Safety Enhancing Mechanisms for Pervasive Computing Systems in Intelligent Environments

Hen-I Yang and Abdelsalam Helal

Computer and Information Science and Engineering Department, University of Florida, Gainesville, FL 32611, USA
{hyang, helal}@cise.ufl.edu

Abstract—Pervasive computing systems provide personalized and intimate services to improve users’ quality of life by weaving computation and communication into the fabric of the environment. With the capability to interact with the physical world and the promise in assisting or managing aspects of users’ daily life, the requirement for safety is high and imminent. The difficulty in providing safety is the result of the dynamicity, complexity, heterogeneity and uncertainly typical in pervasive computing. To devise a viable solution in enhancing safety, we examine and analyze worst-case safety violation scenarios. We then identify four fundamental elements of pervasive computing systems whose individual safety assurances add greatly to the overall system safety. We propose safety enhancing mechanisms for each of the four elements and their interactions.

I. INTRODUCTION

Pervasive computing presents a new computation paradigm that has many unique characteristics, which presents great opportunities but also brings higher risks. The use and integration of sensors and actuators gives these systems the capabilities to interpret and influence the physical world. Many systems provide services that are considered to be personal, and often in environments and settings that are private and intimate. Most applications are designed to be context aware and highly customizable, some even capable of predicting users’ intentions and influencing their behaviors. When the computation and communication are weaved into the fabric of daily life, do users benefit from the convenience of various services, or are they being exposed to unknown risks brought upon them by these same services?

The extensive use of sensors and actuators allows pervasive computing systems to interpret and influence the physical world. They are capable of providing services and assistance by maneuvering electronic devices and physical objects such as appliances, robots or even dumb objects like doors. To the aging elderly population and persons with disabilities, these services mean critical assistance and extended independence; to others, they represent a new level of convenience. However, the capability to interact and influence the physical world implies that the impact of malfunctioned systems is no longer limited to loss of data, waste of time and effort or damage of computing devices; the possibility that users, bystanders, or physical properties can be harmed is as real as those assistance and convenience.

Many pervasive computing systems are intimate [1], because they have extensive interactions with users, and perform personal tasks on behalf of users in close and private settings. Intelligent environments such as smart homes [2] and intelligent vehicles [3, 4] keep extensive records of their users and act upon users’ preferences and limitations. These systems exist in our homes and cars, which are considered to be private, safe and treated with affections. Many services involve our daily routines, including medicine reminder, scheduler, or even personal hygiene [2]. They are integrated into part of users’ life, and users have high expectations and place trust on them. Failure to perform or even causing damages can invoke strong emotions.

The semi-sentient nature of pervasive computing systems allows them to be aware of contexts, to predict and interpret users’ intentions, and to extensively collect fine-grained user information. How do we ensure such systems would not betray the trust placed by their users? The answers might come from similar questions submerged in robotics quite some time back, which are best framed by Isaac Asimov in his famed “three laws of robotics”. Substituting robot with pervasive computing system, the first two laws can be restated as follows,

1. A pervasive computing system may not injure a human being or, through inaction, allow a human being to come to harm.
2. A pervasive computing system must obey orders given to it by human beings except where such orders would conflict with the First Law.

A pervasive computing system shares the same physical abilities as a robot in influencing and interacting with the physical world. However, robots are usually designed as stand-alone systems, while pervasive computing systems are far more user-centric and encompassing, which make these two laws even more pertinent.

Realistically speaking, it is of utmost importance to acknowledge that the impossibility to guarantee the safety of an intelligent environment. Accidents happen, people make mistakes, and Murphy’s law always looms around the corner. A glimpse of local evening news can remind us that despite best intentions and the most diligent efforts, safety may be significantly improved but the risks can never be eliminated. As in real life, weighing risks introduced and the cost to reduce or eliminate them is a continuous process.

