An Infrastructure for Electromechanical Appliances on the Internet

by

Joseph Timothy Foley

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Bachelor of Science in Computer Science and Electrical Engineering

and

Master of Engineering in Computer Science and Electrical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1999

Author .....
Department of Electrical Engineering and Computer Science
May 20, 1999

Certified by .....
Sanjay Sarma
Associate Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

An Infrastructure for Electromechanical Appliances on the Internet

by

Joseph Timothy Foley

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 1999, in partial fulfillment of the
requirements for the degrees of
Bachelor of Science in Computer Science and Electrical Engineering
and
Master of Engineering in Computer Science and Electrical Engineering

Abstract

In this thesis, I designed and constructed an intelligent microwave capable of automatically determining the time required to properly cook commercially-packaged foods. This research involved creating a database infrastructure capable of supporting a huge amount of data distributed throughout the internet. It was accomplished through a number of techniques involving careful application of the existing internet technologies: the Domain Naming System and the World Wide Web. A prototype using Radio Frequency ID tags was built and tested with several test foods.

Thesis Supervisor: Sanjay Sarma
Title: Associate Professor

Acknowledgments

I must thank my Master's Thesis Advisor, Sanjay Sarma, for finding me a topic, and forcing me to stay on-target.

Thanks also go to Erik Nygren, for helping me figure out how to make the whole ONS system possible in a very short time period.

Dave Brock gets a Thank You for his assistance in understanding RFID technology and getting the reader working.

Thanks go to David Rodriguera for doing the final editing.

And lastly, and definitely not the least, many thanks go to Elise Westmeyer for her support and initial editing, without which I might not have finished this document.

Contents

List of Figures

Chapter 1

Introduction

When people think of the future, many have a very "Jetson-like" vision with automation everywhere. Robots would assist humans in assorted tasks, both menial and complex. Every item in the home would know its purpose and be able to complete it without human intervention. Moreover, humans would be able to examine and control the resources of their home more efficiently. This is a possible future but far in the future from now.

The problem lies in the enormous amount of information inherent in an environment. Somehow this information needs to be converted into a format that a computer can manipulate easily. The key is in embedding the information into the environment. Previous attempts have used barcodes and optical recognition systems; each has differing social and technical difficulties in getting people to use them [16, 10].

David Brock, Sanjay Sarma, and Sunny Siu created the DISC[1][5], group to address these challenges and implement generalized solutions using existing technology. This thesis is closely intertwined with the group because their goals are the same: to create a framework with which intelligent application products can interface and communicate. I took this goal and applied it to a specific device to create the Automatic Microwave , of which the design and construction is the focus of this thesis.

[1]Distributed Intelligent Systems Center - disc@mit.edu

1.1 The Technology

1.1.1 RF Tags

The proposed solution to the problem of giving every item an unique identifier is Radio Tags[2]. More commonly known as Smart-Cards, these RF devices can take a simple set of instructions and store a small amount of data able to be recalled later. Until recently, the price of the reader/writers and the cards themselves have been prohibitively expensive. Thankfully, this is no longer true.

The current device we are investigating is the MCRF200 microID(tm). The cost of the reader is close to 70 dollars, and the individual card cost was approximately 30 cents at the time of this writing. We predict from market trends that the readers will soon be in the 20-30 dollar range and the cards around 15 cents within the next three years[6]. If this technology takes off, the decrease in price would probably be significantly faster and deeper. Current rumors suggest Motorola has already developed a tag that meets these specifications. Details were not available at this time.

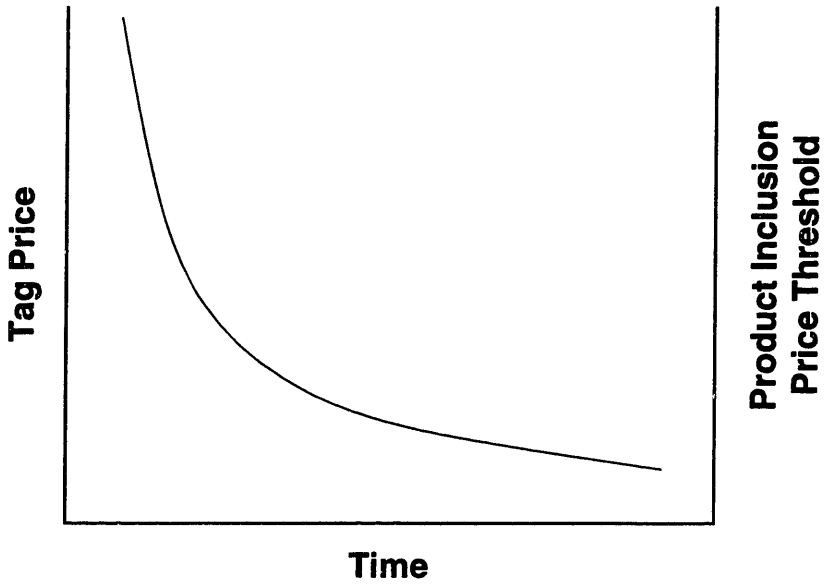


Figure 1-1: Graph of expected RFID tag inclusion into products

[2]I will be using this term interchangeably with the term RF Tag.

1.1.2 Internet Infrastructure

What makes this project a salable design is the explosive growth and development of the Internet. As of 6 years ago, few companies have ever heard of the Internet. Now, it is an integral part of most technology-savvy companies. Groupware and email are critical to keeping the information flowing and people connected. The Internet has become a commodity, much like telephone service, causing people to be less afraid of it and more willing to spend money on it. This attitude means that the connectivity required to connect a company to an individual already exists, at the physical level since wires are being run to fulfill service. [15].

1.1.3 DNS

The biggest challenge for "cataloging the world of food," as some put it, is keeping track of which ID number matches with a given item [10]. Interestingly enough, the Internet has a similar problem: matching human-readable names to the IP numbers, i.e. vice-grips.mit.edu maps to 18.177.0.46. In addition, they have the exact same problem of mapping the numbers back to these names. DNS[3], their solution, is the inspiration for ONS[4], allowing us to use the beneficial properties that DNS exhibits: robustness, scalability, and delegation of responsibility [14].

1.1.4 The Web

Instead of starting from scratch and writing our own protocol to transfer information from client to server and back, we make use of an existing one. The World Wide Web[5] is a simple architecture which can be used to transfer the datafiles using simple HTTP[6]. Once the data is transferred, we can analyze it as per our task. The data's format is PML[7], an extension of XML[8]. It is also well suited as a front end to large

[3]The Domain Naming System
[4]The Object Naming System
[5]Not the greatest thing since sliced bread, but it does make our lives more interesting.
[6]HyperText Transfer Protocol
[7]Product Markup Language
[8]eXtensive Markup Language

databases, using CGI[9]. Using the system described in this thesis, we are able to make use of the web as a growing global information resource with enormous accessibility to large amounts of data[13].

1.2 Synthesis

RF Tag technology has been available, though not cost effective, for 3-4 years. In areas where the cost of the devices has not been an issue, however, RF tags have gone mostly unused. We surmise this is due to a lack of a strong software base such as those supporting UPC bar-codes and other optically-based systems. Unfortunately, UPC numbers are quickly running out and will be fully allocated within 2 years. A new solution must be found. This thesis will attempt to create a modular and extensible design, on both the hardware and software level, that will allow its generalized application to a variety of environments. The hardware design consists of formatting the Tag's data as well as the equipment required to access this data and act upon it. The software design involves creating a sensor-to-database interface and an API[10] for generalized queries.

1.3 Design

This technology appears to have enormous potential, however, it is critical that the initial design be carefully engineered. The "snowball effect" of contemporary development style has an unforgiving attitude towards bad design. The overall system can be seen in Figure 1-2

1.3.1 Hardware

Current RF tag implementation is limited by the range of the readers. A tag must be very close to the reader for them to exchange information. Dr. David Brock, who

[9]Common Gateway Interface
[10]Application Programming Interface

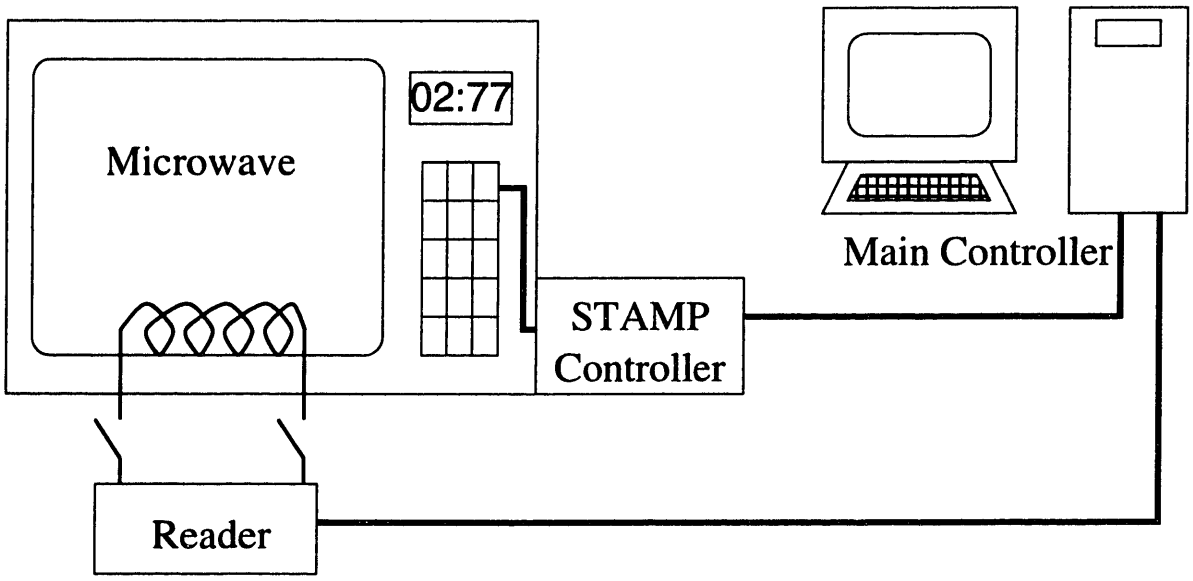


Figure 1-2: Overview diagram of the Automatic Microwave system.

has been working on this problem, has developed a reader with an external antenna which can be configured for a specific distance and application[11]. In addition, the RFID-tags[12] can use CDMA[13] so a single reader can handle many tags.

