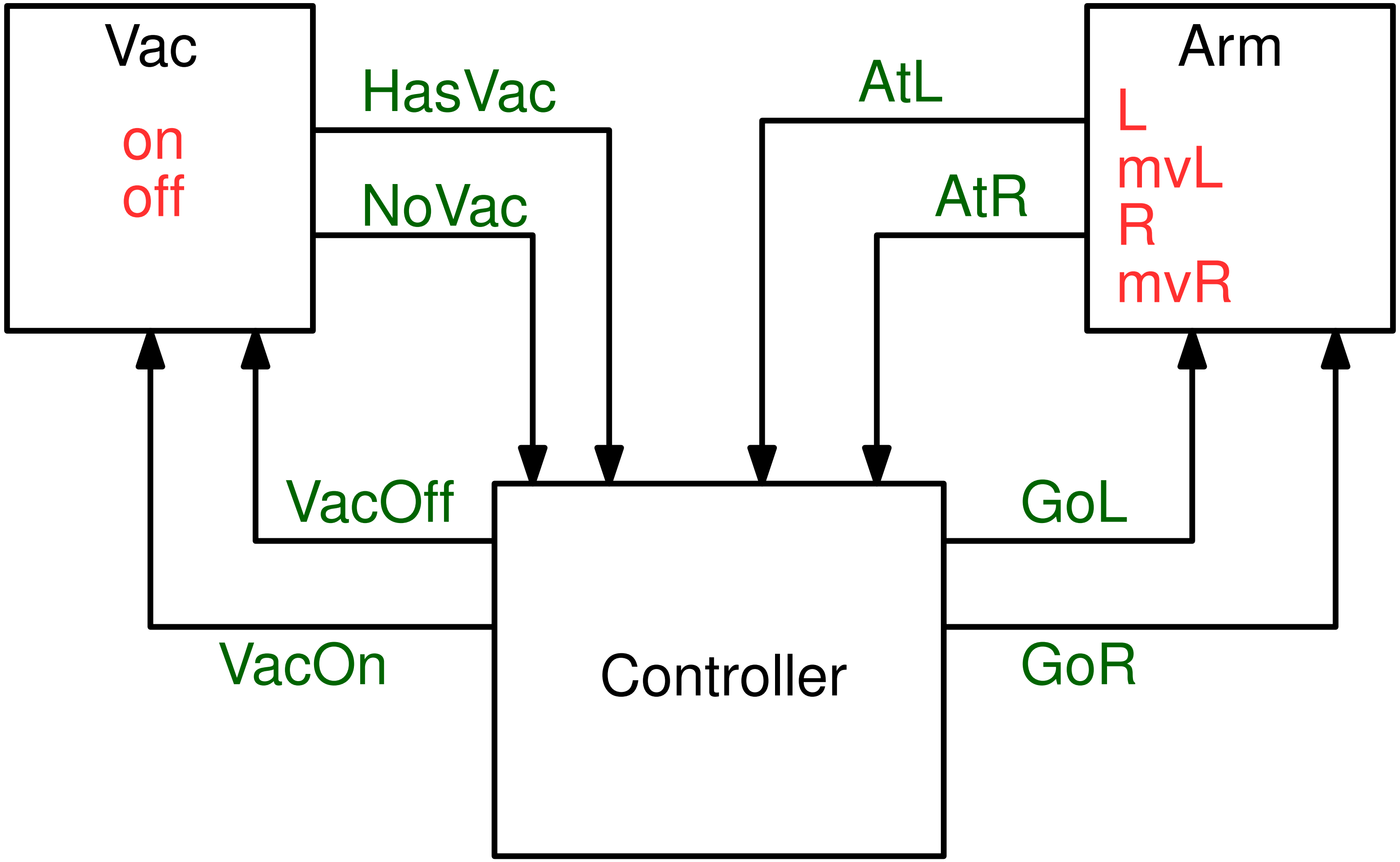
An application is composed by **function block** (FB) connected by **links**.
Links transmit **events** and **data** between FBs.



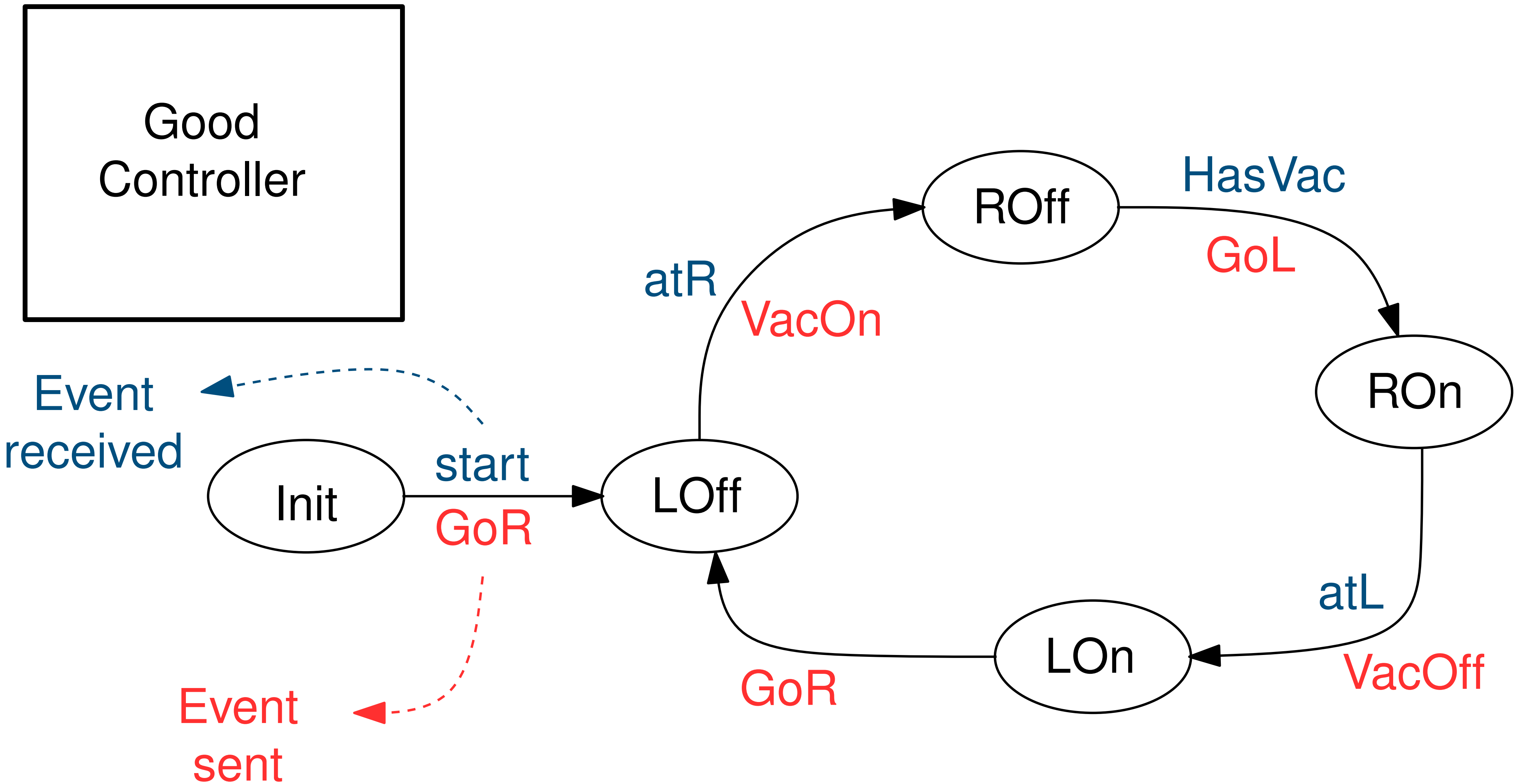
The **behavior** of the FB is specified by a (**deterministic**) Mealy Machine.

