White Paper

CMatos™ — Jini™ services for non-Java devices

By

Steven Knudsen and Serge Brache

September 2002

www.psinaptic.com

PsiNaptic Inc.
216, 200 Rivercrest Drive SE
Calgary, Alberta
Canada T2C 2X5
T: 403 720-2531
F: 403 720-2537

Executive Summary~~Forward~~

The main problem in a world where computing costs become ever less and processors are finding their way into everything, even our clothing, is how to have all these devices communicate and perform tasks on our behalf without having to manage them. Put another way, how do we get all these very useful devices to disappear and still do what we want?

CMatos™ is new software technology that enables very small, embedded processors to offer Java-based services. It's an extension of JMatos™ technology [1] that opens the door for millions and billions of devices to participate in the next era of machine-to-machine communications. With CMatos, a $1 processor can offer information and software services to any device that can act as a Jini client.

The roots of CMatos are in Jini Network Technology. Created and developed by Sun Microsystems, Jini technology enables devices to discover and share services and information without the need for human configuration and management. Devices exchange Java-based services using Jini mechanisms that are dynamic and can adapt to the changes real devices see in real-world computing networks. PsiNaptic has created a Jini technology-compliant implementation for small, Java-enabled devices—JMatos.

Realizing that millions and even billions of processors are too resource-constrained to support Java, PsiNaptic refined JMatos further to create CMatos, a technology that enables Jini behaviour on non-Java devices.

Any processor that is embedded in an object that encounters change can benefit from CMatos. Change can be logical or physical; a device can be connected to a network that changes its configuration on the fly, or the device itself can change networks, change associations as it moves with its owner. As Jim Waldo, one of the creators of Jini technology at Sun, has said [1],

> *That was our original thought. Change is not a rare event—it's constant. We had to figure out a way to allow change to happen without involving people. If change required people, and considering networks now growing into millions of machines and the amount of change those networks experience, we would all have to become system administrators. The only way to avoid that is to automate the ability to deal with change.*

In this white paper, the motivations behind CMatos and Jini technology are described from the perspective of deeply embedded computing. The main features of CMatos are described and some example application areas are explored.

Table Of Contents

~~1~~

-

1 Introduction

A side effect of Moore's Law, of which we are all sadly aware, is that decreasing computing costs have resulted in the increase in the complexity of computing devices. We spend ever more time and money just configuring, managing, and learning about our computers.

As users, that situation is magnified when we begin to acquire multiple computers. These include not only PC-level computers, but PDAs, cellular phones, and soon similar devices built into our appliances at home, our cars, medical instruments, and many other everyday objects. We naturally want these things to share information, to network them together, and that's where the nightmare really begins—a nightmare of catering to our machines.

What about the computers we can't see, the ones that are deeply embedded in things all around us? They too might have useful information and services, but we just don't have the time and energy to manage them all!

Billions of embedded computers are sold every year. The problem just keeps getting worse…

People have described this emerging computing environment as being pervasive. Pervasive computing is about using computers everywhere, at anytime, and by anyone. IBM Chairman and CEO Louis Gerstner has described the goal of pervasive computing as follows [2].

> *Everybody's hardware needs to run everybody's software on everybody's networks.*

This white paper is about helping the very smallest devices participate in a pervasive computing environment. It's about giving such devices the ability to share the information and functions that are intrinsic to them, and how they can do that with one another on our behalf without us having to manage them.

CMatos network technology takes Jini capabilities to levels that no one thought possible. Jini is a network technology created by Sun Microsystems that allows devices to autonomously discover each other, and offer and consume Java-based services. CMatos gives deeply embedded processors, controllers, and even sensors the ability to offer Jini services—that is, the ability to autonomously share their own services and information, to support machine-to-machine interactions in pervasive computing environments.

2 The Cost of Computers

Anyone who has ever used a computer, whether it's one you've bought to use at home or one that you have at work, will know that there are many associated costs. Some are one-time, and others are ongoing, including:

- initial setting of preferences, networking parameters, etc.,
- peripheral purchases,
- peripheral configuration,
- basic applications,
- transfer of all your old work to the new computer,
- anti-virus software,
- software updates, especially due to new viruses,
- mobile networking (for PDAs, laptops, and similar devices)
- system backup,

- periodic data backup.

Nobody likes to pay extra, but often we are willing to when something is very important to us. Think of your car. A single person will have one, a family two or three. We understand that ownership means that we have to pay a professional for periodic service and inspection. That means an investment of time and money on our part, but it's worth it to us because the car is such a useful tool. It seems the same expectation holds true for our computer. At the very least, we invest our own time and money to use them. However, unlike our vehicles, they seem to need upgrading, new software, or even crash (often), requiring more and more of our time.

Why are we willing to invest so much in these devices? Partly it's because our time is offset by their high cost—it seems worth it to us. Cars cost a lot and so do PCs. Let's examine that assumption…

Begin with high-end servers and their costs. These computers cost tens or hundreds of thousands of dollars, even millions. Their installation, configuration, and management are the domain of well-paid professionals. Some receive around the clock support. Since they are the main computers for large organizations, not to mention the Internet, this expense is justified. They support large software for large systems that cater to many concurrent users. Some four million servers and mainframes were sold in 2000.

Down from there we find PC-level computers. With prices in the thousands of dollars, about 150 million were sold in 2000. Even at this price, many of these computers justify professional support. That support is usually found in businesses that have IT departments, but many computer stores and specialty companies offer support for small companies and home users. Since these computers tend to be used by individual users, we've come to expect that we will shoulder some of the cost and burden of ownership. After all, it's rare that you won't want to customize your "personal" computer in some way.

At the next level, things become a little more interesting. Over 400 million PDAs and cell phones were sold in 2000. These are computing devices that are priced at or just below the magic $500 consumer level. At that level, our expectations of them as computers change dramatically. Most software is pre- installed and configured on such devices. Typically, we are interested only in getting some personal information stored on them and them using them as a tool. This is especially true of cell phones where we have been conditioned to expect to use them mainly for communication. At best we are willing to introduce these devices to sources of information, that is, other larger computers or networks. Regardless, we don't expect to pay someone to add software or configure and maintain them for us, and we ourselves certainly can't be bothered [2] ! Again, it's a question of worth. Our time and money is worth more to us.

Imagine now the next level of computing device. Some eight billion embedded processors were sold in 2000. These are embedded everywhere—from toys to appliances to cars. They are inexpensive, costing dollars, and are generally completely hidden from end users. You probably have from ten to forty processors in your car. Others are in your home, built into your microwave, stove, fridge, and environmental systems. Increasingly, these devices need to talk to one another and, eventually, to us. They can provide useful information and services; like monitoring your power consumption and scheduling activities in your home to use the cheapest power rates available. The big problem is how to configure and manage all these devices, get them to talk to one another, especially when we don't know about many of them! Comparing their cost to the cost of your time, it's just not worth it for you to interact with them directly. Not even for a minute.

The situation only gets worse when we imagine all the new devices that will become part of our lives. Things like sensors and tiny controllers costing pennies will find their way into new systems and applications.

The relationship between device cost and the value we are willing to assign is illustrated in Figure 1.

