Managing Heterogeneous Sensors and Actuators in Ubiquitous Computing Environments*

Riccardo Crepaldi‡, Albert F Harris III†, Rob Kooper+, Robin Kravets†,
Gaia Maselli*, Chiara Petrioli*, and Michele Zorzi‡
†Department of Computer Science, University of Illinois at Urbana–Champaign
*Department of Computer Science, University of Rome "La Sapienza"
‡Department of Information Engineering, University of Padova
+National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign
{riccardo.crepaldi,zorzi}@dei.unipd.it, {aharris,rhk}@cs.uiuc.edu,
kooper@ncsa.uiuc.edu, {maselli,petrioli}@di.uniroma1.it

ABSTRACT

With the increase in the number of sensors and actuators available to ubiquitous computing systems comes the need for architectures that can support the development of intelligent applications and expose the rich control and monitoring capabilities provided by these devices to users. In this work, we present a description of the parameters used to define services provided by sensors and actuators. Using this understanding of the devices that provide sensing and control throughout the environment, we present the design of Meditrina, a ubiquitous computing architecture. Meditrina provides clean interfaces for application designers to leverage devices in the environment to support the activities of users in a large variety of scenarios. To demonstrate the power and feasibility of Meditrina, we implemented a prototype, including a room lighting control application. Through the use of the prototype, we show how the architecture facilitates the quick implementation of applications that can react to and affect the environment, even in the face of device failure.

Categories and Subject Descriptors: D.4.7 [Organization and Design]: Distributed systems.

General Terms: Design

Keywords: Ubiquitous systems design, sensors and actuators

1. INTRODUCTION

The interest in "smart" technology in homes and work places has only increased with the improvement in and availability of sensor and actuator technology. Such improvements can now support the visions of ubiquitous computing systems that enable intelligent applications to manage automated office or ambient assisted living environments. Although the end goal of these scenarios differ, all ubiquitous computing systems aim to use such technology to monitor, control, and interact with a number of different components of the "smart" environment, from various personal devices, to environmental controls, to the people themselves. The goal from a systems perspective is to support intelligent applications, which aim to effectively and efficiently manage the devices in support of the given application in the given environment. Additionally, the complexity of the underlying systems and the management of the diverse devices must be hidden to enable easy-to-use interfaces and easy-to-design applications. These requirements drive the need for new system architectures to manage the entire environment, down to interactions with the individual sensors and actuators.

The complexity of the activities expected to occur in these diverse environments requires that the applications themselves have the knowledge and intelligence for the targeted scenarios. Given these application-specific demands, we believe that the role of such a ubiquitous computing system is to provide monitoring, control, and interactive information back to the intelligent application in a timely and resource efficient manner. For example, to support such intelligent applications, the system must provide real-time monitoring information about critical systems and feed back alerts in a timely fashion, as well as support interactions with users.

If we look at the system from the bottom up, the complexity of providing the needed services stems from the heterogeneous nature of the devices and communication technologies used to interact with those devices. For example, the underlying network may include a variety of components that differ in sensing, communication, and computing capabilities, including Internet-enabled workstations, RFID tagged objects, personal sensors, and environmental actuators. This heterogeneity introduces challenges such as coordination of sensing and acting capabilities, communication through different radio technologies, connectivity management, service discovery, and network coverage. Some of these issues can be addressed at the networking level, with routing and network management mechanisms that target the presence of different radios and communication paradigms, while other issues, such as the coordination of sensors and actuators, must be addressed at a higher level, with specific mecha-

*This work was partly funded by NSF CNS 0347468 and the Eureka ITEA project AmIE (Ambient Intelligence for the Elderly).

SANET'07, September 14, 2007, Montréal, Québec, Canada.

nisms that discover and match the capabilities of physical devices to application needs.

One of the major challenges in the design of effective ubiquitous computing systems is that such a complex system must be designed in a distributed manner to support efficient resource usage and fault tolerance. The system must enable information collection and device actuation while not exposing the complexity of the system to either the applications or the users. However, the ultimate goal of such a system is to provide the support required by the applications and users. Therefore, the key to success lies in the design of the interfaces to the applications, the devices, the environment, and the users.

