Facilitating application development for wireless sensor networks

Atis Elsts

University of Latvia

and Institute of Electronics and Computer Science

Riga, Latvia

LATVIJAS UNIVERSITATE ANNO 1919

ELEKTRONIKAS UN DATORZINĀTNU INSTITŪTS

INSTITUTE OF ELECTRONICS AND COMPUTER SCIENCE

About me

- B.Sc. and M.Sc. from University of Latvia (2006 and 2008, respectively)
- PhD. thesis defended 13th December 2013
- Work experience:
  - Institute of Electronics and Computer Science (EDI)
    - Half-time work in science since 2010
    - Full-time work in science since 2011
  - Mikrotik (2005 - 2010)
    - Mikrotik is a Latvian company that makes routers
    - The routers are deployed worldwide (official distributors in 102 countries)
    - I worked on routing protocols (OSPF, BGP etc.) and other networking stuff
    - Adapted & implemented a mesh routing protocol (honorable mention in Latvian IT best student thesis contest in 2008)

LATVIJAS UNIVERSITATE ANNO 1919

ELEKTRONIKAS UN DATORZINĀTNU INSTITŪTS — INSTITUTE OF ELECTRONICS AND COMPUTER SCIENCE

ROUTING THE WORLD MikroTik www.mikrotik.com

The programming of WSN

- WSN software & hardware development is complex because of:
  - **Embedded** device complexity
  - **Networked** device complexity
  - **Energy** constraints
- It may be impossible to shield the developer from implications of low-level choices
- Still, significant classes of WSN **applications** are conceptually simple
  - Sense-and-send
  - Event-detection
  - Data query / reply

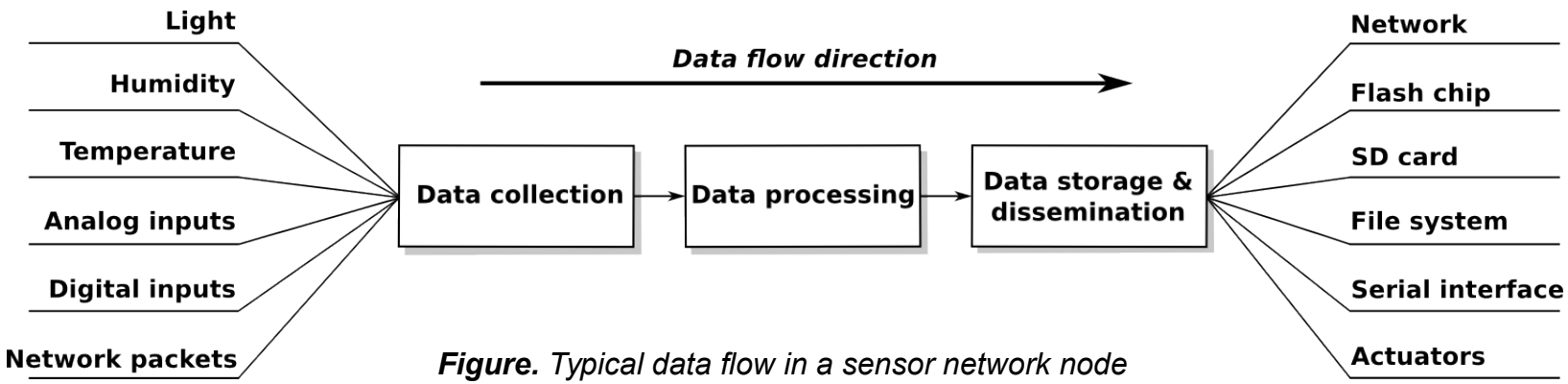


Figure. *Typical data flow in a sensor network node*

The users of WSN

- WSN was envisioned as a tool for domain experts, not computer scientists
- At the moment, WSN app programming for them is hard to impossible
- Financial incentives to empower them:
  - Salary for qualified engineers is the largest cost factor in real-world WSN deployments [Corke 2012]
- Scientific incentives:
  - Make the users contributors, not just observers!

