PlantCare: An Investigation in Practical Ubiquitous Systems

Anthony LaMarca, David Koizumi, Matthew Lease, Stefan Sigurdsson and Gaetano Borriello; Intel Research
Waylon Brunette, Kevin Sikorski, Dieter Fox; University of Washington

IRS-TR-02-007

2002

PlantCare: An Investigation in Practical Ubiquitous Systems*

Anthony LaMarca[1], Waylon Brunette[2], David Koizumi[1], Matthew Lease[1], Stefan B. Sigurdsson[1], Kevin Sikorski[2], Dieter Fox[2], Gaetano Borriello[1, 2]

[1]Intel Research Seattle
[2]Department of Computer Science & Engineering, University of Washington

Abstract. Ubiquitous computing is finally becoming a reality. However, there are many practical issues that stand in the way of mass acceptance. We have been investigating these practical concerns within the context of an autonomous application that takes care of houseplants using a sensor network and a mobile robot. We believe that emphasizing autonomy and thereby minimizing demands on users will help us address the many practical concerns that will arise not only in PlantCare but also in many other ubiquitous applications. In this paper, we discuss the technical challenges that we have encountered while trying to make PlantCare a reality and report on our experience in addressing these challenges.

1 Introduction

A decade ago Mark Weiser first introduced us to the prospect that computers could become truly ubiquitous, something very difficult to imagine at the time, let alone realize. A decade later, however, we can now truly consider embedding computation and communication into virtually all manufactured objects. Yet paradoxically, in many ways we have become a victim of our own success. A steady stream of advances has come at the cost of ever-greater complexity, and our ability to manage this growing complexity has not been able to keep pace. While the potential of ubiquitous technology demonstrated in the laboratory has remained compelling, our ability to realize this potential in practice has continually fallen short. How can we practically support large numbers of heterogeneous, impoverished devices that require individual configuration and maintenance? How do we build long-running applications that guarantee continuous service? How can we utilize existing infrastructure, facilitate interoperability, and enable systems to evolve over time? How can we realize these systems in non-contrived physical environments while introducing minimal, non-invasive infrastructure? Most importantly, how can we do all of this while providing users with pleasant and meaningful interactions with their computers? All of these questions address significant obstacles to the wide-scale

* A version of this paper appears in the *Fourth International Conference on Ubiquitous Computing (UbiComp 2002).*

adoption of ubiquitous computing, but thus far these practical issues have received little attention from the research community.

Recent efforts such as IBM's autonomic computing [1] initiative are beginning to deal with these issues. The focus has been on improving robustness of backend server rooms and data centers as well as increasing the efficiency of system administrators. Our goal is to make life easier for the typical user. We have been investigating how the practical challenges posed by ubiquitous computing systems might be addressed. We believe that some form of autonomous infrastructure capable of configuring, tuning, and repairing without human interaction will be required in order to achieve realistic deployment of ubiquitous systems. Consider, for example, the difficulty in configuring and maintaining the ever-increasing number of devices being developed for sensing and actuation. Manually managing all of these highly constrained, tiny, embedded, heterogeneous devices is not feasible. The model of relying on skilled IT workers cannot scale to meet the enormous, ever-growing complexity of these new challenges. Typical users will not have the time, interest, or technical expertise to do even a minimal amount of the work required while at the same time expecting continuous, reliable, and rich interactions. Instead, we must push the idea of unobtrusive interaction to its limit and investigate autonomous solutions that succeed where human administration cannot.

We are exploring this approach in the context of PlantCare, a ubiquitous and distraction-free system that cares for houseplants. While caring for houseplants may not be the most promising application of ubiquitous technology, we chose this project for its clear goals and the wealth of opportunities it provides us for exploring the set of practical challenges mentioned earlier. Our system monitors environmental conditions impacting plants using wireless sensors placed near the plants. Application logic interprets observed conditions and determines when plants need to be watered. A mobile robot completes the sensing-actuation feedback loop by watering the plants whenever the system requests it to do so.

Our goal in this paper is to elucidate how our work on PlantCare supports the belief that computing systems can be made to operate much more autonomously while still providing a high-quality user experience and how our initial experiences are directing our future work. In Section 2, we describe the specific problem addressed by the PlantCare project and how it facilitates our investigation into the practical barriers faced by ubiquitous computing. In Section 3, we describe our approach and the hardware and software being employed. In Section 4, we discuss the technical challenges we have encountered and report on our experience in addressing them. In Section 5, we describe several lessons we have learned from our experiences. We discuss related work in Section 6, and in Section 7 we conclude with a discussion of future work.

2 The Problem

The challenge undertaken by the PlantCare project is to provide ubiquitous and autonomous care of houseplants in a typical home or office environment. In order to illustrate the system being developed, we briefly sketch a hypothetical scenario below. While much work remains before us in order to fully realize the scenario's vision, subsequent sections will detail aspects of the scenario already achieved and our plans for addressing those aspects yet to be realized. Another goal of the scenario is to ground a more general discussion of what features constitute a practical ubiquitous system.

