Operating Room Sensor Network Design Document

Greg Harfst
gch@lcs.mit.edu

John Guttag
guttag@lcs.mit.edu

March 29, 2001

1 Introduction

The frequently envisioned "operating room of the future" demands a new medical device network. Such a network allows software applications, running on general purpose computers, to access medical devices. Doing so engenders technological improvements, including simplified hardware, and stronger applications. In turn, the technological advance facilitates improved medical care.

In this document, we design a network architecture focused on placing extremely simple sensors on a network, and making their data available to network applications. The architectural components are designed to make the connection and distribution processes both easy and reliable. Also, the design contains sufficient detail to support the implementation of a prototype. In particular, we construct a prototype that supports an intravenous blood pressure (IVBP) device.

First, in section 2, we describe the architectural components. Section 3 explains the principal interactions between those components during normal operation. In section 4, we describe how the system reacts to component failure and works to prevent human error. Section 5 provides additional details necessary to implement a prototype. Finally, we give concluding remarks in section 6.

2 Architectural Components

The network's components fall into two general categories, based on their proximity to the patient. Those close to the patient are smaller and of minimal complexity. The rest reside on powerful computers linked by a high bandwidth, wired, IP network.[1] The patient's components connect to this main network via reliable, possibly wireless, links. Figure 1 depicts the network's overall layout and primary interactions.

Subsections 2.1 and 2.2 start the section by discussing *sensors*[2] and *gateways*, which combine to transfer sensor data to the network. Next, subsection 2.3 introduces *device proxies*, which provide an API for sensor data to client applications. Subsection 2.4 presents the network *resource manager*. Then, subsection 2.5 closes with a description of *client applications*.

2.1 Sensor

Today's medical devices typically consist of a special purpose machine that processes signals from its own sensors. Each sensor measures some physical quantity and converts it into an electric signal. The special purpose device then analyses the data and displays some information to the

[1] We select IP (Internet Protocol) because of our desire to use existing networks, not because the protocol offers any any particular technical advantage.

[2] All *emphasized* words appear in the glossary at the end of this document.

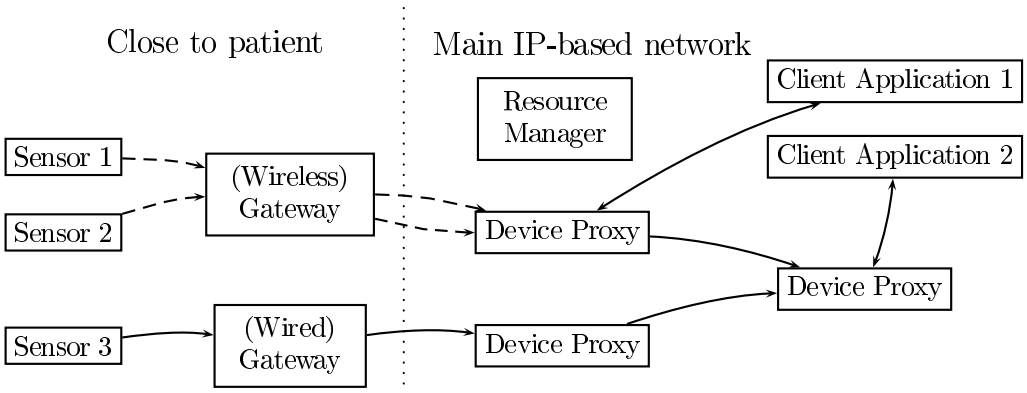

Figure 1: An over-simplified picture of the network infrastructure's components and their interactions. The arrows indicate the direction that information primarily flows. Dashed lines are wireless links. The resource manager might communicate with any other component, so its arrows are not drawn.

medical personnel. Example devices include the electrocardiogram (ECG), and the intravenous blood pressure device (IVBP). The sensors for some of these devices are remarkably simple, with the complexity contained in the way the signal is analysed by the special machine.

Our architecture makes the sensor data available to any application on the network.[3] Doing so breaks open today's restrictive paradigm of closed boxes providing point-solutions. Placing sensor data on the network encourages the multiplexing of expensive resources, such as display monitors. Networking sensors also makes sensor fusion possible.

