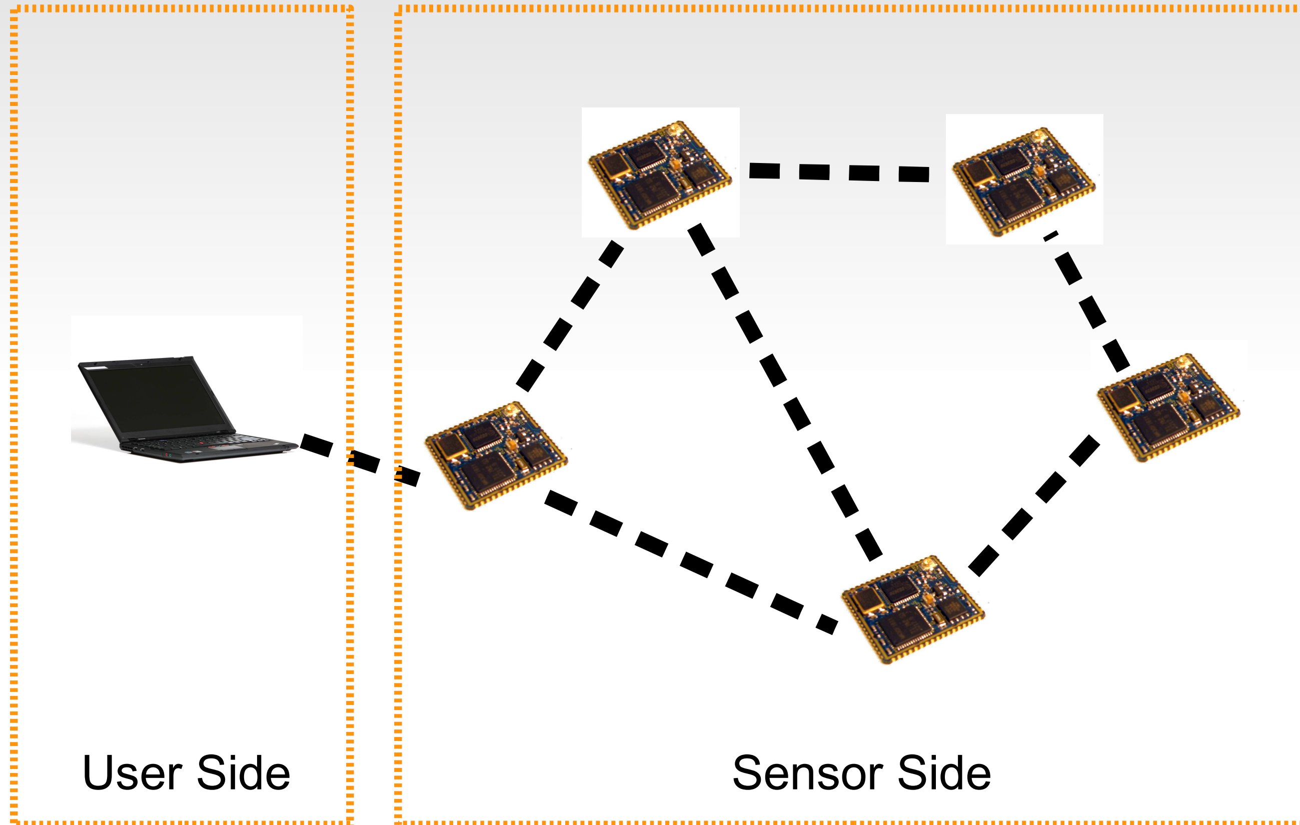
TinyInventor: A Holistic Approach to Sensor Network Application Development