A year ago Ken bought the PlantCare system to take care of the houseplants in his Victorian home. He had just taken on new responsibilities at work and hoped the system would save him some time and worry. Initially he was nervous about buying the system. After all, he had always been annoyed by the learning curve and setup time typically required by new devices (he still hadn't figured out how to move phone numbers from his PC to his cell phone, for example), but he was delighted that after unpacking the new system and turning it on, everything just worked! Not only was it able to find all of the plants around his house, it knew how much water each needed and began watering all of them for him. Better yet, a couple of days later it suggested his ficus would be healthier if he moved it next to a window where it would receive more light. When one day he completely forgot about the system and watered a plant himself, somehow the system knew the plant had already been watered and didn't water it again until the next week. As warmer weather set in, he was delighted to see that the system automatically began giving plants more water. As the months passed and the system continued to operate without any unpleasant surprises, Ken's appreciation of it continued to grow. Sure, it didn't work when a winter storm knocked out a power-line, but once power was restored the system started working again along with the rest of his appliances. All in all, Ken was quite pleased with his purchase – now if only it could help him with his cell phone…

This idealized scenario illustrates many of the practical challenges of ubiquitous system design that motivate our work. For example, we clearly see the need for a zero-configuration, self-maintaining solution. Ken is not a technological expert, and if more than minimal effort is going to be required, he might as well continue watering the plants himself. In the scenario above, the system automatically explored its deployment environment and knew how to care for each of the plants recognized. The system also automatically monitored the environmental conditions of each plant and adjusted its care accordingly.

We also see in the scenario how much Ken values system dependability. While continuous, 24/7 service may not be required in caring for houseplants, if a ubiquitous system is really going to blend into our daily activities, it should rarely, if ever, exhibit a noticeable degradation in service. In the scenario, the system recovered gracefully from power outage as though such were part of its routine operation. Moreover, it recovered without requiring any intervention by Ken.

The scenario also illustrates the need for infrastructure that is minimal, adaptive, and non-intrusive. Ken has no interest in doing major remodeling to his home to add extensive sensing and actuation technology. While assuming everyone lives and works in smart spaces simplifies research (e.g. putting magnetic tape in the flooring greatly eases the problem of robot navigation, adding wired sensors eliminates any concern for batteries, etc.), we believe it is of significant value to consider the minimal infrastructure necessary to bring tangible value into the typical office or residential environment.

Finally, consider when the system asked Ken to move a plant to a healthier location. By being proactive, the system was able to leverage its knowledge and sensor data to provide Ken with proactive assistance rather than offering him assistance only when he requests it. This ability is especially valuable when users do not know to ask for help, as was the case in this example. Such opportunities will only increase as systems become more invisible and contextual awareness improves. This example also demonstrated how whenever limitations in system actuation arise (e.g. the system is not able to move a plant by itself), systems should communicate with the users at an appropriately high level of abstraction clearly understood by a wide range of people.

3 The PlantCare System

The core architecture of PlantCare consists of wireless sensors for environmental sensing, a robot for actuation, and loosely coupled, component-based software to integrate the hardware and provide application logic. Below we describe briefly each architectural component and its overall contribution to the PlantCare system.

3.1 Wireless sensors

Environmental sensing is provided by wireless sensor nodes placed on both the robot and in the soil of the plants being cared for. At the core of each node is the UC Berkeley "mote" wireless sensor platform (see Figure 1). Motes operate at 3V and are assembled from off-the-shelf components that include an 8-bit microcontroller, a two-way 916MHz radio for communication, and an expansion connector for connecting analog sensors. Mote applications are built on TinyOS [2], a small, real-time operating system supporting ad hoc networking and lightweight concurrency.

Sensors used by the PlantCare system include a photoresistor for measuring light level, a thermistor for measuring temperature, and an irrometer for measuring soil

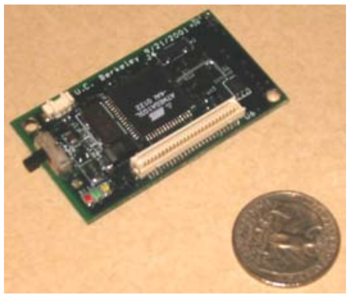

Figure 1: The UC Berkeley mote is a tiny wireless sensor platform for environmental sensing

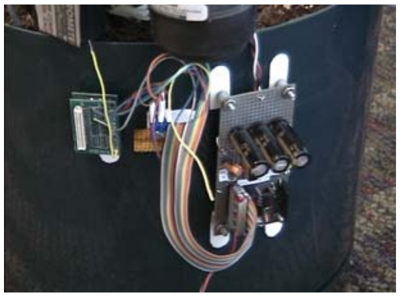

Figure 2: A deployed sensor node. Digital and analog sensors attached to a mote enable sensing local environmental conditions. The node is powered by capacitors that can be inductively recharged

humidity. The nodes deployed at the plants (see Figure 2) periodically emit readings reflecting measurements of environmental conditions and the amount of remaining power. These nodes have also been augmented with a custom power system in which capacitors replace the disposable batteries in order to allow for fast inductive recharging (see Section 4.1). Nodes deployed on the robot use more accurate sensors in order to calibrate and verify the sensors deployed at the plants (see Section 4.2).

3.2 Robots

PlantCare employs a robotic actuation platform to water the plants and to calibrate and recharge deployed sensors. The robot (see Figure 3) is a Pioneer 2-DX [3] augmented with a small water tank, pump, and dispensing spout. A microcontroller board controls the pump and reads the water level in the reservoir. A commercially available laser range-finder provides the robot with the ability to accurately sense its environment for mapping, localization, and obstacle avoidance, without the need for extra markings or reflectors to be placed in the environment [4]. Both the microcontroller and the laser range finder are connected to a laptop that runs the robot's control and collision avoidance algorithms. This laptop is connected to the network via an IEEE 802.11b wireless card. The remainder of the robot control software, including path-planning, particle-filter-based localization, and interface software, is run on a desktop computer.