At the same time, the architecture aims to maintain or increase sensor simplicity. We work to prevent the sensor from doing more than generating its unprocessed electronic signal. Power consumption, and hence the complication of computation and communication are kept to a minimum. To this end, the digitized signal is sent over a simple, short range, unreliable, link technology to the gateway. Whenever possible, this link is wireless to reduce cable clutter. Examples of such links are Bluetooth or wireless RS-232.

Sensors also must contain a minimum amount of state and control functionality to participate on the network. The mandatory state is limited to two IDs that specify their general type, and identity within that type class. We term these the type ID and unique ID, respectively.

So that sensor type information remains updated, the type ID is mappable to a manufacturer-maintained information repository. For example, the type ID might indicate an XML file that is accessible on the manufacturer's web server. The network components could then retrieve the document to learn about the sensor. For performance and reliability reasons, the document should also be cached locally.

2.2 Gateway

The simplicity of our sensors requires an external device to place their data on the main, IP based network. This component, termed a gateway, provides several important functions. First, the gateway supports reliable communication over IP on the main network, which allows it to forward sensor data packets. The gateway also timestamps data packets, and therefore enables sensor fusion by other networked processes.

Typically, several sensors transmit their data to a common, shared gateway. The gateway software runs on a small, yet moderately powerful device. Its small size allows it to be worn by

[3] We are ignoring security, for now. So please do not let the phrase "any application" scare you.

the patient. The gateway's computational power allows it to multiplex data from several sensors and transmit their data reliably out its network link.

Gateways worn by a patient also contain general data about that patient, such as their name or hospital ID. In this way, the gateway functions as a smart ID tag. As section 3 explains, the patient meta-data will prove useful in helping applications find network resources.

Since some devices, such as imaging equipment, require high data rates, we have multiple logical gateways. The high data rate devices communicate through wired gateways, while others connect to a wireless gateway. Regardless, the clocks on all gateways are synchronized using a standard protocol. The required precision of the clocks and their synchronization is still undetermined. Sending multiple sensors through a common gateway will mitigate the synchronization issue.

To prevent any gateway from becoming a single point of failure, we provide multiple, redundant gateways for each logical gateway. The gateways work in concert, balancing sensor load and offering multiple communication paths to the network. If one gateway fails, then a sibling gateway quickly takes its place. Except when the intended meaning is unclear, we conceptualize the collection of redundant gateways as a single gateway.

The concept of multiple gateways raises a number of challenging technical questions. The first is of assigning sensors to gateways. More precisely, suppose n sensors, with transmission requirements d_1 through d_n, and m gateways. How should each of the n sensors be assigned to $l \leq m$ gateways, so that load balancing, power consumption, and communication latency are optimized. A second problem is how to achieve failover among the m gateways. Successful failover includes a number of issues, such as how to share state, recognize failure, and transition connections.

2.3 Device Proxy

Device proxies represent the internal workings of a typical medical device. Consisting of software applications running on powerful networked machines, they embody the information synthesis performed inside today's special purpose machines. Each device proxy provides a public interface for applications that allows them to access the device's information. For example, an ECG device proxy receives data from ECG sensors. Applications then connect to the ECG proxy and retrieve the raw data, the ECG waveform, or some other ECG metric.

Some device proxies receive sensor data via gateways. To limit gateway complexity, it forwards each sensor's data to exactly one device proxy. The device proxy then distributes the sensor data to other applications, according to the proxy's public interface. The same device proxy might also analyse the data and distribute the derived information.

Not all device proxies receive sensor data directly from gateways. *Virtual devices* receive input from other device proxies, as well as alternate information sources. Receiving data from a collection of devices allows the virtual device to perform sensor fusion. While virtual devices are conceptually unique, client applications cannot differentiate between standard and virtual device proxies.

Since each device proxy provides a device-specific interface, we expect the device, or sensor, manufacturer to develop the device proxy software. At a minimum, that software will allow applications to receive the unprocessed sensor data. This will allow innovative developers to use the data for their own novel purposes. Most likely, the device proxy will also add additional value by performing its own analysis on the sensor data. Applications will then be allowed to retrieve the processed data as well.