The main contributions of this paper are three-fold. First, we present a comprehensive discussion of the components and challenges of a ubiquitous computing systems, with an extensive focus on the management and control of the sensors and actuators. Second, we present the design of a ubiquitous computing architecture, called Meditrina[1], that addresses these challenges and can be used to facilitate a ubiquitous computing solution in an effective and efficient manner. Throughout this paper, we use the example of an ambient assisted living environment to describe the complex interactions between the different components of the system. However, Meditrina's design and implementation provide a generic architecture for any activity-based ubiquitous computing application. Finally, we present our prototype of Meditrina and use it to demonstrate its ability to support applications needing to monitor and affect the environment while providing a simple set of interfaces for application developers.

The rest of this paper is as follows. Section 2 carefully defines sensors, actuators, and communication nodes, which are the fundamental building blocks of ubiquitous computing systems. Section 3 presents our architecture, Meditrina. Section 4 describes our implementation of Meditrina, including a discussion of its goals and results. Finally, Section 5 presents some conclusions and future directions.

2. OF SENSORS AND ACTUATORS

In this section, we present definitions of sensor, actuator and node, including which functions they should perform and which parameters describe their behavior. From these definitions, we can discuss the characteristics that should be exposed to the ubiquitous computing system and how these characteristics can be used to optimize the performance of the system.

There are three main functions necessary to support ubiquitous computing systems: sensing, actuation and communication. A *sensor* is a device that "senses" a physical characteristic of the environment. A sensor can measure different quantities, like light intensity, temperature, humidity, or vital signs. An *actuator* is a device that can perform an action of some kind that in fact changes the environment. For example, an actuator could turn on and off a light or control the air conditioning system to change the air temperature. Finally, a *node* is a wired or wireless device that supports communication in the system. A node acts as part of the network, providing routing and any other services the network should need. Nodes also provide limited data storage and processing capabilities, depending on the capabilities of the hardware. The combination of a node with one or more sensors and/or actuators enables the communication between the sensors, actuators and the other components of the system.

The integration of sensors and actuators into a complex system requires a generalized interface, to support the diversity of the devices. To this end, the interfaces to sensors and actuators should export both *control parameters* and *observable parameters*. The control parameters are related to the general behavior of the devices including communication, energy, and sensing/actuating characteristics. The observable parameters deal with the actual values being sensed or changed by the devices. Although the observable parameters are device specific, it is necessary to understand the set of control parameters for each type of device to enable the ubiquitous computing system to achieve the best performance in terms of supporting application requirements and energy constraints.

For a sensor, the control parameters focus on the behavior of the device as it captures environmental values. *Minimum sample rate* tells the system how frequently a device can be queried. *Accuracy* defines the potential error introduced into the sensing of the observable values due to fidelity provided by the physical sensor. It is also necessary to capture the cost of the sensing action in terms of *delay*, intended as the amount of time that the sensing procedure requires before the data is ready to be sent, and *energy*, which captures the energy consumption of the devices required to sample and perform digital conversion of the data. The final control parameter represents the *coverage area* of the sensor.

For an actuator, *delay* captures the time interval between the command and the action taking place. Actuators also have *energy* and *accuracy* parameters. The *accuracy* of an action must be represented by a scale that captures the effect of the action on the environment. For example, an action can have a guaranteed effect, as in the case of turning on and off a light, or can have a result affected by some level of uncertainty, as in the case of the air temperature setting, whose effect is not guaranteed. An actuator may also have a *coverage area* associated with it.

Nodes, which provide the communication functionality to sensors and actuators, have a set of control parameters that influence network performance. These parameters include the *reliability* of the communication, which depends on the network protocols and on the channel. It is important to note that, while this is certainly affected by the reliability of the device (*i.e*, the probability the device will be functioning over some interval), it also incorporates packet-level errors due to collisions and interference in the channel. The *delay* of the communication to and from the nodes, which depends on the discovery time, the duty-cycle, the hop-count and the routing strategy are two additional parameters. Additionally, for wireless nodes, it is necessary to capture the impact of communication on energy constraints. Therefore, it is necessary to have information about the *energy capacity*, in terms of the remaining capacity and the cost of replacement, and the *effective cost* of the network operations, including discovery and communication.

These explicit definitions of the characteristics of sensors, actuators and nodes highlight the types of interfaces to sensors and actuators that must be designed in a ubiquitous

[1]Meditrina is the Roman goddess of health. The original target application for this architecture was ambient assisted living.

