

A Model-Driven Approach to Develop Adaptive Firmwares

Franck Fleurey, Brice Morin and Arnor Solberg

franck.fleurey@sintef.no

SINTEF, Oslo, Norway



This work has been funded by the DiVA (EU FP7 STREP) and MODERATES (SINTEF) projects.

Context

- Internet of things / Sensor networks
 - Resource constrained systems
 - Increasingly complex and dynamic applications
- Current practice
 - Trial and error development of C modules
 - Tangling of different concerns
 - Data processing, communication and networking, adaptation, ...
 - No maintenance, reuse or evolutions considerations
- In the need for software engineering techniques
 - Suited for resource-constrained environments
 - Taking the hardware and physical systems into account

Motivating example

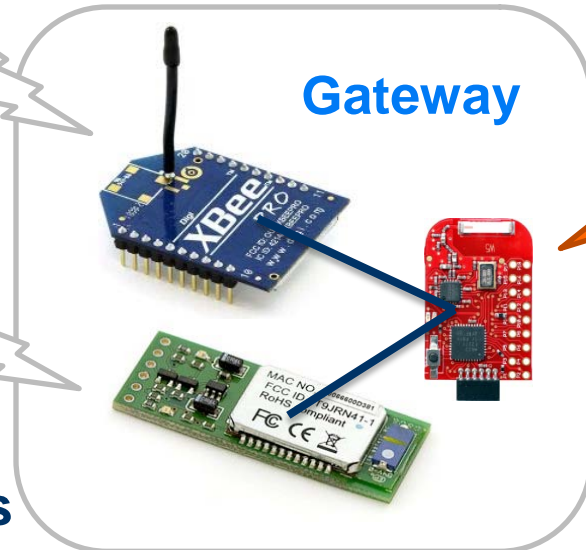
ZigBee Clients

- Data Logging server
- Other Sensor Nodes
- ZigBee Appliances



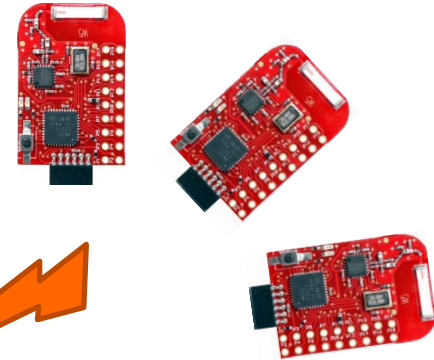
Bluetooth clients

- Smartphones
- Tablet applications
- Laptop computers



Gateway

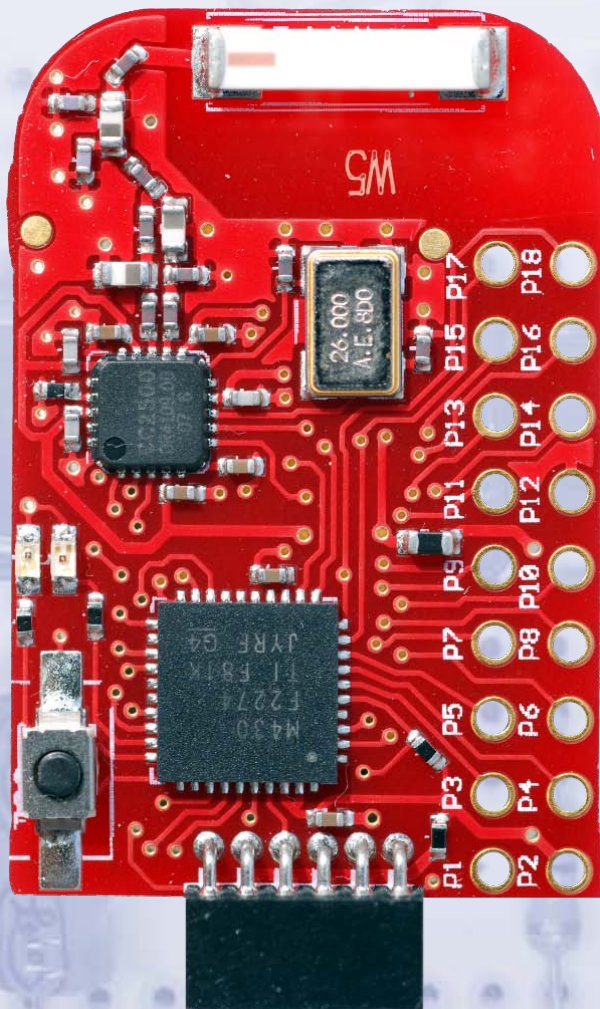
Bridge between low power / low range radio communication to standard radio technologies



Adaptive Wireless Temperature Sensor Nodes

- Adapt to the situation
- Battery powered
- Limited resources
- Low power radio link

The target platform



- MSP430 F2274
 - 16 bits RISC
 - 32 ko Flash
 - 1 ko SDRAM
 - 16 MHz
- USART
- Digital I/Os
- ADC Inputs (10-bits)
- Wireless Transmitter and Receiver
- 2 LEDs
- 1 Push-button

How complicated can it be?

- Very simple functional specification

- Provide temperature data to the applications

```
while (true) {  
    sampleTemperature();  
    sendTemperature();  
    delay(5000);  
}
```

- But... in a resource-constrained environment

- Limited CPU and memory, battery powered, limited bandwidth

- But... Different applications have different needs

- Update rate, notifications, accuracy, response time, ...

How complicated can it be?

■ Very simple functional specification

■ Provide temperature data to the applications

- Adapt sampling frequency and measurement accuracy
- Have a sensor keep track of minimum and maximum
- Provide an “alarm service” to clients
- Gracefully degrades on low battery
- Etc...

■ But...