4Diac implements the **IEC 61499** standard for distributed industrial processes.

Application: Simplified PickNPlace



4Diac Modeling by Example



Challenges

Cyber Attacks can cause Catastrophic Events.

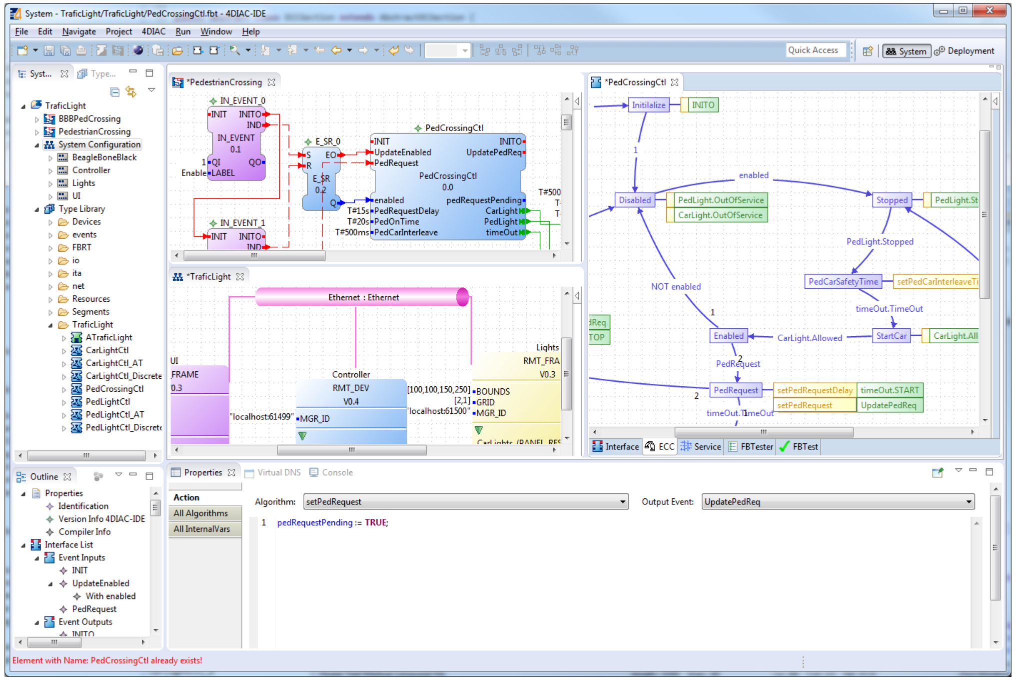
- Cyber-attacks were able to take control of a German mill;
- Ukranian power-station disabled by a cyber-attack;
- Jeep hijacked by hackers;
- Cyber-attacks are able to disable car safety mechanisms.

Levels of Abstraction

- **Application Level:** Logical level without taking into account the deployment nor the network topology of the system.

- **System Level:** FBs are deployed into devices. The network topology is abstract. (**The colours of the FBs in 4Diac.**)

- **Network Level:** FBs are deployed into devices which are connected through a network topology. (Currently being implemented in 4Diac.)



Levels of Abstraction

- **Application Level:** Logical level without taking into account the deployment nor the network topology of the system.

→ **Vast number of examples where formal verification can help, including intruder models.**

- **System Level:** FBs are deployed into devices. The network topology is abstract. (**The colours of the FBs in 4Diac.**)

→ New challenges for formal verification. Devices are weak devices. Powerful Dolev-Yao intruders are not realistic. **We need new intruder models with simpler decision problems.**

- **Network Level:** FBs are deployed into devices which are connected through a network topology. (Currently being implemented in 4Diac.)

→ Additionally, one can use network defences, such as, filtering **suspicious** messages.