The main components of the robot control and navigation system are a reactive collision avoidance module, a module for map building and path planning, and a localization module. All components use proven probabilistic methods [5, 4] to deal

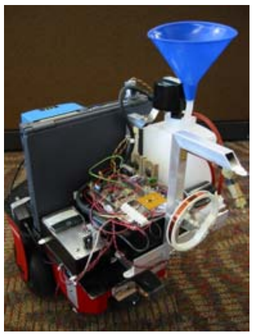
Figure 3: The robot platform showing the spout for filling its water reservoir, the dispensing spout, and the inductive recharging coils, one of which the robot uses to recharge itself and the other to recharge the plant sensors

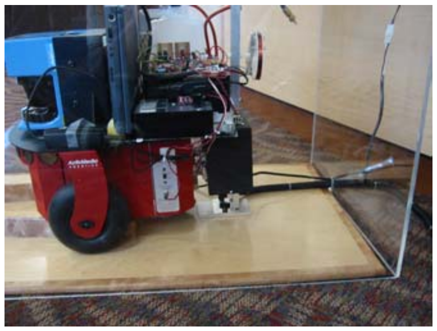
Figure 4: The robot is shown docked in its kitchen cabinet-sized maintenance bay where it can obtain water and recharge

with uncertain sensor information. The high-level task ordering and dispatching software was custom-built for the PlantCare project. Path planning is currently performed with the aid of a static environmental map. The map is constructed off-line by guiding the robot using a joystick and using specialized mapping software to integrate laser scans over time with the assistance of odometry provided by the robot's wheel shaft encoders. Further work is required to automate this process (see Section 4.4).

The PlantCare system also includes a maintenance bay (See Figure 4) that the robot uses to recharge itself and automatically refill its water reservoir. When the robot backs into the bay, sensors on the bay detect the current water level on the robot's reservoir and dispense water if needed.

3.3 Software

The PlantCare application has been developed on top of an in-house library called Rain. Rain is a lightweight messaging system developed as part of a larger effort to explore alternative approaches to providing system support for ubiquitous applications. In Rain, applications are decomposed into a set of cooperating services that communicate with each other via asynchronous messages. Rain also includes a discovery service that allows services to register and find each other. This structure gives the application the opportunity to transparently support the highly dynamic environments envisioned for proactive systems. The asynchronous communication model allows applications to be highly responsive even in the face of widely distributed services running on hardware platforms with widely varying performance. To support heterogeneous computing environments, Rain messages are encoded in XML. On the surface Rain appears very similar to SOAP [6], but whereas SOAP has been used almost exclusively as a synchronous RPC mechanism, Rain has been geared towards support of asynchronous event-based software architectures.

The PlantCare application is composed of fifteen Rain services. Figure 5 shows the organization of these services and how they interact with each other and the environment. These services collectively provide both the high-level application logic as well as the low-level driver-like code that communicates with hardware and external software. To manage the sensor data, there are services that control the sensor base stations (*mote proxy*), unpack the data from its proprietary form (*mote translator*), calibrate the data readings based on previously collected calibration data (*mote calibrator*), and store the readings for future use by applications (*mote data store* and *plant data store)*. Robot services include low-level logic for navigation and activating the robot's sensors and actuators (*robot control*) as well as high-level functionality to perform application-specific robotic tasks such as watering plants and recharging deployed sensors (*robot manager*). The robot's navigation tasks are aided by the data provided by a general-purpose *localization* service that denotes the physical location of tagged objects in the environment. A *task server* acts as a coordinator for activities that need to be carried out. A *gardener* service examines the calibrated sensor data, consults species-specific care instructions in a *plant encyclopedia* service, and posts watering tasks to the task server. Similarly, a

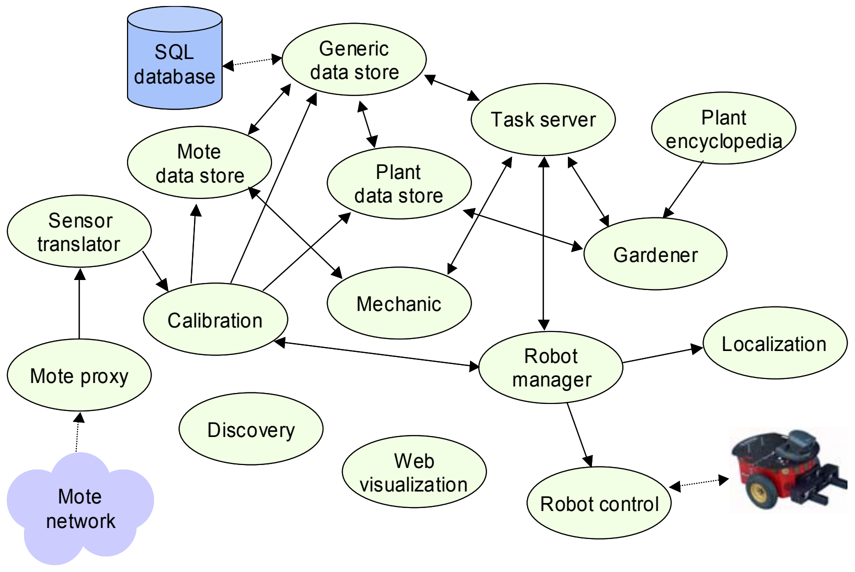

Figure 5: The PlantCare application is composed of a set of cooperating services

mechanic service examines both robot and sensor power levels and posts recharge tasks to the task server. Finally, there are some general support services. The *generic data store* is a Rain service that wraps the functionality of an SQL database. The *web visualization* service acts as a Rain-HTTP bridge to allow services to both generate HTML and to receive CGI requests. Though simple, the web visualization service allows service writers to quickly produce simple user interfaces for services. All together, the PlantCare application consists of approximately 9000 lines of Java Rain services, 150 lines of Perl Rain services, and 400 lines of TinyOS code that runs on the motes.