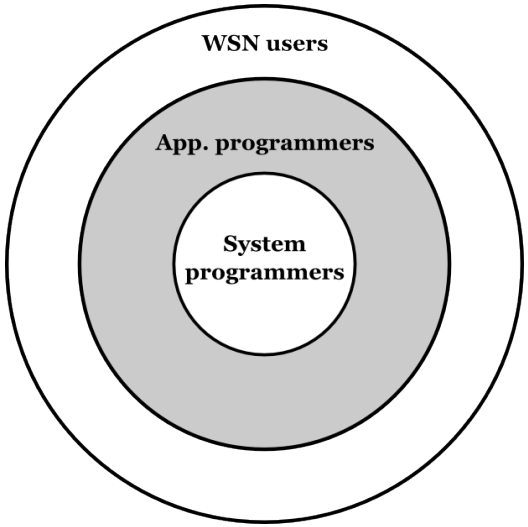


Figure. *Our target audience: the grey area*

Our approach

SEAL – **Se**nsor **A**pplication Development **L**anguage

- A **domain-specific language** (in contrast to a library or an embeded DSL)
- **Minimalistic**
- **Declarative** (in contrast to imperative)
- **Extensible**
- **Bottom-up design**

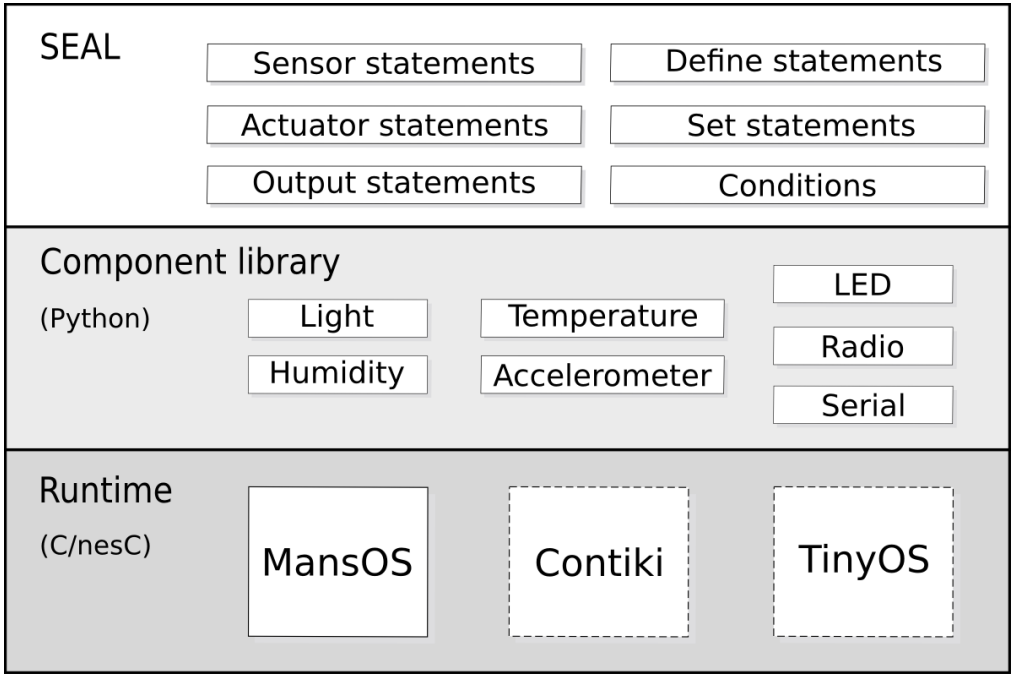


Figure. *Conceptual architecture*

Example application

Light monitoring application

```
read Light, period 10s;
output Radio;

read RemoteLight;

define MinLight min(Light, RemoteLight);

when MinLight < 100lux:
    use Led, on;
end
```

Reads light sensor

Sends light sensor measurements to other nodes

Reads light sensors from direct neighborhood nodes

lesser of...