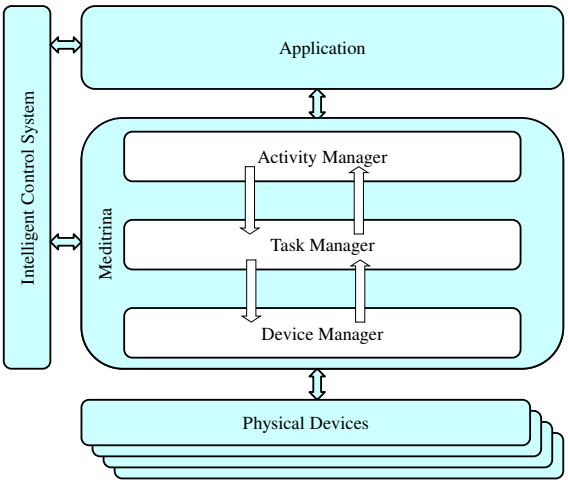

Figure 1: System architecture

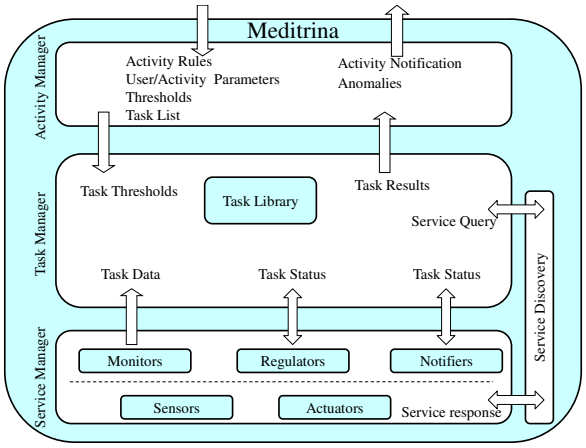

Figure 2: Meditrina architecture

computing system, enabling applications to measure and control their environment. Such an architecture must have the ability to keep track of changes in the availability of the devices in the network as well as the ability to expose these devices for simple use by applications. The next section presents Meditrina, our architecture designed to manage networks of sensors and actuators.

3. MEDITRINA

While a number of architectures for smart homes [1, 3, 9], smart offices [7], and ambient assisted living environments [2, 4, 5, 8, 10, 11] have been proposed, none provide an architecture to support intelligent applications, functioning within guidelines set by users and enforced by the system, through the coordination of heterogeneous sensors and actuators. We envision that these sensors and actuators will not only be distributed in fixed locations throughout the environment, but will also include body-attached devices. In this section, we present Meditrina, which provides just such a system, supporting user activities by providing task management, service discovery and management, device management, and network optimization.

In the design of Meditrina, our ubiquitous computing system, we assume that a number of components are present in the environment (see Figure 1). One of our focus environments is ambient assisted living, and so many of our examples will be drawn from that environment. The highest level component, the application, implements the functionality of the environment. At the lowest level, there is a collection of physical devices, including sensors, actuators, network devices, and personal devices. Finally, there is a control plane, which we call the Intelligent Control System, which facilitates ongoing user activities by providing rules and guidelines within which environmental decisions can be made (*e.g.*, giving safe thresholds for water temperature to be used in a bath).

The intelligent control system is the source of rules that govern actions within the environment. For each activity, it specifies the action to be taken based on the parameters that characterize and personalize the activity for each environment. Violations of these rules are automatically controlled by the system whenever possible. For example, a rule for a user's bath may be that the temperature must be below X, within Y degrees. If the user increases the amount of cold water to the point that the temperature goes below Y degrees, the system automatically increases the flow of hot water to reach the dictated temperature.

The Meditrina architecture provides communication support between the devices, applications, and the intelligent control system, while allowing optimizations at the system level to save energy and match sensor and actuator use to application needs. We call the goal of a specific application *activities*, which are decomposed into the *tasks* required to reach a successful conclusion. Finally, each task requires some number of *services* from the environment.

The idea behind our architecture is that many if not most ubiquitous computing applications can be modeled as a set of activities that follow and possibly interlace with one another. Examples of activities are sleeping, having breakfast, taking a bath, taking medicine, measuring vital signs, and leaving home. The interleaving of these activities is caused by normal user behavior (*e.g.*, a user may need to take medicine during breakfast, causing the suspension of the breakfast activity until the medication activity is complete). During these activities, the environment is monitored and aided by the Meditrina system, which takes inputs from the intelligent control system, senses, and reacts according to current environmental state.