4 Experience

In our lab we have deployed a PlantCare system with four monitored plants, a robot and a maintenance bay. The Rain services shown in Figure 5 have been deployed on a set of desktop machines, one of which is connected to a mote base station. While the PlantCare software is stable and the sensors function well, issues with the robot's power and navigation have limited the system's ability to provide continuous service. While the robotic subsystem is being debugged, we have connected the motes to a wired power supply to keep them from running out of power. To cope with intermittent robotic outages, a task mailer service was added to the application. This service takes watering tasks off the task server and sends email messages to a PlantCare developer.

Despite the preliminary state of our system, a number of interesting results have emerged. In this section, we report our experiences developing and integrating our system.

4.1 Power management

A key focus of our investigation into practical, ubiquitous computing has involved learning how to build long-lived systems that provide continuous service. The immediate challenge this posed for our system's hardware was determining how to keep our robot and wireless sensors from running out of power. The traditional solution for robots has been to rely on human recharging, which is unacceptable for a hands-off solution. As for the sensor network, traditional approaches rely on delivering power by wire [7], disposable batteries [8], or energy scavenging [9]. None of these options are acceptable for a long-lived ubiquitous system: wires are intrusive and inflexible, batteries require manual replacement or severely restrict the network's operational lifetime [10], and current energy scavenging techniques simply cannot generate sufficient energy for sustained operation. Instead, we have adopted a model whereby our system maintains its own hardware in much the same way it cares for the plants. Just as the system must track the humidity of soil to determine when water is needed, we also designed it to track the power levels of both the robot and the individual sensor nodes. Just as the robot was augmented to water the plants, we designed our robot to both recharge itself and the deployed sensor nodes.

We augmented our robot with an inductive charging "paddle" that matches a corresponding charging coil in the maintenance bay. We chose inductive charging for both the sensors and the robot in order to reduce the danger to people and equipment. Whenever the robot needs power, it can recharge itself by navigating back into its maintenance bay. To simplify the robot's power system, the robot's main batteries power all of the robot's accessories, including the laptop and custom hardware. To facilitate recharging the sensor nodes, we replaced their AA batteries with a bank of capacitors and an inductive coil. A corresponding coil has been placed on the robot, allowing it to navigate to the plant, align the coils, and inductively recharge the mote from its own much higher capacity lead-acid batteries. Because inductive field strength falls off at a rate proportional to the square of the distance, close alignment of the inductive coils is required for achieving efficient power transfer.

While the use of inductive charging introduced additional complexity to system hardware, the effect on system software was minimal. Since PlantCare was designed to monitor, record, and issue care instructions, recharging the motes and the robot was a natural extension of the system. Since the robot is intended to spend the majority of its time in its maintenance bay anyway, additional trips for recharging should be unusual. Furthermore, to the extent that plants need water more often than motes require recharging, no extra plant visits are expected.

Once our robot is capable of extended operation, we plan to investigate the potential of overprovisioning and predictive maintenance. By overprovisioning, we mean assuming a more limited energy capacity than is actually present so that the system can cope with unanticipated behavior by users or other system components.

Similarly, by continuously tracking system resources in another Rain service, recharging and watering visits can be planned in advance to improve system efficiency.

4.2 Sensor calibration

Similar to the challenge of providing sustainable power without effort on the part of the user, a practical ubiquitous system must also provide for the configuration and maintenance of its sensors and actuators. One aspect of this challenge involves performing sensor calibration, which is critical to obtaining accurate measurements. While the need for calibration is much less pronounced with expensive sensors, we are concerned with the class of inexpensive mass-produced sensors we expect to be widely deployed. Without calibration, readings taken by these sensors could be wildly inaccurate or even meaningless. The process of calibration derives a function for each sensor that translates the sensor's raw readings into canonical units, taking into account the sensor's unique hardware peculiarities and its environmental placement. In practice, not only does the mapping function vary from sensor to sensor, it often changes over a deployed sensor's operational lifetime. The result is that sensors commonly require both initial calibration and periodic recalibration in order to ensure accurate readings.

Sensors can be calibrated either before or after deployment. Pre-deployment calibration is performed in a controlled space in which the environmental factor measured by the sensor can be carefully regulated. While it is inexpensive, pre-deployment calibration has the disadvantage that sensors are often very sensitive and the act of deployment can change their characteristics. The alternative is to calibrate the sensor in-place, after it has been installed. This is more time consuming and inconvenient, but also potentially more accurate. Although pre-deployment calibration was in fact feasible for us given our limited number of plants, in general this technique does not scale. Furthermore, use of pre-deployment calibration restricts our system's ability to maintain itself in the presence of failing hardware and varying environmental conditions. Instead, we developed the novel approach of deploying uncalibrated sensors and using a robot to perform in-place calibration [11].