Morten Tranberg Hansen
Dept. of Computer Science
Aarhus University
mth@cs.au.dk

Branislav Kusy
Autonomous Systems Lab
CSIRO ICT Center
Brano.Kusy@csiro.au

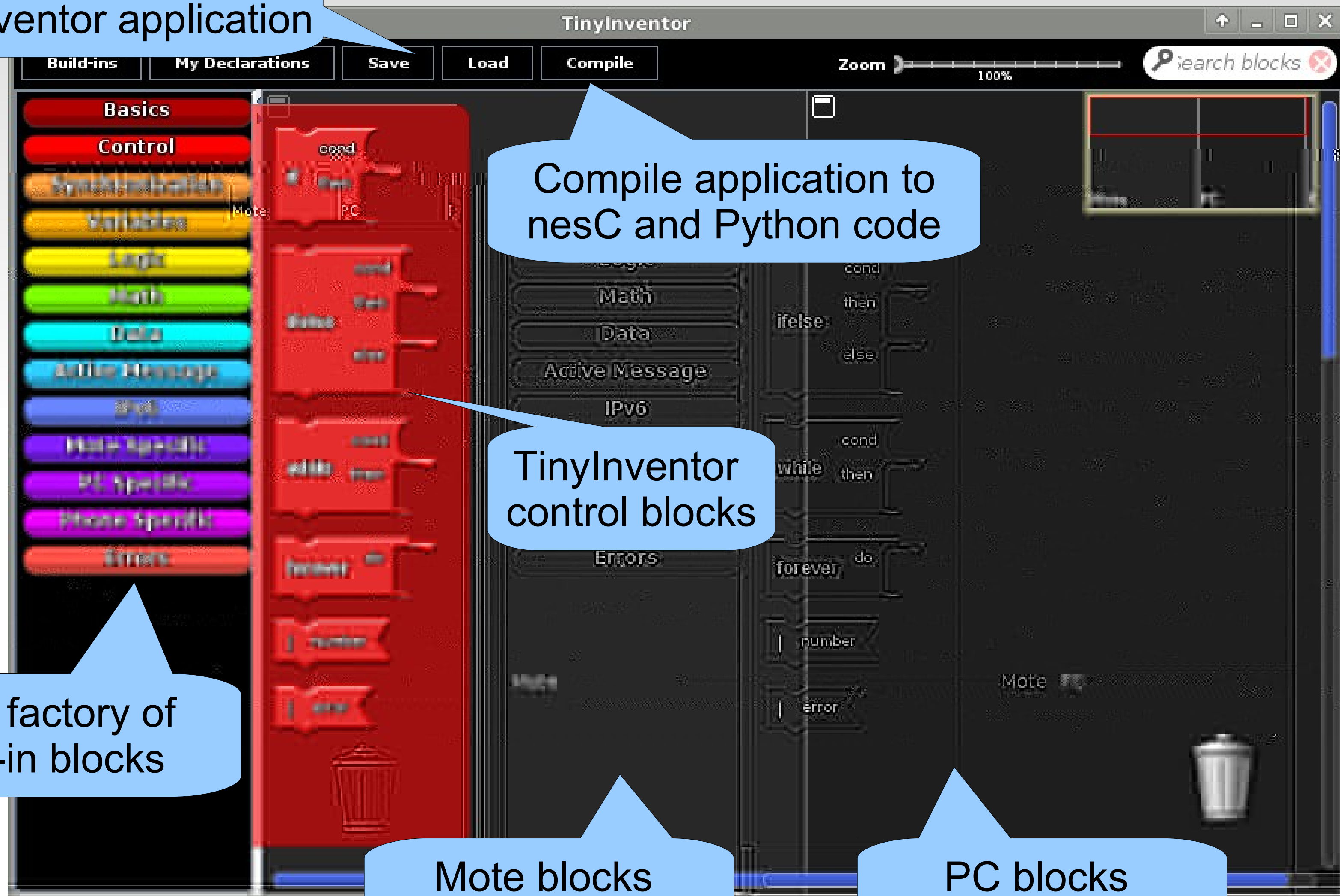
IPv6 and Sensor Networks



TinyInventor

- Ease the task of writing a sensor network application
  - Abstracts implementation details away without really sacrificing efficiency
  - Unifies data access
- Programs are defined as a collection of visual blocks
  - Some running on motes and some on PCs
- Based on IPv6 communication primitives and a multi-threaded environment

TinyInventor Workspace



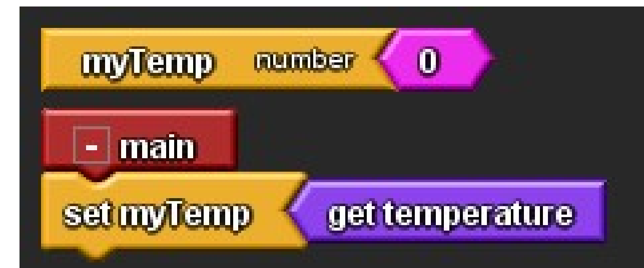
TinyInventor Blocks

- TinyInventor blocks represents data, variables, statements and functions that can be connected in hierarchical way
- TinyInventor provide two groups of blocks
  - Generic blocks representing general language, thread, and communication primitives.
  - Specialized blocks representing specific platform capabilities such as sensors, actuators, and user interfaces.

Block Examples

Generic Block: Sending a UDP packet

Specialized Block
Plotting data on PC



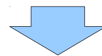
Specialized Block
Getting temperature data on a mote

Block Definitions

- Blocks in the Open Blocks framework are defined in a XML language definition file
- Blocks can be defined without code modifications

```
<BlockGenus name="udp-payload-length" kind="function" initlabel="payload length" color="102 129 255">
  <BlockConnectors>
    <BlockConnector label="" connector-kind="plug" connector-type="number"/>
    <BlockConnector label="UDP message" connector-kind="socket" connector-type="udp-message"/>
  </BlockConnectors>
</BlockGenus>
```

UDP get payload length function XML block definition



UDP get payload length function block

Compilers

- Compile the hierarchical connected blocks to platform specific code
- Compilers represent blocks by components with:
  - Static code: code to be directly part of thefinal program
  - Nested code: code which is to be part of parent components static code
- Compiler components are defined in a XML format similar to the blocks

nesC Compiler

- Compiles a hierarchy of blocks into nesC code
  - TinyInventorC configuration+TinyInventorP module
- nesC components can have five types of static code:
  - Wiring, interface, declarations, initialization, and module code
- Example:
  - In order to set the LEDs in TinyOS one needs to include and wire LedsC (wiring code) and declare the use of the Leds interface (interface code).

Python Compiler

- Compiles a hierarchy of blocks into Python code
  - tinyinventor.py script
- Python components can have 2 types of static code:
  - Declaration and main code
- Example:
  - A main thread needs to be declared (declaration code) before it can be started (main code).

Demo time!

Future Work

- Android phone support
- Explore the possibilities of extending TinyInventor to the Internet of Things were services can be dynamically identified, provided as blocks, connected, and used.