Project Number: DCB-01JF

ADAPTIVE HOME AUTOMATION

A Major Qualifying Project

submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

Joshua W. Frappier

Date: June 1, 2001

Approved:

Professor David C. Brown, Major Advisor
Computer Science Department

Abstract

Can a home be intelligent? Can the tediousness of everyday tasks essentially be removed from our lives by a home that makes decisions and acts as humans do? Current building control systems are becoming inadequate to elegantly support the ever increasing number of devices in the home. An architecture to support intelligent device control that adapts to the behavioral patterns of a user is proposed and evaluated. The results are encouraging, hopefully providing a catalyst for future implementations.

Acknowledgements

First and foremost, I would like to thank the Lord for opening the doors that permitted this project to even happen.

Thanks to HTS for providing the catalyst for the project as well as their generous contributions of hardware resources. Their continuing understanding of my academic situation and consent to allow me to remain true to my vision, has made the MQP a great learning experience.

Thanks also to David Brown, who was brave enough to advise an "on-campus" MQP from 6000 kilometers away.

Table of Contents

Table of Figures

1. Introduction

1.1. Intelligent Homes?

Can a home be intelligent? Can the tediousness of everyday tasks essentially be removed from our lives by a home that makes decisions and acts as humans do?

Such broad questions can only be answered by examining the current trends in home control and automation as well as the rapidly advancing research geared at providing such services. By analyzing the current state, it is anticipated that practical directions for research and development will be determined, bringing artificial intelligence techniques out of the lab and into the development of home control systems. In addition, it is expected that the feasibility of developing a more intelligent home automation system for the open home market will follow such a determination.

1.2. The Current State

Home and building control systems have existed for many years. To date, almost all of these systems have implemented complex schemes to make pre-programmed decisions regarding user comfort and energy conservation. Automation systems are programmed to control lighting, climate control, security, entertainment, and a variety of other resources. Many of these systems operate within the confines of a proprietary system architecture, while others adhere to international bus standards for device control and communication. Complex administrative software is responsible for controlling devices on the control network based on user preferences.

These systems have in the past been adequate for building control. However, with the increasing number of devices in a home, the current architectures are becoming too complex. The amount of time required for system configuration, maintenance, and complicated system interfaces are quickly

becoming overwhelming for the average home user. This can be observed by the lack of such systems currently installed in both new and older homes.

Homeowners and hobbyists who are not able to afford complex, industrial strength control systems, which often require substantial costs for building rewiring alone, have in the past resorted to a variety of novelty control systems, (for example, automation systems based on the X10 protocol). While these systems provide some degree of control over resources, both locally and remotely, their inability to efficiently meld into one cohesive system makes them clumsy and inefficient.

1.3. The Current Research

In recent years, significant research has been focused on creating adaptive environments. These environments are rooms or entire homes that learn about their inhabitants' behavioral patterns. Decisions for building control are made based on those learned behavioral patterns, as opposed to making decisions solely based on pre-programmed control criteria.

One direction of this adaptive environment research centers on high technology, high bandwidth applications that are brimming with "cool" for an expected end-user (speech recognition, speech synthesis, user tracking, integrated intelligent Internet agents, etc.). These systems tend to be very complex as their research goals are oriented toward the theory behind adaptive learning and intelligent agents. In order to operate, these systems require excessive computing power, expensive hardware, and very high maintenance in order to provide the sought intelligence. An example of such a system would be the Michael Coen's HAL project at the MIT Artificial Intelligence Laboratory [Coen, 1999]

A second direction that is being followed in the area of home automation research lies far closer to a practical application of theory. University

research groups, such as Michael Mozer's Adaptive House project [Mozer, 1998], have implemented and tested adaptive control systems in actual homes. The results of these research projects are more likely to predict the immediate future in adaptive home automation.

1.4. The Current Problems

Observing current systems and trends, both in research and in practice, the following weaknesses emerge:

- **Installation/Configuration complexity**: System complexity does not yet allow a normal home user to independently install a complete home automation system. Contractors need to be hired to perform the appropriate hardware and software installation and configuration.
- **Cost**: Installation costs as well as hardware are still expensive.
- **Static binding of devices**: Often, switches and sensors are statically linked with specific devices. What happens when the effect of such a link is undesired? Someone must manually change the link between the devices. In the case of multimedia, consider video playback devices that are connected to one television. This static link requires the purchase of a playback device for *every* television where a user may want to watch videos. This is not efficient use of resources.
- **Lack of total system integration**: Very few home automation solutions provide access to all of a home's resources through one interface. Bringing all home resources under one umbrella of control offers more convenience to the user.
- **Inability to adapt**: Current systems do not *learn* the preferences of users. They merely react to pre-programmed control criteria. Learning user preferences gradually frees the user from repetitive tasks without explicitly programming the control system.