Whenever any sensor is below 100 lux threshold...

Turn on light

Example application

Light monitoring application

```
read Light, period 10s;
output Radio;

read RemoteLight;
define MinLight
    min(Light, RemoteLight);
when MinLight < 100lux:
    use Led, on;
end
```

Component use cases

Example application

Light monitoring application

```
read Light, period 10s;
output Radio;

read RemoteLight;
define MinLight
    min(Light, RemoteLight);
when MinLight < 100lux:
    use Led, on;
end
```

Sensor use case

Output use case

Sensor use case

Actuator use case

Example application

Light monitoring application

```
read Light, period 10s;
output Radio;

read RemoteLight;
define MinLight
    min(Light, RemoteLight);
when MinLight < 100lux:
    use Led, on;
end
```

Parameters

Example application

Light monitoring application

```
read Light, period 10s;
output Radio;

read RemoteLight;
define MinLight
    min(Light, RemoteLight);
when MinLight < 100lux;
    use Led, on;
end
```

Custom suffixes, expressed in familiar physical units

Example application

Light monitoring application

```
read Light, period 10s;
output Radio;

read RemoteLight;
define MinLight
    min(Light, RemoteLight);
when MinLight < 100lux:
    use Led, on;
end
```

Executed periodically

Executed whenever all active local sensors have been read

Executed on reception of net packets

Conditional expression is reevaluatued whenever the parts may change their values

Executed one time whenever this code subbrach is activated

Elements of SEAL

Element	Description	Implementation in C code

Component use case
 	
Examples:
`use LED, period 1s;`
 	
`read Light;`
 	
`output SDcard;`
	Tells that the named component must be used. The usage is type-dependent and means that the component-specific use function from component library must be called.
	
	Sensors are read.
	Actuators: turned on / off.
	Outputs send out latest sensor readings.
		Calls of component usage functions, scheduled with use of timers (possibly periodic).

Elements of SEAL

Element	Apraksts	Implementation in C code

“`when`” statement
 	
Examples:
`when Light < 100lux:`
`  use LED, on;`
`else:`
`  use LED, off;`
`end`
	Depending on a value of binary condition decides which subbranched of code should be activated (selected for periodic execution).	Event-trigerred recalculation of binary expressions, depending on which some timers may be activated and others deactivated.

Elements of SEAL

Elements	Apraksts	Implementation in C code

“`do`” statement
 	
Examples:
`do, once:`
`  read Light;`
`then, times 2:`
`  read Humidity;`
`then, initialTimeout 10s:`
`  read Temperature;`
`end`
	Determines the order of SEAL code subbranches. May determine other execution parameters as well (such as the number of times, period, delay, and duration).	Timer “chaining”. Each subbranch corresponds to a single timer.