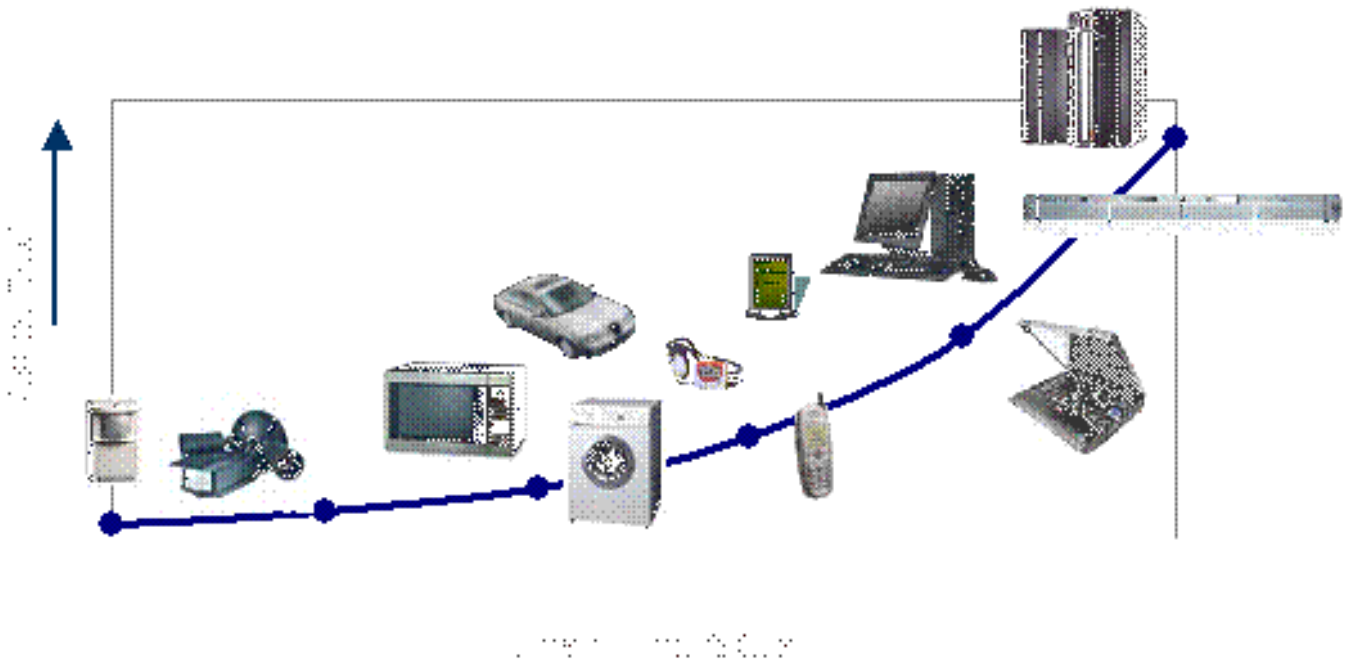

Figure 1 An illustration of the cost of computers and how much we are willing to invest of our own time and money to support them.

We are in danger of becoming servants of our machines. It should be—must be—the other way around. The smaller the device, the more it needs to do things autonomously, on our behalf. Ideally, all devices should be able to discover each other and share information and services without any human interaction, with zero human cost.

Ideally, we would like all computers to share whatever information and software they need to accomplish their tasks.

3 The Misery of Driver Software

What's wrong with how things are done now?

How do devices talk to each other today?

We're familiar with the idea of a software driver. It's a piece of software that gets installed on a computer to enable communications with a device (basically, another computer). A good example is a printer driver.

To use a printer you have to go through several steps. Assuming you have connected the printer to your computer or a network to which your computer is connected, you next

- find the appropriate software driver; either it's on your computer already or it's on a disk someplace (sure hope someone can find it!)
- install it; once the driver is found, your computer may automatically ask to install it. Then again, maybe it won't… plug 'n' pray. Either way, the computer probably wants to reboot.
- configure it; this can involve several steps. First, you'll have to tell your computer where the printer is connected—which port, which network. If it's on a network, then you'll have to know its name and/or address. Once that's done, there may be further information required, like the capabilities of the printer; does it have a duplex unit, how many trays, what sizes of paper, etc.

This process happens every time a printer is added to your environment, or when you move your computer to a new environment, such as a traveler with a laptop is likely to do.

We've come to accept this as a normal requirement, a normal hassle. It would be interesting to estimate how much time is spent every year doing such tasks. It would be frightening to imagine how much time would be wasted if we had to do this for every one of the billions of embedded processors that exist and will exist in the future.

Imagine if you had to do this for every thing you already own or will own, like household appliances, entertainment systems, game players, music players, cell phones, PDAs, even your vehicle. It's just not feasible.

Within all these devices are embedded processors. At that level, further complications arise.

If you are a manufacturer, it's very important how you design your hardware and software. Low-level software is tightly bound to the target processor. A change in processor means you have to update the software, maybe even re-write it. After the change, you now have two sets of software to support. If you change the data communicated by that device to others, or the format of the data, those other device must also be updated.

The obvious answer is for the devices themselves to come already with the right drivers and to offer them to other devices that want to communicate with them. This is not quite as simple as it sounds. Some important considerations include

- Software written for one processor won't probably run on another.
- How do the devices find each other?
- What protocol do they agree to use to communicate?
- What happens to the software after a device is finished using it?

These, and other similar considerations, are common in distributed network computing. Over time, several approaches to solving such problems have evolved. Even the most promising solution, Jini Network Technology, has been applicable only to larger computing systems, those with the processor power and storage needed to accommodate a substantial Java environment.

The challenge for PsiNaptic has been to take the best solution for such distributed network computing problems and bring them down in size and cost enough to be attractive to device manufacturers; we had to find a way to get Jini technology to match the economic realities of deeply embedded systems.

4 Jini Services for Pip Squeaks

In 1999, Sun Microsystems announce Jini Network Technology. In that announcement and in subsequent media coverage, Jini technology was touted as the solution for getting devices of all kinds to talk to each other. Cameras would talk to storage devices to automatically save pictures and then talk to printers to create photographic prints.

The trouble was that the only available Jini technology implementation was too large for small devices. The static and runtime storage requirements of Sun's reference implementation are at least 3 megabytes. Included in that total is the requirement of Remote Method Invocation (RMI), a part of the Java 2 Standard Edition environment, itself at least 10 megabytes in size.

By designing a Java-based Lookup Service implementation

- that does not use RMI,
- shifts processing to client devices,
- and is designed for limited capability hosts,

PsiNaptic has been able to reduce the host device requirements by an order of magnitude. This implementation, called JMatos [3], is fully compliant with the Jini Lookup Service (as per the specification [3]). It has been hosted on all levels of Java environment, including CLDC, CDC, Personal Java, and J2SE. The total storage requirements for the JMatos Lookup Service are less than 100 kilobytes.

4.1 Not Small Enough

Of course, a Java-based Lookup Service (LUS) still requires a Java runtime environment. Even the smallest CLDC environments require at least 200 kilobytes. With a total JMatos system footprint of 300 kilobytes or better, that implementation was still too large for really deeply embedded processor applications. Very low-cost sensors, controllers, actuators, and similar devices are typically designed with as little as 2 or 4 kilobytes of storage, and for only a buck or two.

Java was the problem; not only due to the overhead required to support it, but also the fact that (despite the wishes of every Java programmer), it is still not acceptable in all applications, especially those that are very low end or require hard real-time execution.

The problem was to find a way to provide Lookup Service functionality, offer Java-based services, using a non-Java language.

Two things were key to the solution.

1. We assume that very small devices are mainly interested in telling others something, or doing something that is intrinsic to their nature. For example, a temperature sensor wants only to tell you the temperature, and maybe its context (where it is, with what it is associated, etc.). That is, a small device wants only to offer a service.

2. A Jini federation is not a federation unless client devices are present. In other words, for a service to be used, a Jini client has to discover and download it. The client must have a sufficient *Java environment*.

The Lookup Service interface is the ServiceRegistrar, a Java object. Jini client devices receive and load it when they discover the LUS. Likewise, any services offered by the LUS are Java objects. Again, only the Jini client actually invokes them. Last, a very simple device, like a sensor, that wants to offer a service that is intrinsic to it will not want to run that service itself. Nor is that service likely to change once the device is deployed. The conclusion—*on the LUS, the ServiceRegistrar and all services can remain static bytecodes*.

If, perchance the LUS host is willing to host services from other devices, by definition those devices will be Jini clients. Any Java-related preparation of the service to be registered with the LUS can be done on the client by the ServiceRegistrar on that client device. Thus, having removed the need to execute any Java bytecodes on the LUS host, we were free to implement the remainder of the needed Lookup Service functionality in C.

The result is *a total CMatos system implementation* that, including the static ServiceRegistrar bytecodes and all other related LUS and communications code (TCP/IP stack), is less than 60 kilobytes.

A comparison of Jini technologies is shown in Figure 2. So that we compare apples to apples, where an operating system is required, we have assumed that an embedded Linux OS will be used. Clearly, there is always a cost to using a JVM—at worst more than 10 megabytes (J2SE) and at least 200 kilobytes (CLDC). Based on a test implementation discussed below, CMatos requires less than 60 kilobytes. That includes everything: the network stack, the static Java bytecodes, and the logic to support the Lookup Service.

4.2 Architecture

Many small devices have very simple architectures—a processor, some storage, maybe a communications port, maybe a sensor or actuator of some kind. This is illustrated in Figure 3. Component count is proportional to system cost—the fewer the components, the simpler the device, the better.

To make sure the following discussion make sense, let's quickly review the main steps that happen when a client device enters a Jini federation. A more thorough overview of this is provided on the web, for example, at http://wwws.sun.com/software/jini/whitepapers/jini-execoverview.pdf . CMatos-related details will be discussed later in this section.