1.5. The State of the Future

In attempting to predict the immediate direction of home automation, it is necessary to note the natural progression of real world systems in general. Individual components are designed and employed in industry, each client utilizing the component in proprietary ways. Eventually, the development of hybrid components contributes to simplifying component interaction. Finally, many heterogeneous components are brought under one standardized framework designed integrate the components using one easily manageable interface.

This same trend can be seen in home automation. Many new products are being developed for the home market that are merely combinations of technologies, some of which have existed for decades. Products such as wireless telephone jacks, integrated television/DVD players, voice activated lamps, and net-enabled coffee machines are all of a new genre of devices integrating the high tech with the mundane.

Many of these technologies that are being integrated with our familiar home appliances have a huge potential for affecting the way we spend time in our homes. Imagine a home in which the mere utterance of "Computer, show me CNN," would immediately display the live news feed on whatever video display was closest to you. Or consider an application that would detect the accidental fall of an elderly woman in her home and immediately contact ambulatory services and a family member for her. Applications for home automation could also be as simple as a user's favorite radio station following her as she walks from room to room in a home.

The following technologies are strong candidates for changing the future of our homes and how we will interact with them:

- **Wireless Communication**:

  The recent boom of wireless networking at affordable prices is enabling homeowners to install wireless networking solutions that allow for both easier device installation and the delivery of resources completely independent of location.

- **Improved Multisensors**:

  Desired functionality is often dependent on more than just the intelligence of a central processor. Sensors also provide much of the necessary information for intelligent action. In order to achieve some of the desired applications in home automation, current sensor technology is still quite expensive. Often, the technology does not exist at all. Cheap solutions need to be found for true presence detection, user tracking, and medical surveillance, etc., to allow for many of the desired applications.

- **Universal Multimedia Devices**:

  The desire for multimedia in the home is increasing strongly. In order to eliminate redundant appliances, technologies such as television, video, telecommunications, and the Internet are continually finding new ways to merge. As this process continues, the emergence of devices that are capable of handling many types of digital media will allow extensive freedom for the user in both entertainment and home control.

- **Voice Recognition and Synthesis**:

  Ubiquitous computing is becoming a popular buzzword in home automation circles. **Ubiquitous computing** describes the interaction of a user with a computer in ways that are natural to him or her. In home automation, allowing the user complete control over an environment in a way that is natural and effortless is crucial. Voice, being one of the primary forms for human communication, is a strong interface dynamic in ubiquitous computing. As research in voice

recognition and synthesis advances, the realization of integration grows nearer every day.

- **Integrated Intelligence**:
  Because today's control systems are primarily procedural, the tendency can sometimes be that the computer begins to control the user more than the human controls the computer. Why should a user be required to press a switch if she desires something as basic as light or heat? By creating software architectures that will learn and adapt to the user's actions, mundane tasks, such as lighting and heating control, can slowly be assumed by the system, freeing the user for more interesting activities.

1.6. Project Focus

It is clear that the study of home automation technology, and implementation techniques as a whole, is far beyond the scope of a single Major Qualifying Project (MQP). Each of the key technologies listed above presents its own list of problems, each of which could subsequently constitute an entire MQP.

This research project will concentrate primarily on the software that provides the *infrastructure* for the *intelligent* control of devices within a home. The specific goals of the project are to design and evaluate a system architecture that:

1. eliminates the need for static device binding for control.
2. learns and adapts to an inhabitant's behavioral patterns, adjusting control to:
   a. maximize user comfort.
   b. minimize wasted energy usage.
   c. maintain security.
3. unites all devices under one control architecture.

4. allows devices to function intelligently, even if they are separated from the system during a system crash.
5. provides procedural device control for high-risk situations.
6. allows for plug-and-play operation of newly added devices to the system.