The Meditrina architecture contains three main components (see Figure 2). The first component is the *Activity Manager*, whose job is managing activities for users. The second component is called the *Task Manager*, whose job is to manage specific tasks needed to accomplish activities within the system. The third component, called the *Service Manager*, consists of the interfaces, called *Coordinators*, to the ambient devices providing services in the environment, including user devices (*e.g.*, PDAs), environmental sensors (*e.g.*, light sensors), medical devices (*e.g.*, heart rate monitors), environmental actuators (*e.g.*, thermostats), and networked, computer-equipped household devices (*e.g.*, ovens). In addition to these components, there is a *Service Discovery* component, which enables optimizations related to choosing devices within the system to perform tasks. In the following subsections, we present descriptions of each of the components of the architecture.

3.1 Activity Manager

The *activity manager* is responsible for managing user activities by decomposing each activity into a number of tasks and propagating them along with any parameters from the intelligent control system to the task manager. The upper interface interacts with the intelligent control system to prevent anomalous behavior that may have negative effects. For example, while taking a bath, several factors in the environment must be monitored and controlled (*e.g.*, air temperature, water temperature, and windows), so that the user activity can be performed according to the rules specified by the intelligent control system (*e.g.*, water temperature must fall within a specific range, in the winter the window should be closed), but the user can choose their preferred parameters within that range.

The intelligent control system also comes into play in case of unusual behavior and reactively provides support according to the specific situation. In these situations, the activity manager interacts directly with the control system, passing it the values that define the anomaly. The control system can in turn issue specific queries to the activity manager to better understand the situation (*e.g.*, requesting some vital parameters) and enter commands that specify the action to be taken.

3.2 Task Manager

The *task manager* is invoked by the activity manager for each task of an activity. Consider the previous example of taking a bath, this single activity is broken into regulating air temperature, regulating water temperature and pressure, and monitoring user vital signs. Each of these tasks is handled by a manager that is in charge of guaranteeing efficient task execution. Specifically, task managers are responsible for finding available services from the service discovery component, making specific service requests to the service manager, and monitoring task performance.

The optimization of network costs is performed by the task manager, which takes as input the application requirements from the activity manager and the cost and reliability for each service offered by the devices from the service manager. The objective of the task manager is to trade off application requirements and network costs, so as to achieve a satisfying quality of service, keeping network cost as low as possible. This kind of optimization is performed at the task level because the same task may be performed by using different sensors and actuators, with different costs and quality. This function is performed at the task manager because it is in the unique position to have knowledge about activity requirements and services available in the environment. The task manager is located between the activity and service managers to get application requirements from the former and network costs from the latter.

3.3 Service Manager

At the next level, the Meditrina architecture provides a service manager that defines three types of *coordinators*, through which other system components interact with the various devices in the environment. Each coordinator provides a clean interface to one or more devices for use by task managers.

There are two classes of coordinators to support the various types of devices in the environment, *passive* and *active*. Passive coordinators, which we call *monitors*, sense and output collected data. Active coordinators include both sensing and acting. Within this class, we define two types: *regulators* and *notifiers*. The regulators adjust environmental conditions, while the notifiers interact with the users to take specific actions.

Monitor - Monitors do not perform any action, they passively collect, sense and report data. For example, monitors can watch users' vital signs (*e.g.*, heart rate or blood pressure), or environmental conditions (*e.g.*, light levels, air temperature, or humidity).

Regulator - Regulators are based on the paradigm "sense and adjust," since they adjust the environmental state according to application requirements. Examples of regulators are: water temperature regulators, water pressure regulators, and door openers. Thus, a regulator coordinates sensors and actuators according to a target task. Regulators may have cyclic properties: after triggering an actuator to adjust the environment, the regulator must return to monitoring to ensure the desired effect was achieved.

Notifier - Notifiers are based on the paradigm "notify and wait for response," and involve the interaction of sensors, actuators, and user feedback. After notifying the user, the notifier waits for a response, checks if the task has been correctly performed, and potentially takes further action.

These coordinators provide the interfaces to actual devices, sensors and actuators, in the system. Both types of devices export their control parameters to upper layers. This data can be used by the task manager, along with information from the intelligent control system to choose which sensors and actuators to use for particular activities.