Elements of SEAL

Element	Description	Implementation in C code

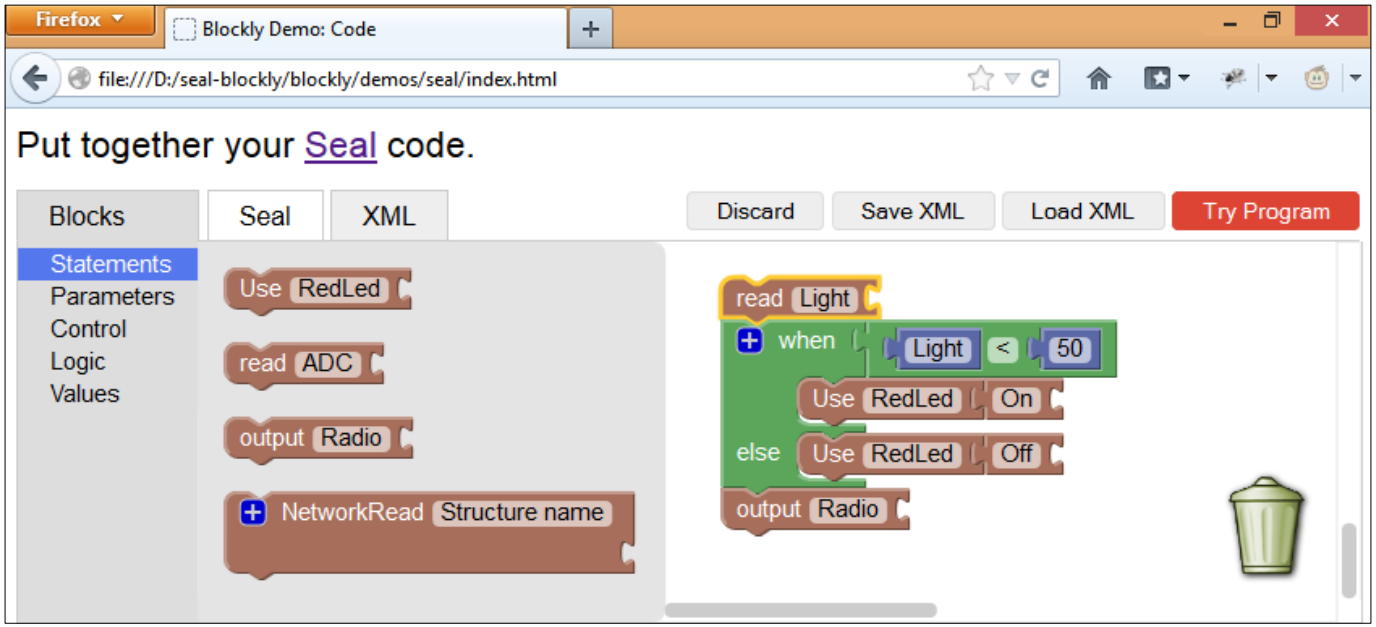
“`define`” statement
 	
Example:
`define MaxTemp`
`  max(Temperature);`
`read MaxTemp;`
	Defines a “virtual” sensor	Function composition between sensor reading functions and predefined data processing functions.

“`set`” statement
 	
Examples:
`set isAlarm True;`
`set count`
`  add(count,1);`
	Initializes or changes a system state variable.	Scalar variable declaration and assignment, respectively.

“`load`” statement
 	
Examples:
`load “extension.py”;`
`load “ext_impl.c”;`
	Imports a Python file (component library extension) or a C file (run-time library extension).	Extension of set of Python modules imported by SEAL compiler.
		Extension of set of C files compiled.

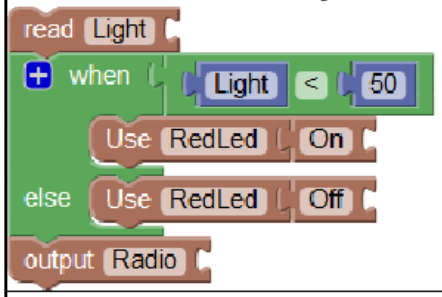
SEAL-Blockly

- *Blockly* – visual programming language & tool from Google
- SEAL-Blockly – *Blockly* adaptation for SEAL
- Pros:
  - Familiar environment (web browser)
  - All possible blocks are predefined and visible in menus
  - Syntax errors are not possible
- Cons:
  - Visual programms can be harder to read (Green, 1992)
  - In some contexts textual programming is more appropriate (Pane & Myers, 1996)



Code comparision

SEAL-Blockly:



SEAL:

```
Read Light;
When Light < 50:
    Use RedLed, On;
Else:
    Use RedLed, Off;
End
Output Radio;
```