In this model, sensors installed on the robot are used to calibrate the deployed sensor nodes. We expect that for a commercial product, the robot would be outfitted with high-quality and highly reliable sensors that had been pre-calibrated at the factory. When a new sensor is deployed, the robot is dispatched to the new sensor's location, and after the robot's sensors have acclimated, a reading is taken. The robot's reading is paired with a reading from the stationary sensor and entered into the calibration database. This routine is repeated until the system has enough data to run a predictive regression technique. Thus, a raw reading from the sensor can be converted to the proper units using the sensor-specific calibration data.

In-place robotic calibration can be naturally extended to adapt to changes in the physical environment. If the environmental conditions change in a significant way, the robot can be dispatched to collect additional calibration data. As an example, consider a deployed light sensor in a room where furniture has been rearranged and now shadows the sensor differently than before. Readings from the robot's visits since the rearrangement are used to adjust the calibration function. Thus robotic calibration can be triggered by a simple heuristic in the regression software and does not require any user attention.

Similarly, continuous robotic calibration can account for changes in the sensors themselves. By placing a lifetime on individual pieces of calibration data, the system can ensure that it is tracking the drift of inexpensive sensors that change their characteristics over time. Incremental calibration also provides an excellent opportunity to detect sensor failure. Without sophisticated models, it is very difficult to detect when a faulty sensor returns plausible, possibly varying, yet uncorrelated data. This means a faulty sensor can potentially remain in service for a significant period of time before the error is detected. In our system, adjusting the calibration data lifetime effectively bounds the detection latency.

Like all techniques, robotic calibration has limitations. While it works well with PlantCare's photoresistors and thermistors, calibrating the irrometers is more problematic. This is because some sensors, like the irrometer, require physical coupling to the environment in order to obtain accurate measurement, and as a result, corroboration cannot be achieved by placing a similar sensor nearby. Merely driving a robot-mounted moisture sensor near the plant will not measure the amount of water in the plant's soil. Another consideration in employing robotic calibration is that it requires a more precise localization system than might otherwise be required: the robot must be able to closely locate the deployed sensors in order to move near enough to calibrate them. We have demonstrated the responsiveness and accuracy of the data in our self-calibration scheme [11].

Once we have robotic calibration operating continuously in PlantCare, we plan to investigate how this technique scales to an increasing number of deployed sensors requiring calibration. We also wish to explore the extent to which the robot's other duties provide opportunities for serendipitous calibration data collection.

4.3 Improving software reliability

Building long-lived systems that provide continuous service is a significant challenge for both software and hardware. To address this, we have developed a 'watchdog' service that can identify and replace faulty or missing required services. Applications that want to use the watchdog facility register a set of checks and reactions with the service. The checks are simply Rain messages coupled with their expected replies.

For example, PlantCare must ensure that there is a *task-server* running at all times. To accomplish this, the system can register a Watchdog check that simply queries the discovery service, expecting to find at least one matching service. More sophisticated checks can be written as well which involve the Watchdog interacting directly with the service being checked. For example, one could test that the database service can store some data and then return it upon query. In the event that a check fails, the reaction is invoked, causing the Watchdog to dispatch one or more messages. These messages can invoke arbitrary services, but in the common case they simply restart the missing service and possibly trigger a cleanup or consistency check. The actual

Watchdog logic is simple. It periodically performs the checks, matches them against the expected result, and fires off reactions appropriately. An obvious extension to this is to use a second Watchdog to watch the first one in order to ensure that this facility is always available.

In using the Watchdog we have found it valuable to carefully consider the notion of state in the design of our services. Stateless services are easier for the Watchdog to replace because they maintain either no state or only soft state that can be regenerated quickly. Generally speaking, there appears to be no disadvantage to having multiple copies of a stateless service running, and it is unimportant which copy of the service a client interacts with. Since stateless services hold no important data and services are removed from discovery quickly upon termination, killing and restarting stateless services is transparent to clients. Stateful services, on the other hand, maintain hard state, the loss of which can affect the correct operation of the system. Rather than trying to solve the problems of loss detection and recovery, we have simply tried to minimize the number of stateful services and make sure they are durable. In PlantCare, for instance, the relational database is the only stateful service. To address durability, we have built our generic data store using a stable version of MySQL and we run this service on a reliable, lightly loaded machine.

As a next step, we have begun monitoring the traffic patterns of a healthy system in order to automatically derive rules that can be directly used to monitor system behavior.

In contrast to the approach taken by Bronstein et al [12], we plan to make the Watchdog aware of these traffic patterns rather than the services themselves. This offers the benefit that third-party services may be introduced into the system at run-time without requiring the services support self-monitoring. We are also interested in monitoring the resource requirements of a given service. By continuously collecting resource utilization data for services, we hope to successfully predict resource exhaustion faults before the fact. In both cases, the emphases are on utilizing the execution history of the system to reason about its current state and hopefully achieve “stability through statistics”.

4.4 Navigating a dynamic world

Another challenge in practical, ubiquitous computing is deriving and maintaining accurate spatial knowledge of the environment. For PlantCare, one of our goals is to support tight integration with the physical environment by automatically detecting the addition of new plants and the movement or removal of known ones. Since our solution involves a mobile robot, we must also enable it to navigate an environment that may change dramatically over the months or years it is in service. This means we must be able to ensure at all times we have both an accurate map of the environment and a satisfactory estimate of the robot’s position and orientation within this environment. We must also make this work without requiring additional infrastructure.

