CS 3468
Lab 2

Jared Wallace

Objectives

1. Understand the design of a sensor application
2. Begin to learn component-based programming languages

Tasks

1. **Architecture and concepts of a sensor application** All of our work will be done in nesC, which is just C with some extra features. A nesC application consists of one or more *Components* assembled, or *wired*, to form an application executable. Components define two scopes: one for their specification which contains the names of their *interfaces*, and a second scope for their implementation. A component *provides* and *uses* interfaces. The provided interfaces are intended to represent the functionality that the component provides to its user in its specification; the used interfaces represent the functionality the component needs to perform its job in its implementation.

   Interfaces are bidirectional: they specify a set of *commands*, which are functions to be implemented by the interface's provider, and a set of events, which are functions to be implemented by the interface's user. For a component to call the commands in an interface, it must implement the events of that interface. A single component may use or provide multiple interfaces and multiple instances of the same interface.

   The set of interfaces which a component provides together with the set of interfaces that a component uses is considered that component's signature. An interface provides the mechanism for a component to use functions in another component.

   - An interface declares a set of functions (*commands* and *events*), but does not implement the functions, nor specify which component provides the interface.

- If a component wants to provide some functions so that other components can use the functions, the component shall provide the interface that declares the functions and implements them as well.
- On the other hand, if a component wants to use a function from another component, it must use the interface provided by the other component.

There are two types of components: *configuration* and *module*.

- A configuration is simply a description of the sub components inside the configuration.
- A configuration assembles sub components together and connects interfaces used by components to interfaces provided by others.
- An application always has a top-level configuration.

A module is the basic unit that provides the functions of the application.

- A module must implement every command of the interfaces it **provides**, and every event of the interfaces it **uses**.
- A configuration has no implementation, but rather sub-components and wirings.
- A module has no sub components or wirings, but rather an implementation.

All of the concepts (in italics), and the hierarchical application architecture, are actually reflected in the programming language (which we will learn next).

2. **Source files of a sensor application** A sensor application has at least three files: a configuration file, a module file and a Makefile. Both the configuration and the module files have the extension ".nc".

   (a) The top component of our example application is called "BlinkAppC". The configuration file is named "BlinkAppC.nc". That file declares that the application needs four components, to wit: MainC, BlinkC, LedsC and TimerC.

```
configuration BlinkAppC {
}
implementation {
  components MainC, BlinkC, LedsC;
  components new TimerMilliC() as TimerC;

  BlinkC.Boot -> MainC.Boot;
  BlinkC.Timer -> TimerC.Timer;
  BlinkC.Leds -> LedsC.Leds;
}
```

   - *MainC* (provided by TinyOS) is the entry of the sensor application. It must be included in every sensor application (just like the main() function must be included in a C program).
   - *BlinkC* (made by you) is the actual component that controls the blinks.
   - *LedsC* (provided by TinyOS) is the component that controls the LEDs.
   - *TimerC* (provided by TinyOS) is the timing component.

Components are connected via interfaces. A connection between components means a component uses an interface provided by another component. In this application, BlinkC uses the Boot interface provided by MainC, as well as the Timer interface provided by TimerC and the Led interface provided by LedsC. (All interfaces are defined by TinyOS)

- The *Boot* interface provides the *booted()* event that notifies connected components when the Mainc component has finished initialization.
- The *Timer* interface provides the *fired()* event that notifies connected components when the timing component completes the countdown to 0.
- The *Leds* interface provides a few commands to turn on or off LEDs on the mote.

Now, read through the file "BlinkAppC.nc" until you understand what components are used in the application, and how they are connected.

(b) The MainC, LedsC and TimerC components are already implemented in system libraries. Only the BlinkC component needs to be programmed in the application. (The implementation is in the file "BlinkC.nc") It declares the BlinkC to be a module using three interfaces.

```
module BlinkC {
  uses interface Timer<TMilli> as Timer;
  uses interface Leds;
  uses interface Boot;
}
```

Recall that a module **must** implement every command of interfaces it provides, and every event of interfaces it uses. Accordingly, BlinkC needs to implement the booted() event (since it uses the Boot interface), and needs to implement the fired() event (as it uses the Timer interface). BlinkC does not need to implement any commands, as it does not provide any interface.

Now, read through the file "BlinkC.nc" until you understand how the module is implemented, how the module calls commands in other components, and what the module does upon an event.

(c) The Makefile always takes the format shown below. (The first line indicates the top component of the application)

```
COMPONENT=BlinkAppC
include $(MAKERULES)
```

Now, compile and run the application and verify that what you see matches the application code.

3. Make a new Blink application so that LEDs blink as a counter Requirements:

- LED **off** means 0, and LED **on** means 1.
- The three LEDs encode a 3-bit number.
- The application generates a counting sequence of 3-bit **gray codes** as 0 1 3 2 6 7 5 4.
- The counting should have a frequency of 1 number per second.

Lab Report

1. Please demonstrate your program to the lab instructor and let him check your code at the end of the current lab project.
2. Your project report is due at the beginning of the next lab.
3. Grading criteria
    - Demonstration, 15 percent
    - Code, 15 percent
    - Report, 70 percent

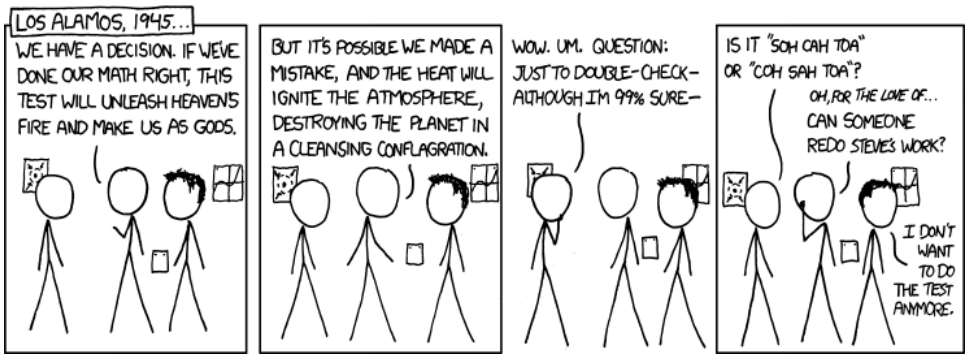
Report instructions

Format:

1. Include your name and ID in the first page
2. Font size of at least 10pt
3. Single spaced
4. Maximum of 5 pages (I will take points off for exceeding this without any good reason)
5. Please submit as PDF online, and turn in a hard copy

Content:

1. Introduction (10 percent of the report grade) Please summarize the task of this lab and what you have learned in the lab
2. Implementation (30 percent of the report grade) Please describe in detail how your made your program. Show what components you used, how they are wired, and through what interfaces.
3. Experiment (30 percent of the report grade) Please describe in detail what can be observed from your program and explain how said observed behavior is a result of your code (and not happy coincidence).



The test didn't (spoiler alert) destroy the world, but the fact that they were even doing those calculations makes theirs the coolest jobs ever.