- Dynamically adapt to provide the right services

■ Limited CPU and memory, battery powered, limited bandwidth

■ But... Different applications have different needs

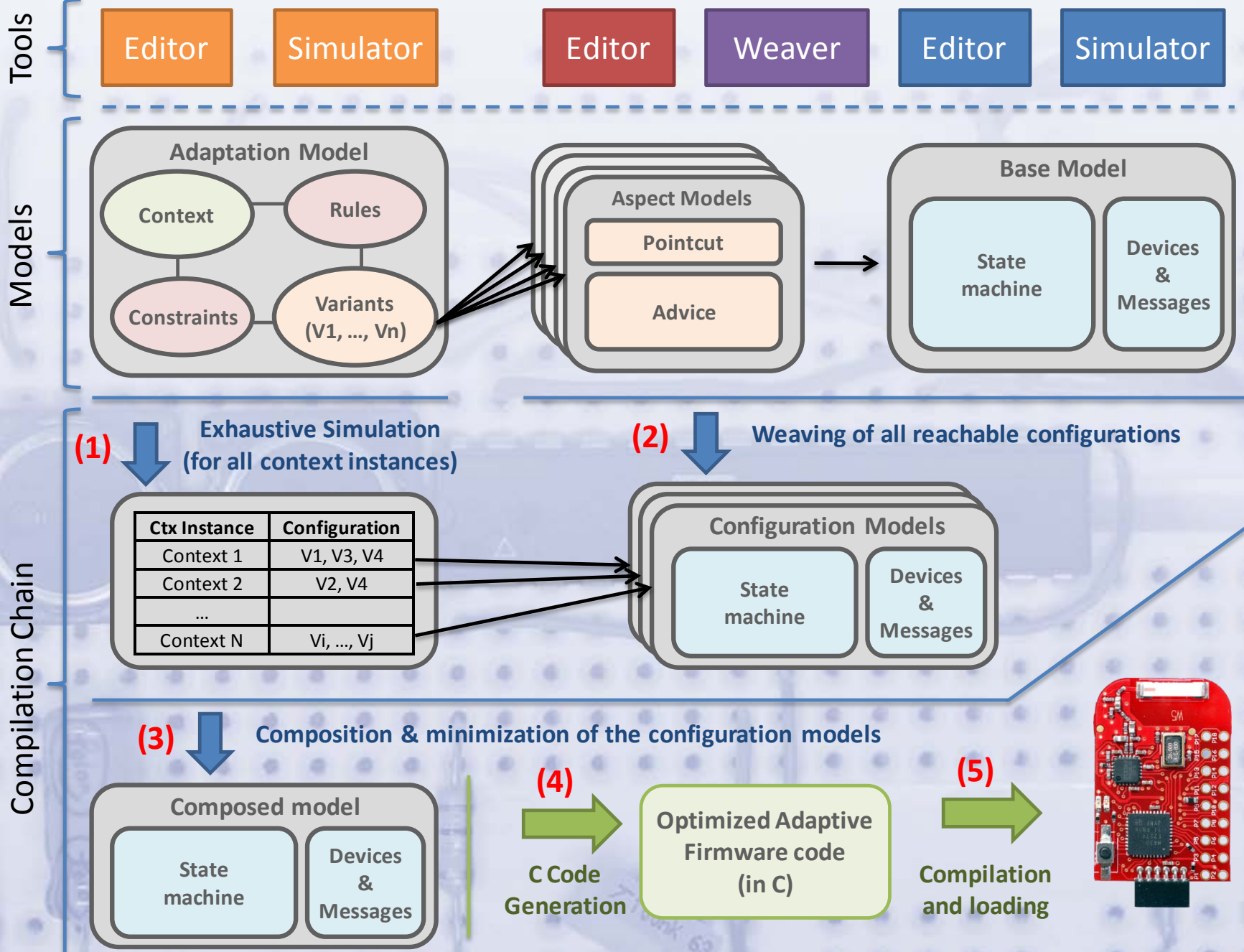
- Update rate, notifications, accuracy, response time, ...

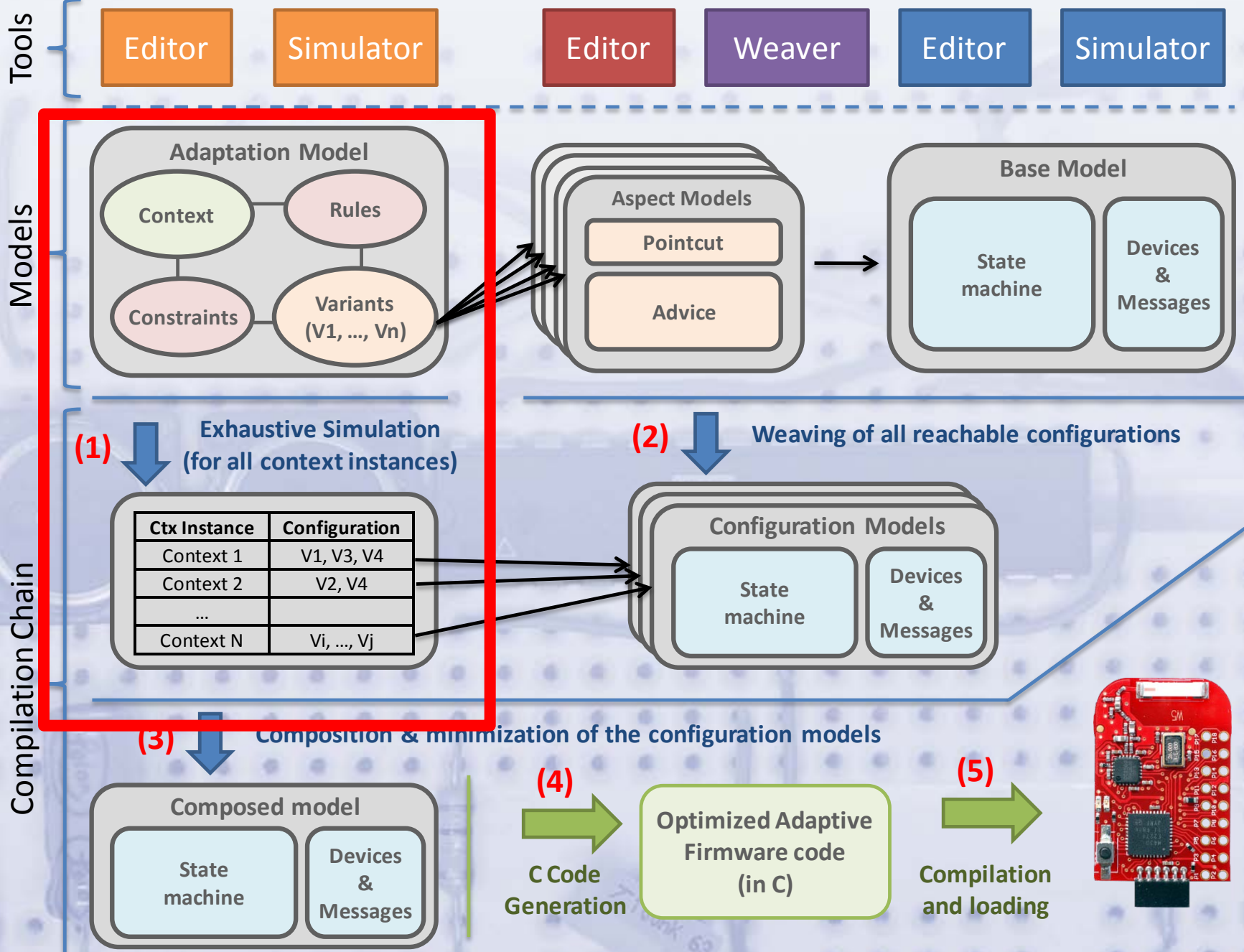
More generally

- Include more logic in the sensors
 - Push the logic as close as possible to the data source
 - Save unnecessary communication which are costly in terms of power and bandwidth.
 - Have the sensor adapt to its environment
 - Find QoS trade-offs according to available resources.
 - Have the sensors adapt the applications needs
 - Adjust the QoS trade-offs to the needs and priority of the applications
- Makes producing the firmware a lot more challenging
- Becomes a self-adaptive system
- Need for software engineering techniques