As mentioned in Section 3.2, our initial approach has relied on manually mapping the operating environment in the form of an occupancy grid, which includes plant positions. This map, along with an implementation of particle-filter-based localization [5, 4], is used to track the robot's position. The map also enables automatic recovery whenever the robot becomes temporarily lost, a situation which must be expected and dealt with for automatic navigation of realistic physical environments. Finally, using the map it is possible to determine optimal navigation routes from the robot's current position to a given destination. Our next step will be to eliminate the pre-generated map and instead let the robot generate its map automatically through exploratory navigation of its environment. In order to accomplish this, we are investigating use of FastSLAM [13], a Simultaneous Localization and Mapping (SLAM) [14] technique. SLAM approaches use particle-filters and/or Kalman filters to simultaneously track the positions of the robot and any identified landmarks in the environment, and eventually find the most probable map and robot pose, given a history of sensor readings.

PlantCare also needs to detect and track plant positions so that the plants can be properly cared for. While today this information resides in the static map, we plan to implement a two level localization approach for an initial dynamic solution. First, approximate plant location will be determined using RF signal strength [15]. Once the approximate location is known, the plant's exact location will be determined by installing an accelerometer in the mote's sensor package and instructing the robot to gently bump any stationary obstacles it senses in the vicinity of the plant. If the accelerometer reports that it was jostled, then the robot knows that it made physical contact with the plant. Determining the plant's orientation (for recharging) will follow a similar approach. Instead of getting feedback from the mote's accelerometer, the robot will get feedback from the mote's inductive charge coil. It will systematically position itself at regular intervals around the plant's circumference and wiggle the charge coil back and forth. When the coil is lined up properly with the mote's coil, the robot will receive the mote's charge message, signaling that is has found the correct orientation with which it should dock with the plant. In the longer term, we plan a more complete solution to the problem by applying vision techniques and fusing the results.

5 Lessons

In this section we describe several lessons informed by our experience thus far in developing and using PlantCare. Our hope is that these lessons may help guide future design of practical ubiquitous systems.

5.1 Integration Is Paramount

Despite a decade of research in ubiquitous computing, our field still lacks a killer application to drive the techniques and technology being developed. One reason for this, it has been argued, is that ubiquitous computing is about a totality of experience, not individually compelling applications [16]. Consequently, we argue integration should be treated as a central task in system design and support for integration plays a critical role in determining a system's value. We saw this firsthand in the case of PlantCare, where we feel our system's value resides not in the sensor network or robot, but in the cumulative value of their synthesis. Therefore integration efforts must be carefully considered when choosing components at design time. For PlantCare, we found it valuable to assess potential integration difficulty along three axes: connections, semantics, and guarantees.

The most basic integration consideration is how to establish a connection so that the system can communicate with the component in question. While a component may offer considerable affordances, it is of little use if it does not offer an accessible API of some sort. Our experience with PlantCare suggests that a basic component with an easy to use interface often provides more value than a more capable component with a proprietary, poorly documented interface. This issue also strongly affects the middleware infrastructure a ubiquitous application is run on top of. Many systems, like SOAP, standardize on a wire protocol, which makes it easy to write application services in a variety of languages. This in turn makes it easier to communicate with external components by allowing a variety of language choices for a proxy or wrapper. When a single language approach is chosen, it is important to realize that not all languages support integration to the same degree. In the case of PlantCare, a JDBC driver was used to interface with the database, a serial port driver was used to communicate with the mote base station, and HTTP requests were used to communicate with the robot's control software. All three of these were well supported by Java and choosing to write our Rain service in Java thus aided our low-level integration tasks.

A second integration difficulty is resolving semantic differences between communicating components. In PlantCare, for instance, both the robot and the location tracking system had a notion of physical location, but the two notions were slightly different. To integrate these components, an intermediary semantic translation was required. While the translation itself was not difficult, our architecture had to accommodate the inclusion of the translation mechanism. As in the connection discussion above, semantic integration is better supported by some system infrastructures than others. Systems that support only simple point-to-point communication, for instance, can make such translations awkward. Some systems, such as Apache SOAP [17], have improved a simple point-to-point communication model by providing mechanisms for message mediation, allowing an opportunity to perform such translations. Other systems are based on different communication models that inherently support semantic translations. As an example, tuple-store based storage systems like TSpaces [18] provide an obvious mechanism for doing both simple and complex translations to bridge the semantic gaps between components.

Finally, it is also important to recognize the importance of establishing guarantees when integrating third-party components. For example, we wanted PlantCare to be self-maintaining and provide 24/7 unattended service. Unfortunately, some of the external software we were integrating was not designed for such usage. The robot's control software, for instance, was not intended for continuous service and had an interactive startup process. Rather than try to make major changes to the robot code, we plan to make use of virtualization. In PlantCare, virtual machine technology will allow us to start up the robotic control software once in a virtual machine and create a checkpoint. Once created, this checkpoint will allow us to remotely shutdown and restart from the checkpoint whenever anomalous behavior is observed. Wrapping software using virtual machines should help us achieve our overall goal of 24/7 operation, even though the individual components are only modestly reliable.

