

Networked Objects

Patrick Dwyer

Spring 2006

Week 3 - February 2nd

Context Awareness

All of our devices will exist in a wider context; places, people, societies, languages. Whether or not we need to take these contexts into consideration depends greatly on what the purpose of our device is. A remote sensor monitoring humidity and wind speed may not need to take as much context into consideration as a device designed to interact with a digital questionnaire.

While the concept of context is variable, we can create a few groups of ideas to be aware of when thinking about the working environment of a device. Deciding which of these our device needs to adapt too, and which of these we can ignore, depends greatly upon the time, money, and interest we have in integrating our device into the larger context in which it functions.

Intersecting the Physical and Virtual

By virtue of being networked devices, our projects exist in both the physical and virtual worlds. What does it mean to link the two? How can we take advantage of this cross-over? Can our devices be aware of both worlds at the same time?

Existing in Real-Time

Our interactive devices are surrounded by the real; people, places, weather, texture, noise, electricity, gravity. How do our devices acclimate themselves to their surroundings, and incorporate that information into their functionality. Does context awareness intersect with calm technology? How *should* context affect our device? How *shouldn't* it? Should our devices exist in four dimensions? More?

Hi, My Name is XPORT

Should devices know about each other? Is this even a different context? How my device acts in the physical and virtual worlds can be impacted and perceived through other devices in the same contexts. How we incorporate the awareness of other devices can determine the effectiveness, reputation, interactivity and interest of our devices.

Back-end Servers in Java / Murmur

One limitation we always work with when using embedded processors is the speed and power of our devices. The standard PIC processor we work with runs at a maximum of 40 megahertz, with about 32 kilobytes of storage space¹; while this type of processor can accomplish quite a bit, it can't do everything. At some point we need to get data off of our embedded processor and into a larger computer, or maybe we want to let a larger computer pre-process data that will later be sent to our device. Either way we need to figure out how to communicate between our device and another computer.

Because we're working with networked devices, we can also work with remote servers. While having a computer sitting next to our device might be convenient, it isn't very aesthetically pleasing, or very scalable. Remote servers have the benefit of being accessible from anywhere on the network by multiple devices at the same time.

¹ PIC 18F252

Previously we would have worked with the Java programming language to create our server side components. While Java is well adapted to the task, it can be difficult to work with, and introduces an extra hurdle in getting any project working. For this class we're going to be trying out a new tool for our server side programming called Murmur.

Murmur?

Yes, Murmur. Murmur is an Integrated Development Environment that we can use to quickly write Java-based servers. Where writing in pure Java can get quite complicated, we can bypass all but the most crucial code when using Murmur; the rest is already taken care of for us. Just as Processing works to make creating graphic applets easier, Murmur works to make server software quick and easy to write.

This is an experimental piece of software, and is under heavy development. If all else fails we can fall back to working in pure Java when need be. Please let me know what does or does not work; the sooner I know, the sooner I can fix it.

<http://www.dwyerdevices.com/software/murmur>

UDP and TCP Servers

Our devices will be communicating primarily over TCP and UDP connections. While we may implement other protocols on top of TCP and UDP, they will form the basis for everything we do. The Murmur software directly supports TCP and UDP servers, as well as sending UDP messages. Examples can be found on the class website, and at the end of the handout.

HTTP Server

We can take advantage of modules within Murmur to enhance how we work with our devices. While our device may communicate with the server through UDP or TCP, we can monitor our server, or have our server process and display data from our device, over HTTP with a standard web-browser. Examples of this can be found on the class website as well as at the end of the handout.

Project Planning

Planning a networked device project can be somewhat daunting; do you start with the hardware or software? Which software? Do I need a server? Engaging in a bit of pre-planning can save quite a bit of time and frustration mid project. Before we step into project planning, remember the most import step of any project: **research**. Always start out with an outline of what you want to do, what you think you might need, problems you might encounter, and other projects that might have things in common.

Each of these planning sections is intended as a rough guide to process and procedure. Your project may need all or none of these steps, and the order in which you approach them is up to you. The detail of your project plans is up to you, but a good axiom to go by is *the better planned, the better executed*.