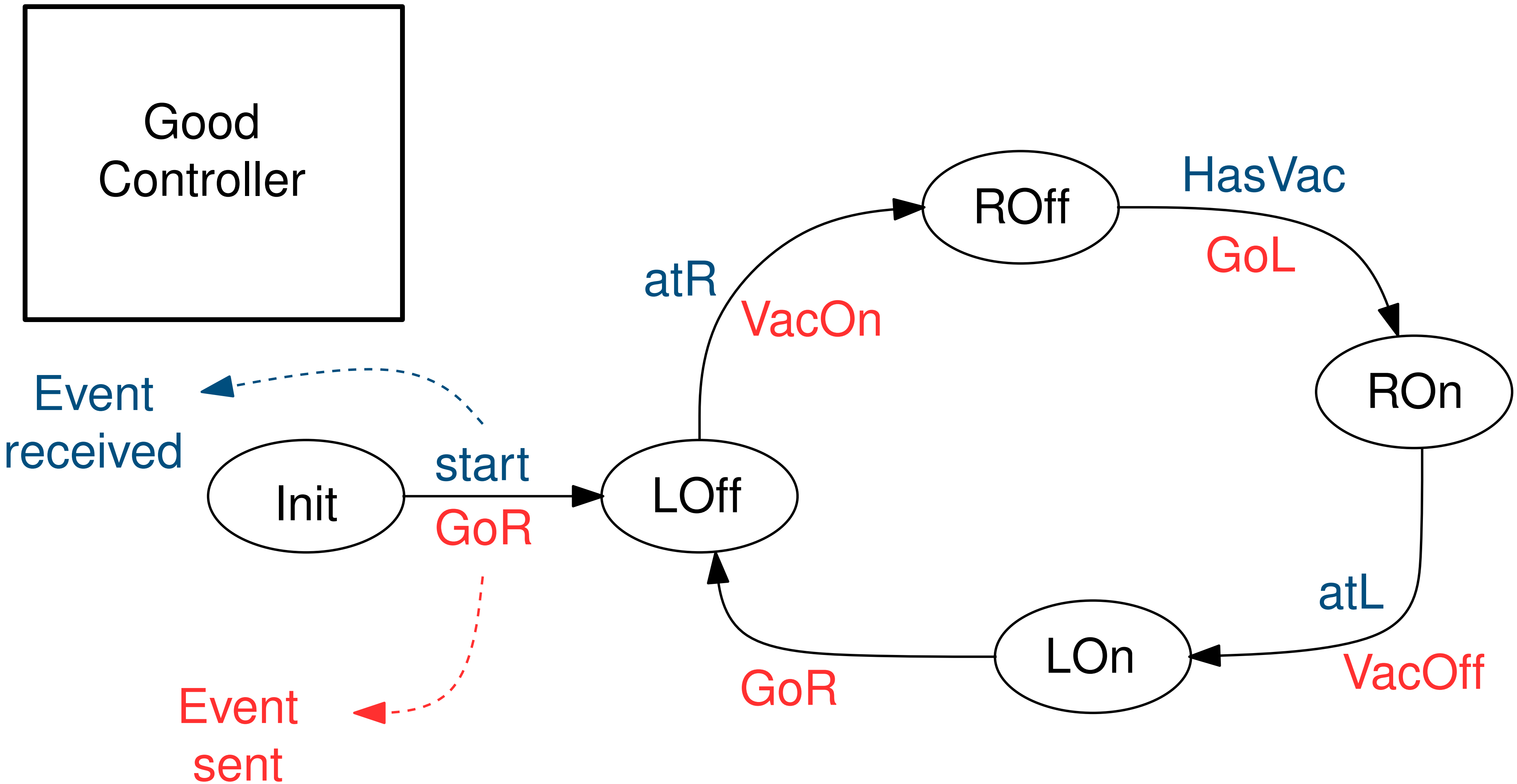
Overview

- We formally specified **an extension with security concerns** of 4Diac semantics in **Maude**.

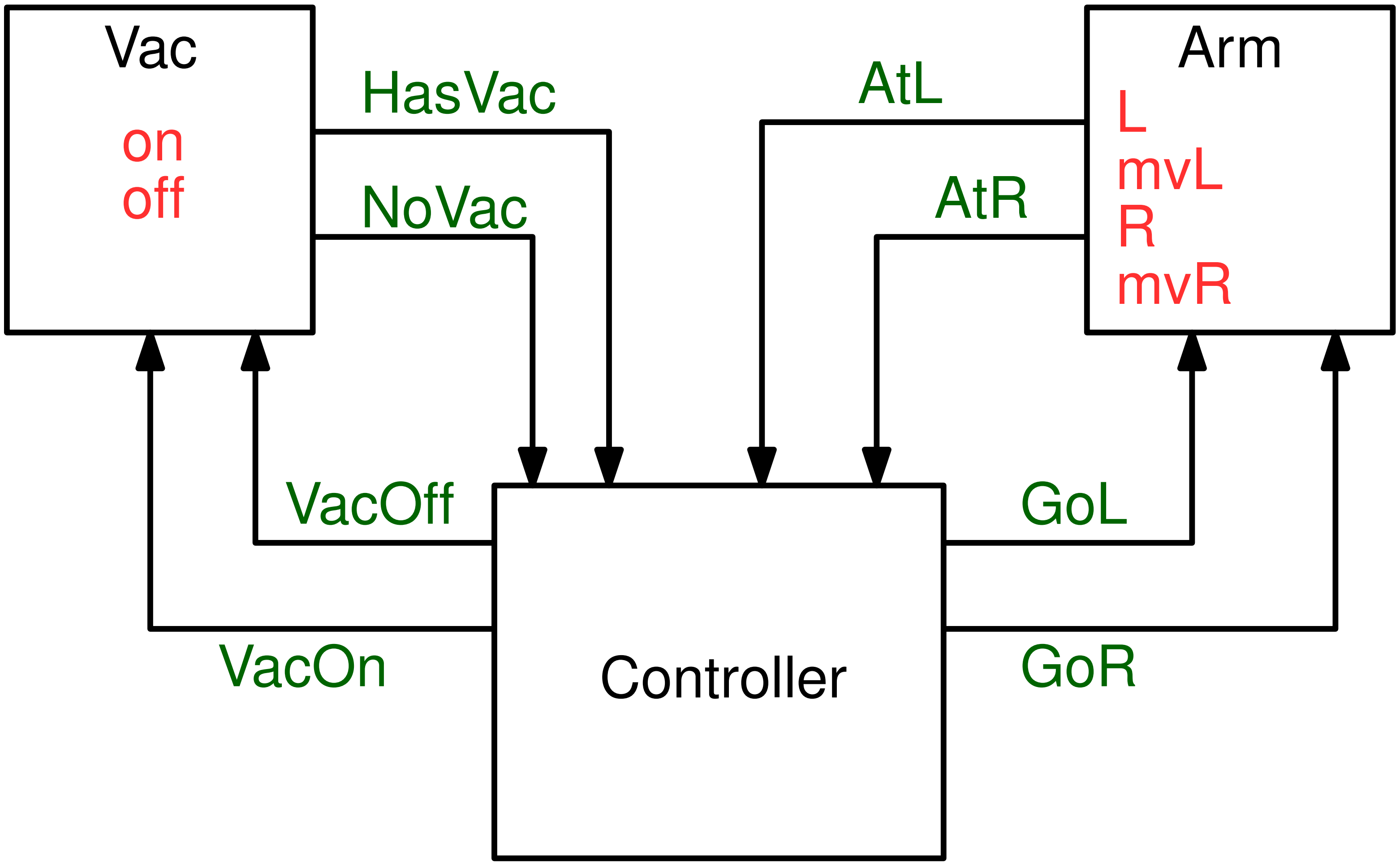
Features:

- **Feature 1 - Specification of Applications**: Model check for **logic errors** in the application;
- **Feature 2 - Deployment**: Deployment of applications in **systems** (sets of devices) as well as in **networks** (sets of devices connected through switches);
- **Feature 3 - Intruders**: Specification of intruders that can **inject messages into channels**;
- **Feature 4 - Defensive Wrappers**: Specification of wrappers relying on **signed messages**;

4Diac Modeling by Example



4Diac Modeling by Example



Bad State: Arm in state **mvL** and Vac in state **off** as may lead to the safety hazard of dropping the cap.

4Diac Modeling by Example

Encoding of example in Maude:

Configurations

```
appInit = [[app, fb(vac, st("off"), none, none)
                 fb(track, st("L"), none, none)
                 fb(ctl, st("init"), none, none)
                 {{ctl,inEv( "start")},ev("start")},none]]
```

Function Block States

Input messages to be processed

State Transitions specified as equations

```
tr(st("init"), st("LOff"), inEv("start") is ev("start"), outEv("GoR") :~ev("GoR"))
```

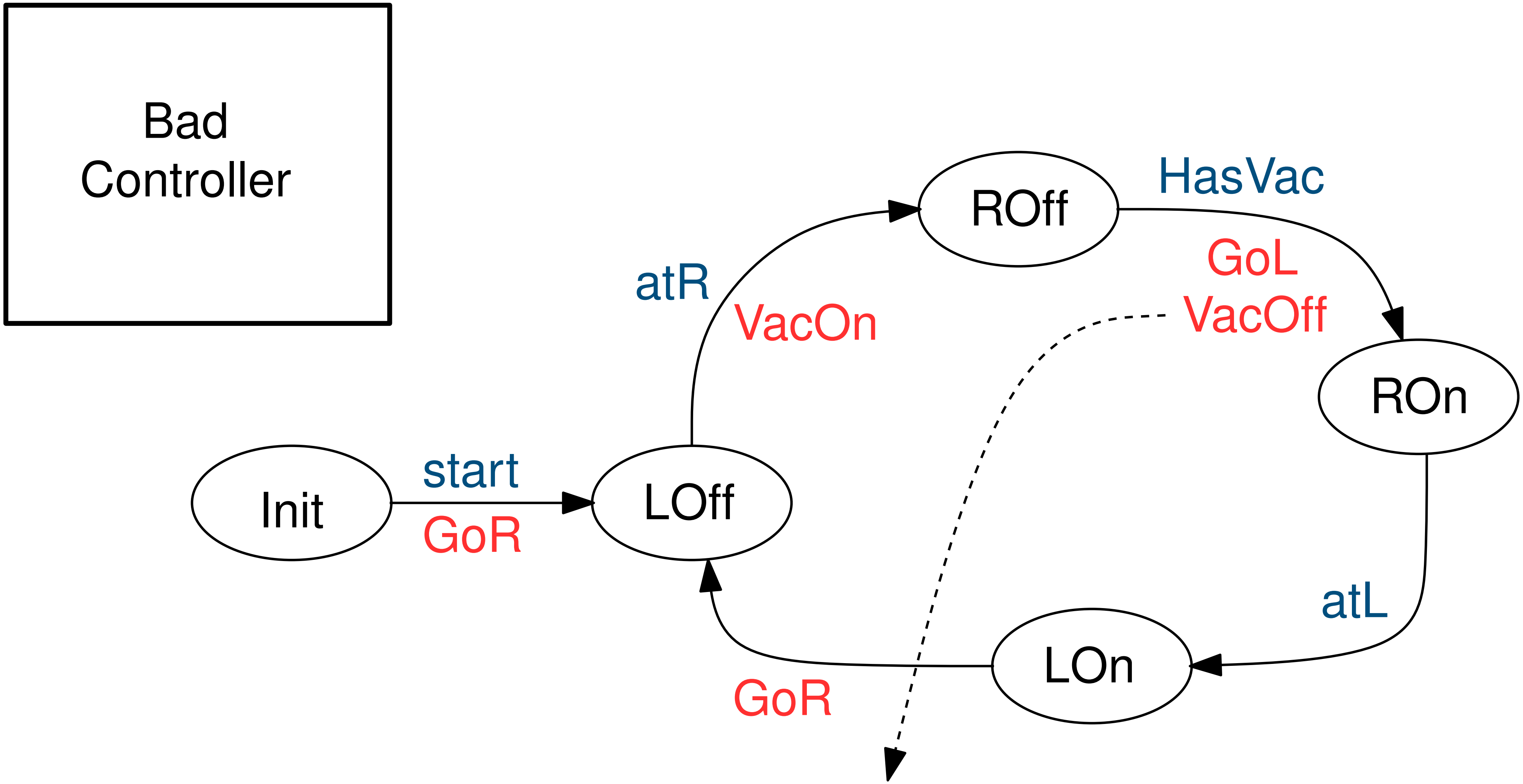
4Diac Modeling by Example

Encoding of example in Maude:

Operational Semantics specified as rewrite rules over configurations

```
crl[app-exe1]: [[id,fbs,iMsgs,none]]
   =>
  [[id,deliverToFBs(fbs,iMsgs),none,none]]
  if not (iMsgs == none) .
```

4Diac Modeling by Example



May lead to a bad state depending on which event is processed first.