The tags will have their ID bit-vector broken into a number of fields. The actual formatting of the tags is still under discussion, so I have arbitrarily picked one. This grouping simply divides the tag into three regions: header, owner, and product.

Security and privacy are also of major concern, especially with the technological push to increase the effective range of the readers which is in conflict with privacy. One of our beliefs is that a large portion of security comes with the severe field or close proximity required in order to read the Tags. In addition, we are working to get the Tags to have some private information and programmability only via direct connection. These precautions should provide enough security for all but the most sensitive of contexts.

[11]Within reasonable limits.

[12]Hereforth called simply tags.

[13]Carrier Detect Multiple Access - a method whereby all devices listen while transmitting to make sure that they don't talk while someone else talks.

1.3.2 Software

An appliance making use of controller and ONS software contains a reader networked to a database computer. When the appliance's controller (which is connected to the reader) notices something has been placed in it, the controller queries a "proxy" server to fetch information about the item. The "proxy" server queries the manufacturer's website for data on the product. The appliance's controller then applies the received information to the item, for example micro-waving a frozen dinner for the correct amount of time.

1.4 Implementation

RF Tags can be read from a distance away by a reader with the proper antenna. The newer tags can even be reprogrammed using radio transmissions.

The "RF" in the name indicates that the information is transferred through standard radio-frequency transmissions. Although this is mostly correct, the transmission occurs in a unique manner called "backscatter modulation." An RF carrier (in the 125 kilohertz range) is set up by the reader. The RF tag then uses its own antenna to act as a parasitic inductance to the field by shorting a capacitor across the coil. Through this modulation, data can be passed by a myriad of encodings. Some new higher frequency tags have a true transmitter which answers on a different frequency.

The obvious next question is, "How do we power the RF tag?" The answer lies in inductive coupling. A RF tag contains an internal antenna, a rectifier diode, and a voltage regulator. Together these components generate the minimal current required to power CMOS[14] devices and allow the tag to function [19].

1.5 Prior Art

The "Things that Think" group [23] at the Media Lab is working on a similar line of research. Their **Home of the Future** project much resembles the **House_n**

[14]Complimentary-symmetry Metal-Oxide Semiconductor

project with which the DISC group and this thesis are associated. They also have a controlled microwave, but it uses UPC's and barcodes as the static source of product information. Their group's focus is mainly on point-to-point communication between the devices using expensive ad-hoc implementations. We believe that this approach is not commercially feasible nor scalable because of its high cost and management issues.

According to CNN Article[2], a similar product is in development: a smart trash-can with an RFID tag reader that does inventory control. This device analyzes one's trash and reorders supplies as the containers are thrown out. The application is qualitatively similar to my thesis, but we are unable to compare its effectiveness of the scalability of its design, because the article does not provide details for analysis.

Chapter 2

The DISC Microwave

Modern microwaves are complex to interface with as they all have dedicated microprocessors and analog logic to control the heating cycle. They also have all sorts of "bells and whistle" features which should be taken into account to properly capture the full capability of the device. The antique microwave, a 1987 GE 2000W model, which I acquired uses simple TTL logic and resistive membrane switches. Because of the lack of a microprocessor, its capabilities are very simple and well defined and surprisingly enough, the platform of choice for this particular project.

2.1 Interface

My focus for the design of this interface was simplicity. With approximately a year to design and test the product, I was unable to begin from scratch. Due to limited resources, it was impossible to get a custom controller chip made to match whatever microwave I acquired. I also had no industrial contacts, so getting a microwave with an easily reprogrammable microcontroller would have been difficult. Despite these limitations, I decided that the easiest and most effective course of action would be to retrofit a older microwave.

On the microwave, there were 17 buttons:

0-9 Give a numeric response for setting time or other parameters.

Temp Cook For use with an optional probe - its goal is to bring the internal temperature of the item (usually a large portion of meat) to the set value.

Time Cook Sets a time for the microwave to be running.

Defrost Presets a power level (usually 30%) appropriate for melting frozen foods.

Hold Timer Function is unclear.

Start Executes programmed commands.

Clear Resets state to nothing.

PowerLevel Sets the Power percentage. It does this through a very slow pulse-width modulation(on order of 1Hz).

When I initially designed the interface hardware, I discovered that I did not understand the hardware as well as I thought I did. My original belief was that the switches were copper contacts to a separate substrate with a conductive matrix. After closer examination of the outside, however, I noticed that the switches did not actually move. This discovery led me to believe that my original model was incorrect. Removal of the panel and case revealed the switch hardware which was in fact quite different. Instead of contacts shorting out the matrix to denote a switch-press, the buttons were the matrix. Pushing on the button area caused the conductive film on top of a plastic substrate to contact a large grounded metal backing. The resulting voltage drop causes the TTL logic inside the controller to acknowledge that the button has been pressed.

This discovery led me to believe that I would need to build an equivalent array of resistors and optoisolators in order to "fake" a button being pressed. Unfortunately, early tests proved this to be the wrong approach. What finally worked was connecting the outputs of the BASIC stamp microprocessor(discussed in Chapter 3 Section 3.1) directly to the matrix. The challenge then became connecting the two components together because the substrate is remarkably thin and solder is impossible. Eventually,

I soldered to the terminals of the Printed Circuit Board's connector directly to the controller I/O cables.

The last challenge was connecting the controller hardware (via a cable) to the workstation running the interpretation software. Unfortunately, all existing holes were plugged, and the power cord hole was sealed. My solution was to saw a hole in the the side and run the cable out behind the display.

2.2 Generalization

This retrofit is reasonably general. Most microwaves I have dissected appear to have carbon-film or carbon-contact switches. The method described above will work on both. In the case of the rare capacitive sensor, a little more modification might be necessary. Such a circuit would need to be analyzed and tapped before going through the preprocessing hardware. In the case of a completely integrated sensor system and control, a retrofit is not reasonable.

The eventual goal of this project is to make simple specifications that can be integrated directly into the device. The ability to retrofit is nice for demonstrations but not economically feasible in the long term except for excessively expensive devices. A microwave, currently costing an average $150, can probably handle having a $15 retrofit kit, but anything more costly becomes economically infeasible. The value added by modifying the internal control hardware ahead of time is most advantageous.

Chapter 3

Microcontroller

From the previous chapter, an interface was discovered to actuate the Microwave. My goal was to have a way for a computer to tell a microwave exactly what to do with a minimum amount of data transferred over the line[1]. In addition, to make the system easy to debug, the protocol had to be human-readable and human-analyzable. For those reasons, I chose to use a simple single character mapping to a button press sent over an RS232 serial line.

3.1 BASIC Stamp

The tight time and resource constraints on this project meant I did not have much time to learn a new language, nor did I have time to build a microcontroller from scratch to do what I wanted. Instead I acquired, from looking through a catalog, a BASIC Stamp microcontroller board.

A BASIC Stamp microcontroller is a custom PIC16C57 microcontroller connected to a Reset circuit, voltage regulator, EEPROM, and serial line drivers [20]. What makes it particularly attractive to this application is that Serial I/O operations are easy in its control language PBASIC. In order to send a "Hello World" across the serial port, one just programs: `Serout 16,84,[''Hello World'']`

[1] I later used a slight modification of this protocol to build the 1999 2.70 Scoring system

The STAMP is also in-line programmable, meaning that the system can be reprogrammed inside of a working device. Lastly, PBASIC, based upon the elementary language BASIC[2], is very easy to learn and has a number of constructs that make it very powerful.

In my device, I used a BASIC Stamp-II because it had more I/O ports available for use than the cheaper BASIC Stamp-I. Even so, I still did not have enough I/O ports[3], so I omitted a control for the Hold Timer button, a lesser function control. I could have probably left out all of the accessory buttons as well, but frugality did not appear critical.

3.2 Code

3.2.1 Guidelines to PBASIC for programmers

1. Lines are not terminated with anything other than the standard EOL character for the OS of choice.

2. The input is line dependent; input can not be split across multiple lines.

3. There are no complex data structures; everything is either a bit, nibble, byte or word, or array of one of these.

4. There are no ELSE statements within IF statements[4].

5. Capitalization does not matter.

6. "=" is both an assignment and equality test operator.

7. Incrementing a variable past its highest value resets it to 0.

8. The comment character is ' and lasts to the end of a line.

[2]Beginner's All-purpose Symbolic Instruction Code
[3]17 buttons, 16 ports
[4]Once synthesizes equivalent behavior through the use of an IF and a GOTO command

3.2.2 Quick introduction to PBASIC

PBASIC follows some standard C conventions but resembles BASIC in its syntax. A program consists of two parts: variable declaration and the program body. The standard mathematical and logical operators exist and exhibit expected behavior. Flow control is accomplished using some specialized functions, but most work can be done using the familiar `IF (condition) THEN (label)`. This statement causes control to pass to a line marked with a label, denoted by a word at the beginning of a line, followed by a colon: "label:". Lastly, there are input and output routines that read pin state and set them to `HIGH` or `LOW`.

`INPUT (pin)` Makes a pin an input.

`OUTPUT (pin)` Makes a pin an output.

`GOTO (label)` Goes to the line with the label.

`HIGH (pin)` Sets an output pin high(5V).

`LOW (pin)` Sets an output pin low(0V).

`(constant) CON (value)` Symbolically defines a constant with a specific value for use throughout the program (as a parameter).

`DEBUG ''(String)'',CR` place a String in addition to a newline in the debugger window.

`SERIN (pin),(baudmode),[(variable)]` reads some amount of serial data from a pin. The baudmode is a complex timing constant best described in the manual[20].

3.2.3 Analysis of Controller Code

The controller code (Appendix A.4.1), owes its shortness in length to the simplicity of the protocol and the power of PBASIC. It begins with a variable declaration, and is followed by a constant declaration (to give pin number symbolic names for ease of

use and reconfiguration). Next, we get into the main control loop at line 18 where we store a single byte value off the serial line at pin 16 (which has a special serial driver circuit). Once we have a character to examine, we complete some comparisons, with the assumption that the encoding is in ASCII, so we can determine first if the value is a number, letter, or something else entirely. At each IF test, if the character matches the class, we pass it on to a post-processing operation that converts the character into the appropriate pin number. Finally, we pulse the pin at LOW, and begin the loop again.