4. PROTOTYPE

To verify our system design, we implemented a prototype system of the Meditrina architecture in a real environment. Our prototype demonstrates how Meditrina can be used to manage an automated living environment, while allowing easy implementation of new applications. The prototype is implemented in a modular fashion, according to the architecture design (*e.g.*, only the service manager deals with the hardware).

In this prototype, we also want to test Meditrina's ability to deal with various network topologies and node parameters. This is a very important feature for an architecture whose main goal is to facilitate applications to assist users' in their daily activities. In our prototype, commands are provided by the user through simple strings, although a real deployment would obviously have a more friendly interface.

The prototype has been deployed in a house (Fig. 3 shows a portion of the deployed system), with the goal to provide easy and reliable access for the user to several different configurations of the lights. For example, turning on a reading lamp on a desk without bothering someone that is resting in the same room, or reacting to a critical event that is simulated by the pressing of a button, which in a real system would be a fire alarm or some other anomaly (*e.g.*, in an ambient assisted living environment, patient life signs could trigger an alarm). The system reacts to a raised critical event by interrupting any running task that is managing the lights and turning on all lights to permit a quick escape from a risky area or to provide sufficient lighting for support personnel. The system uses light sensors and actuators that can turn on and off several lamps. Each device's location is hard-coded into its parameter set, which in a real

Figure 3: Deployment

deployment would be provided by a localization algorithm.

We chose the light control application because it allowed us to test how easily the system reacts to events, how fast this reaction is, and how the service discovery is able to choose the correct devices based on the application's requirements. We simulated hardware failures by unplugging some lamps, and verified that the system could successfully react to the failures by choosing to activate additional lights when the failure was detected.

The communication nodes that are connected to the various sensors and actuators use different frequencies and protocols to communicate. To support all devices, a server is equipped with all of the technologies required to communicate with any sensor or actuator, and contains the implementation of the Meditrina architecture (the service manager, the task manager, and the activity manager). The application, which is currently a simple console program that waits for user input, runs on a laptop that can access the network via IEEE 802.11. A TCP socket connects the application to the activity manager on the server. This allows the application to be installed easily on any PDA or mobile computer. Additionally, a single application can have access to more than one Meditrina system at a time. For example, in an assisted living environment, doctors could have access to different patients' houses via their PDAs.

The following subsections detail our implementation of sensor and actuator hardware and the components of the prototype Meditrina system.

4.1 Hardware

Our prototype is deployed using motes of the Mica family (*i.e.*, Mica2 [12] operating at 900 MHz) and TmoteSky [6] (operating at 2.4 GHz on the IEEE 802.15.4 standard). Both the Mica and the Tmotes have been equipped with light sensors, the Micas with a sensor board expansion, the Tmotes with a photodiode.

Additionally, we designed and implemented ten actuator boards that can drive high current and voltage. These actuators were attached to TmoteSky nodes. The schematics of the actuator is shown in Figure 4, while Figure 5 shows the actual board layout. These boards are powered via a 9 V DC power supply, that also powers the TmoteSky node.

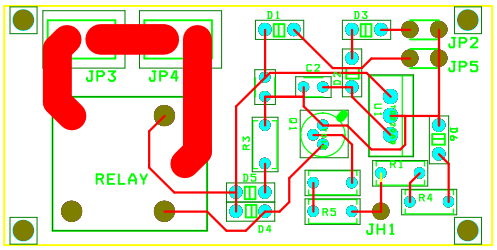

Figure 5: Actuator board layout

The board can control voltage of up to 110 V, and a current of up to 10 A and is perfectly suitable for the control of lamps.

4.2 Node Implementation

Nodes, sensors, and actuators are intended to be very simple and cheap devices, and the software they run should be simple and robust. Most decisions are made on the server side. It is common for cheap devices to experience malfunctions or complete failures. The service manager should track these conditions to prevent unsuccessful queries or commands. The software we developed for both sensors and actuators in our prototype has periodic communication to the service manager, with a KEEPALIVE packet that contains the node address and all of the properties mentioned in section 2.

All of the sensors that we implemented in the prototype have a single-action behavior. The sensor node is in IDLE mode until a QUERY is received. An ACK is immediately sent to the sink node connected to the server, with information about whether the sampling can or cannot be performed. Then, once the data is ready, a packet is sent and the sensor returns to IDLE.

For actuators the behavior is similar. The node is initially IDLE, waiting for a command. When a command is received, the actuator performs the action specified, if possible. An ACK packet is sent back to the sink, including information about the result of the action and the current status of the controlled switch. The actuator then returns to the IDLE state.