What happens when a client device enters a Jini federation?

1. The Jini client device is connected to a network on which a device is running a LUS. In Figure 3, the network is shown as a simple channel.
2. The client "discovers" the LUS via the usual Jini mechanisms (multicast or unicast discovery).
3. The client receives the ServiceRegistrar Java object and loads it into its JVM.
4. The client uses the ServiceRegistrar object to find and obtain services of interest, register its own services with the LUS, or request notification of changes in the federation.

PsiNaptic's initial CMatos development system resembles the architecture shown in Figure 3. It is an 8051 clone based system with external RAM and EPROM, two serial interfaces (one for a debug monitor, one for PPP driven communications), and a bank of LEDs that emulate a sensor/actuator.

With the idea of a simple architecture in mind, what are the main considerations from the perspective of CMatos? Let's look at them by subsystem.

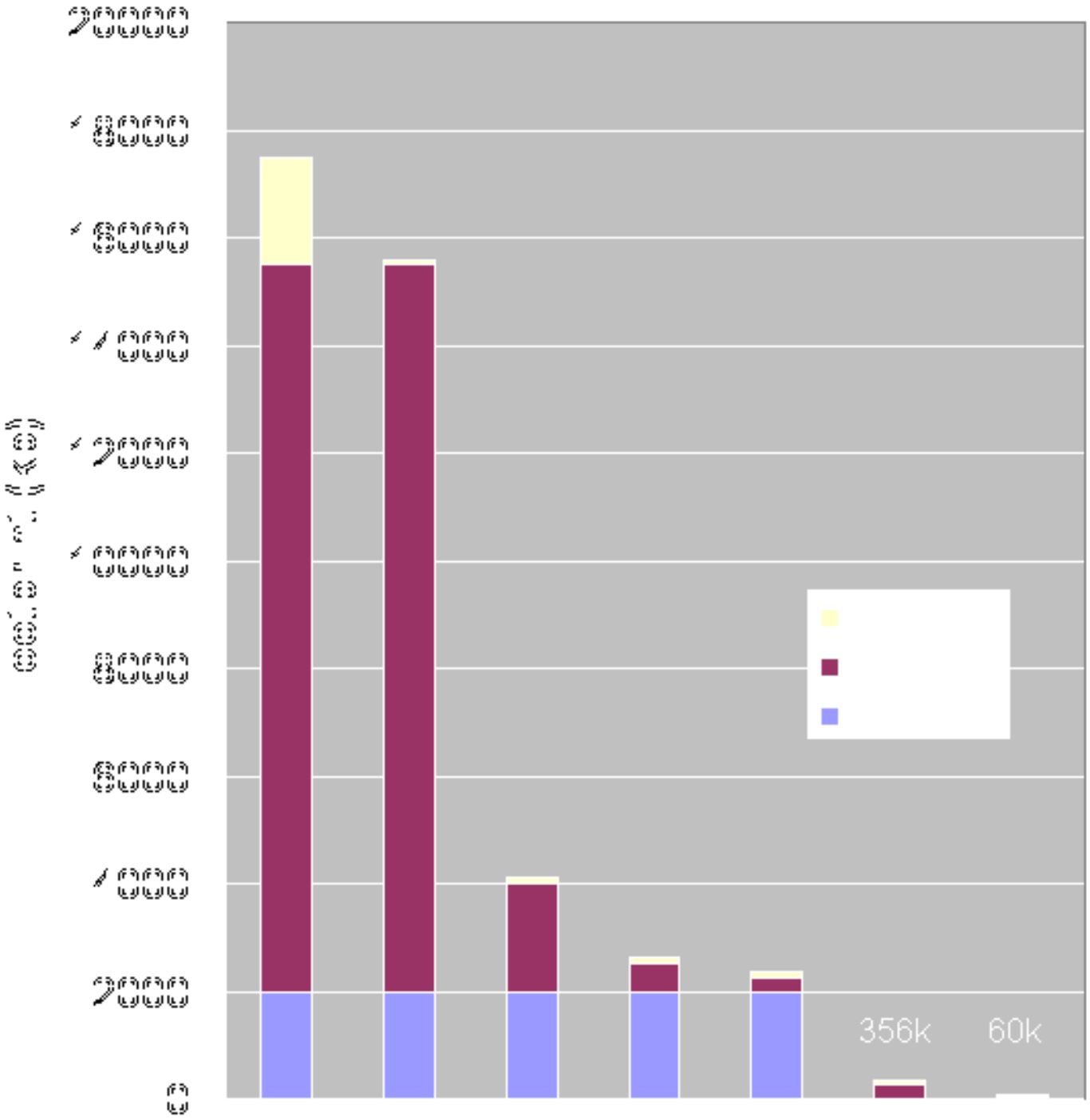

Figure 2 A comparison of total system memory footprint for various Lookup Service implementations [4].

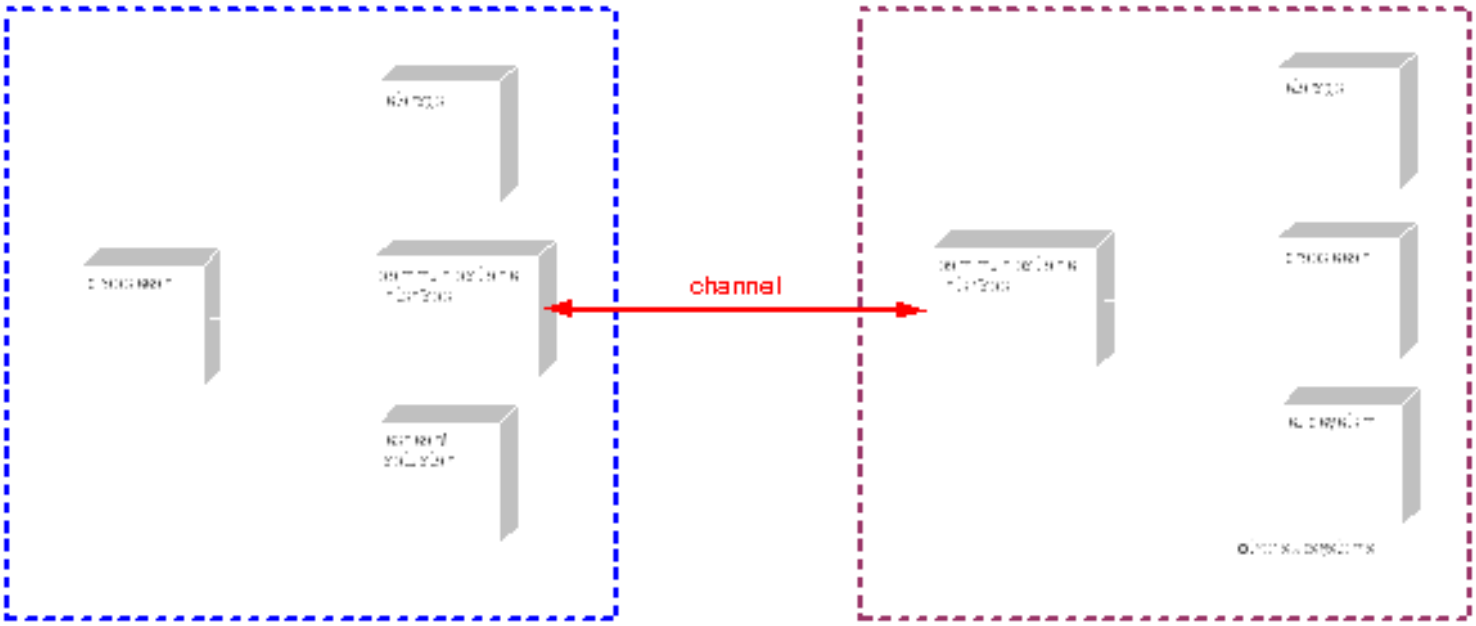

Figure 3 A simple architecture for a CMatos host connected to a Jini client device.

4.2.1 Storage

Storage is required for the subprograms that support the LUS, for any Java-based services including the ServiceRegistrar object, and for communications protocols. In addition, if the device offers its own services, there may be subprograms that run in support of those services. A simple diagram of the memory requirements is shown in Figure 4. Support for the ServiceRegistrar is lumped in with the lookup service subprograms. As seen from the annotations, the total storage required for our development system is about 50 kilobytes. A very simple service that toggles a LED requires about 3 kilobytes for the (static) Java and non-Java (support) components.