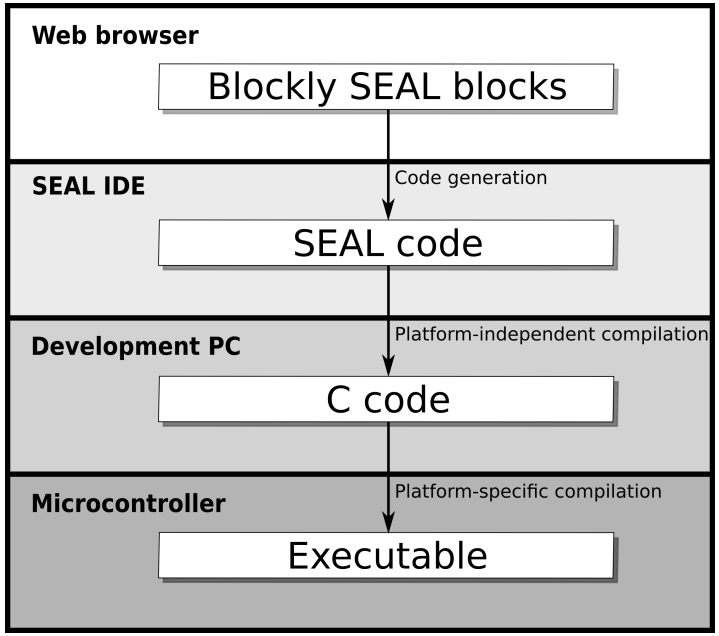
C code:

```
void appMain(void)
{
    alarmInit(&redledBranch1Alarm, redledBranch1Callback, NULL);
    alarmInit(&redledBranch2Alarm, redledBranch2Callback, NULL);

    sealNetRegisterInterest(5, condition1Callback);

    bool newBranchStatus;
    branch0Start();
    newBranchStatus = branch1Evaluate();
    if (newBranchStatus) {

… … and so on
```

~220 lines of code, compiled into 8KB of executable for telosb platform

SEAL-Blockly compared to SEAL, and compared to programming in C

Development workflow



Getting practical

How does SEAL handles common WSN programming idioms?

- Processing (e.g. averaging) data before sending them out
- Waiting on an event (polling or interrupt-based)
- Handling interactive queries
- Implementing state machine logic
- Tweaking system-level stuff for application needs
- Performing actions in strict order & with precise timing

Getting practical

How does SEAL handle common WSN programming idioms?

- Averaging data before sending them out

```
read avg(take(Light, 10));
```

- Waiting on an event (interrupt-based)

```
read InputPin, interruptPort 1, interruptPin 2;
when InputPin = 1: ...
```

- Handling interactive queries

```
const COMMAND 1234;
read RemoteCommand; read RemoteAddress;
when RemoteCommand = COMMAND
      and RemoteAddress = BASE_STATION: ...
```

Getting practical

How does SEAL handle common WSN programming idioms?

- Implementing state-machine logic

```
set AlarmState False;
read Temperature;
when Temperature > 40C: set AlarmState True;
end
```

- Tweaking system-level stuff for application needs

```
output Network, protocol "TDMA", routing
 "CTP", radioChannel 11, radioTxPower 10dBm;
```

- Performing actions in strict order & with precise timing

```
do: use LED, on;
then, once: read Humidity;
then, initialTimeout 1s: use LED, off;
end
```

Language complexity of SEAL

- Smaller grammar & fewer keywords: less time required to learn the language completely
- Fewer special-purpose symbols and operators: code is more intuitive to read

Table. *Language complexity comparison*

	SEAL	TinyScript	TinyDB	snLog	C	nesC
Nonterminal symbols	**36**	31	26	21	70	24 new + 13 changed
Keywords[1]	**19**	15	20	10	31	15 new
Operators	**8**	23	12	16	46	1 new
Non-alphanumeric characters with special meaning	**14**	19	14	13	30	0 new