4.3 Service Manager

The service manager is the component that has direct access to the hardware. It must be able to communicate with each technology used by the various nodes in the environment. The service manager should also be able to react to events coming from the hardware layer or the task manager. Since the network topology is subject to changes, due to hardware failure, node replacement, or node insertion, the service manager must keep up-to-date information about the network topology and the nodes properties, to allow decisions about which nodes to use by the task manager. The knowledge about the network status (*i.e.*, location of devices, hardware failures, energy status) is then used by the service discovery component to generate the list of devices that can be used to satisfy the requirements in terms of quality of service.

Typically, the service manager is in IDLE mode waiting for some event. These events can be raised either by the upper layer (the task manager) or the lower layer (the actual hardware). The service manager is connected, via two TCP

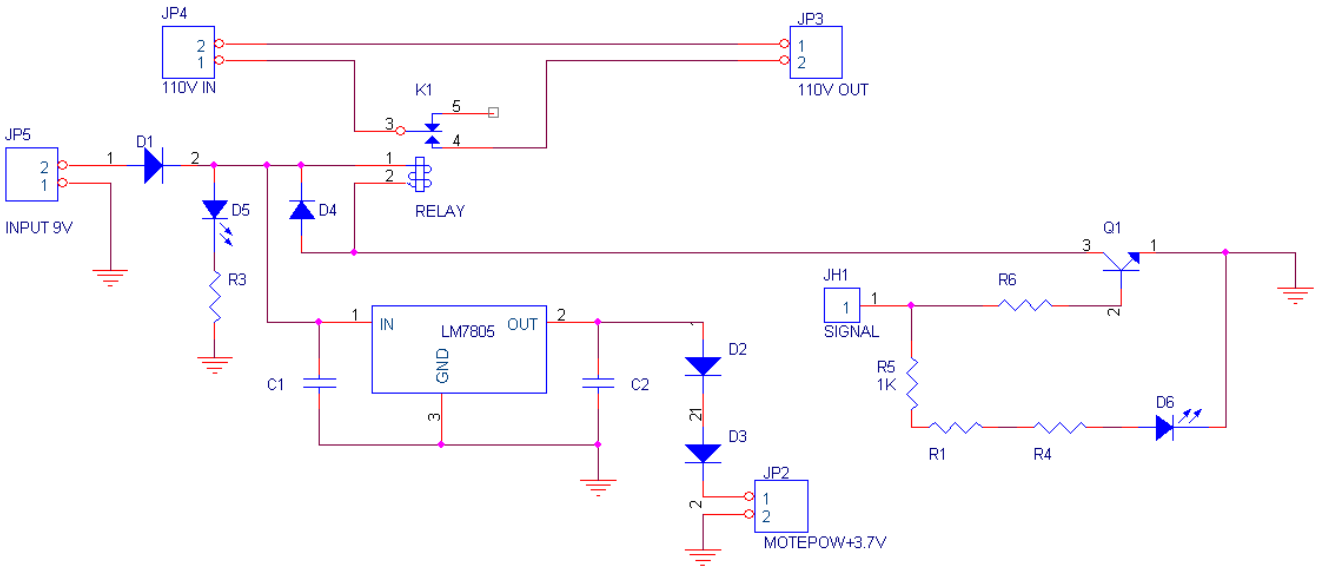

Figure 4: Actuator board schematics

sockets, to a Mica2 and a TmoteSky mote. These devices act as sinks for all messages to and from the network. The service manager provides an interface that the tasks can use to send a command or a query to a specific node, by forwarding the message to the network and keeping track of the associated task ID. When the reply to the query or the ACK to the command arrives, it is forwarded to the appropriate task.

We designed the service manager to automatically collect information about the network status and topology. We use KEEPALIVE messages to keep the information up-to-date. These messages convey data about the nodes (*i.e.*, the position, the battery level) to the service manager. In the future, we plan on implementing more efficient service discovery mechanisms. Each node in the network is linked to a virtual node object that contains all of the information about the control parameters of the device and a validity timer. When a KEEPALIVE message is received, the node list is scanned for an existing instance of the virtual node relative to the message source (each node has a unique ID). If this object is found, the validity timer is reset and the node information updated, otherwise a new object is created. If the validity timer of a virtual node expires before a new packet is received, the virtual node is marked as not working, until a new KEEPALIVE message is received.