Proposed Approach

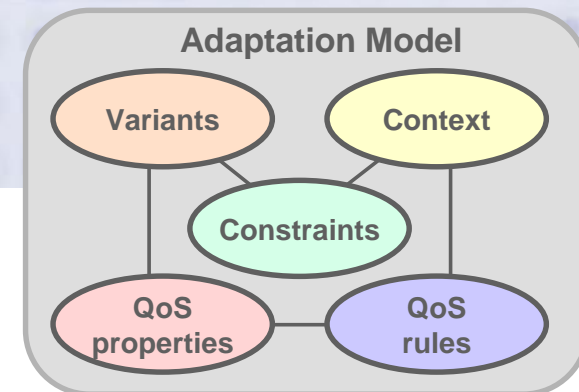
- Model the system expected behaviour
 - Using event driven state machines and asynchronous messages
- Model the runtime variability in the system
 - Using aspect-oriented modelling techniques
- Model the adaptation policies
 - Using a domain specific modelling language
- Perform exhaustive analysis of these models
- Produce an optimized state machine which compiles behaviour, variability and adaptation together
- Produce firmware code which runs on the target platform





The Adaptation Model

- Variability in the system
 - Features, dependencies, constraints
- Variability in the environment
 - What are the elements the system should adapt to
- Adaptation policy
 - What feature should be used in a given context
- The DiVA approach combines
 - Adaptation hard-constraints
 - High-level adaptation rules
 - Expressed on properties of the system



Adaptation model

	Name	ID	Values
▶ Enum	Power Status	power	{normal, low, critical}
▶ Enum	Sensor Mode	mode	{passive, active}
Boolean	Has Subscribers	subscribers	-
Boolean	Reliable Network	reliablenet	-
Boolean	RF Signal Available	rfsignal	-
Boolean	Alarm	alarm	-



Context variables

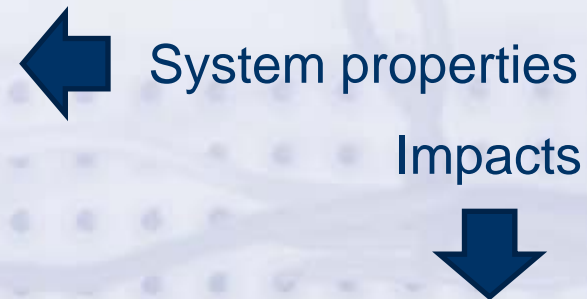
System variability
and constraints



	Name	ID	Lower	Upper	dependency	available	
Dimension	Transmission	T	0	-1	-	-	
Variant	Unicast	UNI	-	-		subscribers	
Variant	Broadcast	BRO	-	-			mode=
Variant	SendMinMax	SMM	-	-	MM and (BRO or UNI)		
Variant	Transmission Errors Detection	TED	-	-	UNI	not power=critical	
Variant	Retransmit On Error	ROE	-	-	TED		
Variant	Buffer On Error	BOE	-	-	TED or ROE		
Dimension	Power Management	P	0	1	-	-	
Variant	Intermittent Operation	INTER	-	-		power=low	
Variant	Critical Operation	SLEEP	-	-	BRO	power=critical	
Dimension	Data Collection Options	D	0	-1	-	-	
Variant	Compute Min/Max values	MM	-	-	SMM or not (UNI or BRO)	power=normal	
Variant	Average 3 Samples	AVG	-	-			

Adaptation model (2)

	Name	Direction	Value
Property	CPU Load	0	-
PropertyLiteral	Negligeable	-	0
PropertyLiteral	Low	-	2
PropertyLiteral	High	-	4
Property	Sensor Data Accuracy	1	-
PropertyLiteral	Improves	-	2
PropertyLiteral	Worsens	-	4
Property	Responsiveness	0	-
PropertyLiteral	Improves	-	2
PropertyLiteral	Worsens	-	4



	CPU Load	Sensor Data Accuracy	Responsivness
Transmission (T)	true	false	true
Unicast (UNI)	Low	-	No Impact
Broadcast (BRO)	Low	-	No Impact
SendMinMax (SMM)	Negligeable	-	No Impact
Transmission Errors Detection (TED)	Low	-	No Impact
Retransmit On Error (ROE)	High	-	Improves
Buffer On Error (BOE)	Low	-	Improves

	Name	ID	context	CPU Load	Sensor Data Accuracy	Responsivness
Rule	Max Accuracy	MA	subscribers and power=normal	Low	High	-
Rule	Alarm	AL	alarm	-	High	-
Rule	Not Alarm		not alarm	High	-	-
Rule	Save power	SP	not power = normal	High	Low	Low
Rule	Bad Network	BN	not reliablenet	-	-	High

Exhaustive simulation

	power	mode	subscribers	reliablenet	rfsignal	alarm
Context	Normal	Active	false	true	true	true
Context	Normal	Active	false	true	true	false
Context	Normal	Active	false	true	false	true
Context	Normal	Active	false	true	false	false
Context	Normal	Active	false	false	true	true
Context	Normal	Active	false	false	true	false
Context	Normal	Active	false	false	false	true
Context	Normal	Active	false	false	false	false
Context	Low	Passive	true	true	true	true
Context	Low	Passive	true	true	true	false
Context	Low	Passive	true	true	false	true