This project attempts to approach the problems of home automation from a fairly broad perspective. As home automation has been a slowly growing field for the past few decades, it may be helpful to define a new architecture, with a fresh outlook, temporarily putting aside the biases of the past few years. It is our hope to do just that.

2. A Background in Intelligence

2.1. Introduction

The idea of intelligent machines often conjures visions of computers like the famous HAL from "2001: A Space Odyssey" [Stork, 1996] or more practically IBM's Deep Blue, the chess juggernaut that defeated world champion chess player Gary Kasparov in 1997 [IBM, 2001]. Whether fictional or factual, the mere concept of such machines is changing the way we look at computing for the next millennium, and more importantly, stimulates curiosity regarding how they will interact with us.

One may ask, "Why are we talking about artificial intelligence in a report about home automation?". The answer is simple. Artificial intelligence often addresses the design and implementation of seemingly intelligent robots. Robots sense and perceive their environment, devise a plan to solve a specific problem, and ultimately affect their environment in such a way as to advance the solution to the problem. Our homes can be seen in a similar fashion. A home can be seen as merely a robot turned inside out. Thermostats, smoke detectors, and other sensors inform a central brain about the environment, control decisions are then planned and then ultimately the environment is altered by radiators, lamps, and other devices.

Realizing intelligent machines in practical ways has been the task of Artificial Intelligence for decades and is still only a budding research field. AI intelligence paradigms are in a constant state of flux, changing with our own observations of both human intelligence and other seemingly intelligent behaviors found in nature.

Presented here are two major approaches to adaptive artificial intelligence that may help us in advancing home automation. The first, classical artificial intelligence, despite its name, is still the primary paradigm for AI systems. The second, embodied cognitive science, demonstrates a shift in perspective happening in some areas of artificial intelligence. This section is not

designed to be a comprehensive introduction to artificial intelligence, rather it merely discusses some of the major points of AI that may apply to achieving adaptive home automation.

2.2. A Definition of Intelligence

As this research attempts to implement an *intelligent* control architecture, a definition of intelligence may be a good starting point. Unfortunately, intelligence means different things to different people. Different experts have different opinions. Here are just a few as cited by [Pfeifer & Scheier, 1999]:

- The ability to carry on abstract thinking. (L. M. Terman)
- Having learned or ability to learn to adjust oneself to the environment. (S. S. Colvin)
- The ability to adapt oneself adequately to relatively new situations in life. (R. Pintner)
- A biological mechanism by which the effects of a complexity of stimuli are brought together and given a somewhat unified effect in behavior. (J. Peterson)
- The capacity to acquire capacity. (H. Woodrow)
- The capacity to learn or profit by experience. (W. F. Dearborn)

Regardless of the specific definition, intelligence generally encompasses the concepts of *learning* from mistakes and new problems being solved by *adaptation*. Let us now look at how this intelligence has been achieved in the realm of artificial intelligence.

2.3. Classical Artificial Intelligence

Since the late 1950's, computer scientists have spent significant energy in advancing the analogy between the operation of the human brain and the

way a computer processes information. For psychologists, it was for the first time that humans were seen as computational beings, perceiving their environment, thinking about it, and consequently behaving in some relevant manner (the "sense-think-act" cycle) [Russell & Norvig, 1995].

The trend quickly became to classify all human activity into information processing terms. It seemed as though all human activity could be quantified into some sophisticated algorithm acting on the input received from the environment and producing meaningful output. Functionalism became a popular paradigm, claiming that intelligent processes need not be tied to specific hardware to reflect the same functionality. For example, both humans and computers can multiply two numbers, showing the algorithm to be key and not the hardware.

Today, research areas for classical artificial intelligence *tend* towards problem solving, knowledge and reasoning, acting logically, uncertain knowledge and reasoning, learning, communication, perceiving, and acting [Russell & Norvig, 1995].

Following the form of classical artificial intelligence, generalized principles have arisen governing the overall design of intelligent agents. The following is a list of design principles for classical artificial intelligence, adapted from [Pfeifer & Scheier, 1999] (remember, this section is only a discussion and the concepts presented here many not directly be implemented in the design):

1. **Model as a computer program**: Assumes that good theories are expressed in information processing terms.

2. **Goal-based designs**: The actions of an agent should be derived from goals and knowledge of how to achieve the goals. From goals, plans are generated that can be executed. Goals are organized in hierarchies.