[1] – *not including aggregation etc. functions*

Application complexity of SEAL

- Metrics: line count, effort according to Halstead's metrics, cyclomatic complexity
- Cyclomatic complexity – a metric for the number of branches in the execution flow
- Higher cyclomatic complexity suggests that code is harder to understand

Table. *Cyclomatic complexity comparison for 8 applications in 5 implementations*

	SEAL	**BASIC**	**TinyScript**	**C**	**nesC**
Empty	**0**	0	0	1	1
Sense & print	**1**	1	2	2	5
Sense & send	**1**	1	2	2	23
SAD	**2**	n/a	n/a	4	n/a
Exercise 1	**1**	1	4	2	2
Exercise 2	**2**	2	4	4	5
Exercise 3	**3**	3	4	4	5
Exercise 4	**1**	3	3	4	6

SEAL resource usage overhead

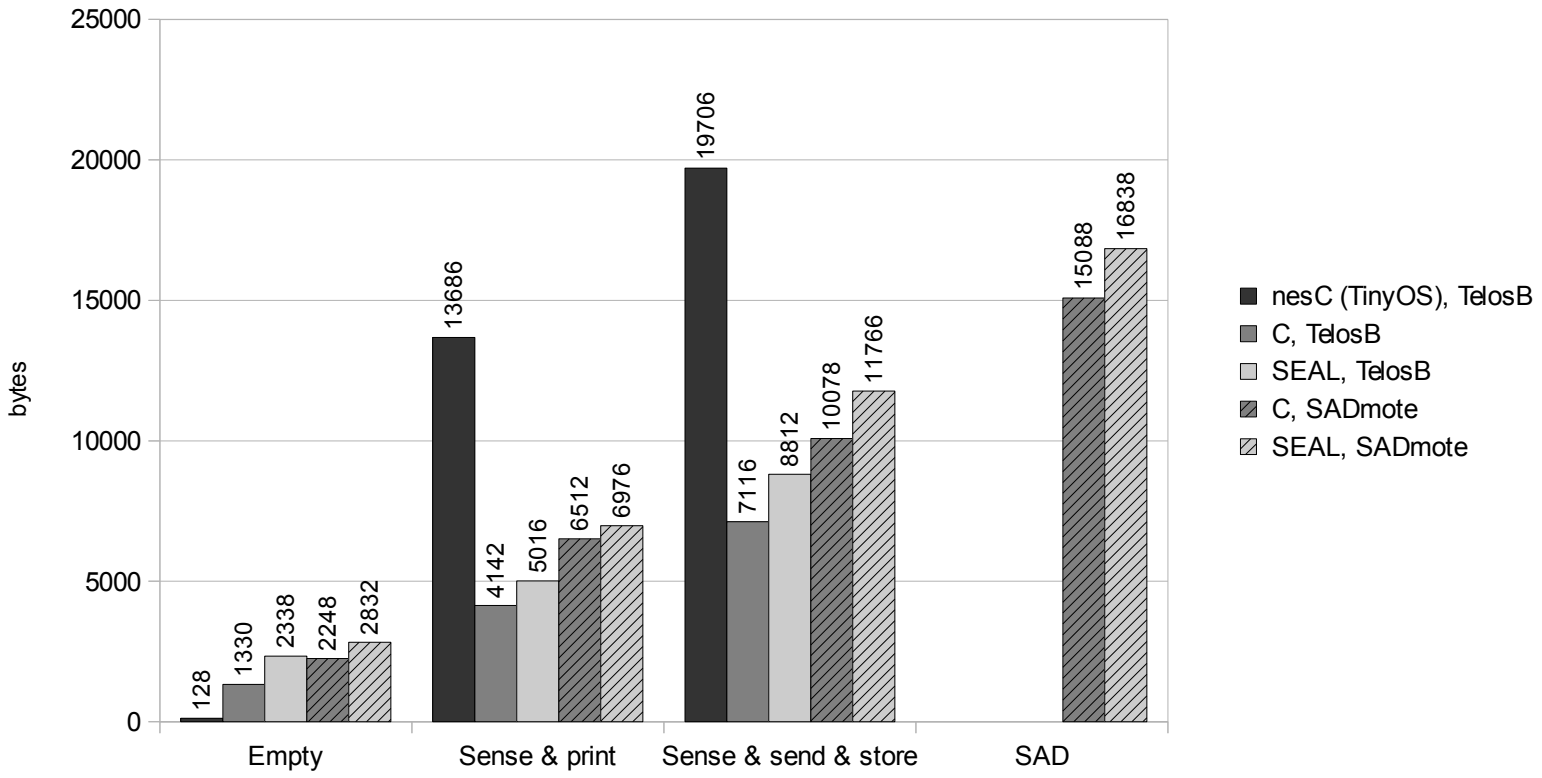


Figure. *SEAL, C, and nesC application binary code usage*

SEAL applications

SEAL applications (not in system level, except the first):

Scenario	Feature demonstrated
Measuring microclimate [Elsts 2012a]	Periodic sampling
Controlling a watering system [Elsts 2012a]	Distributed processing
Collecting light measurements [Selavo 2007]	Interactive query support
Detecting seismic activity [Werner-Allen 2006]	Event-driven processing
Light monitoring in tunnels [Ceriotti 2011]	Intra-node proc. (using C)
Building a backup controller [Zviedris 2010]	Imperative elements