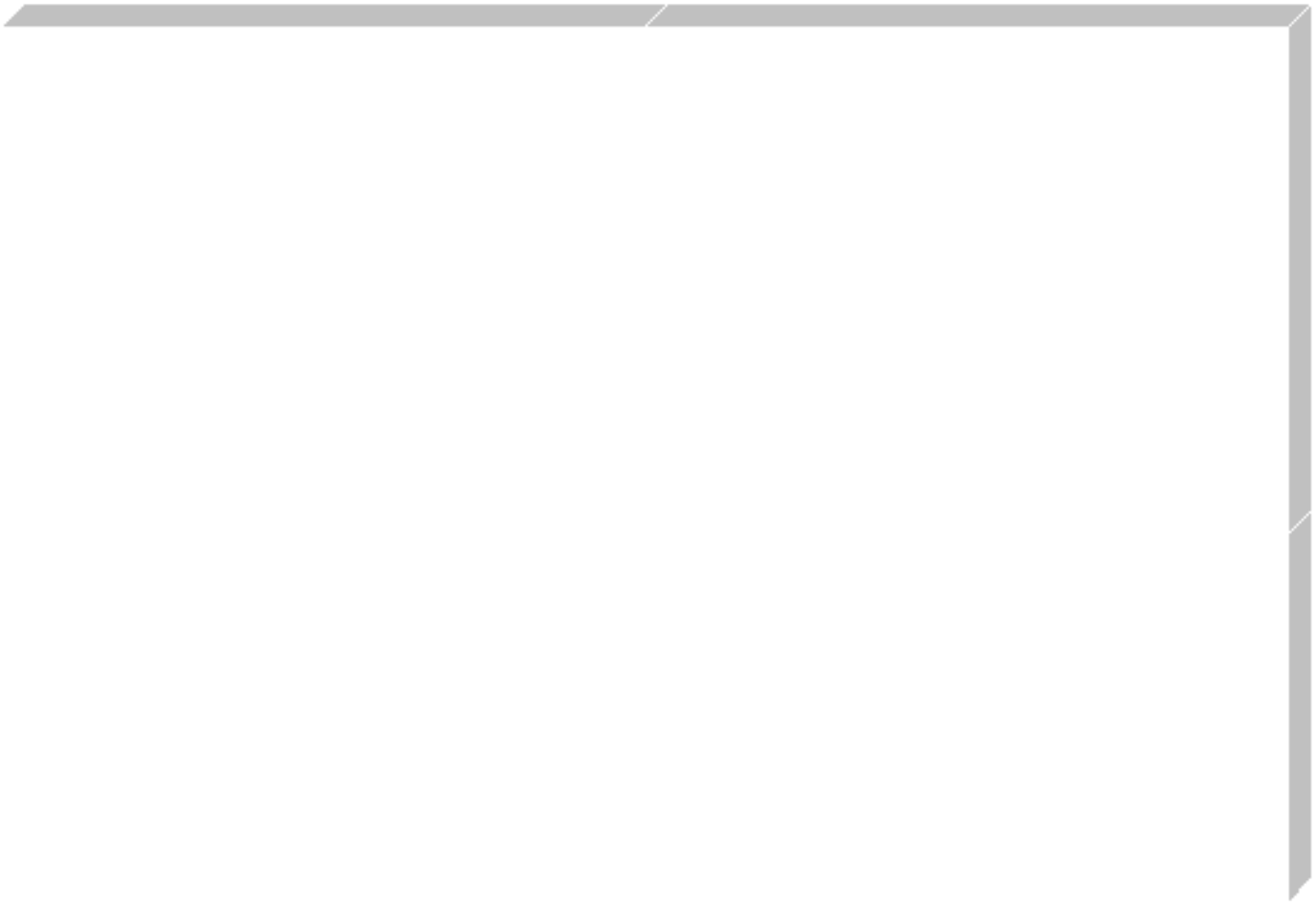

Figure 4 The storage subsystem components broken into functional groups. The actual storage required for the CMatos development system is noted.

4.2.2 Processor

The processor can be just about anything from a 4-bit controller up to the latest AMD, Motorola, or Intel monster. The main consideration is the processor's ability to support a communications protocol. As will be seen in the next subsection, non-TCP/IP protocols can be supported provided a simple protocol bridge is present somewhere in the network. Protocols such as simple two-wire serial, LONTalk (EIA709.1), CAN, I2C, Bluetooth, and many others can be considered. Referring to Figure 4, we see that half the storage requirement is dedicated to the TCP/IP and PPP subsystems. Using a simpler protocol could substantially reduce the communications storage requirement. Then a very inexpensive system on a chip might be considered, like a PIC.

4.2.3 Communications Interface

In an architectural sense, the communications interface is the key to any CMatos system. The choice of interface and protocol impacts the storage and processor requirements, possibly the chip count, and certainly the device cost. In addition, it determines the kinds of networks the device can work with.

Naturally, the choice of interface is very much dependent on the device application. For example, a sensor design may be intended for use in a hierarchical network. The sensor might be one of many attached to an aggregator device. To keep

costs as low as possible, it would make sense to have the sensor-aggregator communications be very simple, perhaps just a serial or 1-Wire cable.

Regardless of the choice of interface, eventually the CMatos device needs access to a TCP/IP based network. After all, the objective is to offer services to Jini client devices, and Jini mechanisms are based on TCP/IP.

Thus, there are two categories of interface to consider; those that support directly TCP/IP, and those that don't. For the latter, a protocol bridge is required

4.2.3.1 TCP/IP-based Communications

If the interface chosen supports TCP/IP, then the architecture is as shown in Figure 3. That is, the channel is assumed to support TCP/IP messaging in some standard way. For example, as described above, PsiNaptic's CMatos development system uses TCP/IP messaging over a serial cable via PPP. The Jini client is a Windows-based PC acting as a PPP host device.

4.2.3.2 Bridge-based Communications

When the interface chosen does not support TCP/IP, then the architecture may resemble one of those shown in Figure 5. Either the Jini client device will include a bridging hardware or software component, or some other device on the network will. The bridging component translates the non-TCP/IP protocol to TCP/IP and UDP messages that match those required by the Jini Lookup Service mechanisms. The non-TCP/IP protocol can be proprietary or not; that is a design choice.

As discussed in introduction of subsection 4.2.3, there are designs in which a bridge-based architecture would be appropriate. One criterion that might be used is the availability of a more capable device with access to a TCP/IP network. If the CMatos devices can be expected to have at least intermittent access to that device, it makes sense to put a bridge on it. On the other hand, if many or all CMatos devices will use the same non-TCP/IP protocol and physical medium, then it might make sense to put the bridging component on the Jini clients. For example, in industrial automation systems, many sensors and controllers may be deployed in different configurations. Computer monitoring systems, such as PDAs, can be Jini clients and access the sensors and controllers as needed. Since the sensor and controller network can be expected to be homogeneous, it is logical to equip each monitoring device with the bridge component.