4Diac Modeling by Example

- **Feature 1 - Specification of Applications**: Model check for logic errors in the application;

- badappInit is the configuration with the bad controller.

search badappInit =>* app such that badState(app) .

Finds 4 ways of reaching a bad state in less than 2ms traversing 35 states.

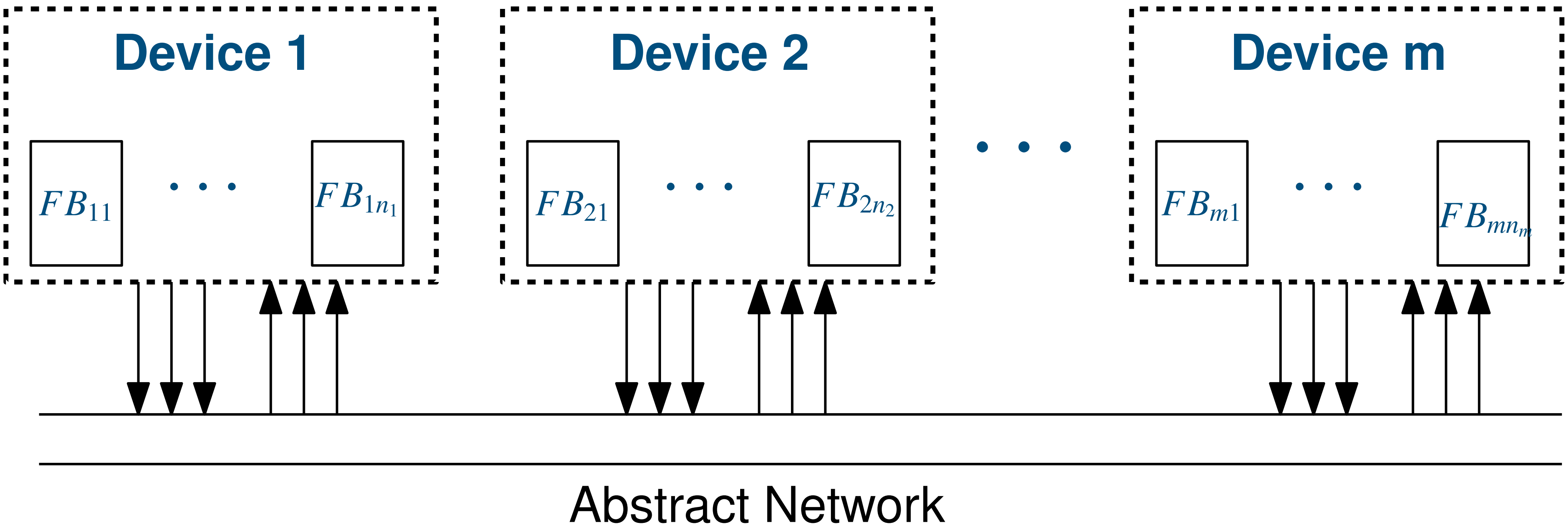
- appInit is the configuration with the good controller.

search appInit =>* app such that badState(app) .

Determines that there is no way to reach a bad state in 1ms traversing 36 states.

4Diac Modeling by Example

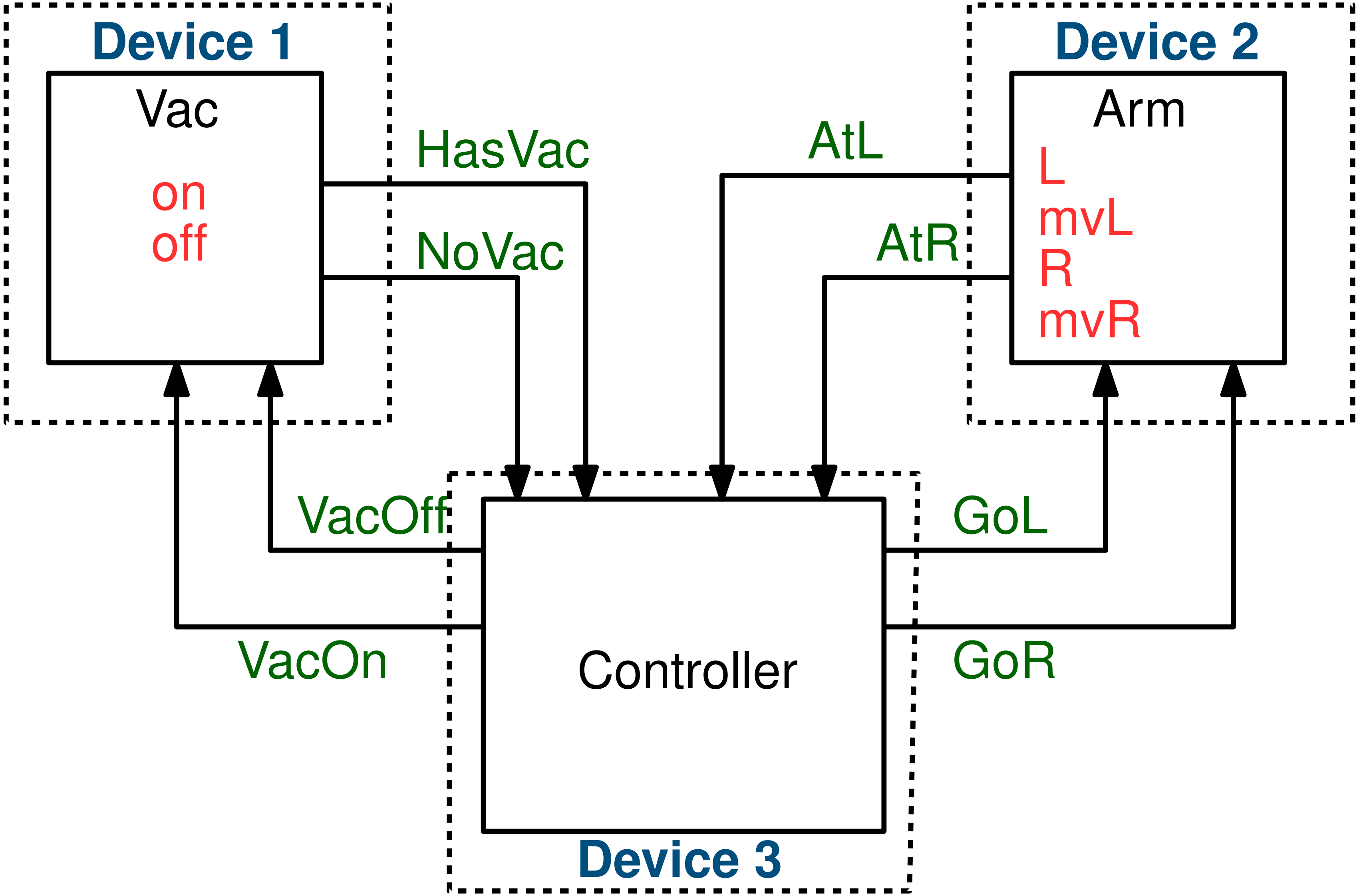
- **Feature 2 - Deployment**: Deployment of applications in **systems** (sets of devices)...;