5.2 Build For Failure

During our work on PlantCare, we have found it valuable to design our system to expect failure as part of routine operation rather than treat errors as exceptional events. For example, our Watchdog discussed in Section 4.3 routinely expects services to fail, and in Section 4.4 we mentioned how our robotic navigation is designed to handle the case of the robot occasionally becoming lost. For a number of reasons, designing for failure is especially valuable in ubiquitous system design. The distributed and dynamic nature of ubiquitous systems make errors more common than in traditional systems. The inexpensive and even disposable nature of sensors and actuators also contribute to an increase in component failures.

For PlantCare, we have employed three techniques to allow our application to better accommodate routine failure: a highly-decoupled architecture, automation, and visualization. As Figure 5 shows, PlantCare was built as a set of peer-to-peer services. While this increased complexity and resulted in greater message traffic, this decoupled architecture greatly aids fault isolation. PlantCare has been built so that with the exception of the generic data store, the failure of any single component only affects a portion of the system. The failure of the robotic control service, for instance, does not affect the gathering and analysis of data, or the queuing of watering events.

Automation has further improved the resilience of PlantCare to routine failures by aiding in error detection and recovery. While simple, our Watchdog service was written to help PlantCare both notice and recover from anomalous behavior.

Finally, adding simple HTML interfaces to our services has greatly aided us in developing and debugging our system. As we have stated earlier, system actuation will always have limits, so it is important that systems support visualization to give users and developers a means of exploring system behavior.

5.3 Traditional Performance Metrics May Not Apply

The quantitative evaluation of a ubiquitous system presents a challenge of its own: which should be measured? For PlantCare, traditional metrics such as bytes-per-message, latencies, and throughputs do not appear to say anything meaningful about the performance of our system. Admittedly, PlantCare is not computationally intensive or driven by hard real-time requirements. Nevertheless, we believe our system is representative of a broad class of valuable ubiquitous systems. Instead of traditional metrics, we find it more compelling to consider more user-oriented performance metrics such as the amount of time people need to spend installing the system, or the amount of time the system takes to restart after a loss of power. Metrics such as these seem especially applicable to the ubiquitous computing domain where the challenges of system configuration and maintenance are becoming increasingly problematic. Unfortunately, these sorts of metrics are difficult to measure. Because these metrics assess overall system properties, components cannot be measured in isolation and the system must be functional before it can be measured. Furthermore, since many of these metrics involve users, measurement is much more expensive and error prone [19]. It is our belief, however, that considering the impact on users is paramount to accurately evaluating the value and practicality of an ubiquitous system and we continue to investigate how to apply such metrics in evaluating our system.

6 Related Work

Autonomic computing is largely focused on reduction of total cost of ownership for large, backend environments like server farms and Internet data centers. Our interests are complementary as we are looking to develop mechanisms that could be deployed in a single home with anywhere from a dozen to thousands of components. Calm computing [20] and invisible computing [21] both relate to our interest in building practical ubiquitous systems from the user's perspective. Calm technology primarily addresses the need to manage information overflow; a steady stream of information is made manageable by presenting it in a format conducive to our peripheral senses. Our approach, on the other hand, seeks to minimize mundane user involvement in managing services. When such interaction is useful or necessary, our goal is to ensure that the interaction occurs at an appropriately high level of abstraction that is clearly understood by a wide range of people. Calm computing techniques could be employed to provide these interactions. Invisible computing strives to seamlessly blend technology and tasks, making the infrastructure effectively invisible to the users. While similar in flavor, we see this as orthogonal to our goal, and we envision both visible and invisible ubiquitous systems that strive for minimal effort on the part of the user.

Many smart environment projects are underway elsewhere, including the EasyLiving [22], iRoom [23] and Aware Home [24] projects. EasyLiving is primarily concerned with providing a coherent user experience, arguing that richer interaction with users requires deeper understanding of the physical environment. Not only do we agree with this, but we believe it applies to achieving more ubiquitous functionality in general. In this regard, our approach complements EasyLiving's user-intensive approach in evaluating the significance of localization and spatial knowledge to achieving ubiquitous functionality. The iRoom project focuses on investigation of human-centric interaction. Like PlantCare, it also recognizes the need to support heterogeneous hardware and software with support for legacy systems, zero-configuration and self-maintenance, and higher-level protocols to achieve maximal interoperability. The Aware Home project is concerned with prototyping new technology. As we have stated elsewhere [25], the great value of such research is its provision of a sandbox where experimental technologies can be safely tested and evaluated. In contrast to this approach, our investigation into practical challenges provides insights into effective design and integration strategies that lead to usable, extensible environments. The approaches are symbiotic in that these insights in turn guide and influence technology-based research.

Numerous applications employ service robots that operate in human environments to assist and serve. Particle filters are often used in robotics for localization [26], and attention has recently been drawn to the Simultaneous Localization and Mapping Problem (SLAM) [14, 13] in mobile autonomous systems. Michaud et al. previously created a self-charging robot [27]. Our efforts are in the proper integration of robots into larger ubiquitous computing applications.

Estrin et al. discuss wireless sensor networks in great detail [28]. We believe the use of a service robot to maintain a sensor network is novel. For more discussion on the relationship of robotics and sensor networks, see our earlier paper [11].

7 Conclusions

In this paper we have described PlantCare, a zero-configuration and distraction-free system for the automatic care of houseplants. This project is part of a larger effort to develop techniques that help ubiquitous computing systems run reliably for long periods of time with little user intervention. We have presented an architecture comprised of wireless sensors, a robot, and a collection of coordinated software services. In addition, we have presented our experiences and lessons learned during the development of the system. While much has been learned from our current efforts, the next steps for the project are to complete our solution and show that it can provide truly autonomous, 24/7 service.