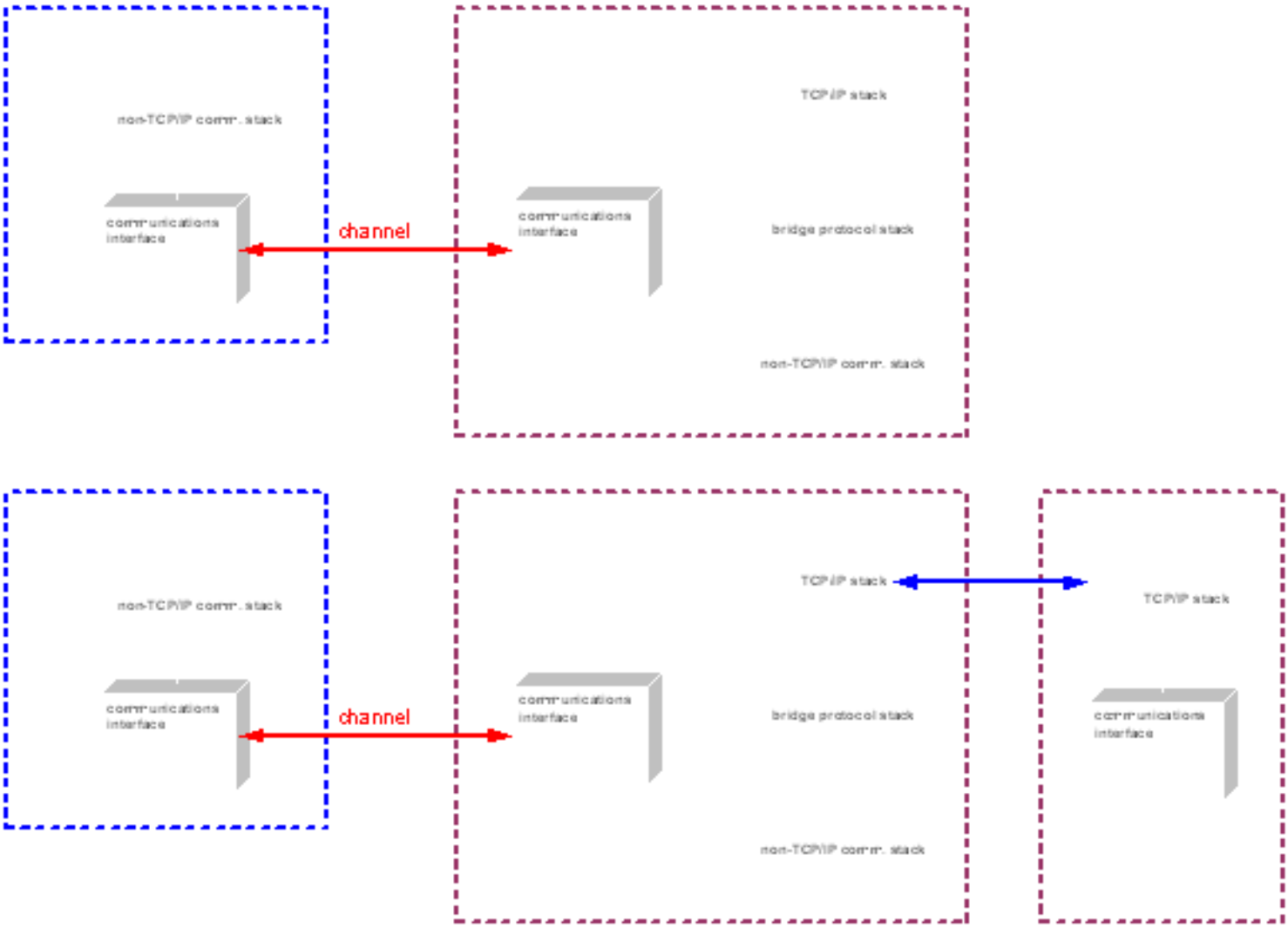

Figure 5 Two possible architectures for a CMatos implementation not employing TCP/IP communications. At the top, the Jini client device contains the necessary subsystems for the protocol bridge. In the architecture at the bottom, an intermediary device provides the protocol bridging.

4.2.4 Sensor/Actuator

The service offered by the CMatos device will almost always be related to some function of the device. We have suggested that some devices may be sensors or actuators. Regardless of the device function, we can expect that the Java service (instantiated on the Jini client) will communicate back to the CMatos device. That communication will be via the communications channel and, hence, use the same protocols as the CMatos LUS implementation. Since the Java service is an object, the details of the communication are hidden from the service client. This highlights one of the main features of using Jini technology for such applications. Because the implementation details are hidden, manufacturers can change the underlying CMatos hardware and software, perhaps to take advantage of new or cheaper technology, and not change the service offered.

4.3 Cost

While it’s not possible to estimate an absolute cost for CMatos devices without specifying a specific application, we can

give a rough estimate based on the PsiNaptic development system described above. Assuming large volume production, so that non-recurring costs can be neglected, an estimate based on the primary component costs can be made.

In Table 1, a breakdown of the costs for a simple CMatos device is shown.

Subsystem	Cost
Processor	0.60
Storage (RAM, PROM)	1.70
Communications [5]	0.50
Power	0.35
Total	3.15

Table 1 Cost breakdown of an example CMatos device that employs TCP/IP and PPP over a serial channel.

This example device is based on an 8051 processor clone that has a built-in UART. As can be seen, the cost for even this basic development-based system is pretty low. With moderate effort, it can be brought even lower.

5 Potential Applications

Here we discuss a few possible application areas for CMatos. In each, low-cost devices are present that have information to communicate to Jini clients. After going through these few areas, it should be obvious that CMatos and Jini technology can be applied in every embedded application.

5.1 Automotive

The controller area network (CAN) is one of several popular bus systems used in vehicles to facilitate communication between various subsystems. Information from these systems is of interest to other subsystems in the vehicle, to the vehicle owner, to service technicians at a dealership or service center, and to the vehicle manufacturer. Jini technology provides the means for services from the vehicle subsystems to be found by clients at all levels of interest.

A simple view of such a system is presented in Figure 6. There are several sensors or subsystems connected to a CAN bus in the vehicle, which is connected to the vehicle's telematics unit. In this case, because the CMatos communications channel is a CAN bus, the telematics unit contains a bridging protocol subprogram that exposes the CMatos interface. The telematics unit acts as a client, using services from the CMatos sensors and subsystems as desired.

The telematics unit is connected to a wireless transceiver in the vehicle which links with a gateway in the owner's home, with access points in the environment, and ultimately to the Internet. Through this link, Jini clients are able to find the auto's services, including those offered by the devices on the CAN bus.

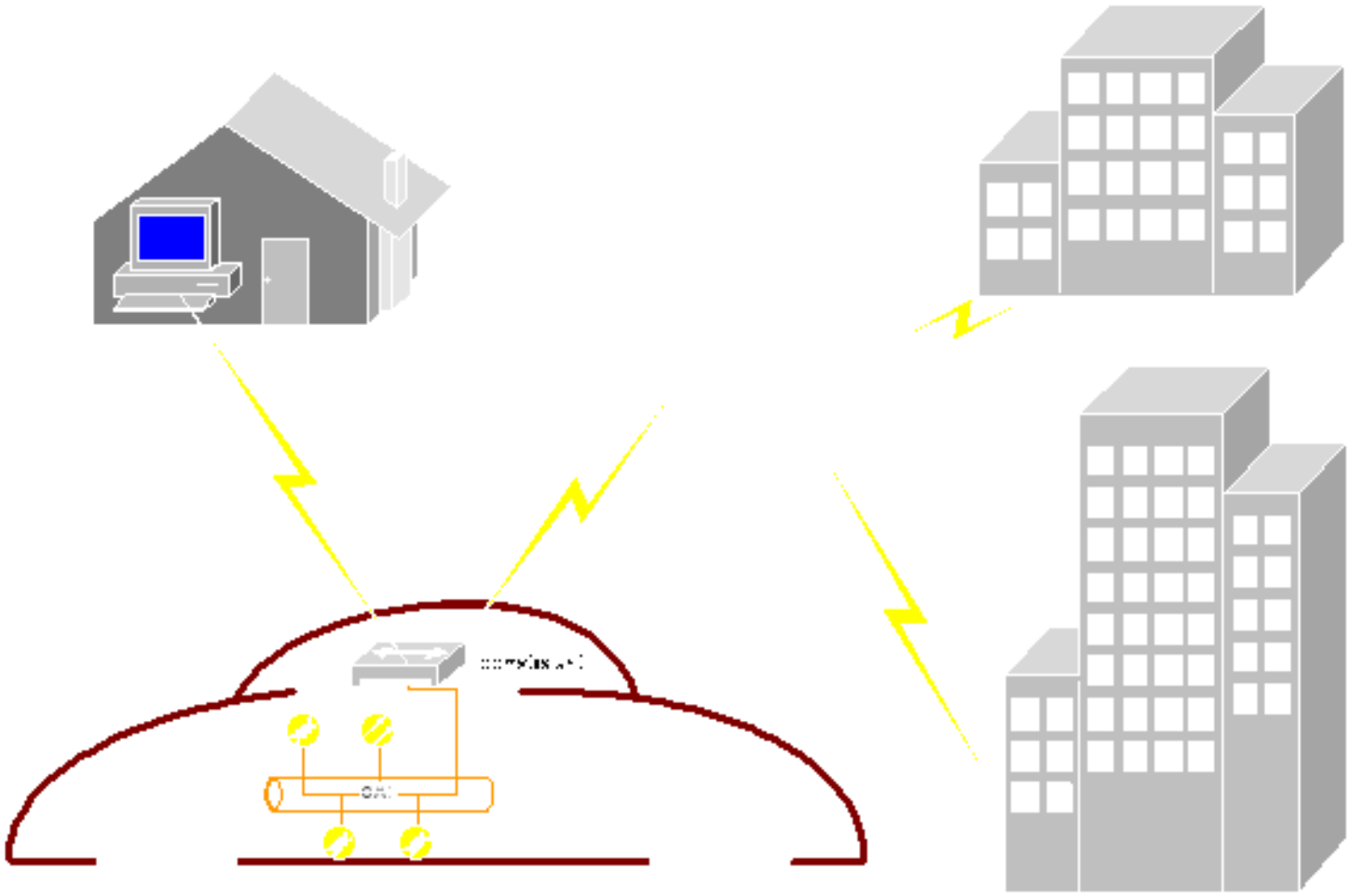

Figure 6 An illustration of an automotive system in which small devices on a CAN bus offer services. They can be found and used by the on-board telematics unit, and then exposed via a standard communications channel (e.g., Bluetooth or 802.11b) to other interested parties. Here we have shown services being pushed up to your home, your service center, and the vehicle manufacturer.

