TinyOS

Daniel Nesbitt
daniel.nesbitt@newcastle.ac.uk

Overview

- What is TinyOS?
- TinyOS components
- Components Example
- TinyOS execution model
- Typical Device

Next

- What is TinyOS?
- TinyOS components
- Components Example
- TinyOS execution model
- Typical Device

What is TinyOS

- Operating System designed for network embedded systems:
  - Sensor networks
  - Embedded robotics
- Is written in nesC programming language
  - A dialect of C programming language
- Has an event driven execution model
- Basic unit of nesC code is a component
  - A nesC application consists of one or more components

Next

- What is TinyOS?
- TinyOS Components
- Components Example
- TinyOS execution model
- Typical Device

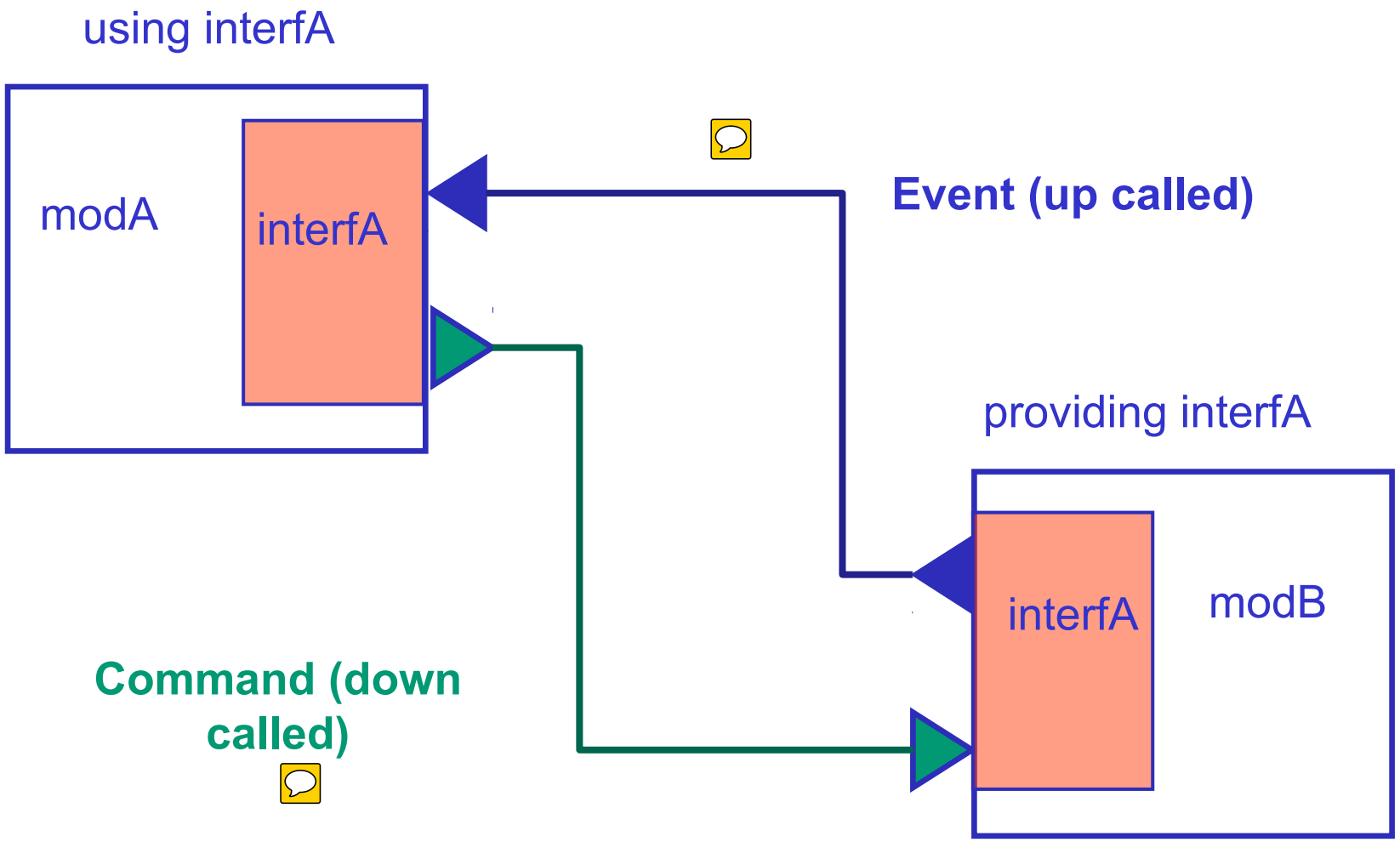
TinyOS Components

- Each component is a file (names must match)
- A component provides and/or uses interfaces
- There are two types of components in nesC: configurations and modules
  - Configuration: is a component that wires (connects) other components together
  - Module: is a component which contains the executable code