All device proxies must handle data buffering in a well defined way. More specifically, the "next data packet" might be defined with respect to either sequence or time. When defined according to sequence, the next packet is the one generated immediately after the previously transmitted packet. Otherwise, the most current reading is desired. Each type of device proxy will solve this problem differently.

2.4 Resource Manager

A device network contains many resources that are vital to the network's proper operation. While one might conceive of a system that manages these resources in a distributed, ad hoc manner, we choose a centralized approach. Centralized resource management offers simplicity, yet increases the risk of singular failure points. Thus, we must carefully ensure high availability and fault tolerance.

The network's resource manager tracks several critical resources. The resources include types of components, plus their instantiated properties. Associations between components are also managed.

The resource manager contains information on every sensor type that might join the network. Some data, such as the sensor's manufacturer's name, are static. The rest consist of questions to be answered when an instance of the device type comes online. Answers to these questions provide information about where and how the sensor is installed. This might include the name of the patient being monitored, for example. In addition, the installation information might include communication parameters, such as the transmission time or frequency. The answers also determine which device proxy receives a sensor's data.

Device proxies are also a managed resource with static and dynamic data. What inputs a device proxy accepts, and how the proxy software is instantiated are examples of static information. Dynamic information relates to what services the device proxy provides at any given time.

The other major dynamic component is the device proxy's address. The information service stores both a network address and a name handle for each device proxy. The name handle provides a level of indirection for applications. This allows a device proxy to change its network address, in case of failure, for example, while maintaining its name handle.

The user is the final resource category. The resource manager must provide an input mechanism that allows the user to enter commands and data. Other user information includes identities, groups, and permissions.

The resource manager enables much of the architecture's functionality by providing a variety of services. These services ease the networks use and strengthen its reliability. Ease of use results from services related to tracking and reporting the network's current resources. For example, the resource manager allows applications to find device proxies by describing desired services. Other resource manager services bring new sensors, gateways and device proxies on the network. Finally, the resource manager provides services that help the network recover from component and user error.

We introduce all of the resource manager's services by describing its four functional units. Subsection 2.4.1 discusses the *update service*, which handles network change. Next, subsection 2.4.2 presents the *query service*, which resolves queries from applications. Following that, subsection 2.4.3 introduces how the *information service* manages persistent state information. Finally, subsection 2.4.4 describes the resource manager's user interface, the *control panel*.

2.4.1 Update Service

The update service encapsulates all of the resource manager's functionality associated with responding to network change. It plays an integral role in the process of registering new sensors. It also processes update messages from device proxies. In addition, the update service performs actions such as starting and stopping other applications. Section 3 provides explicit details on its duties.

The update service works closely with the resource manager's information service and control panel. Most network changes are recorded into the information service. Further, the information service stores static information that the update service uses to complete its tasks. The update service uses the control panel to obtain information or guidance from the user. Section 3 contains

more information on the update service's responsibilities.

2.4.2 Query Service

Applications query the resource manager to find device proxies and learn about the network. The resource manager also resolves network names to addresses. The query service processes all of these queries, effectively defining the network's query language.

The query language is expressive, allowing applications to find device proxies based on descriptions of their properties or services. For example, the query language can express queries such as "What is the address of the IVBP connected to Mary Burnett?" To achieve this, we use a subset of XML[4] as our query language.

The query service resolves queries only using data stored in the information service. After understanding the query, it retrieves all of the necessary information to formulate an answer. It then composes a response and sends it to the requester.

2.4.3 Information Service

The information service manages all of the resource manager's persistent state. Persistent state encompasses all the information that is not easily regenerated. This includes data about the sensor types and instantiated device proxies. The resource manager's other services then use the information to enable network operation.

The information service also records all network communication. Since our main network runs over wired Ethernet, the resource manager is able to listen to, and record, all traffic on the network. This record can then be analysed later to investigate errors, or generate new solutions. Since this data accumulates quickly, it must be stored carefully.

The information service must be designed with features similar to a database. This includes watching for the entry of conflicting or contradictory information. It also must save all transactions carefully so that state can be restored in the case of failure.

2.4.4 Control Panel

The control panel is the resource manager's user interface, providing two main functions. First, it allows the resource manager to communicate with the user. This includes displaying messages and asking the user questions via forms and dialog boxes.