3.3 Expansion Capabilities

As was evident with this initial design, we have a hard limitation of 16 buttons. This is a problem since many microwaves have more buttons than 16 buttons[5]. The answer to this problem is to use a MUX: a device that routes a single input to one of a number of outputs according to an address. We can place one of these on 5 pins to get the equivalent output capability(1 pin to set the value, and 4 pins to set the address since most large MUXes only have 16 output pins). We are able to do this, however, since we only need to push a single button at a time[6]. If this were not true, we would have a hard limit of the number of I/O pins.

[5]Since the goal is integration, this is actually a moot point for everything except for similar retrofits

[6]In fact, if you push multiple buttons simultaneously, the microwave ignores you.

Chapter 4

Tag Reader

4.1 RFID Basics

4.1.1 Intro

Passive Radio Frequency Identification (RFID) systems are composed of three components: an interrogator (reader), a passive tag, and a host computer. The tag is composed of an antenna coil and a silicon chip that includes basic modulation circuitry and non-volatile memory. The tag is powered by a single radio frequency wave transmitted by the reader, called the *carrier*. When the carrier passes through an antenna coil, a small AC voltage is generated across the coil. This AC is passed through a rectifier to supply the necessary power to energize the tag. The information stored in the tag is transmitted back to the reader using backscattering shown in Figure 4-1 [19]. The following list will provide clarification on the terms used:

Reader A microcontrolled unit with a output coil wound for a particular carrier frequency. It consists of peak detector hardware, comparators, and firmware for doing backscatter communications.

Tag The RFID device is a small memory chip with on-board rectification and RF transceiver hardware. It also usually comes with an antenna made of either a finely wound coil or, for the tiniest packages, printed on the board. Some tags

also require a tuning capacitor for low frequencies.

Carrier An RF sine wave generated by the reader used to transmit energy to the tag and to receive data. Common frequencies include 125kHz and 13.56MHz. Higher frequency systems exist, but they generally use different technologies for communication.

Modulation Controlled fluctuations in the amplitude of the carrier, used to transmit data back from the tag to the reader.

Generally a reading process looks like this:

1. The reader continuously generates an RF carrier and watches for modulation which would indicate the presence of a tag.

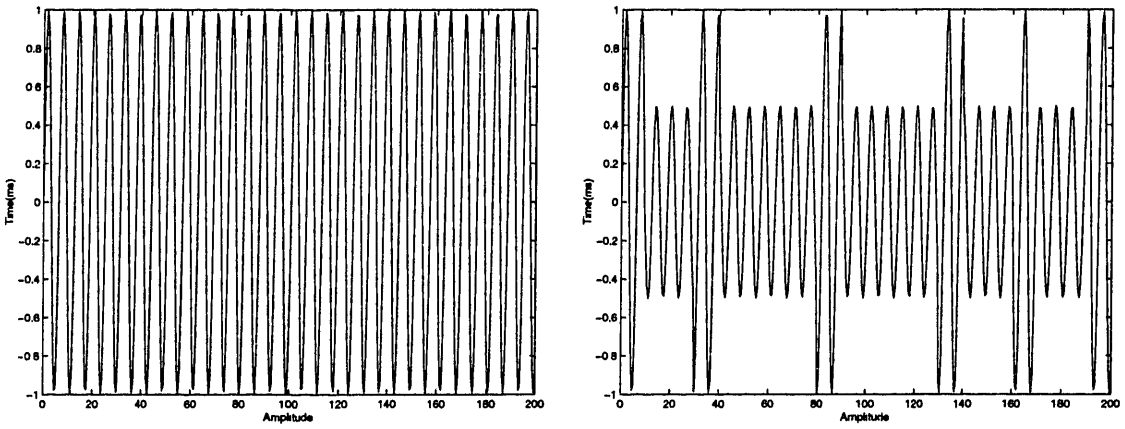
2. A tag enters the RF field. Once it has reached stable power levels, it divides the carrier frequency to generate an internal clock. This clock is used to begin clocking data to an output shunt transistor which is connected across the coil inputs.

3. Shunting the coil causes a momentary fluctuation (dampening) of the carrier wave, shown as a slight change in amplitude of the carrier.

4. The reader detects the amplitude-modulated data and post-processes the stream of bits according to its firmware which decodes the encoding and data modulation.

4.2 Tag ID composition

Now that we have the data from the tag, what does it mean? Until we assign it a meaning, it is simply a bunch of numbers. In the case of the tag used in the prototype, MCRF250, we have either 96 or 128 bits worth of meaningless numbers. In order to give them meaning other than a very simple one-to-one match, we need to group the bits together into fields. Doing so means that we can speed up lookups using routing



(a) Unmodulated Carrier

(b) Carrier Backscatter Modulated

Figure 4-1: Time-Domain Graph of Backscatter Transmission

protocols and bit-masking techniques available from the TCP/IP[1] scheme. The other reason we wish to do this (as will be explained more in depth in the ONS chapter 5.4) is to delegate responsibility in much the same way that the SNMP numbering scheme works [12, 21].

SNMP[2] has the same problem of a large number of bits[3] needing to map to single entities in a changing environment. [18]. The way that it is able to make this problem manageable is that the bits are aligned along the length of a large tree structure, such that each integer is a branch decision along the tree. Different branches are then delegated to different vendors to do as they see fit with that section of the tree, adding or removing integers. Our initial bit-division scheme, shown in the topmost diagram of Figure 4-2, divides the bits quite simply and compares them to an IP address.

The first 8 bits(0-7) of a RFID is the *header field* which gives some idea of the type of data and some critical parameters. The first 4 bits are version information to allow coexistence between different versions. The last 4 are mapping information, which encode where splits should occur if different separations need be specified. Both fields have valid number starting at 1, as 0 has a special meaning discussed later.

[1]Transport Control Protocol/Internet Protocol

[2]Simple Network Management

[3]though they are at least grouped together into integers to begin with

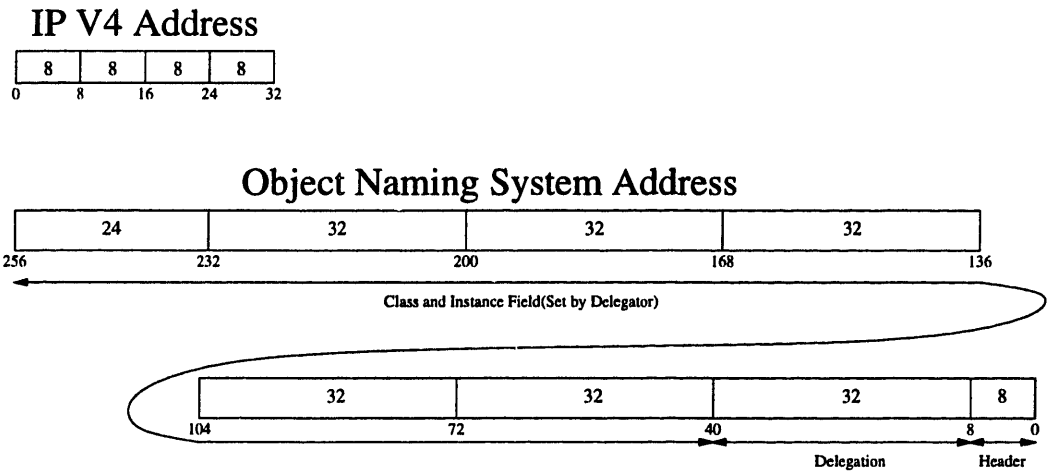


Figure 4-2: Diagram of bit usage in IP addressing versus the RFID tags

The next field of bits (8-40), called the *delegation field*, denotes who is responsible for the information on this tag. It can be a government registry or a product manufacturer. This is what ONS[4] uses to distribute its load across the Internet.

The rest of the bits contain both the class and instance of the object and are called the *data field*. The class is essentially a product model number, and the instance is its serial number. The division between these two is not initially set; instead, it is the responsibility of the delegator to decide where the proper division is to occur[5].

Also, as in the example of an IP-based network, there need to be certain special numbers to denote exceptional cases. When the entire header is all zeros, it denotes the rest of the tag is a UPC code followed by a serial number. This is done for backwards compatibility on products that have already been packaged. When the entire tag is all zeros, it denotes an error condition with the reader or tag. An ID of all 1's is currently undefined.

[4]Object Naming System

[5]This is to allow for companies which make consumables (small types of products, huge numbers of serial numbers) to use the same system as Aeronautics companies (large types of products, small amounts).

4.3 Door Switch

Microwaves work by exciting the molecules inside of food. The water molecules in food are particularly good at being an antenna for the very short wavelengths. Unfortunately, so are pieces of metal such as the tag and its associated reader antenna. A good microwave will heat the pieces of metal until they melt, and the currents induced in the wire have a good chance of killing sensitive electronics connected to them. How are we to successfully and safely place the tag and reader inside such a dangerous environment without killing them?

Thankfully, for this particular application, we do not care if the tag self-destruct. Once the microwave turns on, and the food is being cooked, the tag is no longer of use. We will likely discard it with the packaging when the process is complete[6].

The reader antenna is a tricky matter though still solvable. We can take advantage of another fact: the entire inside of the microwave cavity is lined with metal to act as shielding. This shielding keeps the microwaves from getting out by grounding it. Unfortunately, this fact is the reason the antenna must be inside the cavity – the shielding blocks out the 125kHz signal that our reader uses so we would be unable to put it externally. If we connect the antenna to the shielding, it is protected from the severe environment. However, the shielding will also eat the signal of the reader while connected. We need the shielding to be decoupled when we wish to read the tag.

Once again, behavioral patterns come to the rescue. The door is open when an item is placed in the microwave, and closed when the microwave is operating. A safety interlock roller-switch will keep the microwave from operating while the door is open. We can use this same feature and mechanism to decouple the antenna from the reader when the microwave is running and recouple it when the door is opened. This feature allows us to operate even under these hostile conditions.