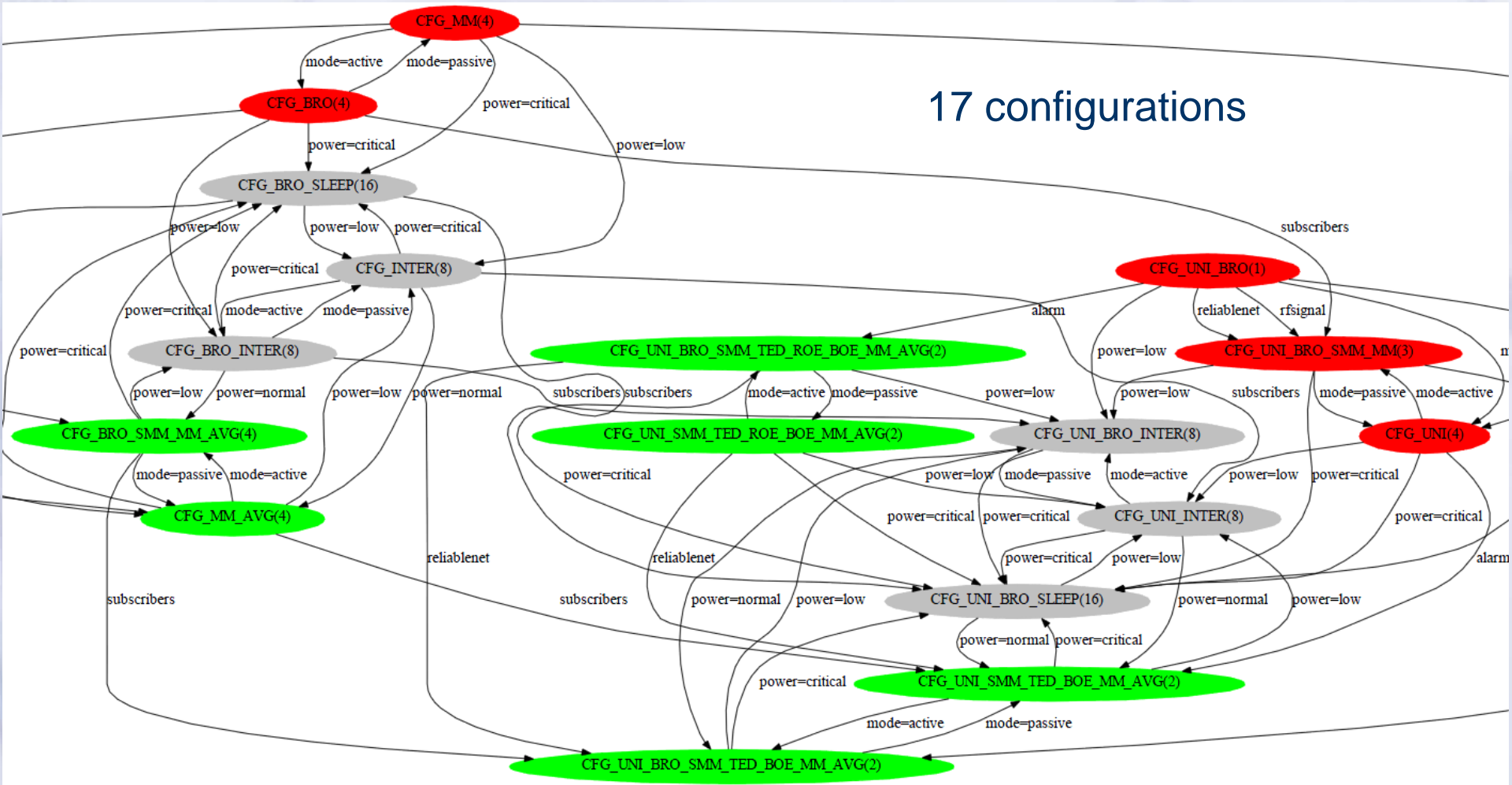
Exhaustive enumeration
of the contexts

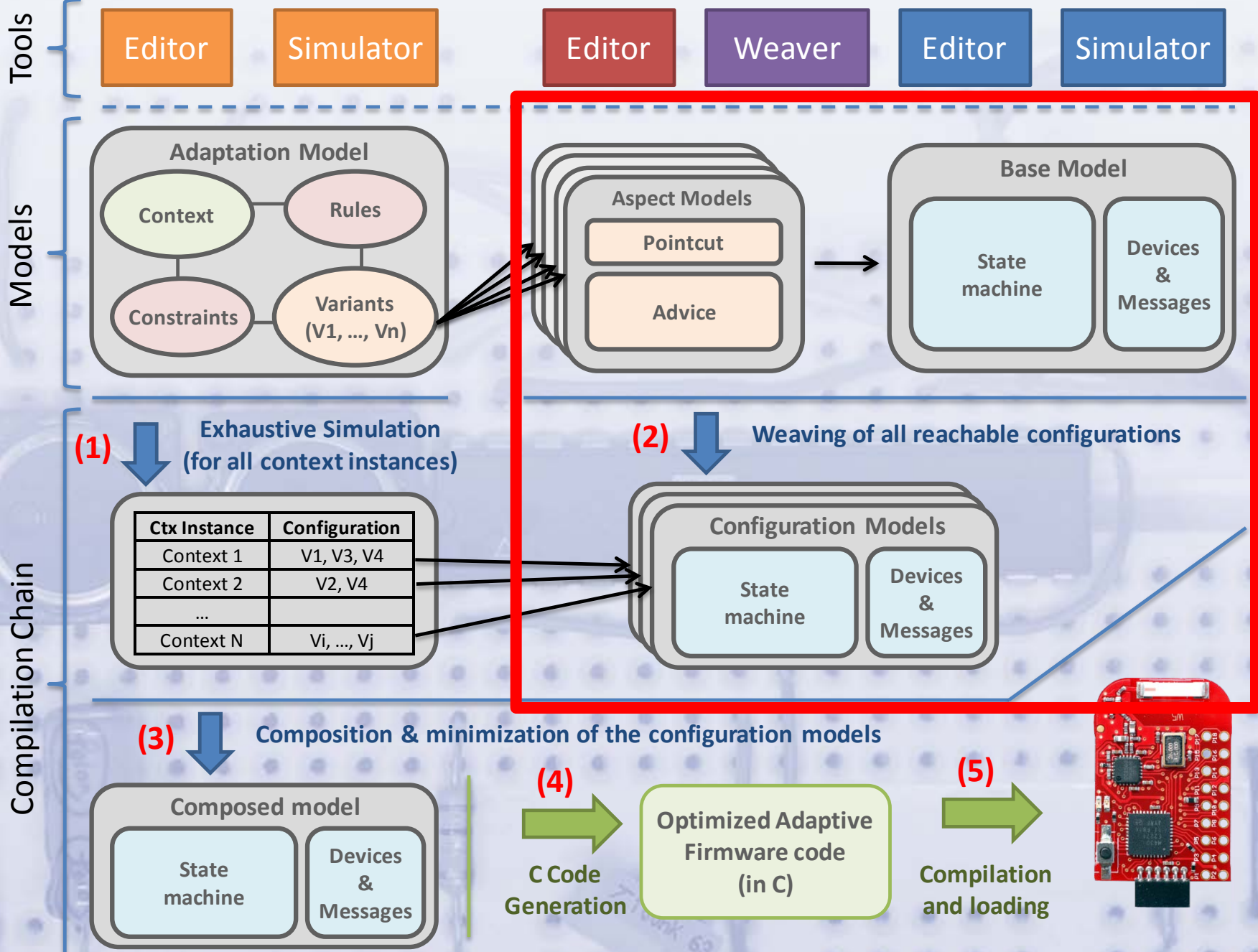
Simulation results



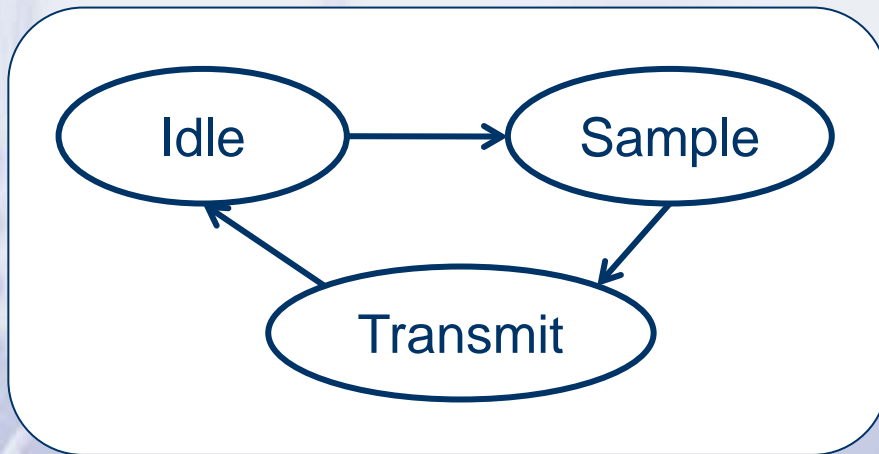
	CPU	ACC	RES	Total
Scenario EXHAUSTIVE	-	-	-	-
Context	Low	High	Lowest	-
Configuration (34)	-48	80	2	34
Unicast (-8)	-8	0	0	-8
SendMinMax (0)	0	0	0	0
Transmission Error	-8	0	0	-8
Buffer On Error (10)	-8	16	2	10
Compute Min/Max	-8	32	0	24
Average 3 Sample	-16	32	0	16
Context	High	High	Lowest	-
Configuration (-32)	-64	32	0	-32
Unicast (-32)	-32	0	0	-32
SendMinMax (0)	0	0	0	0
Compute Min/Max	-32	32	0	0
Context	Low	High	Lowest	-
Configuration (34)	-48	80	2	34
Unicast (-8)	-8	0	0	-8
SendMinMax (0)	0	0	0	0