The control panel also allows the user to input commands and hence manage the network. For example, the control panel can display all of the data stored in the information service. This includes all of the device proxies participating on the network. The control panel also allows the user to perform tasks such as opening and closing applications.

2.5 Client Application

Client applications allow users to interact with the network and its resources. The applications can run on any network accessible computer. A typical application obtains data from one or more device proxies and then displays information to the user. For example, an IVBP client application might display a patient's blood pressure on a monitor in the operating room.

3 Component Interactions

With each of the components described, we give a detailed explanation of their interactions under normal operation. First, we discuss the process of registering a new sensor on the network. Next,

[4]Extensible Markup Language

we cover how a client application collects and displays data from a device proxy. Finally, we describe the process of removing a device proxy or sensor from the network.

3.1 Registration

A sensor must register on the network before it participates. Registration allows the resource manager to learn about network change, and connect the sensor with a device proxy. After elaborating on these goals, we detail how the actions are carried out.

3.1.1 Registration Goals

Gathering information about all new sensors enables the subsequent processes of connection establishment, resource discovery, and security. The collected information pertains to the general sensor type, as well as the particular installation instance.

The resource manager obtains general information by resolving the type ID. These data indicate the sensor's capabilities and specifications, such as transmission rate and link technology.

Instance information includes where and how the sensor will be connected to gather its readings. Often this includes the patient name, or what device will receive the data. User dialogs, or other instance methods, might be employed to collect the data.

Connection establishment involves associating the sensor with a collection of gateways, plus its device proxy. The selected gateways optimally support the load and provide redundancy. The device proxy must be appropriate for the sensor's type and instance. This might require instantiating a new proxy, or notifying an existing proxy of the new sensor.

The resource manager records all the steps taken and decisions made. The data saved includes the time of registration, and what services the device proxy now provides. For new device proxies, its address is kept.

3.1.2 Registration Procedure

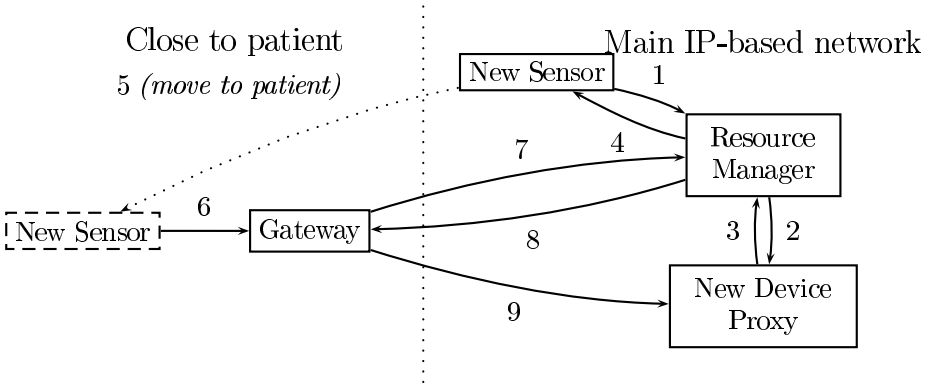

Figure 2: The registration process. The sensor first contacts the resource manager directly, and then later connects to the gateways. The numbers indicate the message passing sequence.

Registration begins when the new sensor contacts the resource manager through a special registration gateway. The registration message includes the sensor's unique and type IDs. These IDs allow the resource manager to begin collecting information about the sensor.

After gathering information, the resource manager associates the sensor with a device proxy. This association procedure might include starting a new device proxy. In any case, the selected device proxy should expect to receive messages from the new sensor some time later.

The new sensor must also be associated with a collection of gateways, which are presented to the medical staff. The specifics of the indicated gateways depends upon the collected information. At a minimum, it includes the bandwidth requirements. When data such as the patient name has been collected, then the set of appropriate gateways is further limited.

When information gathering and associating complete, the resource manager sends an acknowledgment back to the sensor. Later, the sensor is installed and connected with the gateways. The primary gateway then queries the resource manager for the address of the sensor's device proxy.