What are attainable goals for safety mechanisms in a pervasive computing system? The following is a list of goals in order of difficulties and proactiveness:

1. Addition of new technology, devices and services should not be the primary culprit for surging risks. Similar to cost/benefit analysis for business decisions, the risk/benefit ratio needs to be assessed to ensure additional convenience significantly outweigh the extra risks introduced.
2. With capabilities to sense the environment, interact with physical world and make calm decisions,
pervasive computing systems are excellent candidates to handle emergency should they occur.
3. To push the boundary further, we would like these systems to proactively detect, prevent and manage existing risk factors.

For example, a house can catch fire if an absent-minded person leaves the stove on and forgets. A deployed pervasive computing system should not increase the chance of fire because of improper implementations or invalid operations. Actually it is much preferred if the system could monitor for and eliminate potential causes of fire hazards, or at least respond quickly should a fire starts.

Before safety mechanisms can be established, it is essential to pinpoint the potential safety hazards by examining realistic worst-case scenarios in an intelligent environment. In section 3, we identify and analyze four fundamental elements of pervasive computing systems and devise safety mechanisms based on these findings. Related work is presented in section 4, followed by the conclusion.

II. SCENARIOS AND ANALYSIS

A. Scenarios

1) Conflicting usage of shared resources: The lamps in the house start to turn themselves on and off randomly like a haunted house ever since the new energy saving service is added to the existing subscription of the lighting control services.

2) Invalid operational directives: The meal preparation service follows the exact steps in the recipe and gives instructions when the time comes to turn or add additional ingredients, and automatically shutoff to prevent overcooking. But because of a simple typo by the programmer, the meal preparation service has mistakenly set the target temperature inside the steak to 3500 °F instead of 350 °F and turns the oven into an incinerator.

3) Risks of conflicting side effects: The refrigerator is not closed properly and the cold air spills onto the stove right next to it. To make sure the stew is well heated, the meal preparation service turns up the heat. To maintain constant temperature inside the refrigerator to prevent food from spoiling, the compressor amps up the power. Before long, both have been drawing so much power and a fuse is blown and the house sinks into darkness.

4) Violation of user centric computing: When he wakes up in the morning, the alarm is blaring, the bed is shaking, and he feels his head is throbbing so badly as if it was to explode. But the morning call service decides extra push is necessary this morning. He struggles to climb out of the bed and turns off the alarm, and feels like screaming and throwing up at the same time. He vows that there will never be any “smart” thing in the house from this day on.

B. Analysis

Service oriented architecture (SOA) is widely adopted by pervasive computing systems, with which each individual software or service focuses on accomplishing its own predefined goals. While it is an effective model to handle the dynamics, heterogeneity and complexity of the environment, compatibility issues between services do arise, especially when services depend on one another, interact, or compete and race for shared resources. In a traditional computing system such as PC or server, the operating system usually monitors the use of shared resources and keeps track of the dependency. In pervasive computing systems, the lack of reliable monitoring and arbitration mechanism to oversee the large number of shared heterogeneous resources presents a major problem. The flickering lights exemplify this issue when energy saving and lighting services each unaware of the other’s operation on the shared resource (the lamp). Without proper coordination, the lamp flickers when receiving conflicting directives from two services.

Even when services do not compete for shared resources, the side effects (also known as environmental effects) of the actions can still cause conflicts between independent services. While it is tedious and costly for each service to specify how to resolve contention for shared resources, it may not even be possible to account for all possible interferences caused by the side effects. The unintentional side effects are rather limited in traditional computing, but can be a major source of risks in intelligent environments. The conflict between the fridge and stove in achieving their respective predefined goals in the scenario exemplifies this issue.

Every device has its own supported operations, and the appropriate conditions under which the operations are to be performed. Improper operations and errors made by human operators usually are the result of carelessness, ignorance or misunderstanding. A well-designed device would embed hints and cues to minimize the occurrence of these human errors. For instance, the temperature range on a thermostat and the temperature marks on the dial of an oven inform users the limits of their operations. However, when operated by pervasive computing services, these cues, hints and visual feedbacks would fall to deaf ears, resulting in more frequent errors and worse damages. Without proper assistance and restraints, programmers who are usually not onsite are more detached from the actual devices, hence do not have the same access to visual cues and physical feedbacks that might have signaled a mistake. The software that controls devices can also receive unreasonable commands because of an error as simple as a typo. The incineration of the steak is the result of an erroneously added a 0 in the cooking direction.