Deployment is computed automatically from the applications and a function block to device mapping.

4Diac Modeling by Example

- **Feature 2 - Deployment**: Deployment of applications in **systems** (sets of devices)...;



4Diac Modeling by Example

- **Feature 3 - Intruders**: Specifcation of intruders that can **inject messages into channels**;

Intruder Device

Msg 1

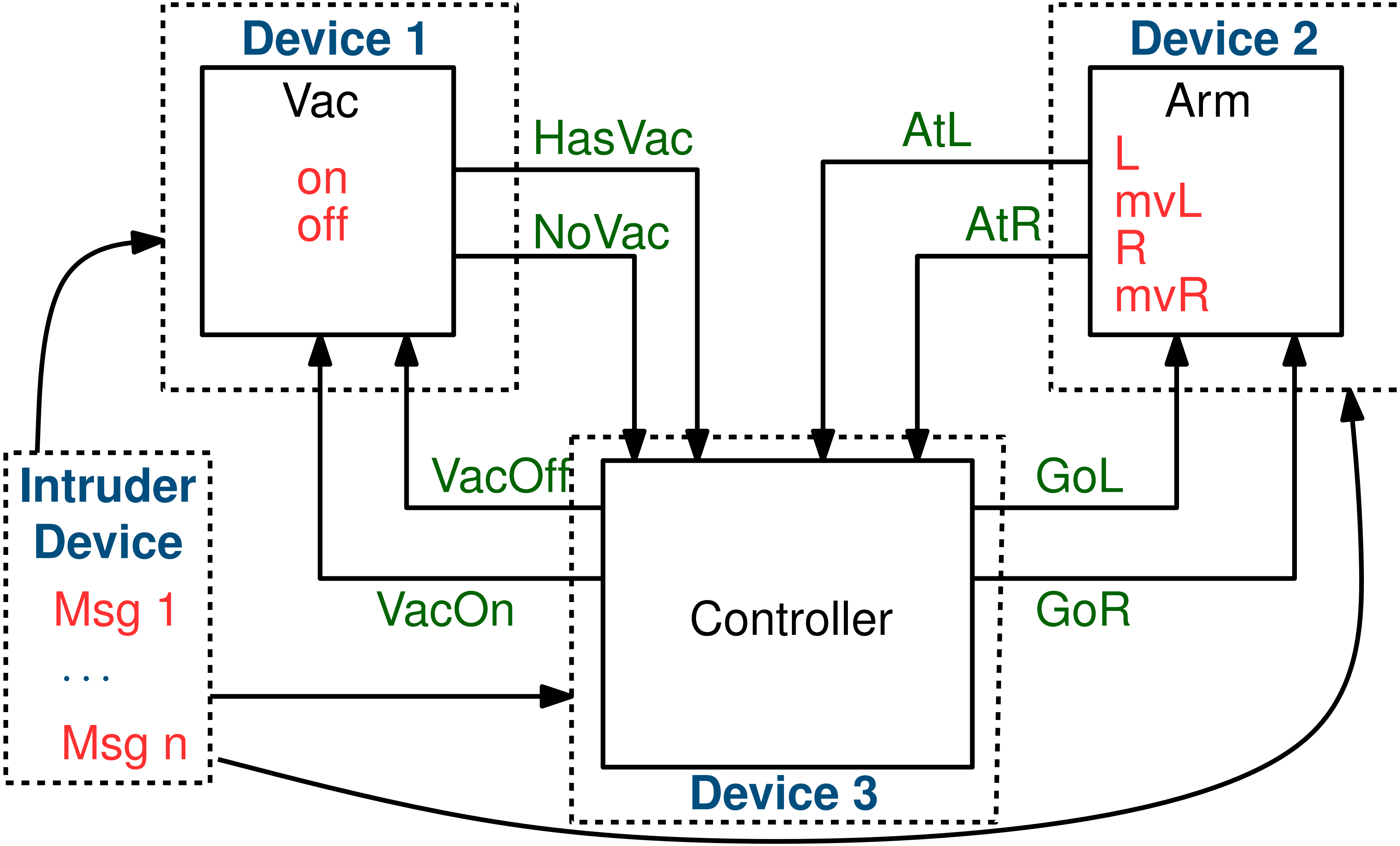
. . .