Upon receiving the query, the resource manager validates that the sensor has connected to the proper gateway. In the case of an error, it alerts the medical staff. In addition to verifying the gateway association, the resource manager might gather information. In particular, meta-data stored in the gateway might supply the patient's name.

Finally, the gateway connects with the indicated device proxy. After connecting, the gateway forwards on all sensor messages to the device proxy.

3.2 Data Collection and Display

When a client application desires information from a device, or one of its sensors, it first queries the resource manager for the appropriate device proxy's address. The query describes the desired device or service. The returned result contains a list of network names for all matching device proxies. The client application then selects one of these names as the device proxy to contact, and resolves the name to an address.

Using the address, the application contacts the device proxy with a request for data. The format of the request matches a published specification for communication with the particular type of device proxy. Typically, the request includes the type of information desired, and how that information should be delivered.

Meanwhile, sensors continue to push data to the device proxy via their gateway. The device proxy collects the data and uses it to respond to the application's requests. This might entail simply distributing the raw sensor data, or some information derived from that raw data. The way in which the device proxy distributes the information depends on its implementation. It might push the data, distribute the data on request, or use another method.

3.3 Leaving the Network

This section discusses the process of how a sensor or device proxy properly exits the network. See section 4 for a discussion of how to handle components leaving due to error or failure.

First, suppose a sensor leaves the network. Perhaps the sensor is removed from the patient because its data is no longer needed. The knowledge of this change must be propagated to the resource manager.

The sensor's device proxy is responsible for this propagation, since we work to keep gateways agnostic about sensors. The method by which a device proxy detects a sensor's removal depends on the protocol that they share. Possibilities include an exit message, loss of connection, or cessation of data packets.

The device proxy is also responsible for communicating any changes in its services to the resource manager. Further, the device proxy might need to inform its client applications of the lost sensor. For device proxies with only one sensor, the proxy should also begin its exit procedure.

When a device proxy closes, it must inform all dependent components of its intentions. It should first send a terminating message to any connected gateways. The gateway can then close its connection with the related sensors. Just as when a sensor exits, the device proxy must notify client applications and the resource manager.

4 Fault Tolerance

Unfortunately, we can not always depend on every network component to operate as expected. In this section we describe how the network reacts to a component that stops responding, or starts behaving unexpectedly. In doing so we must consider both the cause and severity of the failure. The cause might be related to failure of either hardware or software. Further, errors might be intermittent or persistent.

In the event of failure, the remaining network components should work to restore normal operation as quickly as possible. We also limit a failed component's sphere of influence, so that the number of affected components is minimized. Along these lines, we eliminate single points of failure.

We divide the discussion into subsections that focus on the failure of each of the different components. Subsection 4.1 describes how to replace a malfunctioning sensor. Next, the failover procedure for a gateway is in subsection 4.2. Subsection 4.3 discusses when a device proxy closes unexpectedly. Following that, a crashed client application is discussed in subsection 4.4. Then, subsection 4.5 discusses how the network recovers when the resource manager fails.

4.1 Sensor

Suppose a sensor malfunctions. Since sensors are simple, we assume the hardware has failed and needs replacement. If it fails during the registration process, then we restart the process with a new sensor. This is viable since registration should happen quickly.

Next, consider a sensor that fails during data collection. The failure might be detected by the device proxy or client application. In either case, assume no redundant sensor exists on the network, so that the application cannot simply switch to using the other sensor. In this case, we must replace a sensor.

When replacing a registered sensor, it is often not necessary to perform every registration step on the new sensor. In particular, the resource manager skips the most information gathering steps. Once the new sensor is indicated as a replacement, much of the old information is retained. The replacement sensor is then connected, and connection establishment proceeds as usual.

4.2 Gateway

Suppose that a gateway communicating for one or more sensors fails. This is especially troublesome because the gateway is a single point of failure for all of its sensors. To eliminate this problem, we always have a set of several, redundant gateways. Then, if the primary gateway fails, one of its siblings can quickly step in.

Before a switch happens, the primary gateway's failure must be detected. Detection might occur in a number of ways, depending on how it fails. However the failure is detected, a sibling gateway must be notified of the need to take over, and the failed gateway must be shut down. The shutdown either results from the resource manager sending a message or the user physically manipulating the gateways.