Acknowledgements

We would like to thank Eugene Shih, Jason Jenks and Kevin Chan for their support during the project. We would also like to thank David Patterson for a fruitful discussion about performance evaluation.

References

1. Autonomic Computing Manifesto, http://www.research.ibm.com/autonomic/ manifesto/autonomic_computing.pdf, visited June 2002.
2. J. Hill, R. Szewcyk, A. Woo, D. Culler, S. Hollar, K. Pister. 2000. System Architecture Directions for Networked Sensors. Architectural Support for Programming Languages and Operating Systems 2000.
3. ActivMedia Robotics, http://www.activrobots.com, visited Feb. 2002.
4. S. Thrun, M. Bennewitz, W. Burgard, A. Cremers, F. Dellaert, D. Fox, D. Haehnel, C. Rosenberg, N. Roy, J. Schulte and D. Schulz. 1999. MINERVA: A second generation mobile tour-guide robot. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).
5. Burgard, W., A. Cremers, D. Fox, D. Haehnel, G. Lakemeyer, D. Schulz, W. Steiner and S. Thrun. 1999. Experiences with an interactive museum tour-guide robot. Artificial Intelligence.

6. D. Box et al. Simple Object Access Protocol (SOAP) 1.1, World Wide Web Consortium (W3C), May 2000. http://www.w3.org/TR/2000/NOTE-SOAP-20000508, visited Feb. 2002.
7. M. Kohno et al. An adaptive sensor network system for complex environments. Robotics and Autonomous Systems, Vol. 28, Issues 2-3, August 1999, pp. 115-125.
8. A. Wang et al. Energy-Scalable Protocols for Battery-Operated Microsensor Networks. IEEE Workshop on Signal Processing Systems, 1999, pp. 483-492 .
9. J. Rabaey and et al. PicoRadio Supports Ad Hoc Ultra-Low Power Wireless Networking. IEEE Computer, July 2000, Vol. 33, No. 7, pp. 42-48.
10. M. Bhardwaj, A. Chandrakasan, and T. Garnett. Upper Bounds on the Lifetime of Sensor Networks. IEEE International Conference on Communications, 2001, vol.3 pp. 785-790.
11. A. Lamarca et al. Making Sensor Networks Practical with Robots. To appear in the 2002 International Conference on Pervasive Computing. Intel Research, IRS-TR-02-004.
12. Bronstein et al. Self-Aware Services: Using Bayesian Networks for Detecting Anomalies in Internet-based Services. 2001 IEEE/IFIP International Symposium on Integrated Network Management Proceedings, pp. 623 -638. HP Labs, HPL-2001-23R1.
13. M. Montermerlo, S. Thrun, D. Koller, B. Wegbreit. FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. AAAI, 2002.
14. G. Dissanayake, P. Newman, S. Clark, H.F. Durant-Whyte, M. Csorba. A Solution to the Simultaneous Localization and Map Building (SLAM) Problem. IEEE Transactions of Robotics and Automation, 2001.
15. J. Hightower, R Want, and G Borriello. SpotON: An indoor 3d location sensing technology based on RF signal strength. UW-CSE Tech Report 00-02-02, University of Washington, Department of Computer Science and Engineering, Seattle, WA, Feb. 2000.
16. G. Abowd. Killer applications vs. Killer experiences: achieving a palatable human experience in ubiquitous computing. CHI 2001Workshop: Building the Ubiquitous Computing User Experience.
17. Apache Soap. http://xml.apache.org/soap, visited June 2002.
18. Tobin J. Lehman et al. Hitting the distributed computing sweet spot with Tspaces. Computer Networks, vol. 35, no. 4, Mar. 2001, pp. 457-472.
19. S. Consolvo et al. User Study Techniques in the Design and Evaluation of a Ubicomp Environment. To appear in the 2002 International Conference on Ubiquitous Computing. Intel Research, IRS-TR-02-012.
20. M. Weiser and J. Brown. The Coming Age of Calm Technology. Technical Report, Xerox PARC, October 1996
21. D. A. Norman, The Invisible Computer, MIT Press, 1998
22. B. Brumitt et al. EasyLiving: Technologies for Intelligent Environments. Proc. 2nd Int'l Symp. Handheld and Ubiquitous Computing (HUC2K), Lecture Notes in Computer Science, vol. 1927, Springer-Verlag, Berlin, 2000, pp. 12-29.
23. A. Fox et al. Integrating Information Appliances into an Interactive Space. IEEE Computer Graphics and Applications, v. 20 no. 3, May/June 2000, pp. 54-65.
24. C. Kidd et al. The Aware Home: A Living Laboratory for Ubiquitous Computing Research. Proc. Of the 2nd Int'l Workshop on Cooperative Buildings (CoBuild99).
25. L. Arnstein, G. Borriello, S. Consolvo, B. Franza, C. Hung, J. Su, Q. Zhou. Labscape: Design of a Smart Environment for the Cell Biology Laboratory. To appear in IEEE Pervasive Computing
26. F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte Carlo Localization for Mobile Robotics. ICRA, 1999.

27. F. Michaud et al. Experiences with with an autonomous robot attending AAAI. IEEE Intelligent Systems, 2001, Volume: 16 Issue: 5, pp. 23-29.
28. Embedded, Everywhere: A Research Agenda for Networked Systems of Embedded Computers. Computer Science and Telecommunications Board (CSTB) Report.