Exhaustive simulation (2)





Base model with state machines



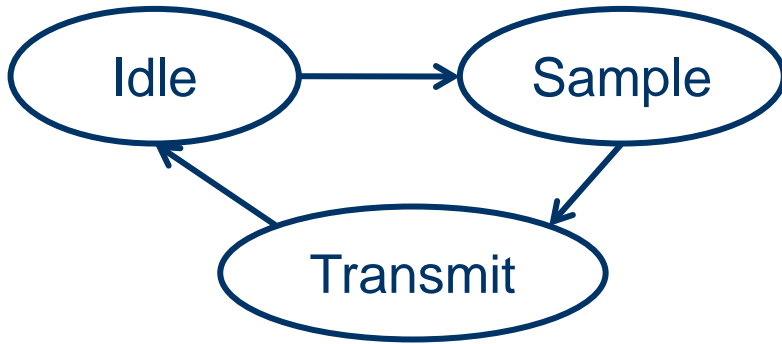
State Machine

Device model

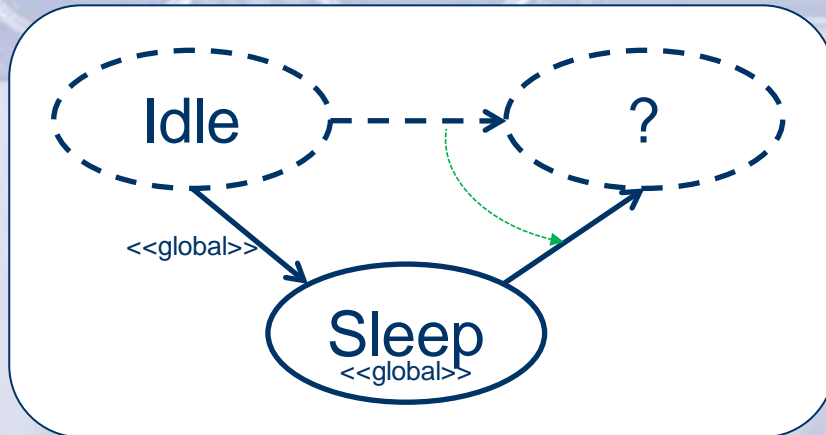
```
device Timer
{
    // Start the Timer
    message start(timer_id : Integer, delay : Integer);
    // Cancel the Timer
    message cancel(timer_id : Integer);
    // Notification that the timer has expired
    message timeout(timer_id : Integer);

    sends timeout
    receives start, cancel
}
```