It is interesting to explore this scenario a little further. In the design of this system, it seems sensible to expose different services to different end users. The information that the owner wants to see is probably different from what the service technician and the manufacturer need. Since the telematics unit can act as a Jini client, it might be interesting to create a meta-service on it that discovers and uses the services from the CAN bus devices, processes the information supplied, and distills it into a service suited to the owner. Information that the telematics unit holds, such as the auto's maintenance record, could be combined with the CAN bus data to provide an up-to-the-minute report of the vehicle's status. Even better, that information could be exposed to a service center to be analyzed. There, the vehicle's maintenance plan could be dynamically updated and provided to the owner.

The manufacturer will have interests that differ from both the owner and the service technician/center. For example, the manufacturer may wish to interrogate the vehicle to see what subsystems are present and their version numbers. Such information might be useful for performance tracking, quality control, or even for recall and repair notices.

In all these scenarios, the Jini clients don't have to worry about the peculiarities of the CMatos-enabled devices. The services provided hide (encapsulate) the details of the particular CAN bus messages; the client needs only look for services from a specific class of device. That means that the client software doesn't need to know about the large number of vehicle/sensor combinations. Put another way, the client doesn't need to have the software drivers needed to talk to all the possible sensors and subsystems for every vehicle it's likely to encounter. CMatos ensures that the client gets the right

software when it needs it.

5.2 Home Automation

The “Smart Homes” being built today and in the future include many smart devices. Embedded, networked processors are found in every system and room in a house—from light switches to kitchen appliances to heating and air conditioning systems. Home gateways that connect to these devices are becoming common. Through them, services can be offered and discovered. For example, with your utility company you can monitor and manage your energy consumption. Using CMatos, monitoring and control devices can be found and used by you on your home or work PC, or anywhere else using a PDA or cell phone. CMatos controller services for your appliances can be linked to energy price monitoring applications run by your supplier to ensure that specified tasks, such as dish washing, water softening, etc, are run when prices are lowest.

Instead of speculating on all the various combinations of services and applications, let’s look at a simple control service example. In it, a room in your house will offer a hierarchy of services to a simple PDA Jini client.

[6]

Imagine you have a PDA that runs a simple Jini client. The client “looks” for services that have a “room” attribute . A simple module embedded in the wall of each room or plugged into a convenient power outlet (alá X10) provides the room service. The module supports only one service that describes the room.

Walking into the house, your PDA presents a simple dialog as shown in Figure 7. Several room services have been discovered and listed. Let’s say you select (click on the door of) your home office.

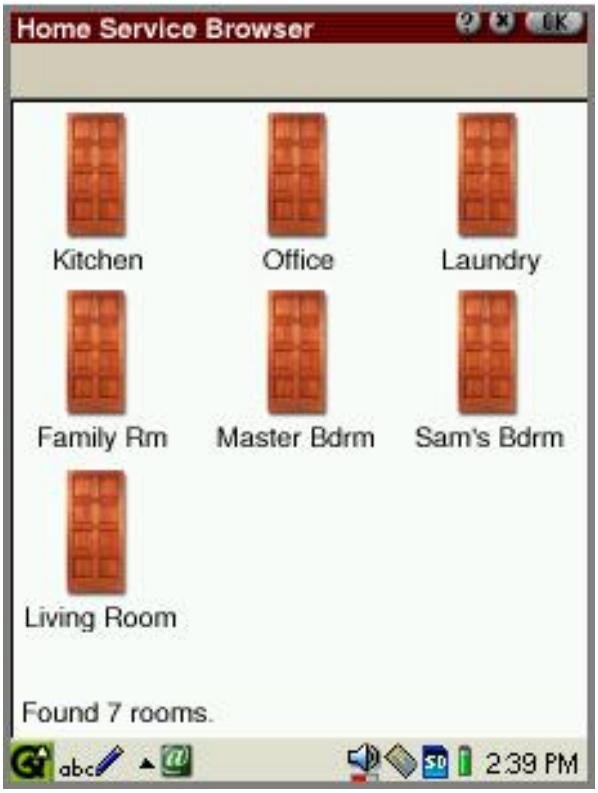

Figure 7 A Jini client application running on a PDA. It has been designed to discover services that possess a “room” attribute. In the example, seven such services have been discovered.

The PDA downloads the service object that represents your office. It's a simple service that does two main things. First, as illustrated in Figure 8, it draws a simple floor plan of a room, in this case, your office.

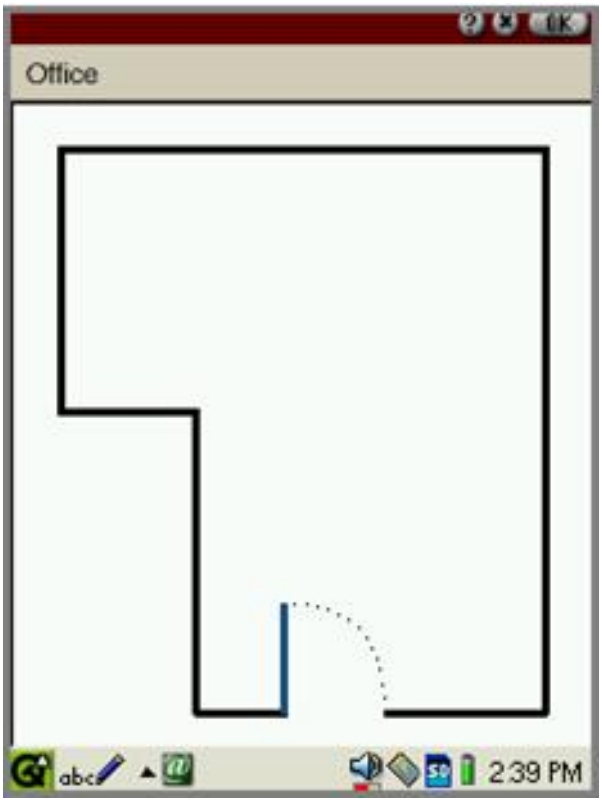

Figure 8 The floor plan of your home office as initially presented by the selected room service.

Second, the room service is itself a Jini client; that is, it searches for services that are associated with this particular room. Such services might have a named attribute called "Office", for example. In Figure 9, we see that the room service has found three lights, one music source, and a desk. The desk appears to have a service associated with it, namely a lock. Note that each service found will provide it's own UI. The lights each offer an icon to represent their service plus the light's coordinates in the room.

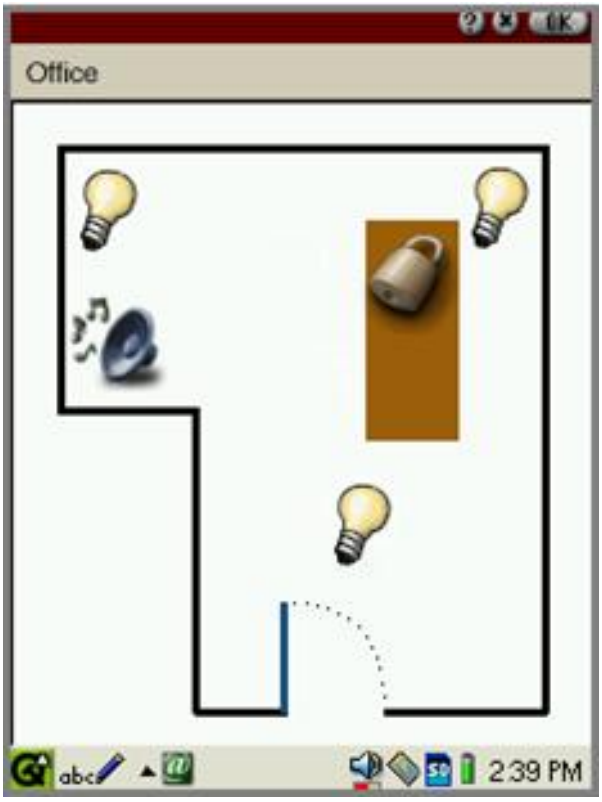

Figure 9 The room service has found several other CMatos services associated with your office; three lights, one music system, a desk and embedded in the desk some kind of lock.