[6]Though this makes the intelligent trash-can mentioned in Section 1.5,[2] less useful

Chapter 5

Main Controller

As shown in Figures 1-2 and 5-1, the design for the post-processing step is not very complex. Our prototype made use of a Dual Pentium II-Xeon machine running NT. This hardware was selected because it was easily available in my work environment and not because of a processing power requirement. The Windows/NT operating system was selected because there is a working Java Communications library associated with sending and receiving data to and from the serial ports. Java was selected for its portability. Current research trends show that there is active development of Java controllers in small form factors.

5.1 Design

As with all good software design, the pieces of the main controller program needed to be modular enough to facilitate quick debugging and interchangeability. These design qualities were a critical concern because both industrial and home applications quick turnaround. The modular design shown in Figure 5-1 (which goes hand-in-hand with Java's object-oriented programming framework[8, 9, 3]) means that errors can be compartmentalized and dealt with by the module that knows how. Information about error conditions is passed to higher abstractions through an specified exceptions, so isolation of errors tends to be simple. For an instance of interchangeability, it would be relatively easy to replace the ReadReader module with code for interpreting UPC

symbols or bit patterns of a UPS reader.

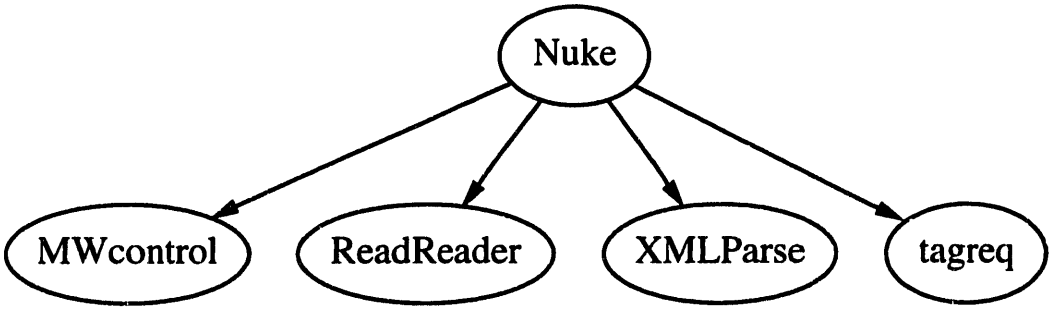


Figure 5-1: A Modular Dependency diagram of the Workstation Software

MWcontrol Java interface to the microwave controller protocol.

Nuke Main part of the control program. It invokes the methods of the other modules to accomplish its goal.

ReadReader Java interface to the RFID Tag reader.

XMLParse Takes an PML file and parses it into an internal datastructure designed for this application.

tagreq Takes an ID and converts it into a URL for acquiring the PML specification.

General operation goes as follows:

1. An item is placed in the microwave. The reader reads it, and passes the information to `ReadReader`.
2. `ReadReader` signals `Nuke` to become active. It does so, and passes the ID to `tagreq`.
3. `tagreq` uses the method mentioned in Sections 5.2 and 5.4 to generate the URL pointing to the ONS object.
4. `Nuke` takes the URL, and passes it to `XMLParse` which fetches it and parses it
5. `Nuke` queries `XMLParse` for the relevant information for this device.

6. `Nuke` uses the information to control the device using `MWControl`.

Full code listings are in Appendix A, Section A.1.

5.2 ID Analysis

Once the reader has acquired the ID, what should it do with it? If we examine the code specifically designed for this purpose, it breaks the ID into the above-mentioned fields, and composes an URL according to the rules. This URL is passed to the next module, the PML parser.

5.3 PML

PML stands for Product Markup Language. It attempts to describe the functionality and form of any physical device or object. Such a format is the key to controlling arbitrary devices. This problem is much similar to the visualization problem mentioned in Gershon's paper[7].

Visualization is the link between data/information and the human mind. Gershon discusses the complexity of taking raw data and turning it into a format that is easily understandable while also having high information density[1]. Our requirements for the controller data format are identical.

Thankfully, a language template already exists to assist with this task: XML, a more generalized version of the Web's HTML[2]. XML only gives a generalized specification for a data format, unfortunately, not an actual application, so we still need a custom parser. Because PML is built on top of XML, we will use the terms interchangeably except where explicitly specified.

After evaluating several XML parser API's[3], I decided to use the IBM implementation[11], version 2.0.6, because their parser has both a validating and non-validating mode with

[1]For instance, the amount of data a modem can push through a phone line is many times faster than a voice conversation, but there are few people who can understand the hiss without assistance.

[2]Hypertext Markup Language

[3]Application Programming Interface

excellent documentation. Using their API, I was able to make a custom XML parser to parse PML and verify its format using a DTD[4]. An analysis of the DTD is beyond the scope of this paper; refer to [11] for a good overview.

5.3.1 Dissection of a PML document

A PML document is a tree structure, as shown in Figure 5-2. (The full sample file used in the prototype is in Appendix A, Section A.3.2

```
<food-package-home-heated>
<Device type="MicrowaveOven">
<Text cycle="1"> Caution, Food may
become very hot in center </Text>
<Power cycle="1">100</Power>
<Time cycle="1">240</Time>
<Wait cycle="1">30</Wait>
</Device>
</food-package-home-heated>
```

Figure 5-2: Sample PML food description

Regions in XML are denoted by using a structure known as a "tag[5]." To lessen confusion, we will continue using the term "region". A region specifies a node in the tree datastructure with its leaves symbolizing all the regions it encloses. The notation for a region begins with a string label "<label>" and a different matching label at end: "</label>."

This document is for an object of the class "food-package-home-heated."[6] The class contains a number of nodes (in this case, one) or Device types. The Device type node has the nodes "Text," "Power," "Time," and "Wait," which specify the actual parameters for operating the device on this item.

[4]Document Type Definition
[5]for further confusion
[6]better known as a "Healthy Choice"(tm) TV dinner

5.4 ONS

Some of the most worrisome issues about handling any form of data is scalability and speed. In the case of the identifying objects, the problem comes with the enormous name-space that all need unique identifiers.

The Internet has an almost identical problem. How do you map the hard-to-remember IP addresses to the easier-to-remember names like "vice-grips.mit.edu." DNS is the answer.

DNS/BIND (Domain Name Service/Berkeley Internet Name Domain) is an IP address mapping system that maintains this mapping through a hierarchical system of multiple servers. The main way it works is as follows: A machine wanting to find out the IP address of the machine "vice-grips.mit.edu"

1. It finds out the ".edu" servers by contacting the DNS root servers.
2. The ".edu" servers refer them to the ".mit.edu" servers.
3. The ".mit.edu" server returns the IP address "18.177.1.46"[14].

Even more interesting is that the ".mit.edu" servers can make subdomains and refer queries about the subdomains to other servers. For instance, the "swiss.ai.mit.edu" hostname is pointed at the ".ai.mit.edu" servers by the ".mit.edu" servers[17].

We can apply this to the 128 bit RF-id tags by first separating out our initial header field and simply breaking the rest up into 4 byte hexadecimal encoded chunks separated by dots. Lastly, we add on the ID type, the default being "rfid:"

`(objidpart4)...(objidpart1).(header).rfid.objid.net`

`802A69.6E74656C.6C696765.6E746F62.6A.rfid.objid.net`

The "magic" occurs in the separations. A machine seeking the webpage with the data on the object:

1. The machine goes to the root name servers and returns ".net" server.
2. The ".net" server returns the ".objid.net" server (which happens to be vice-grips.mit.edu).

3. The ".objid.net" server returns the ".A.rfid.objid.net" server which is one of the DISC servers[7].

4. This server returns the address for the "E746F62.6A.rfid.objid.net" server which would be under control by the manufacturer.

5. We then have IP address of the webserver that has information on that tag.

The IP number is stored using either a standard DNS server or a DENTS [4] database back-end nameserver[8]. The machine would then query that webserver in a standardized format:

`GET http://(server-ip)/RFID.cgi?ID=802A696E74656C6C6967656E746F626A`

and if that fails:

`GET http://(server-ip)/RFID/802A69/6E74656C/6C696765/6E746F62/6A`

The amazing thing is that this is done seamlessly and invisibly. Pointing a web-browser at that URL with the (server-ip) replaced by:

`802A69.6E74656C.6C696765.6E746F62.6A.rfid.objid.net`

gets the correct XML file. We call this service (layered on top of DNS) ONS for Object Naming System.

Standardization needs to occur between the manufacturer/website and the specific type of object. In the end, we want an address that basically encodes $< instance >< model/type >< manufacturer >$ in some number of fields/bits that are fixed in size, but this has been discussed previously in other DISC literature.

We can even deal with UPC[9]'s by creating a separate domain: `(UPC).(header).upc.objid.net`. Assuming that the correct datafiles have been given to ONS servers, the UPC act just like RF tag identification numbers.

Another useful feature of ONS is that a record can also contain a TXT record, embedding a string. This string could be a special URL like "http://vice-grips.mit.edu/objid/tv-

[7]or whomever this service is farmed out to

[8]DENTS is an alternative DNS server system. It is mostly a DNS front-end to a database system, that allows for more generalized DNS functions, i.e. it could be configured to answer any *.E746F626.A.rfid.objid.net DNS query with 18.78.1.38.

[9]Universal Product Code

dinner/foo.bar," which might be a faster way to get at the PML code for that particular item.

To make the system even more robust, ONS has a local cache that it checks first. The expiration date for the information in the cache is all specified by the initial server that gave the data, which minimizes load on the servers and on the network. In addition, load is automatically balanced among the servers (if multiple ones are listed), thereby making this a scalable solution.

The use of DNS as a massive, structured database is not new. Similar applications in other regions already exist, for example is `TPC.INT` or MIT project Athena's hesiod service.

5.4.1 TPC

`TPC.INT`[22] stands for The Phone Company, a collection of FAX servers one may use to fax by e-mail to many locations around the world. They way it works is rather elegant:

1. One composes an email message containing the text of the message you wish to fax with a few special control characters for formatting.

2. One sends it to a very carefully specified email address: `remote-printer.Arlington_Hewes/Room_403@0.1.5.2.8.6.9.5.1.4.1.tpc.int`

This email address is parsed as follows: the left-hand part identifies the kind of access (remote-printer) along with the identity of the recipient (Arlington_Hewes/Room_403)[10]. This information shows up on an automatically generated header page. The right hand side is the phone number for the fax destination, reversed (and with the country code), such that our example destination would be the phone number 1 (415) 968-2510.