3. **Rational agents**: If a rational agent has a goal and it knows that a particular action will bring the agent closer to the goal, it will choose that action for execution. Essentially, a rational agent is one that does the right thing [Russell & Norvig, 1995].

4. **Modularity**: Models should be built in modular ways. Modules include perception, learning, memory, planning, problem solving and reasoning, plan execution, language, and communication.

5. **Sense-think-act cycle**: The operating principle is as follows: first the environment is sensed and mapped onto an internal representation. This information is processed, leading to a plan for an action. The action is then executed.
6. **Central information processing architecture**: Information from various sensors must be integrated into a central representational structure in short-term memory. This integration requires information from long-term memory. Memory consists of structures that are stored and later retrieved.

7. **Top-down design**: The design procedure is as follows: specify the knowledge level (specify what the agent should be able to do), derive the logical level (formalization of how the initial specification is to be achieved), and implementation level (produce the actual code).

It is now important to note, these briefly described design fundamentals are naturally not without their complications. The classical approach to intelligence has in the past decade received much criticism because of its failure to address many practical implementation issues:

1. **Robustness**:

   Traditional AI systems tend to lack fault-tolerance unless exception handling for specific situations is explicitly programmed into the

system. More importantly, these systems tend to lack simple methods for generalizing their environment to account for novel situations. For agents that are designed to operate in the real world, this can be a huge problem as two real world situations are never exactly the same.

2. **Sequential processing**:

   The sequential nature of today's processing architectures naturally leads to the development of sequentially processing agents in AI applications. Essentially this is not a problem, however, when attempting to design intelligent agents that mimic the massively parallel nature of the human brain, this approach serves only to complicate design.

3. **Real-time processing**:

   With systems that require a real-time response to their constantly changing environments, such as home automation, a typical central processing approach can prove quite problematic. If all sensory information must first be collected by a central device, be processed (integrated, mapped, be used to generate action sequences), and ultimately be converted to motor control signals, real-time response can be difficult at best.

4. **Frame problem**:

   As agents interact with their environment, it is possible that the environment could change. These changes could severely impact the future decision-making processes of the agent. Updating the agent's internal representation of the environment is a costly operation whose execution time will greatly affect any agent's effectiveness in a demanding environment. The frame problem is intrinsic to any world-modeling approach to agent design.

5. **Symbol-grounding**:

   As agents interact with their environment, it becomes necessary to "ground" symbols to actual situations or objects that the agent encounters for later reference. It turns out that this is not at all a simple task for symbolic systems as the number of possible associations that are related to one symbol can branch at what seems like an infinite rate.

6. **Embodiment and Situatedness**:

   The embodiment problem addresses the fact that abstract algorithms do not affect the real world. For an agent that has not been given a "body", it cannot be determined with confidence that it truly can cope in the real world. Situatedness is very closely related to embodiment. Only by giving an agent the ability to perceive its environment on its own, can it begin to make intelligently planned actions. An agent's unique interpretation of its environment is the first step to solving the symbol-grounding problem.

While classical concepts in artificial intelligence still dominate mainstream application of computer intelligence, it is clear that the magnitude of some of the issues seen so far will inhibit the ultimate future of *certain* intelligent applications.

Let us now discuss an alternative to the classical mindset that may begin to solve some of the problems discussed above.

2.4. Embodied Cognitive Science

The past ten years have brought about a significant (and still growing) paradigm shift in the academic realm of artificial intelligence. Observing natural behavior in both humans and other organisms has led to new notions

of how *seemingly* intelligent behavior may occur. The most astounding of these observations is that intelligent behavior may not always be reduced to one specific internal mechanism. This discovery, along with the desire to mimic natural systems, has led to some new design methods which build on the currently accepted paradigms [Pfeifer & Scheier, 1999].

2.4.1. Complete Autonomous Agents

For the rest of the paper, the word agent will be employed quite liberally. However, for reference, a general, well-known definition will be provided for reference. An **agent** is anything that can be viewed as perceiving its environment through sensors and acting upon the environment through effectors [Russell & Norvig, 1995]. It is important to note that the term "agent" can be used to describe an entire system *or* an agent can be one of many agents whose individual functionalities, observed collectively, comprise an entire system.

It can be argued that the observed tasks of an agent are merely the result of many instinctual actions and not from an agent's explicit sense of the task at all. This "frame-of-reference" problem points to the fact that what the observer sees is not always what it appears to be.