This list is used by the service manager when a query is received by the service discovery component. Given the parameters of the service required by the requesting task, the service manager returns a list of devices that could be used to successfully perform the task with the minimum energy consumption.

4.4 Task Manager

The task manager deals with simple activities that involve some of the hardware devices. Examples of tasks are collecting a light intensity sample in a specific area, deciding if one or more light bulbs should be turned on or off, and checking the heart rate of a user. The task manager should manage more than one task at the same time, since an activity is potentially composed of multiple tasks, and more than one activity can be performed at the same time. Furthermore, certain conditions require an immediate reaction, as in the case of a heart attack sensed in a user. This means that the design of this component should allow high-priority tasks to interrupt any running low-priority tasks and act as soon as possible.

The task manager provides a communication interface to the activity manager. When a new task is activated from the upper layer, the task manager creates a new thread that manages the request. A task thread's first action is to create the list of services that it requires to achieve the requested result. This list is compiled from a set of simple hard-coded definitions. The desired services do not specify which hardware should be used, but only the type of information and/or actions, the location where the task is located, and the quality of service parameters required. The task queries the service discovery component to find the devices that should be involved in the operation to achieve the goal, while respecting the given time and energy constraints. Service discovery returns the list of devices that can be contacted, and the task manager sends to the service manager the appropriate commands. The rest of the behavior is task dependent.

We defined four types of tasks: the *single-action* task, where no feedback is expected after the action, the *one-feedback* task, where the action is followed by a single response (*e.g.*, a sensing to check if a light has turned on), the *continuous feedback* task, where the first action starts an action-feedback cycle until a certain condition is reached, and the *continuous action* task, where the same action is repeated until a stop command is issued.

Our prototype includes only *single-action* tasks, that assume that, once the sensing and the actions have been done, tasks are successfully completed if no hardware failure occurred. A more complex behavior, with *feedback-based* tasks,

would be useful if other actuators were included, such as window shades. In this case, a task instructed to make the room dark could first perform a check of the environment status, then turn off all the lights if the light intensity is over the *dark* threshold, then make a new query and, if it is still too bright, send a command to lower the window shades.

4.5 Activity Manager

The activity manager is the highest level component of the Meditrina architecture. It must control and send requests to the Task Manager and provide the user or the intelligent application with a simple interface. The external entities that access the Meditrina system should be able to simply schedule the execution of one or more activities based on time or some raised event. The activity manager should then generate one or more tasks per activity. All of the hardware information and the optimization issues should be hidden from the user and managed by Meditrina. This implies that the activity manager must be able to understand the user requirements and translate them into tasks.

The list of activities that the user can perform is provided either during the initial setup of Meditrina or every time a new feature is installed. Each activity specifies which tasks should be invoked to achieve the goal, and specifies quality of service requirements in terms of maximum delay and success probability. Activities implemented in this prototype are creating different lighting configurations for the room, for example, a resting configuration with all lights turned off, a reading configuration, where two different lamps can be turned on with different costs to create a bright zone, and a critical configuration, whose goal is to have all of the lights turned on. A priority level can be assigned to each activity at setup time or during the normal execution, depending on the environmental status. Activities with higher priority can interrupt those with lower priorities. The activity schedule is defined in a database that contains a set of rule definitions that specify the events that can be caught, the activity that should be started in response, and the priority that should be assigned to it.

Many events can start an activity: a timer can expire, another running activity can signal an event (*e.g.*, human presence, sound, fire, bad weather, or threshold exceeded), and the user can send a command. In our prototype, these rules are based on time and on asynchronous user commands. When one of these conditions happens, the activity manager executes a query on the rules database, looking for the activities that should be started. Once the activity list is generated, for each entry, the activity database is scanned and a list of tasks that must be run is provided. The activity manager instructs the task manager about the list of required tasks and then returns to the IDLE state, waiting for another event. If the activity requires some feedback, a specific task is run and then the activity manager monitors the environment and after a specified timeout raises an event signaling the current status of the monitored quantities.

4.6 Service Discovery

The goal of the service discovery is to provide the Task Manager with the address of one or more hardware devices that can satisfy the requirements for a desired task. However, the choice should be performed in order to optimize the network energy consumption. These two goals are often hard to combine, and the network characteristics can vary due to hardware failures, hardware replacement, or new devices. Additionally, the QoS required for a task can vary each time the task is generated. The service discovery time is added to the total delay for the result.