TPC is a volunteer fax service, distributed around the country who is willing to run a TPC server on their computers at home or work. Once a computer registers its

[10]Because many mailers cannot handle email addresses with spaces in them, spaces are replaced with underscores(_)

phone number (through a web form), a phone table is consulted to determine what area codes are local to that number. These numbers are delegated to that IP address internally and mapped inside of the DNS configuration file. Since the phone number is reversed, doing a DNS resolve will return a server that is in the `1.tpc.int` zone, the ones serving the United States. The next level down, `5.1.4.1.tpc.int` returns an IP address, which is where the email is sent and spooled by the server at that machine. The extra numbers are for completeness/expansion and for international phone support.

5.4.2 Hesiod

The previously-mentioned problem of a massive distributed database also troubled the designers of MIT's Project Athena. They needed a method for telling Athena workstations where arbitrary services, such as email, home directories, and printing were located. Such a service is known as a Directory Service. They also needed it to be distributed since the key to Athena's scalability is its distribution factor. In the end, they decided to modify the DNS service to accept a new query, "HESIOD"[11]. Doing a resolve of a specially formatted hostname and query of class HESIOD would return a TXT record specifying information on that service.

For instance, in order to find out the "/etc/passwd" entry for me, one would type at a unix prompt:

```
nslookup
set class=hesiod
set type=txt
foley.passwd.ns.athena.mit.edu
<\overview>
```

Unfortunately, this meant that Athena had to deploy a special set of name servers specifically devoted to answering hesiod queries. It was eventually realized that a special class was not required for query resolution since hesiod lived in its own domain.

[11]Standing for Hesiod Service

As a result, the service could be integrated into the standard name service instead of a special class. This alteration improved the input of service information into software because no modification of name service libraries would need to occur. As an added benefit, only minor configuration changes would be necessary to the old hesiod servers.

There are competitor directory services available such as LDAP (Lightweight Directory Access Protocol) and YellowPages (a Sun Microsystems implementation). However, Athena continues to use the Hesiod service for its strengths at the time of this writing.

5.5 Proxying

Proxying is defined as having someone else do work, in place of you. In the case of DNS, proxying is native to the system. A DNS client can usually[12] ask the another client to do a lookup for it. In the case of the house automation system, we need a similar framework in order to get the most functionality out of a single connection to the Internet.

Once again, we make use of the SNMP framework to make an interesting feature possible. In the case of a zero'th level proxy, the tag is self-referential, meaning that it contains the data required to control the products with no outside intervention. The conversion to data becomes application-specific at this point. My proposal is to do a conversion using the ASCII encoding[13], then run the results through the XML Parser to look for data appropriate for our appliance. This process becomes much more feasible and useful when tag size increases to 1 KByte, which is expected to occur soon.

In the general case, however, the tag is just a pointer to a data structure somewhere living on an webserver which would be considered to be a first-level proxy.

[12]Assuming that the DNS client is not brain dead.

[13]Though Java of course prefers the 2-byte-to-1-character conversion of Unicode characters, unfortunately this means that twice the bits are required in order to encode anything.

5.6 Security

Most aspects of this thesis have been explored, at least superficially, by other companies, but almost none have considered the issue of security. Such lack of concern presents a grave danger to society where information is a malleable and powerful tool. Certain aspects of the information environment enable the access, collection, collation, and analysis of large databases containing personal information from public and private sources. There are significant social implication of such collection, manipulation and analyses, known as profiling[1]. The idea of an unmarked van pulling up in front of your home and determining its contents is a violation of privacy as is the knowledge of companies monitoring movement and action. The technologies we are applying make this process easier to accomplish, however, with foresight and planning, these issues can be dealt with while maintaining the positive aspects of the design.

Another question draws upon the network traffic that enters and leaves the house. Someone could surreptitiously listen in on the packets and match the products to queries being made. One safeguard to prevent this uses a "static" caching model, whereby the most commonly used information is stored on a CD or DVD local to the controller computer. For a subscription fee, a new updated one is sent out each month to users of the system. This minimizes traffic and does not expose much information about purchasing models or other private information.

The fact that the ID's are pointers to dynamic information allows for another security technique. To prevent some individuals from gaining too much information about an object, a webserver can be configured to only return specific directories or pages to certain parties. In the case of a particularly smart webserver, only parts of the original PML file might be returned, instead of the whole document.

We can also apply some simple cryptographic models to the tags to enhance privacy. Since many of the tags are reprogrammable, the data can be stored in an encrypted format on the tag. The main controller computer would take the output of `ReadReader` and apply a decryption process before passing it onto the `tagreq`

converter.

Chapter 6

Conclusion

6.1 Status of Prototype

The prototype has been built and performs according to specification. Unfortunately, because of the scarcity of associated hardware, the acquisition of tags and readers has been expensive and slow. This lack of tags has made me hesitant to destroy any of them by leaving them in the microwave, and permanent installation of the reader into the device has been delayed until replacement parts arrive.

Also, because of problems in getting Network Solutions Inc. to delegate the domain `objid.net` correctly, the ONS system currently works only if a machine has `vice-grips.mit.edu` set as one of its primary name servers. This problem will be resolved in the near future.

6.2 Analysis

The scalability of this system meets its requirements. The ONS system is capable of handling a large number of requests because it is able to hand off most of the work to the delegated servers.

The main limitation that has not been addressed is the number of devices that can be connected to a single main controller. In the current prototype, the connection limit of devices is two; each system requires two serial ports, and most computers

have a maximum of 4 serial ports. Changing to another type of serial interface such as USB[1] solves this problem as the devices can be hooked to a "hub". USB, however, incurs the need for a slightly more complicated controller.

6.3 Suggestions

For future projects, direct access to the microcontroller system inside the device is suggested instead of doing a retrofit operation. In terms of labor and cost effectiveness, a retrofit is definitely neither efficient nor cheap.

6.4 Future Plans

The technology and infrastructure of this project is easily expanded to other devices. Current projects include an array of reader coils which are polled to locate an object, which can subsequently be identified by the ONS system. Tracking movement of items is critical in large volume operations such as shipping or any inventory system.

For instance, a businessperson wants to know the location of a package during shipment. Current systems have the location granularity of knowing which shipping office it was last seen at. Given a slightly more intelligent system and a minimal set of hardware (RF-ID reader, Transceiver with home office, and GPS), the granularity of position can be made even smaller. At that point, an empowered executive can find out within the limits of latency and GPS the exact location of their package. This improves planning efficiency and a more accurate ETA.

Another exciting aspect of this system is the decrease in lost packages. Once the package enters the system, it leaves a trail each time it passes near a reader. This accountability means that the level of service guarantee can be upgraded at a minimum of cost. Such systems can be retrofitted with a little effort around a current accounting system.

[1] Universal Serial Bus

An increase in accountability allows a paradigm shift, as packages can be envisioned as though they were packets of data being routed through a network. This abstraction means that standard network routing protocols and addressing can be applied to the system to make it more efficient, and in cases of office shutdown, far more reliable.

For instance, Federal Express currently routes all its packages through a central office, then sends them off to their final destination. This makes management easier and increases speed of delivery. Processing package delivery in this fashion comes at a high cost, however, and this is reflected in the price to the customer. Even worse, if a catastrophic event such as a blizzard takes that home office out of service, the entire system grinds to a halt.

In contrast, if there were a hierarchical structure is utilized with main "routers" of packages that would successively pass them to local offices, the system can be likened to an IP network. If one of the nodes goes down, traffic is simply rerouted automatically around it. In some cases, if links are close enough, the packets can jump from local office to local office until reaching their final destination. This example can be extended further to a truck being given instructions to drive to a particular destination at which point another truck by the same carrier takes the packages.

6.5 Summary

We have designed and implemented the basis for an Internet device infrastructure. Furthermore, we have applied this infrastructure successfully to a particular product, the microwave, to create a device capable of making appropriate decisions using dynamic data and able to act upon this data. This system and technique hold much promise for the future of automation of household products.

Appendix A

Program Listings

A.1 Java Main Controller Code

A.1.1 Nuke.java

```
/* $Id: Nuke.java,v 1.6 1999/05/19 18:59:59 foley Exp $
 * Main Calling routine for the MicroWave controller
 *   and cooking procedure
 *
 *   by Joe Foley<foleymit.edu>
 *   Copywrite MIT DISC, 1999 <discmit.edu>
 */

import java.io.*;

public class Nuke {
    private static boolean DEBUG = true;
    public static final int WAIT_EXTRA = 2; // to allow clock delay
    public static void main(String[] argv) {
        // parse arguments
        MWcontrol control = new MWcontrol();
        String id;
        if(argv.length >0) {
            // Testing purposes
```

```
    id = argv[0];                                                    20
}
else {
    /// Daemon mode, just wait for reader to do something
    // SerialDaemon daemon = new SerialDaemon(COM2);
    /// Event driven (soon)

    ReadReader reader = new ReadReader();
    while(reader.currentID == null) {
        try{Thread.sleep(100);} catch(Exception e) {}
    }                                                                30
    id = reader.currentID;
    System.out.println("Got a TAG: "+id);
}
String url;
XMLParse ourparser = null;

int i = 1;
for(i=1;i<=tagreq.MAXconvert;i++) {
    //      System.err.println("URL try:"+i);
    try {                                                            40
        url = tagreq.converttoURL(id,i);
        ourparser = new XMLParse(url);
        break;
    }
    catch(Exception e) { }
}
if(i>tagreq.MAXconvert) {
    return;
}
//          System.out.println("Power: "+ourparser.getInstruction(1,"Power"));   50
i=1;
while(ourparser.getInstruction(i,"Time")!=null
      ||ourparser.getInstruction(i,"Text")!=null) {
    System.out.println("Cycle "+i+" Starting");
    try { System.out.println("Text: "+ourparser.getInstruction(i,"Text")); }
```

```
        catch(Exception e) { e.printStackTrace(); }

        int power = new Integer(ourparser.getInstruction(i,"Power")).intValue();
        control.PowerLevel(power);
        int timecook = new Integer(ourparser.getInstruction(i,"Time")).intValue();
        control.Time(timecook);
        System.out.println("MicroWave set to "+power+"% Power, "+timecook+" Seconds.. Starting.");
        // brrr.. I really want a check before this happens - foley
        control.Start();
        // Wait till cook cycle is done
        System.out.println("Waiting for MicroWave to finish.");
        try {Thread.sleep(timecook*1000);}
        catch(Exception e) { e.printStackTrace(); }

        // pausing is optional
        try {
            int waittime = new Integer(ourparser.getInstruction(i,"Wait")).intValue();
            System.out.println("Waiting specified "+waittime+" Seconds.");
            waittime+=WAIT_EXTRA;
            Thread.sleep(waittime*1000); }
        catch(Exception e) { e.printStackTrace(); }
        System.out.println("Cycle "+i+" Complete");
        i++;
    }
    System.out.println("Food is Cooked. Share and Enjoy!");
}

private static void dprint(String text) {
    if(DEBUG) System.out.print(text);
}
private static void dprintln(String text) {
    if(DEBUG) System.out.println(text);
}
private static void dprinterr(String text) {
    if(DEBUG) System.err.print(text);
}
```

```
    private static void dprinterrln(String text) {
        if(DEBUG) System.err.println(text);
    }
}
```