System Diagrams

A networked device can be a much more complex system than a typical embedded project. The overall system may include a variety of hardware and sensors, communications links, remote or local servers, and remote processes. A System Diagram is a map of all of these parts, showing the path of interaction and information throughout a system. This diagram doesn't need to go into specifics of the exact structure of information, but it should be a detail of what connects where, why, and how. Many future problems can be quickly solved through a detailed system diagram.

The System Diagram is ideally two parts; a drawing and a text. The drawing should be a literal map of how the system pieces connect to one another. This could be boxes and lines, shapes and arrows, or whatever visual device you see fit. The textual component should explain what each part of the system is doing, and how that affects the overall operation of the system. Again, even a simple diagram and a para-

graph of text is good, but the more detailed the System Diagram, the easier your life will be later on in the building and development phase.

Protocol Diagrams

The Protocol Diagram fills in the communications details of a System Diagram, offering insight into the structure and mode of how your project communicates. This diagram should include an outline of what a packet or piece of information being transmitted across the network should look like, what it can or can't contain, and why. The protocol diagram is typically a small drawing detailing where in the project this communication is taking place, and a longer textual component that outlines the structure of the communication.

Flowcharts

Flowcharts are the life-blood of the engineering world. For our purposes the flow chart can be a guide to the procedure and steps we take in our project. We can use flowcharts in a few different aspects of our projects; Design, Development, and Programming.

A design flowchart can be the same thing as our System Diagram, an outline of how our project works, what pieces interact with each other, and how.

The development flow chart is part time line, part to-do list; it links together development tasks and deadlines to form a schedule of what needs to be done in the project. Each piece of the flow chart is dependent upon those before it, giving us insight into what we need to work on, in what order.

Of the various flowcharts we can create, one of the most useful can be the Code flowchart. Whether code flowcharting is useful to you is very much dependent upon your comfortability with programming and the project you are developing; some may find a code flowchart useful, some may already create an ad-hoc code flowchart whenever they write a program, and some may find it more confusing. Code flowcharting can be as simple as writing a set of comments before you code, so you get a feel for the individual tasks your code must complete, or as complex as a diagram with boxes, arrows, and comments.

Readings

- IDEO Interactive Display
 - <http://www.ideo.com/portfolio/re.asp?x=50186>
- Blinkenlights
 - <http://www.blinkenlights.de/gallery/index.en.html>
- Remote Lighting (Vectorial Elevation)
 - <http://alzado.net/>

Assignment

- Response to Readings
- Prepare to build XPort boards

udpExample.mur

```
/* Welcome to Murmur, this new program has
   an empty setup method ready for you to
   use.
*/

void setup() {

    // Fill in your setup method

    add("UDPServer", "udpServer", "udpCallback");
    set("udpServer", "port", 10000);
    start("udpServer");
    info("Starting server...");
}

void udpCallback(ServerEvent se) {

    info("Server message: " + se.getMessage());
}
```

tcpExample.mur

```
/* Welcome to Murmur, this new program has
   an empty setup method ready for you to
   use.
*/

void setup() {

    // Fill in your setup method
    add("TCPServer", "tcpServer", "tcpCallback");
    set("tcpServer", "port", 10000);
    start("tcpServer");
}

void tcpCallback(ServerEvent se) {
    info("Got TCP Message: " + se.getMessage());

    TCPServerEvent tse = (TCPServerEvent)se;
    tse.finish();
}
```

udpHTML.mur

```
/* Welcome to Murmur, this new program has
   an empty setup method ready for you to
   use.
*/

int hitCount = 0;

void setup() {

    // Fill in your setup method

    add("UDPServer", "udpServer", "udpCallback");
    set("udpServer", "port", 10000);
    start("udpServer");

    add("HTTPServer", "httpServer", "httpCallback");
    set("httpServer", "port", 8080);
    start("httpServer");

}

void udpCallback(ServerEvent se) {

    hitCount = hitCount + 1;
}

void httpCallback(ServerEvent se) {

    HTTPServerEvent hse = (HTTPServerEvent)se;
    hse.OK();
    hse.send("UDP Hits:" + hitCount);
    hse.finish();
}
```