Following these notifications, the actual switch must be negotiated. For the negotiation to succeed, the gateways must continually share their persistent state information, as discussed in subsection 3.1. All of the old gateway's connections must then be migrated to the new gateway.

Such a switch raises a set of challenging semantic issues. For example, we must avoid sending duplicate messages. While these problems are tough, they have been studied.

4.3 Device Proxy

Now consider the failure of a device proxy. In such an event, gateways and applications may experience trouble with their connections to the proxy. Otherwise, the user might detect an

error.

Since device proxies are typically written by outside developers, we do not have full control over how they handle failure. Some device proxies might incorporate their own set of failure procedures. Once a device proxy is started by the resource manager, it takes on the responsibilities of a standard, highly available server. Still, the resource manager provides a suit of services to assist a device proxy in recovering from failure.

For each type of device proxy, the resource manager can store several operations. First, the resource manager could know how to check if the device proxy has failed and needs restarting. Often, the check involves periodically sending a keepalive message. The resource manager also stores instructions for restarting the failed device proxy.

Every device proxy is responsible for handling its own state and re-establishing lost connections. The resource manager can assist in re-establishing connections through its network name resolution service.

4.4 Client Application

When a client application fails, the user obviously notices. Also, the device proxy might lose a connection or be unable to deliver data. After a few failed attempts, the device proxy should give up and close the connection.

The architecture does not provide services to help client applications recover from failure. When a client application fails, it must restart and reestablish its connections. The control panel can be used to restart the client application.

4.5 Resource Manager

If the resource manager crashes, the network is disabled, but not useless. While the manager is down, new sensors cannot join, and applications cannot query for devices or resolve network names. Further, changes to the network cannot be recorded. So if a device proxy changes its services or address, that information will not be recorded until the device manager resumes operation.

Regardless, existing applications, device proxies, and sensors can continue to run and communicate. To that end, they should run on separate hardware from the device proxy.

To resume full network operation, a new instance of the resource manager should be started. Keeping all persistent state in the information service engenders a successful restart because information is not scattered around the entire device manager. If the resource manager restarts at a new location, the other components must find it. They do this using a discovery protocol.[5]

5 Prototype Details

This section contains the low level details needed to guide a physical manifestation of the preceding discussion. The prototype's initial goal is to transfer data from simple sensors to a device proxy, and then client application, running on the network. It also implements the basic functionality of every network component, including the resource manager.

We organize the discussion into sections based on components. For each component, we describe how it will be created, and specify its interfaces.

5.1 IVBP Sensor

We chose the IVBP as the network's first sensor because of its simplicity. An IVBP sensor employs basic circuitry, connected to an intravenous sensor, to generate a differential voltage

[5] Or by having a list of locations to try.

signal. Additional hardware amplifies and digitizes the analog signal. This digital signal is what we wish to send over the network.

A PIC processor captures the digital signal and creates packets to send out its communication link. Initially, we use wireless RS-232 as the sensor's simple link technology. This will have to be manually configured in advance. In the future we hope to use BlueTooth, which will provide ad hoc connections.

The PIC also stores the sensor's type and unique IDs. Both IDs are, somewhat arbitrarily, tow bytes in length. The IVBP, as the pioneering sensor, has a type ID of 1.

Field name	Length	Description
Registration		
Msg	1 byte	Message identification (0x00)
TypeID	2 bytes	Sensor's type ID (0x01)
UniqID	2 bytes	Sensor's unique ID
Data		
Msg	1 byte	Message identification (0xFF)
TypeID	2 bytes	Sensor's type ID (0x01)
UniqID	2 bytes	Sensor's unique ID
Len	1 byte	Number of bytes (l) in payload
Payload	l bytes	The data reading

Table 1: The content of registration and data packets sent from an IVBP sensor. Parenthetical information specifies a field's value. Also, I believe the payload length, l, will be 1 for the IVBP.

When a sensor first powers up, it sends out a registration message to the gateway. After this, it begins sending data packets. This consists of reading in a digital signal, forming the packet, and transmitting it out the communication interface. The gateway will drop these packets until registration completes. Table 1 describes the information contained in the registration and data message packets.