Aspect to specify variability

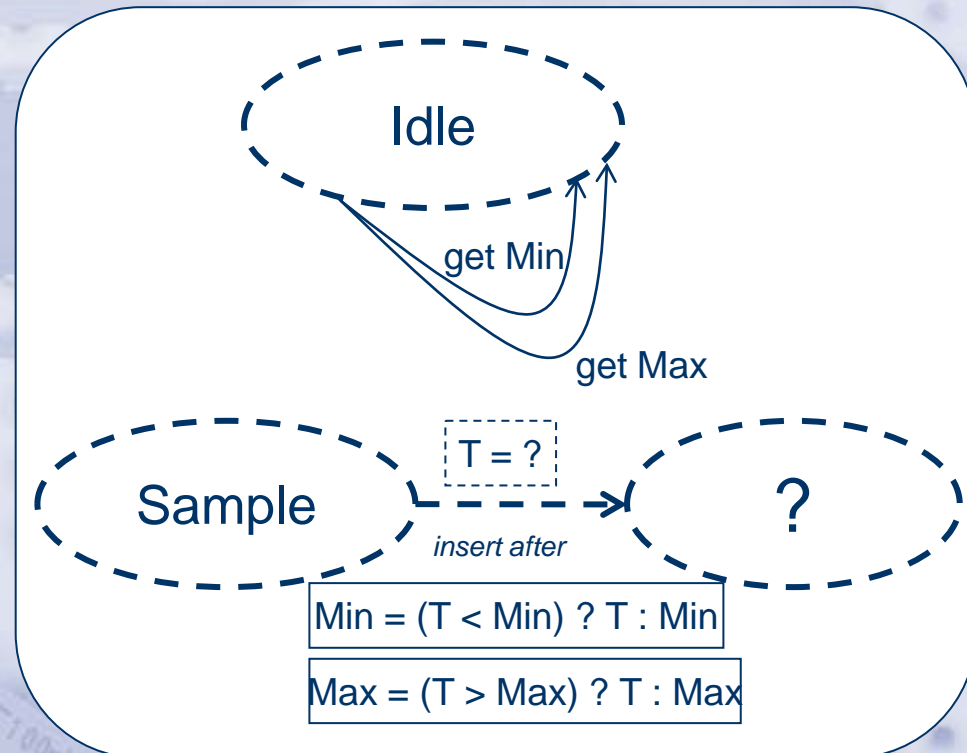


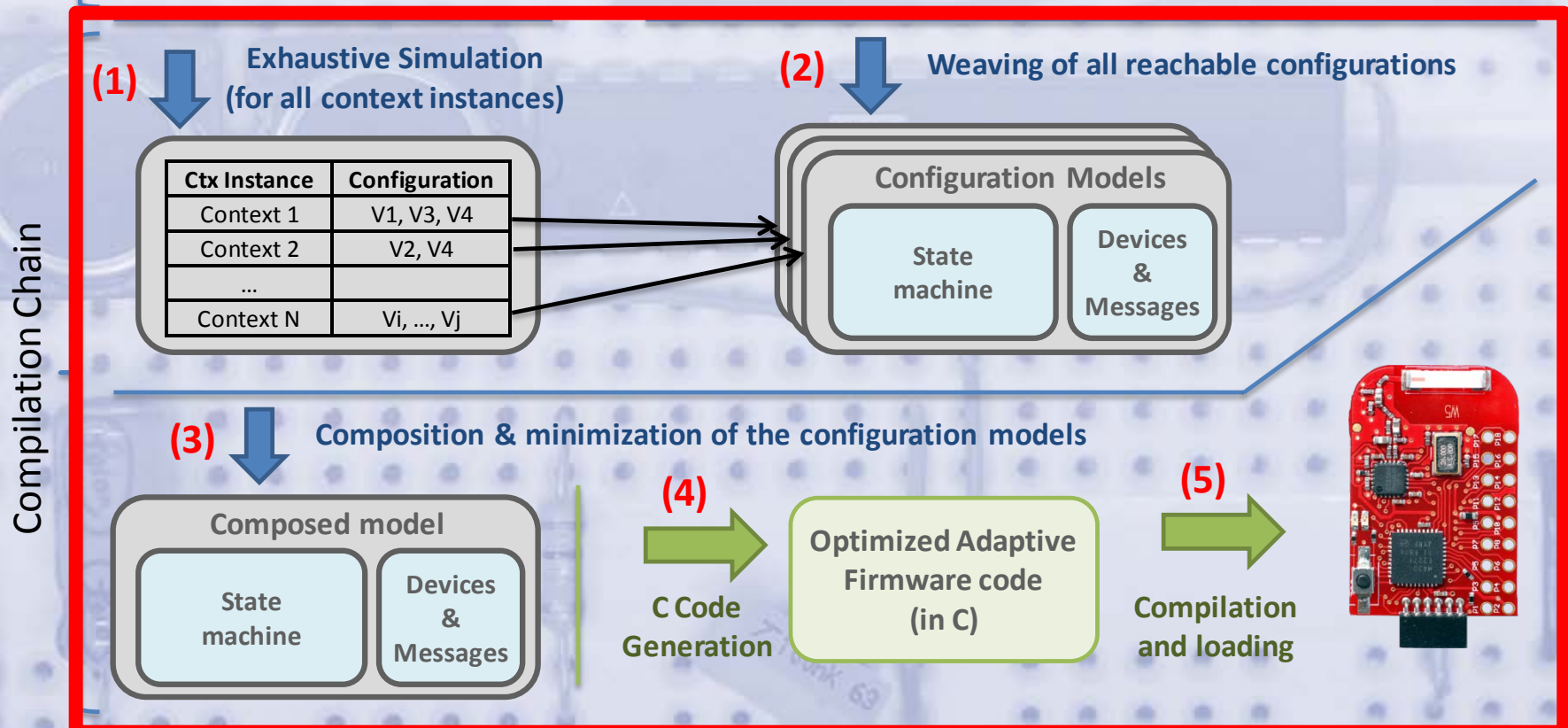
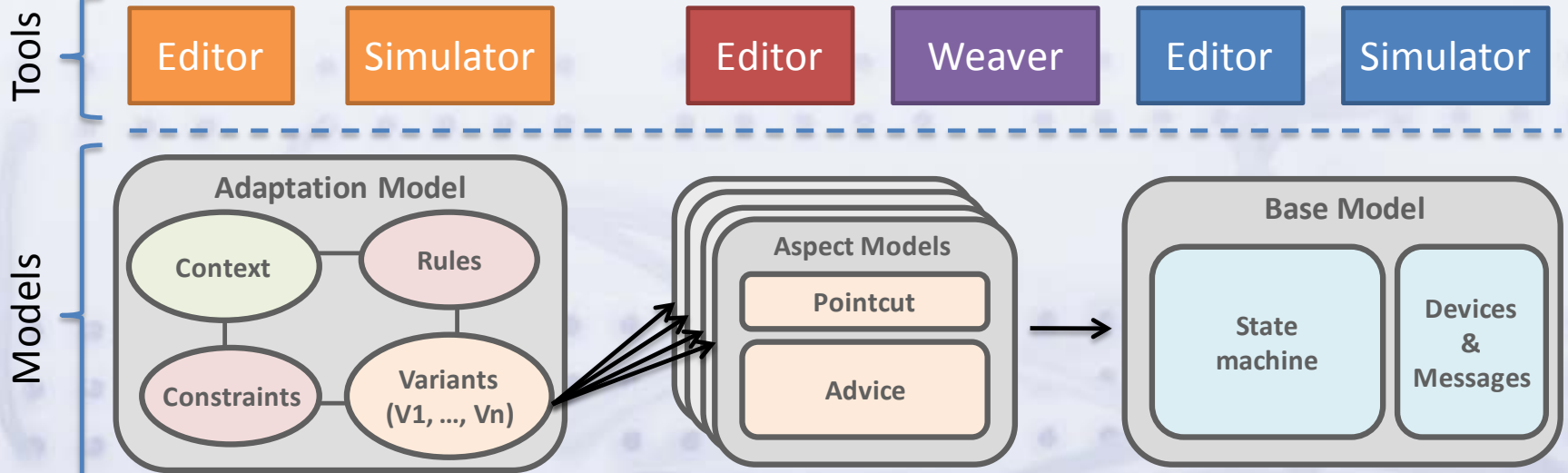
Base model