A.1.2 MWcontrol.java

```
/* $Id: MWcontrol.java,v 1.8 1999/05/19 18:59:57 foley Exp $
 * MWcontrol - microwave serial control, for interfacing with a BASIC Stamp
 *   and cooking procedure
 *
 *   by Joe Foley<foleymit.edu>
 *   Copywrite MIT DISC, 1999 <discmit.edu>
 */

import java.io.*;
import java.util.*;                                                    10
import javax.comm.*;

public class MWcontrol {
    private Enumeration portList;
    private CommPortIdentifier portId;
    private SerialPort serialPort;
    private OutputStream outputStream;

    public static void main(String[] args) {
        MWcontrol control = new MWcontrol();                           20
        /* Cheapo Client - mostly for debugging */
        for(int x=0; x<args.length;x++) {
            for(int i=0; i<args[x].length();i++) {
                System.out.println("Arg: "+args[x].charAt(i));
                control.sendchar(args[x].charAt(i));
            }
        }
    }

  /*  public MWcontrol() { } /* test stub */                           30

    public MWcontrol () { this(9600); }
    public MWcontrol (int baud) { this("COM2",baud); }
    public MWcontrol (String port) { this(port,9600); }
```

```
 public MWcontrol (String port, int baud) {
    portList = CommPortIdentifier.getPortIdentifiers();

    while (portList.hasMoreElements()) {
       portId = (CommPortIdentifier) portList.nextElement();
       if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
          if (portId.getName().equals(port)) {
             try {
                serialPort = (SerialPort)
                   portId.open("MWcontrol", 2000);
             } catch (PortInUseException e) {}
             try {
                outputStream = serialPort.getOutputStream();
             } catch (IOException e) {}
             try {
                serialPort.setSerialPortParams(baud,
                                               SerialPort.DATABITS_8,
                                               SerialPort.STOPBITS_1,
                                               SerialPort.PARITY_NONE);
             } catch (UnsupportedCommOperationException e) {}

          }
       }
    }
    // as part of the startup process, clear the display
    Clear();
 }

/* API */
 public boolean NumPress(int num) {
    char ourchar;
    ourchar = (char)(num + '0');
    return(sendchar(ourchar));
 }
 public boolean NumPress(String strnum) {
    int i = 0;
```

```
        for(i=0;i<(strnum.length()-1);i++) {
            sendchar(strnum.charAt(i));
        }
        // get return code on last char
        return(sendchar(strnum.charAt(i)));
    }
    public boolean Start() {
        return(sendchar('s'));
    }
    public boolean Time() {                                          80
        return(sendchar('t'));
    }
    public boolean Time(int time) {
        Time();
        return(NumPress(timeconvert(time)));
    }
    public boolean Clear() {
        return(sendchar('c'));
    }
    public boolean TempCook() {                                      90
        return(sendchar('m'));
    }
    public boolean Defrost() {
        return(sendchar('d'));
    }
    public boolean PowerLevel(int power) {
        // actually, only wants first digit if not 100%
        if(power != 100) {
            power = power / 10;
            PowerLevel();                                            100
            return(NumPress(power));
        }
        return(true);
    }

    public boolean PowerLevel() {
```

```
        return(sendchar('1'));
    }

                                                                    110
  /* serial code */
    private boolean sendchar(char ch) {
        System.out.println("Sending to MicroWave: "+new Character(ch));

        /*
          try {
          Thread.sleep(1500);  // takes about 1.5 sec between keypresses
          outputStream.write(ch);
          } catch (IOException e) {return(false);}
          catch(Exception e) { e.printStackTrace();  return(false) ;}      120
        */
        return(true);
        /* later we'll put some checking instead of delay
           since the stamp can actually respond */
    }

    public static String timeconvert(int time) {
        // convert integer seconds to MIN SEC SEC format
        // We use a string, because it's easier to index down
        String retval = "0";                                             130
        int min=0,sec=0;
        min = time / 60;  sec = time % 60;
        if(sec < 10) {
            retval = Integer.toString(min)+"0"+Integer.toString(sec);
        }
        else {
            retval = Integer.toString(min)+Integer.toString(sec);
        }
        return(retval);
    }                                                                    140

}
```

A.1.3 ReadReader.java

```
/* $Id: ReadReader.java,v 1.4 1999/05/19 19:00:01 foley Exp $
 * XML Parser for Microwave Controller
 *   by Joe Foley<foleymit.edu>
 *   Based upon SimpleRead.java by Sun Microsystems Corporation
 *   Copywrite MIT DISC, 1999 <discmit.edu>
 */
import java.io.*;
import java.util.*;
import javax.comm.*;

public class ReadReader implements Runnable, SerialPortEventListener {
    static CommPortIdentifier portId;
    static Enumeration portList;

    InputStream inputStream;
    BufferedReader inReader;
    SerialPort serialPort;
    Thread readThread;
    Vector TagsSeen;
    String[] Tag;
    int TagIndex;
    public String currentID;

    public final static int TAGEND = 7;
    public final static int MODEL = 1, ENC = 2, UNK = 3, SIZE = 4, POLARITY = 5, ID = 6;
    public static void main(String[] args) {
        ReadReader reader = new ReadReader();
        while(true) {};

    }

    public ReadReader() {
        Vector TagsSeen = new Vector();
        Tag = new String[TAGEND];
```

```
        portList = CommPortIdentifier.getPortIdentifiers();

        while (portList.hasMoreElements()) {
            portId = (CommPortIdentifier) portList.nextElement();
            if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
                if (portId.getName().equals("COM1")) {
                //if (portId.getName().equals("/dev/term/a")) {

                    try {
                        serialPort = (SerialPort) portId.open("RFID Tag Reader", 2000);
                    } catch (PortInUseException e) {}
                    try {
                        inputStream = serialPort.getInputStream();
                    } catch (IOException e) {}
                    try {
                        serialPort.addEventListener(this);
                    } catch (TooManyListenersException e) {}
                    serialPort.notifyOnDataAvailable(true);
                    try {
                        serialPort.setSerialPortParams(9600,
                                                       SerialPort.DATABITS_8,
                                                       SerialPort.STOPBITS_1,
                                                       SerialPort.PARITY_NONE);
                    } catch (UnsupportedCommOperationException e) {}
                    readThread = new Thread(this);
                    readThread.start();
                }
            }
        }
    }

    public void run() { // the real work begins
        while(true) {
            try {
                //System.out.println("Tag:"+Tag[ID]);
```

```
            currentID=Tag[ID];
            /*              int x = TagSearch(Tag[ID]);
            if(x<0) { // doesn't exist
                TagsSeen.addElement(new Visiblity(Tag[ID]));
            }
            else {
                TimesSeen.elementAt(x).sightings+=2;
            }
            TagExpire();
            */                                                          80
            Thread.sleep(500);
        } catch (InterruptedException e) {}
    }
}

/*
  public int TagSearch(String TagID) {
    for(int i=0;i<TagsSeen.size();i++) {
        if(((Visibility)TagsSeen.elementAt(i).Tag).equals(TagID)) {return(i);}
    }                                                                   90
    return(-1);
}
public void TagExpire() {
    for(int i=0;i<TimesSeen.size();i++) {
        TimesSeen.elementAt(i)-;
    }
    for(int i=0;i<TimesSeen.size();i++) {
        if(TimesSeen.elementAt(i)<=0) {
            TagsSeen.removeElementAt(i);
            TimesSeen.removeElementAt(i);                               100
        }
    }
}
*/

public void serialEvent(SerialPortEvent event) {
```

```
        switch(event.getEventType()) {
        case SerialPortEvent.BI:
        case SerialPortEvent.OE:
        case SerialPortEvent.FE:                                          110
        case SerialPortEvent.PE:
        case SerialPortEvent.CD:
        case SerialPortEvent.CTS:
        case SerialPortEvent.DSR:
        case SerialPortEvent.RI:
        case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
            break;
        case SerialPortEvent.DATA_AVAILABLE:
            try {
                // The reader uses Newlines.  DOS does not like newlines...  120
                inReader = new BufferedReader(new InputStreamReader(inputStream,"UTF8"));
                while(inReader.ready()) {
                    String strbuf = inReader.readLine();

                    // if we have a line with a Tab in it, it's the first one
                    if(strbuf.indexOf("\t")==0) { TagIndex=0; }
                    Tag[TagIndex] = strbuf;
                    // System.out.println(TagIndex+":"+strbuf);
                    TagIndex++;
                    if(TagIndex >= TAGEND) {TagIndex = 0;}                130

                }
            } catch (IOException e) {}
            break;
        }
    }
}
```