Hmmm, how does a particular light service know its coordinates? There are several simple ways to have that information associated with the service. Imagine that you buy a new lamp for your home office. Once home, you put it in the office, plug it in, and turn it on. From the factory the only thing it knows how to do is offer its service *without* a room association or room coordinates. You turn on your PDA, fire up the Home Service Browser, and open your Office service. It's configured to look for all services associated with the office and for any services that aren't associated with any room (such as your lamp). The new lamp service pops up, and you drag its icon to the proper place on the room layout. That information is fed back into the lamp service by the room (Office) service. The lamp stores the coordinates and its new association with the Office.

Pretty simple! Sure, there are some other issues to deal with, such as what happens if you move the lamp to another room, but the principles are straightforward.

Now, when you select a light service, the light is turned on. Being a simple service, the light service provides simple feedback. Say you selected the light icon in the middle of the room that's associated with your main room lighting. When the light is on, the icon changes state as shown in Figure 10.

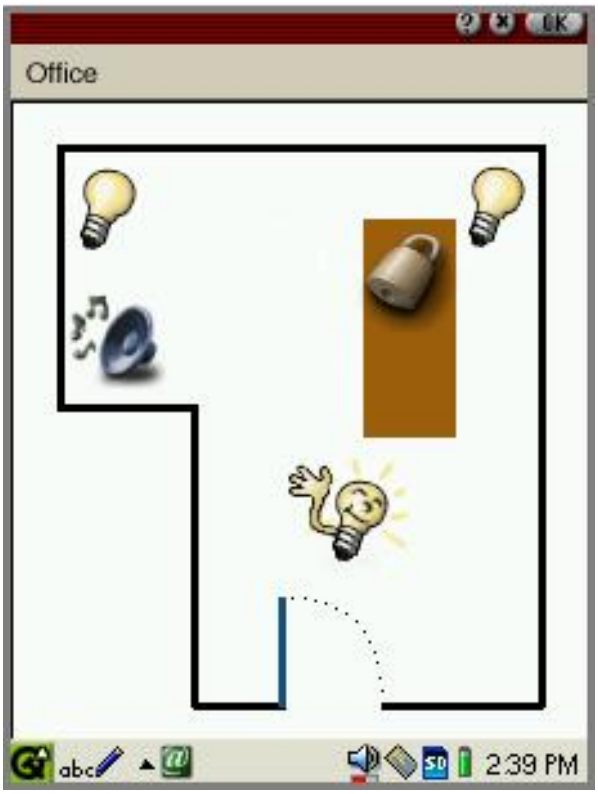

Figure 10 After clicking the light service icon in the middle of the room, the main room lighting is switched on and the service updates the icon to represent the state change.

The other services offered are a little more complicated. For example, selecting the music service might pop open a dialog that allows you to control your music system. The dialog would be specific to the system manufacturer and, since it's an uploaded Java service, neither the room service nor the PDA need to care about its details.

Likewise, the lock service might bring up a simple dialog asking for a password to unlock your desk, or it might be more complicate, like a voice recognition challenge. It really doesn't matter to the room service since it's just a container for the other services.

Last, consider a residential gateway instead of the PDA. It's really no different except that it allows the services to be exposed to the outside world. As discussed right at the start of this section, that might be very useful to service providers, such as your utility company.

This example illustrates the beauty of Jini technology. Services can be deployed where they are needed without the need to reconfigure the network.

With CMatos technology, you can bring into your home any device that supports a Jini client and it will discover your room services and allow you to control things. Best of all, with CMatos you don't have to spend a fortune just to have such capabilities—its cost is in line with the cost of these simple, household devices.

5.3 Industrial Controls

From the previous application area discussions, a pattern should be emerging in your mind. Small, CMatos-enabled devices are able to offer interesting services to any device that is a Jini client. So long as that device has access to a network to which the CMatos device is connected, it can obtain the service.

In this last example application discussion, we look at two aspects of CMatos use. First, we describe the idea of exposing Jini services at all levels of an organization—from the shop floor where CMatos enabled devices gather information, control machines, and do many other things, right up to the main corporate information system. Second, we explore a few questions related to cost, legacy systems, and distributed control.

5.3.1 Hierarchies of Services

Using Jini technology, services from equipment deployed in the field, on the shop floor, or anywhere else we might imagine, can be deployed wherever it makes sense. Not only that, intermediate systems can capture services from lower systems, analyze the information, and create meta-services to be pushed up through the higher levels of an organization.

Devices providing services can register them with Lookup Services at different levels of the network. Lookup Services can register their own services (ServiceRegistrars) with other Lookup Services. Thus, a hierarchy of services can be created. Applications can capture and use these services from whatever level is appropriate. Moreover, as the system changes, for example, due to re-deployment of equipment, computers, sensors, etc., these changes are reflected in the hierarchy by the change in registered services. The system dynamically responds to changes in state.

Consider the architecture illustrated in Figure 11.

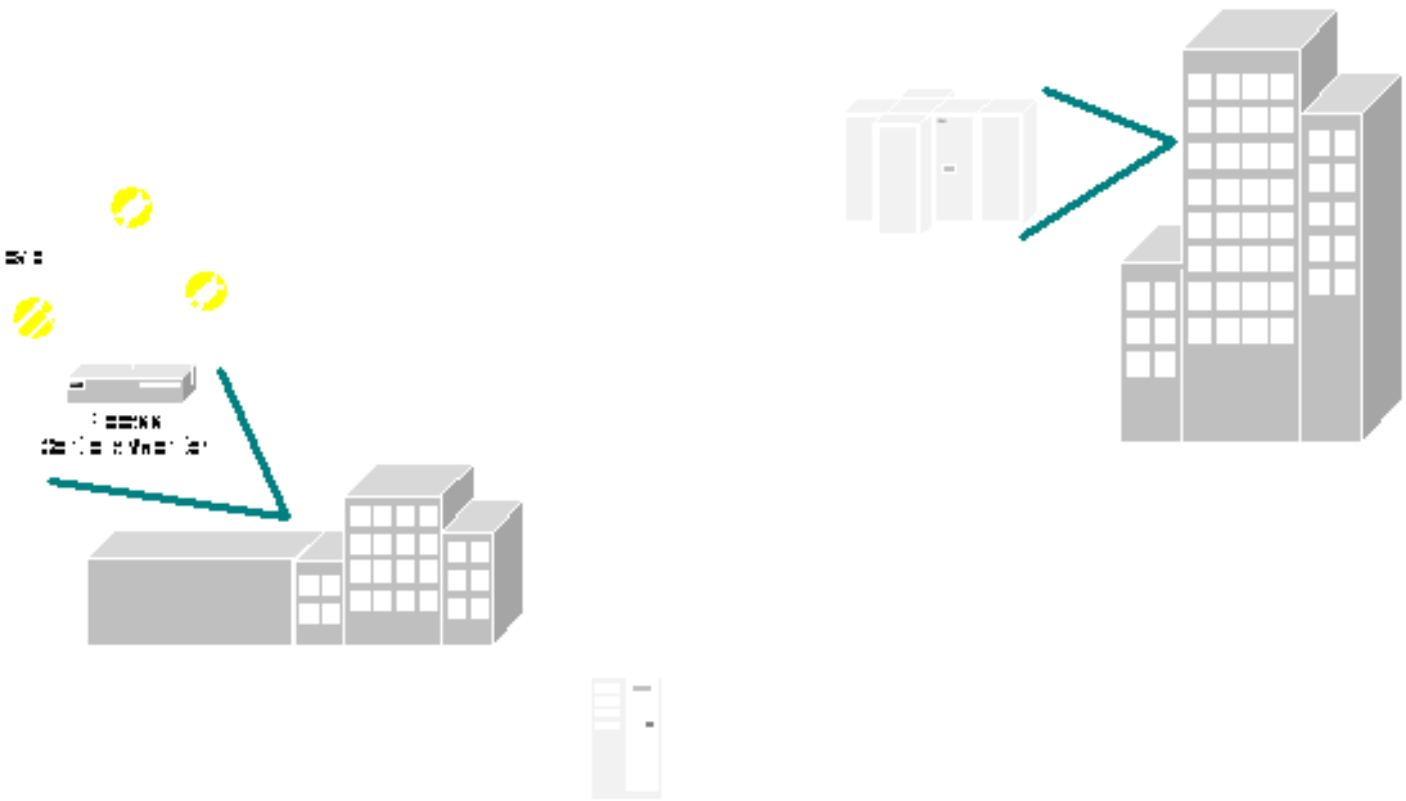

Figure 11 An architecture illustrating how Jini services can be deployed, aggregated, analyzed, and propagated in an organization.

