Software Architectures for Things That Think

Matthew Gray

August 12, 1998

1 Introduction

The Media Lab vision of Things That Think[1] is rapidly taking form. A wide range of data previously unavailable is now accessible through many things that think. Further, many everyday "things" can now computer actuated, as output methods. The storage, organization, sharing and acting on this data, still is the domain of individual, special purpose software architectures.

1.1 The Problem

In the case of many past projects, handling the data after it was collected has been a heavily manual task. Data was often not merged or correlated with other related data sets, and was not readily accessible or clearly labeled.

In the case of the ongoing Marathon Man project, all of the data collection and collection has been primarily in the form of flat data files and by-hand generation of plots and the like. Some statistical analysis of the data has been done[2], however even this was done in an isolated environment and involved a unnecassarily large amount of human attention. Integration of multiple data sources or sharing of data between systems has essentially not been done at all.

As more and more data sources and thinking environments become available, the value of shared, scalable access to data and resources becomes extremely valuable. Requiring the use of a wide range of custom interfaces to access the "things", whether the data or control over them, makes a true Things That Think environment prohibitive.

2 The Solution

In order to create an effective network of Things That Think, a number of pieces are needed. We think of this network in terms of a distributed object model, and the following pieces define behaviors and interactions of those distributed objects.

1. A discovery system, to allow "things" to find each other, find local processing power, and find Internet resources. This includes naming, contexts, and configuration of the things as well.

2. A standard proxying system, to allow "dumber" Things to easily be proxied by a piece of code.

3. A standard interface, to allow interaction between things, processing power, and the rest of the Internet. This includes introspection, and ability to engage in dynamic interface interaction.

[1] http://www.media.mit.edu/ttt/Vision.html
[2] http://www.media.mit.edu/~mkgray/army/

4. A standard labeling, to allow semantic understanding of the interfaces to things.
5. A standard data repository, for access to current and historical data and event logs from sets of things.
6. A standard set of components, for anlysis, presentation, and organization of data and actions coming from things.
7. A user interface, for building and interacting with things at a programmatic level.

In all of these cases, the standard is necassarily an *interface* standard, with the implementation of the piece as unspecified as possible.

This paper discusses the details of these pieces, what systems provide these pieces and the issues surrounding them, and makes a recommendation as to which system to use.

2.1 Existing Systems

A number of pre-existing systems implements some of these pieces. Some of the terminology from these distributed object systems will be used, with a tendency to presume the java model.

Specifically, [stuff about jini]

2.2 Conventions

Throughout this paper, there will be many references to "things that think". These will be referred to as Things, with a capital T to distinguish from any other usage. A network of such Things is called a "TTT network".

2.3 Scenarios

One of the most direct ways to identify what the requirements of a system like this are, is to give detailed descriptions of some of the scenarios under which it would be used.

2.3.1 BioPak

A person dons the BioPak, which as soon as it is powered on begins collecting a variety of biometric measurements, which are logged. After wearing this for some time, they take the BioPak and "plug it in"[3], and the following steps occur. The BioPak, which is assumed to have limited onboard processing power,

[3] In all cases, "plugging it in" is taken to include actually physically plugging it in, or initiating some sort of wireless connection

identifies itself in some way the the computer system. The system then initiates a transfer of the data. This data is now available easily to other systems on the net, and a presentation system either can discover this information, or will be notified of the newly available data and take some automatic action, such as presenting it to the user.

2.3.2 Sponsor Sensor

A number of motion sensors, optical tripwires, and the like are configured in a room, and are plugged in to one or more computers. The data coming off these sensors is accessible over the net and automatically discoverable. Further, something which is using this data should be able to automatically become aware of any new sensors of this type. This data feeds into an inference system, which can distinguish the arrival of one or a small number of people in the room from the arrival of a large group. Upon this decision, various actions, such as playing sounds, taking pictures, changing lighting are executed.

2.3.3 Personal Monitor

A desktop application that helps manage one's health and diet is capable of automatically discovering and being semantically aware of new types of data that may be available. A new TTT bathroom scale, would automatically be discovered and the data collected would be understood by the program to be a weight. It could be easily graphed, as just a value, or it could be integrated more meaningfully by a program that knows what kind of weight changes are healthy, given different exercise regimes, as reported by something like the above BioPak.

2.3.4 Mr. Java

A project like the Mr. Java coffee machine should be reworkable in this context: An identity tag reader which triggers events which causes the right kind of coffeee and news to be generated.

3 Components

3.1 Discovery and Naming

3.2 A Standard Proxy

3.3 Interface Definitions

3.4 Semantic Labeling

3.5 Data Repository

3.6 Standard Components

3.7 User Interface