Context awareness is central to many pervasive computing systems. Context reflects the condition of the surrounding environment and summarizes users’ current status and intentions. When dealing with user related contexts, it often involves proper interpretations or even predictions of users’ intentions. However, even human beings cannot always read other people’s intentions correctly, let alone asking a computing system to do so. The misinterpretations, however, may cause serious harm and injury to users, such as the case when morning call service does not properly recognizes Jason’s physical condition and worsens his headache.

C. Discussion

Just like many issues in computing systems such as binding and introduction of parallelism, how and where to implement and enforce safety mechanisms presents an interesting tradeoff between flexibility and efficiency. On
one hand, middleware is widely employed in complex pervasive computing systems such as those in intelligent environments. It is the nexus of information flow where components are brokered, common utilities or discovery mechanism are available, hence presents a great opportunity to intercept, examine and intervene with the data coming in and commands going out. Middleware is effective in detecting the risks resulting from dynamic interactions and exceptions arose during operations. On the other hand, the overall safety and the quality of the system can be greatly enhanced if safety features can be included as the system is being developed. Proper APIs and programming tools can introduce and validate that appropriate measures are in place to provide certain level of assurance. Compile time support is more effective at defining exception handling routines, enforcing event driven program model, prioritizing operations and alignments, or specifying impermissible contexts.

Safety issues in pervasive computing are broad and complex, relying on safety measures in either middleware or programming model alone will not be sufficient. It requires a collaborative effort between the middleware at run time and the programming aid at the implementation time. The middleware handles the enforcement of safety principles and dynamic behaviors, while the programming tool ensures the target systems support safety API and eliminates statically determinable unsafe operations.

III. SAFETY MECHANISMS FOR PERVERSIVE COMPUTING SYSTEMS

A. Applicable Existing Safety Mechanisms

How do we make an intelligent environment safer? Safety is not a new concept, and many existing mechanisms are applicable to pervasive computing systems. 1. Implementing fail-safe physical safety mechanisms in addition to electric or software safety is a prudent design decision. 2. Most of the safety mechanisms for computer systems and networks such as authentication, capabilities and security protocols can be applied with little or minor modifications. 3. Since failure is the norm in pervasive computing, any mechanisms that enhance the robustness and availability to allow systems work through failures can enhance the safety of the overall system.

It is a wise decision to leave the original physical device or interfaces in place as a backup when integrating with “smart devices”. For instance, to make sure the elderly is safe when the fall detection service fails to detect an emergency, it is better if the residents still carry an emergency button. To ensure the escape route is open when a fire disrupts the network or power line for the automatic door control, the regular door knob should be kept in place so people can always open the door. Using the physical device as backup in cost-effective, as these mechanisms are often already in place in regular environments. The reliability is also greatly increased because they seldom fail.

Typical safety measures for computers and networks, such as authentication, capabilities, security and privacy have all been well studied and plenty of mechanisms have been widely adopted. Applying these techniques to pervasive computing systems usually require little or minor changes. Because of the number of devices in a typical intelligent environment, and many of them are low-end sensors, failure is considered the norm during operations. Mechanisms allow dynamic brokerage [5], failure compensation using virtual sensor and service re-planning [6] to improve the availability of services all contribute to the safety of the system.

Our research effort focuses on the safety mechanisms that are specific to pervasive computing systems. The potential higher penalties associated with the risks justify designing implementation time and run time mechanisms to enhance the safety of pervasive computing systems.

B. Four Fundamental Elements

To devise effective safety mechanisms, we have to understand where the risks may arise, and what the components that need to be protected are. A quick survey of existing pervasive computing systems and an analysis of their capabilities reveal to us that there are four fundamental elements in pervasive computing systems: device, service, user and space, as shown in Figure 1.