Msg n

- Intruder can **inject at any time any** anyone of the messages Msg 1, . . ., Msg n in the system.
- In the current implementation, intruders can only use a message once. **On the one hand, it reduces state space**, but **on the other hand, it reduces the capabilities of the intruder.**

4Diac Modeling by Example

- **Feature 3 - Intruders**: Specifcation of intruders that can **inject messages into channels**;



4Diac Modeling by Example

- **Feature 3 - Intruders**: Specifcation of intruders that can **inject messages into channels**;

Configuration: wsysIntruderNoPol

- Intruder possess only one message:

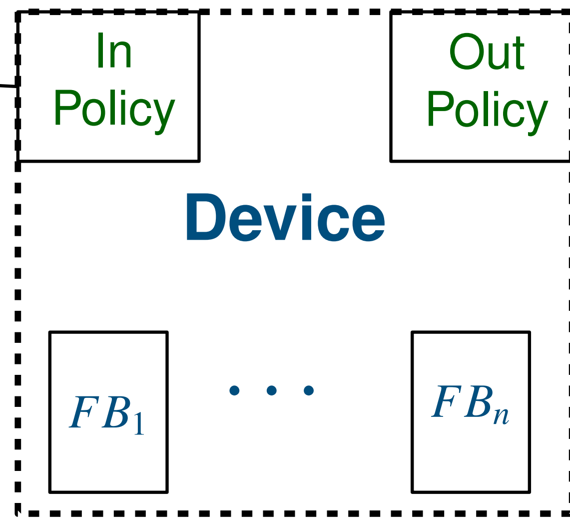
```
{{dev3,{ctl,outEv("VacOff")}},{dev1,{vac,inEv("VacOff")}},ev("VacOff")} .
```

```
search wsysIntruderNoPol =>* wsys such that badState(wsys) .
```

Finds 7 attacks in less than 4ms traversing 154 states.

4Diac Modeling by Example

- **Feature 4 - Defensive Wrappers**: Specifcation of wrappers relying on **signed messages**;



Set of pairs of input events and devices, $\{(ev_1, dev_1), \ldots, (ev_n, dev_n)\}$.

Device only accepts these events if signed by the correct device.

For example, device with Vac in policy: {(VacOff, Device 1}.

Set of out events specifying which events, $\{ev_1, \ldots, ev_n\}$ that are sent to the network signed: $sign(ev_1, sk), \ldots, sign(ev_n, sk)$.

For example, out policy of Device 1 with the controller: {VacOff}.

4Diac Modeling by Example

- **Feature 4 - Defensive Wrappers**: Specifcation of wrappers relying on **signed messages**;

Configuration: wsysIntruderPol

- Intruder possess only one message:

  {{dev3,{ctl,outEv("VacOff")}},{dev1,{vac,inEv("VacOff")}},ev("VacOff")} .

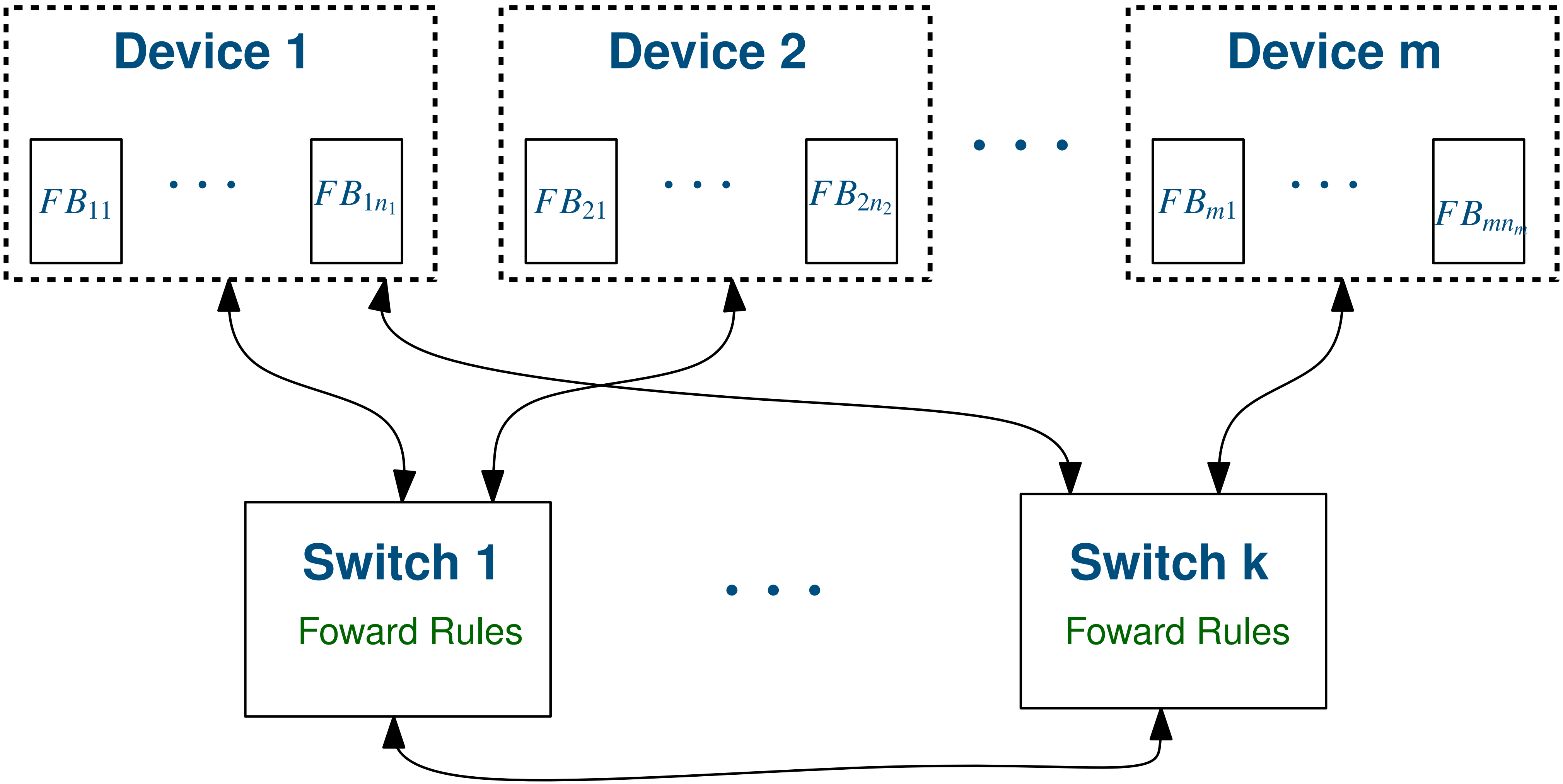
- Devices with defensive wrappers.

search wsysIntruderPol =>* wsys such that badState(wsys) .