The 'Msg' field indicates the message type. This method of indicating the type of message is used throughout the prototype.

5.2 Compaq iPaq Gateway

Our prototype gateway consists of an application running on a Compaq iPaq handheld computer. We choose this device since it is small, yet has both serial and network interfaces. These features allows a gateway to take data readings in from the serial port, packetize them, and transfer them out the network interface.

Using an iPaq eases the development process. First, the Linux operation system has been ported to it.[6] This provides a TCP/IP stack, familiar utilities, and open source code. The iPaq also has plenty of computing power, and an LCD display. Finally, it's add-on sleeves allow it to utilize hardware such as 802.11 network interface cards.

The gateway software is basic and unadorned. We develop it using the C programming language and compile it for the iPaq's StrongARM processor using the public skiff clusters. Its main tasks are to listen for messages on the serial port and network interface. The program must also send messages out its network interface.

As an intermediary between sensors and the network, the gateway presents two interfaces. To sensors, it must receive registration and data messages, as described in the previous section. The gateway also sends two related messages, and receives one, on its network interface. Table 2 defines this network interface. Notice that we only include the first type of registration message.

[6] See `http://www.handhelds.org` for more information.

Field name	Length	Description
Network Registration		
Msg	1 byte	Message identification (0x02)
TypeID	2 bytes	Sensor's type ID
UniqID	2 bytes	Sensor's unique ID
Len	2 bytes	Number of bytes (l) in meta-info string
Meta	l bytes	A meta-information string.
Network Data		
Msg	1 byte	Message identification (0x03)
TypeID	2 bytes	Sensor's type ID
UniqID	2 bytes	Sensor's unique ID
Time	2 bytes	Time stamp
Len	1 byte	Number of bytes (l) in payload
Payload	l bytes	The data reading
Network Registration ACK		
Msg	1 byte	Message identification (0x04)
TypeID	2 bytes	Sensor's type ID
UniqID	2 bytes	Sensor's unique ID
Name	4 bytes	Network name of sensor's device proxy

Table 2: This table defines the messages that comprise the gateway's network interface. It sends the first two and receives the third.

The gateway's network data message includes a timestamp field. This field is set based on the time that packet is received on the network interface. For the prototype we do not worry about synchronizing this clock with others on the network.

The 'Name' field of the network registration ACK message contains the network name of the newly registered device proxy. For the prototype, this is a four byte token generated, and resolved, by the resource manager. When registration fails, these four bytes are all set to zero.

Finally, the gateway stores meta-information about the patient who wears it. For the prototype, this is read from a data file on the iPaq. In the future we might enhance its interface to receive meta-information from the network.

5.3 Resource Manager

The resource manager runs on any networked, general purpose computer. It implements all of the basic features of the update, information, and query services. It also provides a control panel interface to the user.

We choose the Java programming language as the prototyping language. Java provides many built-in features that ease the gateway's realization. In particular, its classes greatly simplify the process of writing networked programs. Further, it is platform independent, and supports advanced graphical interfaces.

The resource manager must accept a variety of messages. For now, we only implement the registration and query parts of the interface. These messages will allow sensors to join the network, and applications to query for devices. Table 3 defines the content of the accepted messages.

When an IVBP sensor registers, the update service must start an IVBP device proxy. The device proxy runs on the same machine as the resource manager. Further, the resource manager specifies the proxy's port, so it need not receive a message with that information. It also passes the sensor identification and gateway's address as command line arguments when it starts the

Field name	Length	Description
Network Name Resolution		
Msg	1 byte	Message identification (0x05)
Name	4 bytes	Network name to resolve
Network Name Resolution ACK		
Msg	1 byte	Message identification (0x06)
IP	4 bytes	IP number
Port	2 bytes	Port number
Device Query Resolution		
Msg	1 byte	Message identification (0x07)
Len	2 bytes	Length of query string (l)
Query	l bytes	The query string
Device Query Resolution ACK		
Msg	1 byte	Message identification (0x08)
Name	4 bytes	Network name to resolve

Table 3: This table defines the resource manager's interface. The registration messages are contained in Table 2.

device proxy.