A.1.4 XMLParse.java

```
/* $Id: XMLParse.java,v 1.8 1999/05/19 19:00:05 foley Exp $
 * XML Parser for Microwave Controller
 *   by Joe Foley<foleymit.edu>
 *   Based upon DOMFilter.java by IBM Corporation
 *   Copywrite MIT DISC, 1999 <discmit.edu>
 *
 *   Usage:
 *    construct with the URI
 *    call getInstruction(i)
 *                                                                    10
 *   Bugs/Notes:
 *    1. need to make check to start count at lowest number
 *    2. The maximum might be useful too
 */

import java.io.*;
import java.lang.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import com.ibm.xml.parsers.*;                                          20
import org.w3c.dom.*;


public class XMLParse {

    public static String[] nodetypes;
    private static boolean DEBUG = false;

    private NodeList instructions;
    private int instructionCount;                                      30

    private String elementName = "Device";
    private String DeviceType = "MicrowaveOven";
    private String attributeName = "Type";
```

```
private String parserClass = "com.ibm.xml.parsers.NonValidatingDOMParser";

public static void main(String[] argv) {
    if (argv.length == 0) {
        printUsage();
        System.exit(1);                                                    40
    }
    XMLParse ourparser;
    try {ourparser= new XMLParse(argv[0]);
    dprintln("Power1: "+ourparser.getInstruction(1,"Power"));
    dprintln("Power2: "+ourparser.getInstruction(2,"Power"));
    }
    catch(Exception e) {}
}
public XMLParse(String uri) throws FileNotFoundException {
    dprintln("XMLParse(): start");                                         50
    initnodetypes();
    dprintln("Init done");

    try {
        dprintln("Creating a "+parserClass+" Parser");
        Parser parser = ParserFactory.makeParser(parserClass);
        dprintln("Parsing URI:"+uri);
        parser.parse(uri);

        // The next line is only for DOM Parsers                           60
        dprintln("Converting to Document");
        Document doc = ((NonValidatingDOMParser) parser).getDocument();

        // get elements that match
        dprintln("Extracting elements with Tag:"+elementName);
        NodeList elements = doc.getElementsByTagName(elementName);
        if (elements == null) { // sucko!
            System.err.println("No elements");
            return;
        }                                                                  70
```

```
    // now go over the NodeList looking for ours
    dprintln("Got the elements, enumerate looking for attribute "+attributeName);
    int elementCount = elements.getLength();
    Element devicedata = null;
FOUNDRECORD:
    for (int i = 0; i < elementCount; i++) {
        Element     element    = (Element)elements.item(i);
        dprint("\nelement("+i+"): "+element.getNodeName());                    80
        NamedNodeMap attributes = element.getAttributes();
        if (attributes != null) {
            int attributeCount = attributes.getLength();
            for (int j = 0; j < attributeCount; j++) {
                Attr attribute = (Attr)attributes.item(j);
                //dprint("("+j+")");
                //dprint(attribute.getNodeName());
                //dprint("=\"");
                //dprint(normalize(attribute.getNodeValue()));
                String attribbuf = normalize(attribute.getNodeValue());        90
                if(attribbuf.equals(DeviceType)) {
                    dprint("  GOT IT!");
                    devicedata = element;
                    break FOUNDRECORD;
                }
                //dprintln("'");
            }
        }
    }
    //              if (attributes.getNamedItem(attributeName) != null) {      100
    //print(element,element.getAttributes());
    //              }

    dprintln("");
    if(devicedata == null) {dprintln("Could not find "+DeviceType); return;}
    instructions = devicedata.getChildNodes();
```

```
        instructionCount = instructions.getLength();
        if(instructions == null) { dprintln("No instructions for "+DeviceType); return;}
    }
    catch (SAXException se) {                                                    110
        se.printStackTrace();
    }
    catch (DOMException de) {
        de.printStackTrace();
    }
    catch (FileNotFoundException fnfe) {
        throw(fnfe);
    }
    catch (IOException ioe) {
        ioe.printStackTrace();                                                   120
    }
    catch (Exception e) {
        e.printStackTrace();
    }

}

public String getInstruction(int cycle,String field) {
    dprintln("getInstruction("+instructionCount+"): start");
    if(instructions == null) { dprinterrln("No instructions for "+DeviceType); return(null);}  130
   ELEMENTLOOP:
    for (int i = 0; i < instructionCount; i++) {
        if(instructions.item(i).getNodeType() == Node.ELEMENT_NODE) {
            //dprint("(E)");
            Element element = (Element)instructions.item(i);
            //              System.out.println("ele: "+element.getTagName());
            if(! element.getTagName().equals(field)) { continue ELEMENTLOOP; }
            NamedNodeMap attributes = element.getAttributes();
            if (attributes != null) {
                int attributeCount = attributes.getLength();                     140
                for (int j = 0; j < attributeCount; j++) {
                    // dprint("(A)");
```

```
                    Attr attribute = (Attr)attributes.item(j);
                    if(attribute.getNodeName().equals("cycle")
                       && new Integer(normalize(attribute.getNodeValue())).intValue() == cycle) {
                        return(grabtext(element));
                    }
                }
            }
        }
    }
    return((String)null);
}



private static String grabtext(Element element) {
    String retval = "";
    NodeList ournodes = element.getChildNodes();

    if(ournodes == null) return("<no text>");
    for(int i=0;i<ournodes.getLength();i++) {
        if(ournodes.item(i).getNodeType() == Node.TEXT_NODE) {
            retval += normalize(((Text)ournodes.item(i)).getData());
        }
    }
    return(retval);
}

/** Prints the specified element. */
private static void print(Element element, NamedNodeMap attributes) {

    dprint("<");
    dprint(element.getNodeName());
    if (attributes != null) {
        int attributeCount = attributes.getLength();
        for (int i = 0; i < attributeCount; i++) {
            Attr attribute = (Attr)attributes.item(i);
```

```
            dprint(" ");
            dprint(attribute.getNodeName());
            dprint("=\"");
            dprint(normalize(attribute.getNodeValue()));
            dprint("\"");
        }
    }
    dprint(">");

} // print(Element,NamedNodeMap)

/** Normalizes the given string. */
private static String normalize(String s) {
    StringBuffer str = new StringBuffer();

    int len = (s != null) ? s.length() : 0;
    for (int i = 0; i < len; i++) {
        char ch = s.charAt(i);
        switch (ch) {
            case '<': {
                str.append("&lt;");
                break;
            }
            case '>': {
                str.append("&gt;");
                break;
            }
            case '&': {
                str.append("&amp;");
                break;
            }
            case '"': {
                str.append("&quot;");
                break;
            }
            case '\r':
```

```
            case '\n': {
                str.append("&#");
                str.append(Integer.toString(ch));
                str.append(';');
                break;
            }
            default: {
                str.append(ch);
            }
        }
    }

    return str.toString();

} // normalize(String):String

private static void initnodetypes() {
    nodetypes = new String[40];
    nodetypes[Node.ELEMENT_NODE] = "Element";
    nodetypes[Node.ATTRIBUTE_NODE] = "Attribute";
    nodetypes[Node.TEXT_NODE] = "Text";
    nodetypes[Node.CDATA_SECTION_NODE] = "CDATA Section";
    nodetypes[Node.ENTITY_REFERENCE_NODE] = "Entity Reference";
    nodetypes[Node.ENTITY_NODE] = "Entity";
    nodetypes[Node.PROCESSING_INSTRUCTION_NODE] = "Processing Instruction";
    nodetypes[Node.COMMENT_NODE] = "Comment";
    nodetypes[Node.DOCUMENT_NODE] = "Document";
    nodetypes[Node.DOCUMENT_TYPE_NODE] = "Document Type";
    nodetypes[Node.DOCUMENT_FRAGMENT_NODE] = "Document Fragment";
    nodetypes[Node.NOTATION_NODE] = "Notation";

}

private static void dprint(String text) {
    if(DEBUG) System.out.print(text);
}
```

```
    private static void dprintln(String text) {
        if(DEBUG) System.out.println(text);
    }
    private static void dprinterr(String text) {
        if(DEBUG) System.err.print(text);
    }
    private static void dprinterrln(String text) {
        if(DEBUG) System.err.println(text);
    }

                                                                        260
    /** Prints the usage. */
    private static void printUsage() {
        System.err.println("usage: java XMLParse (options) uri . . .");

    } // printUsage()

}
```

A.1.5 tagreq.java

```
/*
 * $Id: tagreq.java,v 1.6 1999/05/20 16:00:06 foley Exp $
 * tagreq.java - conversion routine for ID -> URL
 *
 */

import java.net.*;
import java.io.*;
public class tagreq {

    public static final int MAXconvert = 2;
    public static void main(String[] args) {
        String thisLine;

        if (args.length > 0) {
            try {
                URL u = new URL(converttoURL(args[0]));
                System.err.println("trying "+u);
                // turn the URL into a DataInputStream
                try {
                    BufferedReader theHTML =
                      new BufferedReader(new InputStreamReader(u.openStream()));

                    try {
                        while((thisLine = theHTML.readLine()) != null) {
                            System.out.println(thisLine);
                        }//
                    }
                    catch(Exception e) {
                    System.err.println("Lost on the readline(): "+e);
                    }
                }
                catch(Exception e) {
                    System.err.println("Lost on the InputDataStream conversion:"+e);
```

```
            }
        }
        catch (MalformedURLException e) {
            System.err.println(args[0] + " is not a parseable URL");
            System.err.println(e);
        }
    }
}

public static String converttoURL(String tag) {
    return(converttoURL(tag,1));
}

public static String converttoURL(String tag, int trial) {
    //      System.err.println(tag);
    String retval="";
    String host="",directory="";
    String head="",manu="",prod="",type="";
    if(tag == null) return((String)null);

    type = "RFID";
    // ok, if its damn short, its probably a UPC
    if(tag.length()<=12) {
        if(tag.length()<=12) {
            type = "UPC";
            tag = "0"+tag+"0"; // HealthyChoice sucks!
        }
    }

    // divide into domains/directories
    if(type.equals("UPC")) {
      head="0";
      manu=tag.substring(6,tag.length());
      prod=tag.substring(0,6);
      host = manu+"."+prod+"."+type+".objid.net";
    }
```

```
        else {
          head=tag.substring(tag.length()-2,tag.length());
          manu=tag.substring(tag.length()-10,tag.length()-2);
          int i;
          for(i=tag.length()-10;i>=8;i-=8) {
            prod=tag.substring(i-8,i)+"."+prod;
          }
          if(i>=0) {
            prod=tag.substring(0,i)+"."+prod;
          }                                                          80
          host = prod+manu+"."+head+"."+type+".objid.net";
        }

        // convert prod's "."s into "/"s
        char[] charbuf = prod.toCharArray();
        for(int i=0;i<charbuf.length;i++) {
            if(charbuf[i] == '.') {charbuf[i]='/';}
        }
        prod=new String(charbuf);
                                                                     90
        // this is the secure redirect method
        //   host = manu+"."+prod+"."+type+".objid.net";
        if(trial == 1) {
            retval = "http://"+host+"/"+type+".cgi?ID="+tag;
        }
        else {
            retval = "http://"+host+"/"+type+"/"+manu+"/"+prod;
        }
        return(retval);
        // less secure for demo                                      100
        //return("http://disc/"+type+"/"+manu+"/"+prod);

    }

}
```