At the left side of the figure, a manufacturing plant is shown. In it there is some kind of industrial process, such as an

assembly line, a chemical processing system, etc. A process monitor is connected to several sensors, controllers, and equipment line cards. Each device offers one or more services via CMatos. Some of the devices, like the line card shown, may even support JMatos and be Jini clients. Even at this low level of the system, such devices are able to function autonomously and use the services of their peers. This provides a greater degree of distributed processing and system robustness—the system is able to operate even without the process monitor!

At the first level of the system, an important client of the device services is the process monitor. It analyzes the information provide by each and presents the results as its own Jini service. If the devices connected to the monitor don't directly support TCP/IP communications, we assume the monitor provides the necessary bridging service (see the discussion in subsection 4.2). Regardless, the devices' services and the monitor's service(s) are available to locally interested user on the same network as the process monitor. One such user has a laptop on which she is running an appropriate Jini client.

It's important to note that even at this level, there are very strong advantages to using Jini technology. They all boil down to the ability to handle change. A technician working in the plant doesn't have to have his equipment updated every time there is a manufacturing line reconfiguration. If sensors, controllers, or other pieces of equipment are changed, it doesn't matter to the technician and his equipment. The services associated with the changed equipment will be discovered and used just as before. This applies to new equipment as well as legacy systems. The abstraction and encapsulation provided by using Java-based services means less time and effort is spent reconfiguring and testing the line.

Consider next the monitoring and control systems in the plant. They are the same as the process monitor. They can find and use the services of the process monitor and its associated devices. Some example functions include load balancing, maintenance monitoring and scheduling, inventory control, environmental monitoring and control, and even dynamic task reassignment.

In addition, the plant can aggregate and repackage information and services for other systems at a higher level. These new services are offered across a WAN, or the Internet, and may be consumed by the systems at the company headquarters.

In this way, as little or as much of the lower systems services can be exposed to the higher organization levels as desired. The key to getting services from the lowest level equipment is to use CMatos. It's the right tool for the job.

5.3.2 Other Considerations—legacy systems, standards, and distributed computing

A challenge that all new technologies face is adoption. Standards, de facto or otherwise, are by definition used widely. However, introduction of a standard can be expensive. Sometimes a lot of infrastructure is required, as is the case with cellular technologies for example. Often legacy systems are already in place. Replacing them outright is expensive and time consuming. Other considerations such as industry practice, coding standards, retooling costs, and such can stand in the way. People start to wonder if there isn't an incremental way to introduce new technologies and reap incremental benefits.

In an industrial automation setting legacy systems are a fact of life. Many systems work well enough in isolation, but could benefit from greater awareness of their context and peers. Likewise, control and management processes could stand to have greater access and control to subsystems. Jini services can really help to achieve these aims, but come at the cost of fitting a JVM onto the subsystems. That can be expensive, or perhaps not even possible in the case of legacy systems—the resources just aren't available.

CMatos avoids the need for a JVM. Java-based services can be made available from any device that has a processor with just a small increase in memory. Since many systems aren't designed to use their processor's full address space, adding memory usually involves only a small hardware change, or none at all—no additional hardware is required.

Since CMatos is written in C, and can be written in other languages, current coding and implementation practices and standards can be followed. Situations where Java can not and has not been used because of concerns over real-time

performance, reliability, and so forth, are not a concern. No Java runs on the CMatos-based system—but the benefits of Java are made available to client devices in the system.

With the above considerations in mind, we can see how it's possible to incrementally introduce Jini capabilities into a system while preserving legacy function. At the highest levels the task is straightforward; Jini clients and Lookup Services can be introduced at the enterprise level using Sun's freely available systems. As we work down the organizational hierarchy, the strategy becomes one of introducing JMatos to systems that have JVMs and suitable resources already. The key benefit is using CMatos for the large number of other, resource-constrained systems and devices, including those that are legacy. The introduction to Jini technology can be done first for the systems of highest value, and attrition or replacement strategies used for other systems.

Using Jini technology also provides better distribution of computing, control, decision-making, and logic. In the example of the previous subsection, a mixture of CMatos- and JMatos-enabled devices was considered, with the assumption that, in general, CMatos will be on the lowest-level devices. A JMatos device can be both a Lookup Service host and a Jini client. That means that it can be aware of services offered by other LUSs and use them. This allows designers to consider highly distributed and autonomous systems of devices capable of peer-to-peer interactions. The result is a greater distribution of function and intelligence and ultimately a more robust the system.

The benefits gained by employing Jini, JMatos, and CMatos technologies are ultimately manifest as reduced capital and operational costs.

6 Conclusion

As In a world where most people have a very hard time configuring and managing their computing devices, and in which ever more computers will be embedded in ever more things, Jini network technology is the best way to get our devices to work between themselves, for us, and not us for them.

CMatos is an extension of the JMatos implementation of Sun Microsystems' Jini technology. Like JMatos, CMatos helps resource-constrained devices offer Java-based services to other devices. Written in Java, these services can be run anywhere. Using Jini technology, these services can be discovered and used everywhere. Unlike JMatos, CMatos is implemented using a non-Java programming language. C has been used for PsiNaptic's initial implementation, but just about any other language would do as well.

The big advantage in using a non-Java language is the size of the implementation. A total CMatos system requires less than 60 kilobytes of storage and can be run using an 8-bit (or even 4-bit) processor! This means that the cost of the system can be kept very low.

In a world where billions of inexpensive, embedded processors are sold every year, CMatos is the only technology that will allow them to share information and services autonomously. That is, CMatos helps these devices do things on our behalf without bothering us unnecessarily.

CMatos devices work for us, not us for them.

7 References:

[1] Frank Sommers and Bill Venners, "Jini-talk with Jim Waldo—Full Transcript,"

JavaWorld, http://www.javaworld.com/javaworld/jw-11-2001/jw-1121-waldointerview.html , November 21, 2001.

[2] U. Hansmann et al., Pervasive Computing Handbook, Heidleberg, Germany: Springer-Verlag, 2001.

~~[8]~~[3] K. Arnold et al., K. Arnold, et al, *The Jini Specification*, Reading, Mass.: Addison-Wesley, 1999.

8 Acronyms

Definition	Description
CAN	Controller Area Network
CDC	Connected Device Configuration
CLDC	Connected Limited Device Configuration
EPROM	Erasable Programmable Read-Only Memory
LED	Light Emitting Diode
OS	Operating System
PC	Personal Computer
PDA	Personal Digital Assistant
PPP	Point-to-Point Protocol
J2ME	Java 2 Microedition
J2SE	Java 2 Standard Edition
JVM	Java Virtual Machine
LAN	Local Area Network
LUS	Lookup Service
PAN	Personal Area Network

PDA	Personal Digital Assistant
RAM	Random Access Memory
RMI	Remote Method Invocation
TCP/IP	Transmission Control Protocol / Internet Protocol
TTL	Transistor-Transistor Logic
UART	Universal Asynchronous Receiver/Transmitter
WAN	Wide Area Network

9 Legal Notice

---

[1]
JMatos is PsiNaptic’s Java-based implementation of Sun Microsystems’ Jini network technology.

[2]
To be sure, some people upgrade to new devices to get new software. Sometimes the features are compelling; sometimes they just want the latest thing. Regardless, this doesn’t really solve the problem.

[3]

J for Java, and the Greek word *Matos*, for self-acting, as in “automatic”.

[4]

Embedded Linux is assumed for the JVMs requiring an OS. The hardware CLDC implementation can be any current Java chip or core with minimal libraries. The J2SE runtime footprint was measured for release 1.3.0_02 on Windows 2000. The Personal Java footprint is inline with that described in Sun’s technical note at http://java.sun.com/products/personaljava/MemoryUsage.html . The CDC and CLDC footprints are consistent with Sun’s J2ME specification.

[5]

The communications channel is RS232. A transceiver is used to bring the signal levels to RS232. It may be eliminated if TTL levels are acceptable for the application.

[6]

Jini services can have associated attributes. This is one way of differentiating service types.