Sleep Aspect

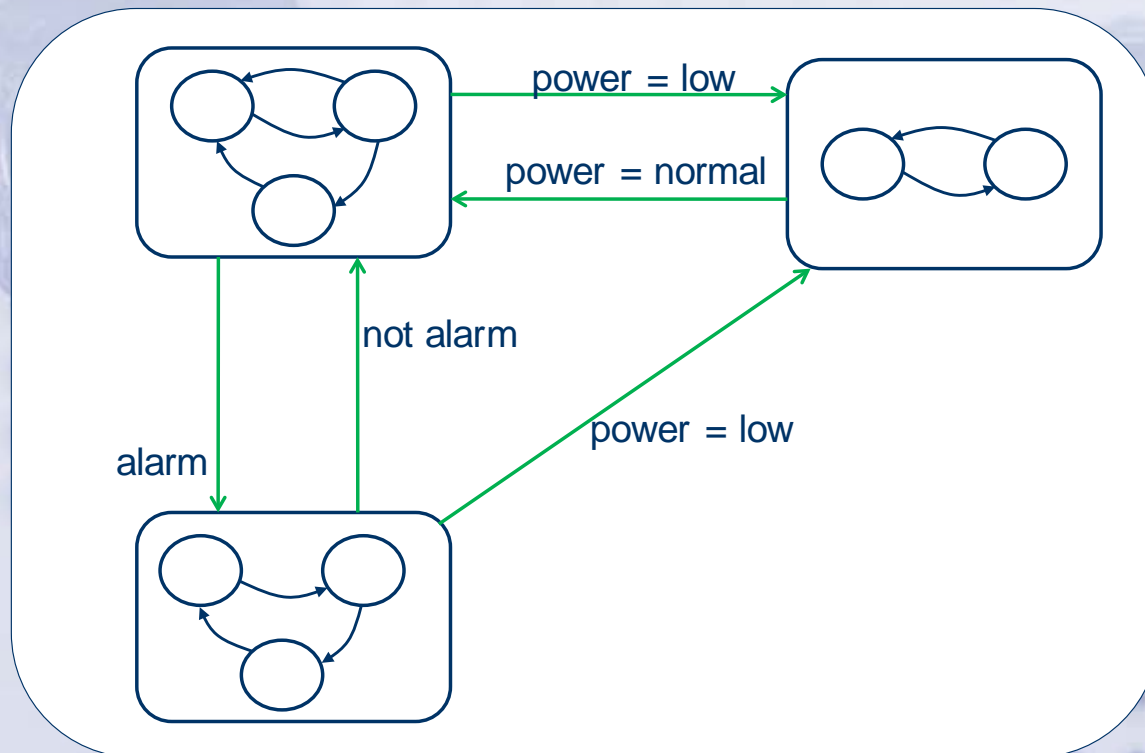
Min/Max Aspect





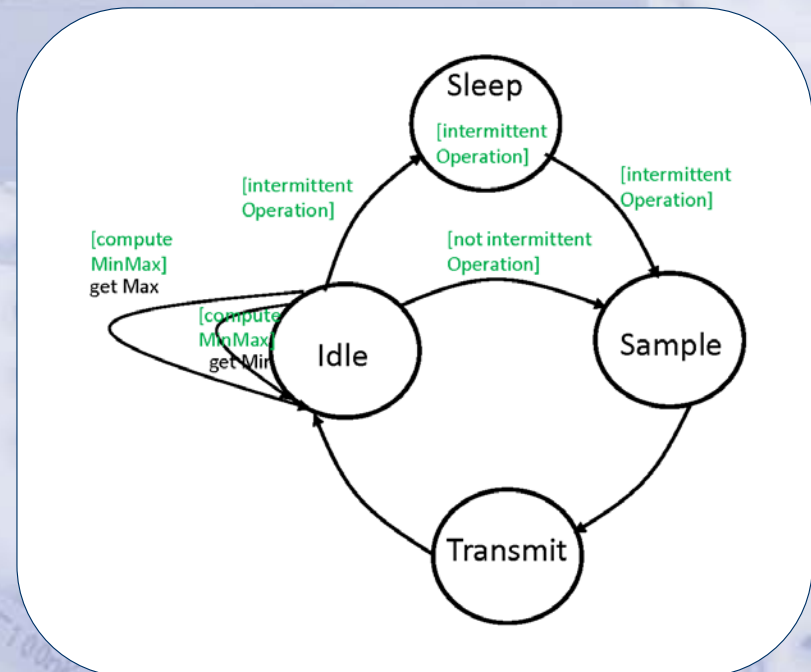
Composed model: Strategy 1

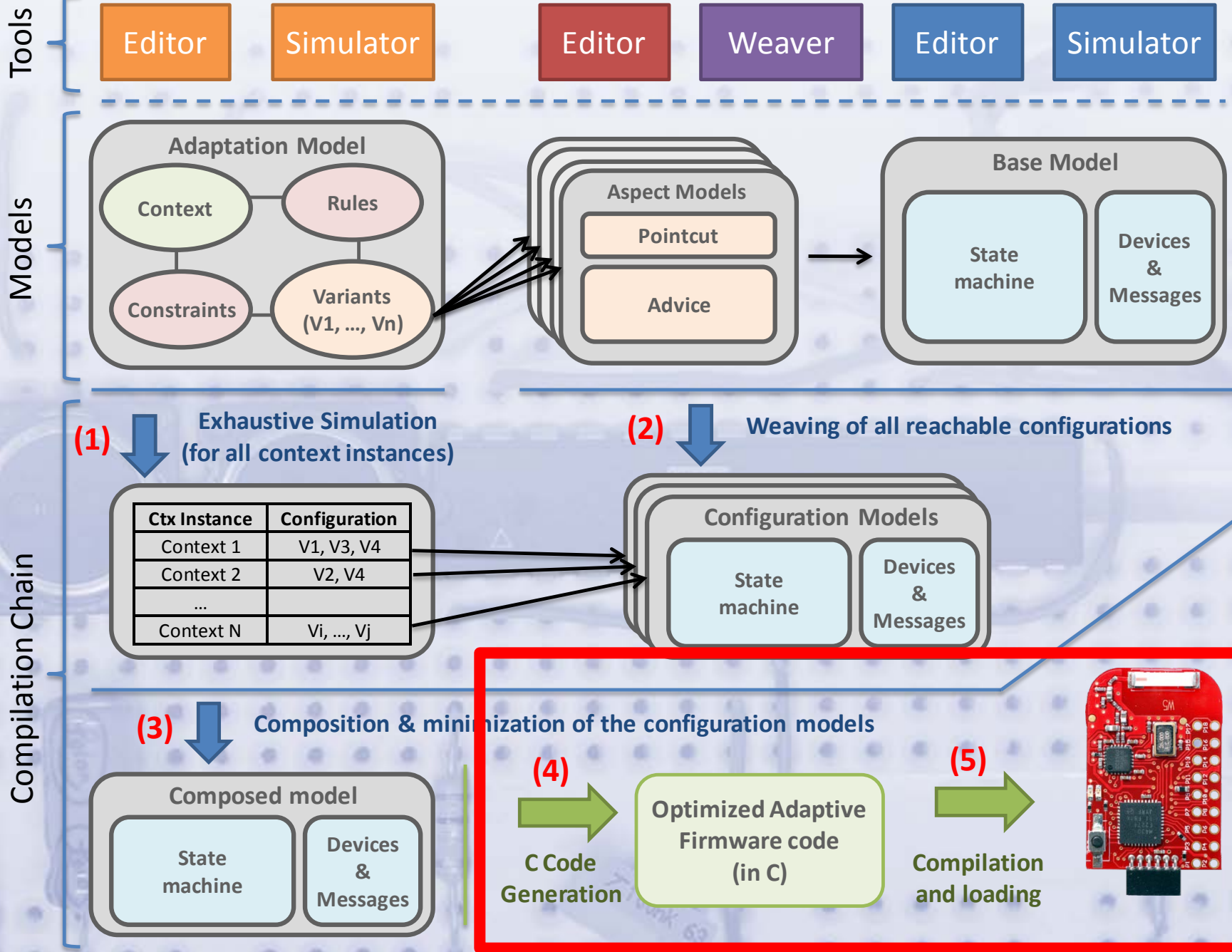
- One composite state per configuration
- Good readability but far from optimal



Composed model: Strategy 2

- Equivalent to the previous state machine
- Obtained by flattening and minimization
- Much more efficient but not as readable
- Used for code generation





Summary

- Sensor networks and Internet of things
 - Highly dynamic adaptive and distributed systems
 - Need proper software engineering techniques
- Even the smallest nodes need to adapt
 - Challenge: resource-constrained environment
 - Opportunity: reasonable scale
- The approach uses domain specific modelling languages
 - Adaptation model combining constraints and goals
 - Behaviour model using state machines and aspects
- Exhaustive simulation and optimisation
- Generation of firmware in plain platform specific C