A.2 Object Naming System Configuration

A.2.1 objid.zone

```
1   ; $Id: objid.zone,v 1.2 1999/05/14 16:37:27 foley Exp $
    ; SOA generated automatically by granitecanyon.com

    objid.net       IN      NS          VICE-GRIPS.MIT.EDU.
5   objid.net       IN      NS      NS1.GRANITECANYON.COM
    objid.net       IN      NS      NS2.GRANITECANYON.COM

    objid.net.      IN RP foley.mit.edu. Joe.Foley.objid.net.

10  Joe.Foley.objid.net.    IN      TXT     "Joe Foley, NIC handle JF11811"

    ; These are just sample default entries.  Not for real consumption.
    ; celestial seasonings
    1111.2222.3333.objid.net.  IN      A       209.38.196.65
15  ; imdb
    3333.4444.5555.objid.net.  IN      A       169.207.1.237
    ; healthy choice
    50100.40806.objid.net.     IN      A       209.224.0.133

20  ; the subdomains
    RFID            IN      NS      VICE-GRIPS.MIT.EDU.
    RFID            IN      A       18.177.1.46
    UPC             IN      NS      VICE-GRIPS.MIT.EDU.
    UPC             IN      CNAME   RFID.OBJID.NET.
25
    WWW             IN      A       18.78.1.38
```

```
; localhost kludge
localhost.objid.net.    IN      A       127.0.0.1
```

A.2.2 rfid.objid.zone

```
;
; Authoritative data for rfid.objid.net
;
;
@       IN      SOA     VICE-GRIPS.MIT.EDU. DISC.MIT.EDU. (
                                1999042200      ; Serial
                                30      ; Refresh 30 seconds
                                300     ; Retry 5 minutes
                                3600    ; Expire 60 hours
                                216     ; Minimum 3 minutes
                        )
        IN      NS          VICE-GRIPS.MIT.EDU.
        IN      NS          SLEDGEHAMMER.MIT.EDU
802.A696.E746.56C6.C696.7656.E746.F626.A IN A 18.78.1.38

```

A.2.3 upc.objid.zone

```
@       IN      SOA     VICE-GRIPS.MIT.EDU. DISC.MIT.EDU. (
                                1999041601      ; Serial
                                30      ; Refresh 30 seconds
                                300     ; Retry 5 minutes
                                3600    ; Expire 60 hours
                                216     ; Minimum 3 minutes
                        )
        IN      NS          VICE-GRIPS.MIT.EDU.
```

```
            IN      NS          SLEDGEHAMMER.MIT.EDU
10
    ; healthy choice
    50100.40806     IN      A       209.224.0.133
    ; real healthy choice
    050100.408060   IN      A       209.224.0.133
```

A.3 Object Description Language Code

A.3.1 food.dtd

```
<!ELEMENT food-package-home-heated (Device)+>
<!ELEMENT Device (Power | Temp | Wait | Time | Text)* >
<!ATTLIST Device type ID #REQUIRED>
<!ELEMENT Power (#PCDATA)>
<!ATTLIST Power cycle CDATA "1">
<!ELEMENT Time (#PCDATA)>
<!ATTLIST Time cycle CDATA "1">
<!ELEMENT Temp (#PCDATA)>
<!ATTLIST Temp cycle CDATA "1">
<!ELEMENT Wait (#PCDATA)>
<!ATTLIST Wait cycle CDATA "1">
<!ELEMENT Text (#PCDATA)>
```

A.3.2 food.odl

```
<?xml version="1.0"?>
<!DOCTYPE food-package-home-heated SYSTEM "food.dtd">

<food-package-home-heated>
<Device type="MicrowaveOven">
        <Text cycle="1"> Caution, Food may
        become very hot in center </Text>
        <Power cycle="1">100</Power>
        <Time cycle="1">240</Time>
        <Wait cycle="1">30</Wait>
        <Power cycle="2">95</Power>
        <Time cycle="2">240</Time>
```

```
        <Wait cycle="2">120</Wait>
    </Device>
    <Device type="Oven">
        <Temp>450</Temp>
        <Time>900</Time>
    </Device>
    <Device type="ConvectionOven">
        <Temp>400</Temp>
        <Time>600</Time>
    </Device>
    <Device type="ToasterOven">
        <Temp>350</Temp>
        <Time>1200</Time>
    </Device>

    <Device type="Toaster">
        <Text>Error: This Food Package not
        compatible with this Device</Text>
    </Device>
    </food-package-home-heated>
```

A.4 MicroController Code

A.4.1 micrwave.bs2

```
1
      serData          var      byte
      numOffset        con      48
      TEMPCOOKB CON 10
5     TIMECOOKB CON 11
      DEFROSTB CON 12
      'HOLDTIMERB CON 12
      STARTB CON 13
      CLEARB CON 14
10    POWERLEVELB CON 15
      ' ASSUME PF IS ON SCREEN AND SETUP
      'LOW TIMECOOKB
      'PAUSE 100
      'HIGH TIMECOOKB
15    'PAUSE 100
      'LOW CLEARB
      'PAUSE 100
      'HIGH CLEARB
      ' MAIN LOOP
20    MainLoop:
      SERIN 16,84,[serData]
      debug cr
      ' 0-9 are obvious(48-57)
      ' alphas are 65-90,97-123
25    ' S=Start,T=TimeCook,C=Clear
      ' M=TempCook,D=Defrost,L=PowerLevel
      if serData >= 48 AND serData <= 57 then Numpress
```

```
    if serData < 65 OR serData > 116 then MainLoop
    if serData > 90 AND serData < 97 then MainLoop
30  if serData = "S" OR serData = "s" then Start
    if serData = "T" OR serData = "t" then TimeCook
    if serData = "C" OR serData = "c" then Clear
    if serData = "M" OR serData = "m" then TempCook
    if serData = "D" OR serData = "d" then Defrost
35  if serData = "L" OR serData = "l" then PowerLevel
    PushB:
    LOW serData
    'debug "LOW:",DEC serData,cr
    PAUSE 250
40  HIGH serData
    'debug "High:",DEC serData,cr
    INPUT serData           ' allow manual override

    GOTO MainLoop
45
    Numpress:
    'debug "Numpress",cr
    serData = serData - numOffset
    GOTO PushB
50  Start:
    'debug "Start",cr
    serData = StartB
    GOTO PushB
    TimeCook:
55  'debug "TimeCook",cr
    serData = TimeCookB
    GOTO PushB
```

```
      Clear:
      'debug "Clear",cr
60    serData = ClearB
      GOTO PushB
      TempCook:
      'debug "TempCook",cr
      serData = TempCookB
65    GOTO PushB
      Defrost:
      'debug "Defrost",cr
      serData = DefrostB
      GOTO PushB
70    PowerLevel:
      'debug "PowerLevel",cr
      serData = PowerLevelB
      GOTO PushB
```

Bibliography

[1] Roberta Brody. Consequences of Electric Profiling. *IEEE Technology and Society*, 18(1):20–27, Spring 1999.

[2] CNN Article on a Smart Trashcan using RFID tags. http://www.cnn.com/TECH/ptech/9904/10/smart.trash.ap/

[3] Cay S. Horstmann & Gary Cornell. *Core Java 1.1*, volume 1-Fundamentals. Prentic Hall PTR, Upper Saddle River, New Jersey, 1997.

[4] Website for DENTS, a generalized DNS server front-end. http://www.dents.org

[5] The DISC Homepage. http://disc.mit.edu

[6] Discussion during DISC meeting of RFID Tag market.

[7] Nahum Gershon & Stephen G. Eick. Guest Editors' Introductionn: Information Visualization. The Next Frontier. *Journal of Intelligent Information Systems*, 11(3):199–204, November-December 1998.

[8] David Flanegan. *Java in a Nutshell: A Desktop Quick Reference.* O'Reilly & Associates, Inc., Sebastopol, California, 1997.

[9] Elliotte Rusty Harold. *Java Network Programming.* O'Reilly & Associates, Inc., Sebastopol, California, 1997.

[10] Richard Hasha. Needed: A common distributed-object platform. In *IEEE Intelligent Systems and their applications* [16], pages 14–16.

[11] IBM XML Parser for Java. http://www.alphaworks.ibm.com/formula/xml

[12] Mitchell M. Tseng & Jianxin Jiao. A variant Approach to Product Definition by Recognizing Functional Requirement Patterns. *Journal of Engineering Design*, 8(4):329–340, December 1997.

[13] David Konopnicki and Oded Shmueli. Information Gathering in the World Wide Web: The W3QL Query Language and the W3QS System. *ACM Transactions on Database Systems*, 23(4):369–371, December 1998.

[14] Paul Albitz & Cricket Liu. *DNS and BIND.* O'Reilly & Associates, Inc., Sebastopol, California, second edition, 1997.

[15] Frank Manola. Technologies for a Web Object Model. In *Journal of Engineering Design* [12], pages 38–47.

[16] Michael C. Mozer. An intelligent environment must be adaptive. *IEEE Intelligent Systems and their applications*, 14(2):11–13, March-April 1999.

[17] Discussion on ONS design with Erik Nygren¡nygren@MIT.EDU. 1999.

[18] Marshall T. Rose. *The Simple Book: An Introduction to Internet Management.* Prentic Hall PTR, Upper Saddle River, New Jersey, second edition, 1994.

[19] Pete Sorrells. *microID(tm) 125kHz RFID System Design Guide*, 1998.

[20] *BASIC Stamp Manual Version 1.8.*

[21] W. Richard Stevens. *Unix Network Programming*, volume 1, appendix A, pages 889–891. Prentic Hall PTR, Upper Saddle River, New Jersey, second edition, 1998.

[22] Free Distributed Email to Fax Gateway. http://www.tpc.com

[23] The "Things That Think" Group at the MIT Media Lab. http://www.media.mit.edu/ttt/