![Figure 1. Four fundamental elements](image)

Device: The capabilities to interpret and interact with physical world starts with devices. Whether it is a sensor, actuator or smart appliances, a device can be classified as the source of data, the recipient of commands or both. Devices can become the source of risks when they receive improper instructions or operate outside of normal operational range. Device description is used to explicitly capture or model semantic information, domain value bounds (temperature, luminance, others), operation constraints (range, maximum usage frequency, enumeration of discrete values), physical medium description, interaction protocol, and others. By providing such device description information, the middleware should be able to operate and utilize the device while enhancing the device and overall system safety.

Service: The dynamicity and heterogeneity of these systems make SOA or SOA-like architectures a prevalent practice in pervasive computing. Service is at the core of most of these systems. When employing SOA, even devices are represented as services. Services frequently exchange information and interact with others, and sometimes they can be chained or connected to create more complicated services. Each service is usually designed to fulfill very specific purposes, and has very specific requirements on the availability of other services or the existence of configurations in order to accomplish its goals. In the dynamic environment of pervasive computing, services present the smallest and simplest entity that can be well-regulated. Services can introduce risks when they violate rules on sharing resources, engage in race conditions or deadlock, or own conflicting objectives from other services. The lifecycle of services can be appropriately modeled as a state machine and their interactions can be described by interfaces.

User: Any system modeling and analysis attempts to model human behavior is laborious rarely generate any accurate results. But users are critical because they have
the final say in deciding what the system need to do, and sometimes even how they should be done. Users are the single most risky factor of all, more danger is caused by the carelessness, ignorance or misunderstanding of users than problems in the system. There is not too much we can do to prevent users from doing what they do, but we can monitor users’ status to keep them out of harms’ way, and align systems’ behavior with users’ intentions as long as they are safe. User profile is designed to model users, which preferably should be both human and machine readable. User profiles are divided into two parts, the static profile, which describes users’ preferences, limitations, and priorities, is relatively stable and usually not affected by the system; the dynamic profile, on the other hand, consists of the user related data gathered by the system, for instance, the current location of the user, the blood pressure and glucose level attained by medical sensors, and is continuously updated by the system.

Space: Many do not consider a space itself to be a critical element in the system. However, a space is an important element because the status of the space as a whole is critical for context-awareness; a space also encompasses all other elements within, in particular the status of services, devices and users as well as the interactions between them; side affects usually cannot be captured by devices or services, but can be described by measurable changes in a space. Based on functions, social contexts and the locale of a space, there are different restrictions and interpretations of the contexts. For instance, the interpretation of cleanliness in a clean room will be different from a butcher shop; the acceptable temperature and humidity also drastically differ in an engine room from an operation room. The differences can be defined in the context interpretation descriptions so raw data can be interpreted accordingly. The state of a space can be captured using ontology-based context graph, which not only tells the system how to interpret low-level sensor readings into contexts, but can be used to monitor and visualize the currently active contexts.

Operating in a dynamic, heterogeneous and uncertain environment, the safety issue of pervasive computing systems is complex. Since they are all composed of these four fundamental elements, any potential risks would involve at least one of them. Employing the strategy of “divide and conquer”, we devise the safety mechanisms by first securing each element, and then reducing risks occurred during their interactions. Finally, since the space is the all-encompassing element, we use the safety mechanisms for space as a safety net to capture anything unintentional or fall through the crack.

C. Safety Mechanisms for Pervasive Computing Systems

1) Static safety mechanisms – Prioritized Safety API

a) State machine and service safety interface: This is a service protection mechanism to allow smooth operations and enhance control and tracking of services in dynamic environment typical for pervasive computing, we adopt the event-driven programming paradigm, which is based on a state machine for all services as shown in Figure 2. init is the state when a service object first instantiated; a service is binded when successfully acquired all dependent services; before execution a service needs to be aligned with users’ preferences and limitations; only then can a service start executing. When emergency hits, a service can move into emergencyPowerDown from any other state. Service safety interface is the manifestation of the transitions in the state machine.

b) Mandatory power down sequence for all service: onEmergencyPowerDown() method, part of the service safety interface, is mandatory for every service, in which programmers would describe the necessary steps, such as setting variable to a safe value (e.g. set the oven temperature to 0 °F before stopping the service), releasing any shared resource it currently holds, report power down status to service registry, and notifying services it currently collaborates with, to ensure a service can be parked safely. This sequence is required to be reliable, atomic and follows a well defined protocol for termination in case of failures.