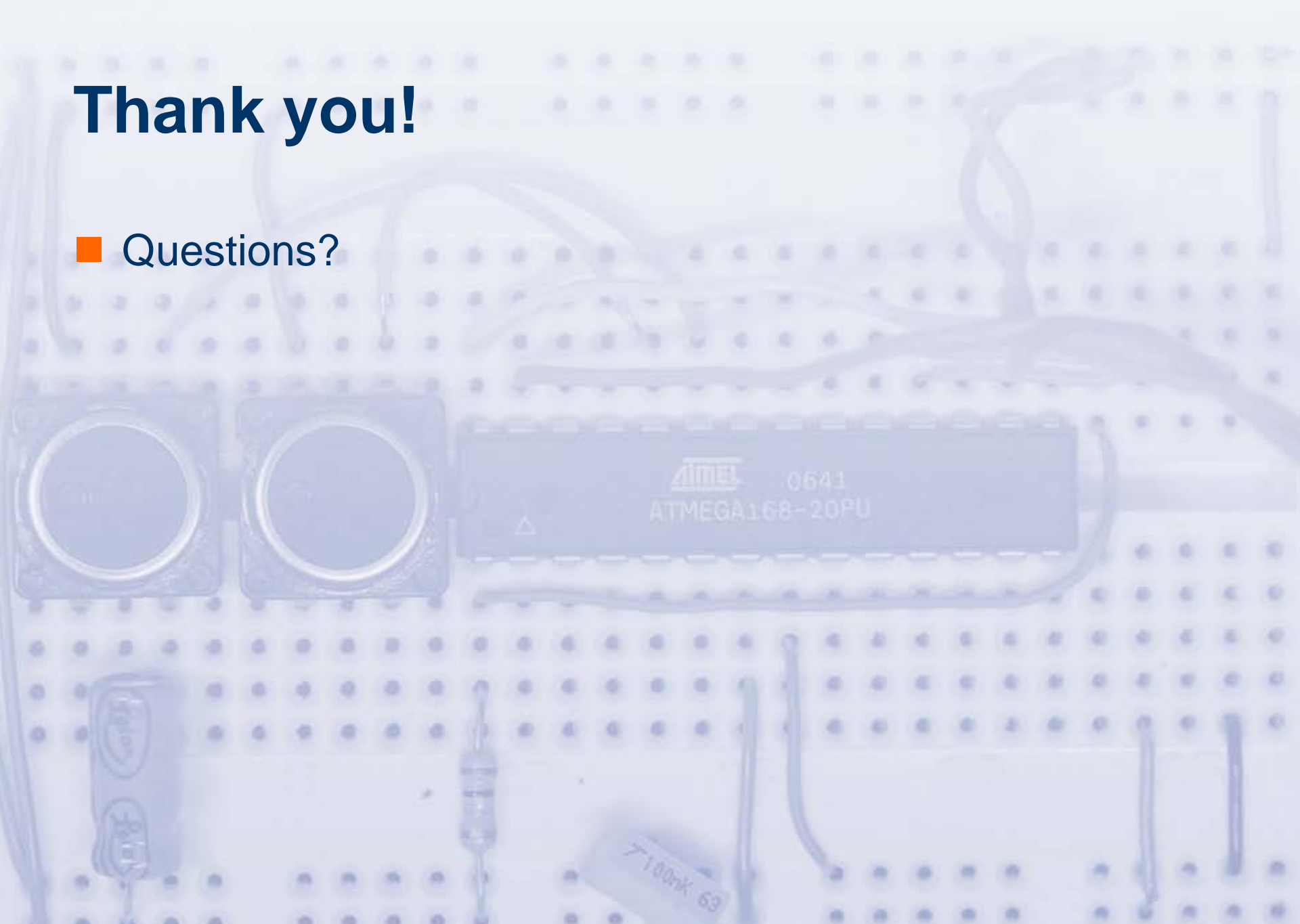


Limitations and Future work

- Automated detection of quiescent states
 - Allow reconfigurations as soon as they are safe
- Validation with model checking
 - Of the adaptation model and combined state machines
- Dealing with “unforeseen” adaptations
 - Firmware live updates
 - Finer grained reconfigurations
- Coordinating the adaptation of a set of devices
 - Connect the adaptation of a set of devices
 - Distribute the adaptation logic
- Integrations of a complete tool chain
 - Individual tools are already available as open-source

Thank you!

■ Questions?



Sate Machine model

```
composite state ACTIVE init ReadSensorValues {

    state WaitingForCmd {

        on entry {
            send Timer.start('1', '10000')
        }

        transition update_data -> ReadSensorValues {
            event Timer#timeout
            guard 'timer_id == 1'
        }

        transition GetData -> WaitingForCmd {
            event WTSCClient#GetData
            action send WTSCClient.SensorData('temp','0','0','batt')
        }

        transition GetStatus -> WaitingForCmd {
            event WTSCClient#GetStatus
            action send WTSCClient.SensorStatus('interval','0','0','0')
        }

        transition GetName -> WaitingForCmd {
            event WTSCClient#GetName
            action send WTSCClient.SensorName('name')
        }

        transition SetName -> WaitingForCmd {
            event WTSCClient#SetName
            action 'strcpy(name, new_name);'
        }

    }

    state ReadSensorValues {

        on entry {
            send LED.light_on('2') // GREEN
            send MSP430Sensor.mesure_temperature()
        }

        on exit {
            send LED.light_off('2')
        }

    }

}
```

Aspect implementing Min/Max

```
pointcut {
    // The system root component in which
    // to store the min/max info
    component WTS <PC_SystemRoot> {}
    // The idle state where we add the c
    // for Get Min and GetMax
    state WaitingForCmd <PC_Idle_State>
    // The sampling state where to add t
    // min and max
    state ReadSensorValues <PC_Sample_Sta
        transition temperature <PC_setDa
    }
    // Detect all places where data is s
    // to add the min/max data
    <PC_message_to_change>
    send WTSClient.SensorData('temp','0')
    // Things which we refer to but are
    device WTSClient {
        message SensorData(temp : ?, min
        receives SensorData
    }
}

advice {
    // add the min/max variables
    component WTS <AD_SystemRoot> {
        property min : Integer
        property max : Integer
    }
    // add the message handler to reset min/max
    state WaitingForCmd <AD_Idle_State> {
        transition ResetMinMax <AD_ResetMinMax> -> WaitingForCmd {
            event WTSClient#ResetMinMax
            action 'min = temp; max = temp;'
        }
    }
    // add the actions to compute the min/max
    state ReadSensorValues <PC_Sample_State> {
        transition temperature <PC_setData> -> ? {
            action {
                'if (temp > max) max = temp;'
                'if (temp < min) min = temp;'
            }
        }
    }
    // add the data to the SensorData messages
    <AD_message_to_change>
    send WTSClient.SensorData('temp','min','max','batt')
    // Things which we refer to but are already in the base
    datatype Integer;
    device WTSClient {
        message SensorData(temp : ?, min : ?, max : ?, batt : ?);
        message ResetMinMax();
        sends ResetMinMax
        receives SensorData
    }
}
```