Check that there are no attack in less than 4ms traversing 38 states.

4Diac Modeling by Example

- **Feature 2 - Deployment**: ... as well as in **networks** (sets of devices connected through switches);



4Diac Modeling by Example

- **Feature 2 - Deployment**: ... as well as in **networks** (sets of devices connected through switches);

Configuration: netIntruderNoPol

- Intruder possess only one message:

  {{dev3,{ctl,outEv("VacOff")}},{dev1,{vac,inEv("VacOff")}},ev("VacOff")} .

- **Devices without defensive wrappers;**
- Devices connected through one switch.

search netIntruderNoPol =>* nsys such that badState(nsys) .

Finds 12 attacks in around 60ms traversing 488 states.

Configuration: netIntruderPol

- **Devices with defensive wrappers;**

search netIntruderPol =>* nsys such that badState(nsys) .

Concludes that there are no attacks in 25ms traversing 200 states.

4Diac Modeling by Example

- **Feature 2 - Deployment**: ... as well as in **networks** (sets of devices connected through switches);

Configuration: netIntruderPol

- Intruder possess only one message:

  {{dev3,{ctl,outEv("VacOff")}},{dev1,{vac,inEv("VacOff")}},ev("VacOff")} .

- **Devices with defensive wrappers;**

- Devices connected through one switch.

search netIntruderPol =>* nsys such that badState(nsys) .

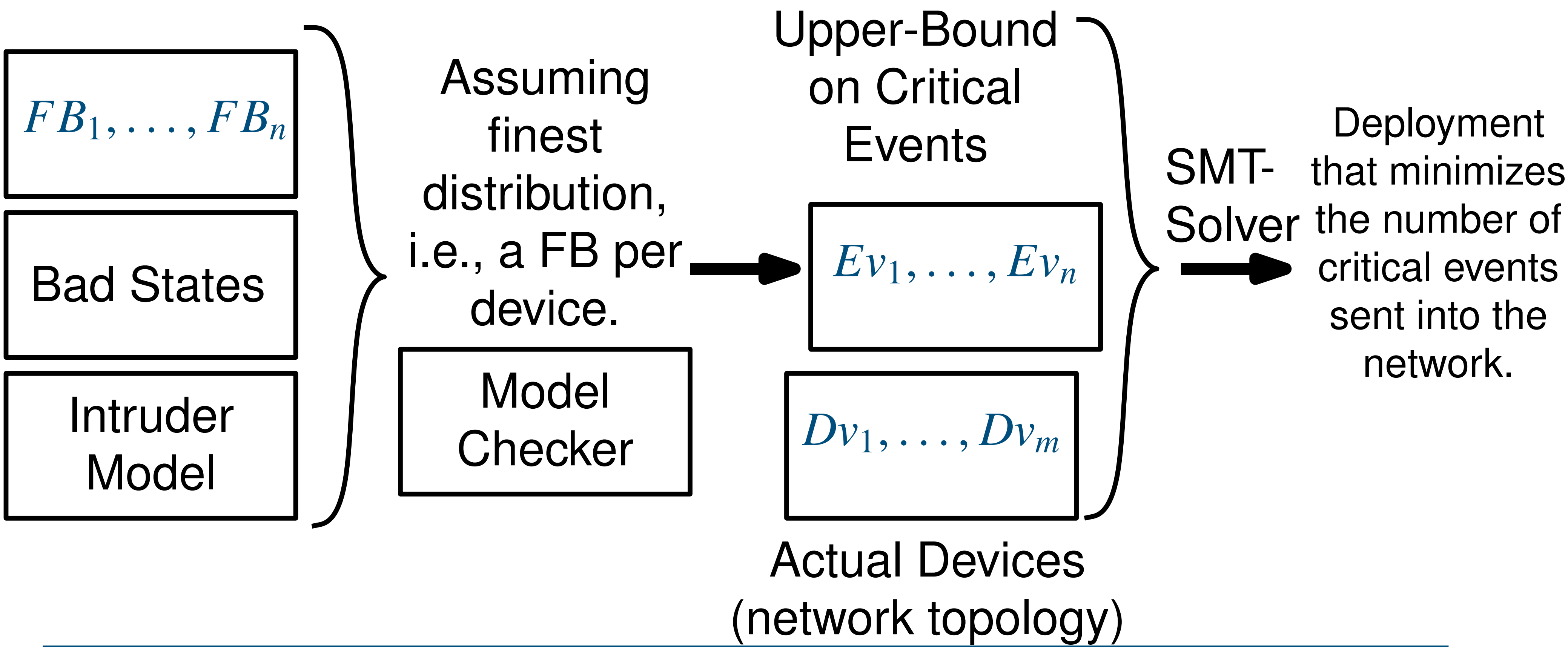
Checks that no attack is possible in 13ms traversing 98 states.

Other uses of Formal Verification

Deployment Optimization: Encryption is **heavy and undersirable** as it leads to delays. Unfeasible for applications such as the **Tomato Separator**:

https://www.youtube.com/watch?v=EBddJjYNp0g

Our Idea (still under development)



Issues / Questions

- Performance:
  TSN communication;
  What is the impact of signature? Weaker signature schemes?

- Defenses using Network:
  Can we use firewall to mitigate attacks, by, e.g., isolate sub-nets?
  Port Security?

- Intruder Model:
  Is it running on a good device?
  Is it running on its own device?
  What information can we rely on the HW/wiring to provide that is not fungible?

Future Work

- Model defenses for switches: firewalls?, SDN rules?
- Decision problems and complexity results;
- Include data channels;
- Stronger Intruder Models: Symbolic Analysis
- Stronger defensive wrappers;
- Investigate scalability;
- Extend model with TSN.

Questions?