Figure 2. State diagram for services

- Preemptive methods with priority: Methods are designated with one of the two priorities, high priority for emergency handling methods, and regular priority for all other methods. Regular priority methods should be implemented as preemptible using similar techniques as preemptive concurrent programming.

2) Dynamic safety mechanisms in the middleware: Figure 3 summarizes the dynamic safety mechanisms supported by the middleware. The details of some of these mechanisms are given in the following subsections.

a) Device safety checker: Each instruction issued to the devices needs to go through additional checking to ensure their conformance to the limitations of the target device, including the method invoked is supported, the operands are within the normal operational range, and the frequency of alternating instructions is acceptable. From the perspective of object-oriented programming, it is preferred these checkings are performed within the device service object, but devices with small footprint or higher code reuse, they can also be outsourced to Device Safety Checker with device id and the parameters of instructions.

b) Context manager and emergency detector: Context manager uses context graph to interpret raw sensor readings into higher-level contexts Contexts are effective ways to describe the overall status of everything in the space, including the users of the system. Active contexts are provided to services for making context-aware decisions. Emergency detector also check these active contexts to see if any dangerous impermissible contexts has come true. Should any impermissible context
be triggered, the emergency detector would invoke the handling routine in the emergency handler vector (EHV) using the associated emergency number.

d) Emergency Handler Vector (EHV): This is the centralized emergency response routine similar to the interrupt vector. When each device and service first joins the system, they are required to deposit emergency handlers into EHV should any major problem arises; there is also a handler for each impermissible context defined in the context graph. When an emergency handler is activated, EHV issues preemptive high-priority calls to suspend services, park devices, decouple dependencies or issue overrides to reverse the emergency situation.

![Figure 3. Dynamic safety mechanisms in the middleware](image)

IV. RELATED WORK

There are limited studies on the safety issues of pervasive computing systems. A compile time semantic checking mechanism on a functional/logic programming language is proposed for implementation and verification of safe pervasive computing systems [7]. But the result cannot be directly applied to systems implemented using more popular languages, as is the case for most systems implemented. Leveson uses software engineering practice to devise safety modeling using fault tree and probability analysis, and identifies requirements and design for safety [8]. Her model and practice help to shape our solution, but do not provide any immediately applicable design specifics for pervasive computing systems.

There are other safety studies on pervasive computing in particular application domains, more specifically, those applicable to intelligent vehicles [3, 4] and smart hospital [9]. Our goal in this study is to create a generic safety mechanism that is applicable to a wide range of pervasive computing systems in intelligent environments.

Many existing middleware have been designed for pervasive computing [9, 10], but they have other primary design goals such as quality of contexts and ease of integration. None was designed to address the safety issue.

V. CONCLUSION

Safety is defined as “the condition of being safe from undergoing or causing hurt, injury or loss”. For pervasive computing systems, especially those deployed in intelligent environments, the concern over safety should as important as the effectiveness or usability. These systems have the ability to influence the physical world, and often interact with their users in a much more intimate way than traditional ones, hence any unsafe operation would cause more drastic physical and psychological damages. With all the assistance and convenience that come with these systems, it is crucial to ensure there is no unreasonable or unnecessary increase in risks to the well-being and property of users.

The dynamicity, complexity, heterogeneity and uncertainty of pervasive computing systems pose serious challenges in guaranteeing safety. The lack of a reliable monitoring and arbitration mechanism such as the ones provided in operating systems in traditional computers contributes to the problem. By examining fundamental elements in pervasive computing systems, the analysis and modeling of devices, services, users and space shed some lights on how to make these systems safer. Securing the safety of each element and managing the risks of the interactions among them provides a promising approach in making pervasive computing systems in intelligent environment a safer endeavor.

REFERENCES