TinyOS Components

- components connect with each others by a set of interfaces; A component can:
  - provide a set of interfaces to other components
  - Use a set of interfaces provided by other components
- Interfaces specify commands and events where
  - Commands are down called
  - Events are up called

TinyOS Components



TinyOS Components

- The nesC code uses more explicit data types by declaring their size

	8 bits	16 bits	32 bits	64 bits
Signed	int8_t	int16_t	int32_t	int64_t
Unsigned	uint8_t	uint16_t	uint32_t	uint64_t

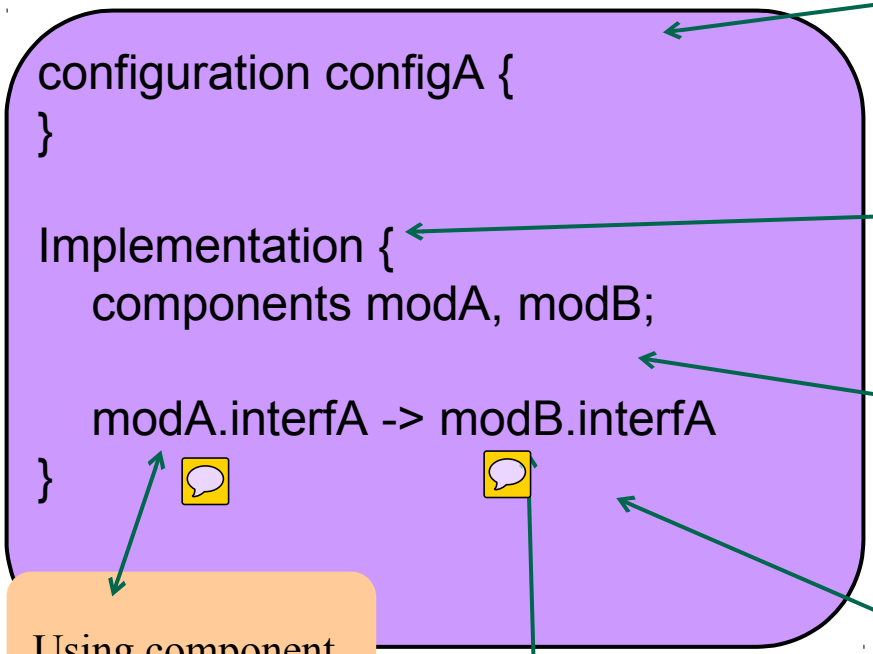
- Standard C types (*int, long* or *char*) can be used, but this might raise cross-platform issues
- Most platforms support floating numbers (*float* almost always, *double* sometimes)
- The *bool* type can be used in nesC

Next

- What is TinyOS?
- TinyOS Components
- Components Example
- TinyOS execution model
- Typical Device

Components Example

- Configuration



The first two lines just to indicate that this is a configuration file (file name has to match)

Actual configuration implemented in the pair of braces following the key word implementation

The components lines specify the set of Components which are referenced (to be wired)

The remainder connects interfaces used by Components to interfaces provided by others

Using component

Providing component

Components Example

- Module:

```
interface interfA {
  event void fired ();
  command void start ();
  command void stop();
}
```

```
module modA {
    uses interface interfA;
}
Implementation {
    uint16_t counter = 0;

    event void interfA.fired () {
        call interfA.start ();
        counter ++;
        …..
        …..
        if (counter >= 10)
            call interfA.stop ();
    }
}
```