We also implement a rudimentary query language. Its format follows that of a CGI[7] query string. In particular, it is a text string of 'key=value' pairs. In the future, we will add support for an expressive language such as XML.

In addition to queries for devices, the resource manager resolves network names to network addresses. The network name, generated by the resource manager, is four bytes in length. The network address consists of a four byte IP number, and a two byte port number.

The control panel provides the network operator with a graphical interface to the resource manager's functionality. It displays what sensors and device proxies are currently running. It also the user to access the resource manager's information service. In short, the control panel provides full access to the resource manager's functionality.

5.4 IVBP Device Proxy

The IVBP device proxy receives sensor data packets and distributes information to interested applications. It is implemented in Java for the same reasons discussed above. Further, it runs on the same machine as the resource manager. The messages comprising its interface are in Table 4.

To request, and then receive, data, an application makes a socket connection to the device proxy. The device proxy behaves like any standard server application. In particular, it listens for requests on a standard port number. It then processes the request type and negotiates a persistent connection on an alternate port. The persistent connection is then used to respond to the request, or receive future information.

Both gateways and client applications make socket connections to the device proxy. A gateway connects and then begins sending sensor data packets across the persistent connection. Applications request a type of service, and then receive the desired data via the connection.

The IVBP device proxy provides two services. First, it can forward on every sensor data packet that it receives. The client application can then perform its own processing on that primary data. Next, the device proxy can provide a blood pressure reading every second. In this case, it performs its own processing on the data.

[7] Common Gateway Interface

Field name	Length	Description
Gateway Connection		
Msg	1 byte	Message identification (0x09)
TypeID	2 bytes	Sensor's type ID
UniqID	2 bytes	Sensor's unique ID
Application Service Request		
Msg	1 byte	Message identification (0x0A)
Serv	1 bytes	Number indicating requested service
Sensor Data Response		
Msg	1 byte	Message identification (0x0B)
TypeID	2 bytes	Sensor's type ID
UniqID	2 bytes	Sensor's unique ID
Time	2 bytes	Time stamp
Len	1 byte	Number of bytes (l) in payload
Payload	l bytes	The data reading
Device Data Response		
Msg	1 byte	Message identification (0x0C)
Len	1 bytes	Length of data (l)
Payload	l bytes	The data payload

Table 4: This table defines the IVBP device proxy's interface.

The device proxy does not make any effort to recover lost connections. Either the gateway or a client application must perform the steps necessary to reestablish a connection.

5.5 IVBP Client Application

The IVBP application aims to provide all of the information currently offered by the proprietary display hardware. It displays the pulse waveform, along with the current blood pressure reading. The major differences are that it obtains the sensor data from a network, and hence can run on any computer.

Another exciting version of the application would be a web applet. This would allow the IVBP data to be displayed on any web page that supported Java.

6 Concluding Remarks

The operating room of the future is an exciting concept. This document contains the first run at a specification for a fundamental piece of that concept, the device network architecture. In doing so, we define all of its components, and how they interact. We also discuss the possibility of failure among those components. Finally, we give details to steer the development of a prototype network.

In writing this document I have tried to include all the necessary details. Some topics, such as security, are noticeably absent. This document does not give close consideration to authentication and authorization of identities and commands. Also, we have only considered sensors in this design, and have ignored actuators. Such information must be inserted later.

I am sure other information is missing too. Please get back to me with your comments, so that they can be incorporated into the design. The next step will be to actually implement the components, and see how they interact.

A Glossary

Control Panel The user interface component of the resource manager.

Device Proxy An application that presents an interface to services provided by a medical device. It is owned by the resource manager.

Virtual device A device proxy that does not receive its inputs directly from sensors.

Client Application An application that gathers data from device proxies.

Gateway A piece of hardware that places data from sensors on the IP-based network. A single gateway often receives data from several sensors.

Information Service The resource manager's component that handles all of the network's persistent information.

IVBP An intravenous blood pressure device.

Update Service The resource manager's component that receives messages related to network change.

Resource Manager A primary network component that provides services that make the network easy and reliable to use.

Sensor A device that measures some physical quantity and convert it into an electric signal, that is then transmitted to a gateway.

Query Service The component of the resource manager that resolves queries from applications.