As an example, let us briefly look at Herbert Simon's illustration of an ant on the beach. When observing an ant returning to its nest, the path that it follows can look quite complex from the observer's perspective. The ant, however, has no concept of the nature of its path. It is possible that the ant's behavior is composed of many simple rules, that when followed, produce the desired path. For instance, the rules may be: (a) if obstacle on left, turn right, and (b) if obstacle on right, turn left. The complexity of the ant's path would then be dependent on the interaction of the ant with its environment and not solely on internal mechanisms [Simon, 1969].

When speaking in terms of agents that have the aforementioned type of controlled activity, it becomes necessary to speak about them as complete entities. Complete agents are defined as those who are autonomous, self-sufficient, embodied, and situated. The ability to adapt with experience is also a key property of a complete autonomous agent. [Pfeifer & Scheier, 1999]

Autonomy describes an agent's ability to function with some degree of freedom from external control. Agents will always have some degree of dependency upon either their environment or other agents, or possibly both. Autonomy can also be seen as a measure of the relationship between agents (assuming environmental resources can be viewed as agents). An example of an autonomous agent would be a land-roving robot that is designed for exploration of a distant planet. The lack of operator control requires the robot to make rational decisions based solely on it's own perception and experience.

Embodiment is an attribute of an agent that physically has a "body" that interacts with its environment. It is fundamental to the usefulness of agents in the real world. By opening an agent up to the harsh reality of a true environment, it provides the agent the ability to better evaluate a situation. Real world interaction also allows the agent designer to take advantage of the physics of the environment for increased performance.

A **situated** agent is one who interacts with the environment completely independent of human intervention. An agent that is situated will perceive its environment only through the use of it's own sensors and interaction with the environment.

Self-sufficiency describes an agent's ability to maintain itself over extended periods of time. In the case of robotic agents, this would translate to the maintenance of power supply, internal temperature

regulation, and the avoidance of harmful obstacles. In order to maintain these states within a constantly changing environment, it is also necessary for the agent to learn about and adapt to its environment over time.

Adaptation, or the ability to adjust oneself to the environment, is directly related to agent intelligence. Every agent has certain functional requirements, and occasionally an agent needs to dynamically alter its behavior to meet those requirements. Adaptation is the process of an agent dynamically meeting those requirements. Adaptation can be broken into four specific categories: (1) evolutionary adaptation, genetic changes in a species over time; (2) physiological adaptation, individual agent changes to adapt to long-term environmental conditions; (3) sensory adaptation, sensors becoming accustomed to certain environmental stimuli; and (4) learning adaptation, the ability for agents to adjust to all other environmental changes. Evolutionary and learning adaptations most heavily affect the study of embodied cognitive science in artificial intelligence.

When designing an autonomous agent, its "ecological niche" must also be carefully considered. An agent's **ecological niche** defines the environmental conditions under which the agent is expected to be able to function. Every agent is designed for certain ecological niche, and while that at first appears to be a disadvantage, it actually allow for certain simplifications in design. By knowing specifically the environment in which an agent will function, it can be designed to take advantage of the physical properties of that environment.

2.4.2. The Subsumption Architecture

In 1986, Rodney Brooks from the MIT Artificial Intelligence Laboratory published a paper that had a significant impact on the field

of AI [Brooks, 1986]. Brooks presented the first comprehensive engineering approach to elegantly bringing multiple sensors and actors under an architecture that was robust as well as incrementally extendable – the subsumption architecture.

The subsumption architecture organizes agent behavior into multiple layers of functionality. Less complex behaviors are implemented first and more complex behaviors are built later, as they most likely require lower levels of behavior to accomplish their task.

All layers of the system receive all sensory information and can control devices without consulting or passing through other layers. This means that higher levels of behavior can inhibit the behavior of lower levels. Lower levels of behavior function independently of higher levels unless higher levels specifically need to temporarily inhibit some lower level behavior.

The subsumption architecture should be given serious attention in the study of adaptive intelligence because of its following advantages:

1. **Interaction focus**: takes focus away from central information processing and places focus on the interaction between sensors and actors. This coincides with a more neurobiological view of system architecture.

2. **Embodiment oriented**: subsumption very much assumes that the system to be created is to interact with the real world and exploit the environment.