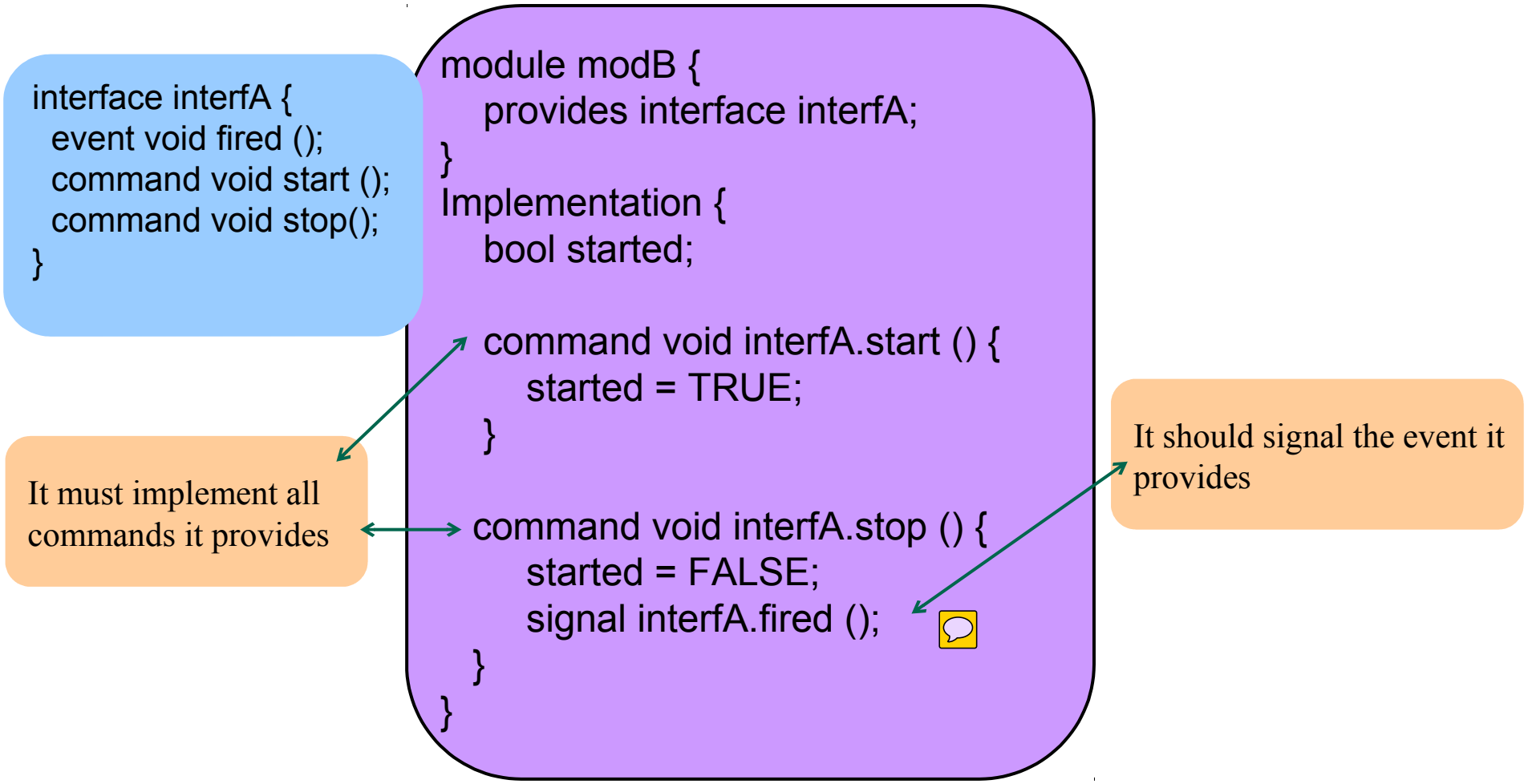
```
configuration configA {
}
Implementation {
    components modA, modB;
    modA.interfA -> modB.interfA
}
```

We declare a variable of type uint16_t

It must implement all events it uses

It can use the commands in interfA which are provided by modB

Components Example



Next

- What is TinyOS?
- TinyOS Components
- Components Example
- TinyOS execution model
- Typical Device

TinyOS execution model

- All the demonstrated commands and events so far run synchronous (concurrent) activities
  - When synchronous code starts running, it does not relinquish the CPU to other synchronous code until it completes
- Synchronous approach does not work well for large computations
- A component needs to be able to split a large computation into smaller parts, which can be executed one at a time
- There are times when a component needs to do something, but it is fine to do it later
- Task: Is a function which can be declared in a component
  - The component can order TinyOS to run the task in later time

TinyOS execution model

- Task example:

```
module modA {
    uses interface interfA;
}
Implementation {
    uint16_t counter = 0;
    task void computeTask () {
        counter ++;
        …..
        …..
        if (counter >= 10)
            call interfA.stop ();
    }
    event void interfA.fired () {
        call interfA.start ();
        post computeTask ();
    }
}
```

Task declaration

Posting the task

TinyOS execution model

- A component can post the task in a command, an event, or a task
  - A task can post itself
  - A task can call commands and signal events
- The post operation returns an *error_t* , whose value is either *SUCCESS* or *FAIL*
- The post operation places the task on an internal task queue
- The TinyOS scheduler runs tasks according to the FIFO scheduling policy

Next

- What is TinyOS?
- TinyOS Components
- Components Example
- TinyOS execution model
- Typical Device

Typical Device

http://www.sownet.nl/download/G301Web.pdf

Summary

- The TinyOS application consists of one or more components, there are two types of components
  - Configurations: Wire interfaces of different components together
  - Modules: Hold the implementation of interfaces
- Different components communicate using interfaces
  - Commands: down call
  - Events: up call
- It is a good practice to split a large computation into smaller parts (tasks)
  - A component can post the task in a command, an event, or a task
  - The TinyOS scheduler runs tasks according to the FIFO scheduling policy

Summary

- To write a TinyOS application:
  - Select interfaces which you want to use
  - Provide interfaces if necessary
  - Wire interfaces to other components
  - Implement events and commands from the interfaces