The service discovery component lies between the task manager and the service manager. When a task requires a sensed value in a specific area, or has to change a specific condition, it invokes service discovery, querying for one or more actual devices that can address its quality of service parameters in terms of delay, energy consumption, or accuracy. Given these parameters, the service discovery scans the entire device database, looking for all sensors or actuators that can satisfy the requirements. A cost function is computed for each resulting device, based on the energy cost of the activity or other parameters the programmer can specify. The cheapest solution is then provided to the task that made the request. The task will then send to the service manager a command specifying which actual device to contact. In our prototype, we use simple energy costs as our metrics for choosing devices; however, the system itself is general enough to support any cost function. For example, in ambient assisted living environments, the cost function will include some measure of accuracy of measurement, which may be more important than simply conserving energy.

5. CONCLUSIONS AND FUTURE DIRECTIONS

The increased availability of sensors and actuators brings with it the ability for ubiquitous systems architectures to more finely monitor and affect the environment. This ability requires architectural changes to present these devices to intelligent applications, hiding the low-level aspects of the devices and only exposing parameters about available services that are meaningful to the applications. In this paper, we presented the design of Meditrina, a comprehensive ubiquitous system architecture supporting the integration and control of sensors and actuators.

To demonstrate the ability of the Meditrina architecture to facilitate rapid application design by providing interfaces to the services provided by the devices in the environment, we implemented a prototype. The implementation includes a hardware implementation of actuators capable of controlling any 110 V device pulling currents of up to 10 A. Our application is a light control application that allows users or the system to control the light levels in a room. We tested the system's reactions to both user inputs and hardware failure, further demonstrating the flexibility of our architecture.

There are a number of directions for future work. First, service discovery in the prototype is very simple, many further optimizations over a large number of network parameters can be researched. Additionally, the task manager activity of choosing set of devices to use during the normal activities can be optimized to further improve system performance. In our current prototype, we use hard-coded rules for activities, while the vision of the architecture includes an intelligent control system. Additionally, there are many human-computer interface questions about how to represent and allow the control of so many devices throughout the environment. Finally, the development of full applications to support specific scenarios, such as ambient assisted living, will act as final proof of the usefulness of the architecture.

6. REFERENCES

[1] Georgia Institute of Technology. Aware home. http://www.ee.gatech.edu/fee/ahri, 2007.

[2] W. Heinzelman, A. Murphy, H. Carvalho, and M. Perillo. Middleware to support sensor network applications. *IEEE Transactions on Network*, 18(1):6–14, January-February 2004.

[3] Intel Corporation. Age-in-place. http://www.intel.com/research/prohealth/cs-aging_in_place.htm, 2007.

[4] D. Malan, T. Fulford-Jones, M. Welsh, and S. Moulton. CodeBlue: An ad hoc sensor network infrastructure for emergency medical care. In *International Workshop on Wearable and Implantable Body Sensor Networks*, 2004.

[5] I. Mohomed, M. Ebling, W. Jerome, and A. Misra. HARMONI: Client middleware for long-term, continous, remote health monitoring. In *UbiHealth*, 2006.

[6] MoteIV. Tmotesky. http://www.moteiv.com.

[7] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R. Campbell, and K. Nahrstedt. Gaia: A middleware infrastructure to enable active spaces. *IEEE Pervasive Computing*, pages 74–83, October-December 2002.

[8] University of Rochester. Center of future health. http://www.futurehealth.rochester.edu, 2007.

[9] University of Virginia. Smart In-Home monitoring system. http://marc.med.virginia.edu/, 2007.

[10] Q. Wang, W. Shin, X. Lui, Z. Zeng, C. Oh, B. Alshebli, M. Caccamo, C. Gunter, E. Gunter, J. Hou, K. Karahalios, and L. Sha. I-Living: An Open System Architecture for Assisted Living. In *IEEE SMC*, 2006.

[11] A. Wood, G. Virone, T. Doan, Q. Cao, L. Selavo, Y. Wu, L. Fang, Z. He, S. Lin, and J. Stankovic. ALARM-NET: Wireless sensor networks for assisted-living and residential monitoring. Technical Report CS-2006-01, University of Virginia, 2006.

[12] XBOW. 2nd generation micamote. http://www.xbow.com.