Analog signal processing [Strazdins 2011]	Intra-node processing
Generating a signal [Mednis 2012c]	Sequential execution
Detecting potholes #1 [Mednis 2011]	Intra-node processing
Detecting potholes #2 [Mednis 2011]	Intra-node processing
Detecting vehicle movement [Mednis 2012a]	Optimized intra-node processing
Finding parking space [Kothari 2007]	Distributed processing

Application: environmental monitoring

Environmental monitoring in precision agriculture

Distributed microclimate measurements in an orchard:

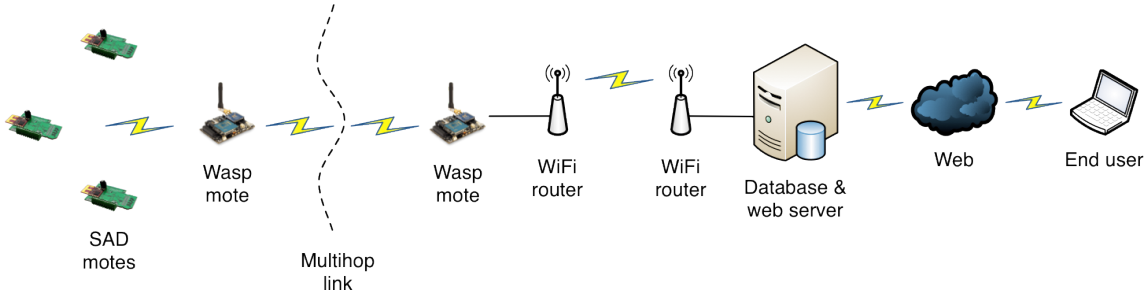
- temperature, humidity, light

The potential users are agroscientists in Latvia State Institute of Fruit-Growing (*http://lvai.lv*)

17 node network in 2011, 19 node sensor network in 2012

>50 000 sensor measurements collected in 2011
>200 000 sensor measurements collected in 2012

In 2012: application logic of the whole network: described with 14 lines of SEAL code!



Conclusion

- A new WSN programming language SEAL proposed
  - Aimed directly at domain experts
  - Motivated by real-world applications
  - Extensible
  - Implementation is feasible on low-power MCU
  - A front-end for visual programming implemented (SEAL-Blockly)
- Future work:
  - More of macroprogramming
  - Formal methods (e.g. for test set generation)
  - Support of other platforms (e.g. TinyOS, Arduino)

Thank you!