3. **Parallel control**: subsumption provides for many parallel independent processes as opposed to a centralized control process.

4. **Evolutionary**: subsumption assumes that once a layer of functionality has been designed, it should not have to be later redesigned. This principle is inspired by evolutionary factors.

2.4.3. Emergence

"We realize interesting and complex behaviors can be had via the aggregates of simpler ones; groups of simple agents can be combined to do interesting things." [Coen, 1997]

In designing agents, it is often advantageous to take into account the known nature of the agent environment and attempt to provide for *emergent* behavior. **Emergence** can be described in the following ways [Pfeifer & Scheier, 1999]:

1. It is the property of a system that is not contained in any one of its parts.
2. It concerns behavior that results from agent-environment interaction when the behavior has not been explicitly preprogrammed.
3. Emergent behaviors are often behaviors that are not fully understood.

Designing for emergent behavior requires that an agent designer first develops low-level ontologies, provides sensory redundancy, and allows for self-organization within the system. Specific emergence design methodologies do not currently exist, so design tends to be very tailored to a specific project and requires designer ingenuity.

2.4.4. Design Principles

Generalized principles have arisen governing the overall design of complete autonomous agents. The following is a list of design principles (adapted from [Pfeifer & Scheier, 1999]):

1. **The three-constituents principle**: designing autonomous agents always involves three constituents: (1) definition of ecological niche, (2) definition of desired behaviors and tasks, and (3) design of the agent.

2. **The complete-agent principle**: the agents of interest are the complete agents, i.e., agents that are autonomous, self-sufficient, embodied, and situated.

3. **The principle of parallel, loosely coupled processes**: intelligence is emergent from an agent-environment interaction based on a large number of parallel, loosely coupled processes that run asynchronously and are connected to the agent's sensory-motor apparatus.

4. **The principle of sensory-motor coordination**: all intelligent behavior (e.g., perception, categorization, memory) is to be conceived as a sensory-motor coordination that serves to structure the sensory input.

5. **The principle of cheap designs**: designs must be parsimonious and exploit the physics and constraints of the ecological niche.
6. **The redundancy principle**: sensory systems must be designed based on different sensory channels with potential information overlap.

7. **The principle of ecological balance**: the "complexity" of the agent has to match the complexity of the task environment. In particular, given a certain task environment, there has to be a match among the complexity of sensors, motor system, and neural substrate.

8. **The value principle**: the agent has to be equipped with a value system and with mechanisms for self-supervised learning employing principles of self-organization.

2.5. Conclusion

It examining the evolution of artificial intelligence over the past decade, it is clear that the field is in a heavy state of flux and will most likely continue to be in the coming years. In this project we will attempt to follow the cognitive design approach whenever possible, opting for the more fluid, less procedural solution of the modern outlook.

3. System Design

Now that we have a sufficient background in artificial intelligence concepts as well as specific system design goals, it is necessary to bring the two together and discover how artificial intelligence design principles will most likely be able to aid us in meeting our goals.

Our study of artificial intelligence led us quickly to the concept of an "agent". It is not difficult to imagine every device in a home being represented by an agent. The distributed nature of home automation encourages us to use the concept of an agent in a distributed manner. The collective functionality of all of the devices (agents) in a home is what creates our complete home automation solution. So the first design goal will be to create a distributed, multi-agent system.

We now have the task of organizing these agents in some logical fashion. Potentially, hundreds of agents could exist within a home. These agents can not simply *exist* in an environment and be expected to function in an intelligent fashion. The subsumption architecture, discussed in Section 2.4.2., could very well be the solution to the organization of these agents. The subsumption architecture provides for distinct levels of functionality that will operate independently unless they are overridden by a higher level. This type of operation is optimal for a home automation system where individual devices should contain some base behavior and operate with that behavior until some higher level dictates otherwise.

In this section, we will discuss a potential design for a multi-agent home automation system that uses the subsumption architecture as its foundation.

3.1. The Tree We Call Home

Everything in an environment has a "state". A television can be on or off. A lamp can be at 50% power or possibly it is off. A room can be in a state that represents the occurrence of an intrusion, or the bedroom can be in a state denoting sleep. Devices have states, rooms have states, floors have states, and even a campus of many buildings can have a state. *Everything in an environment has a state.*

Often it is desired that an entire room, or even an entire building, would go into a specific state at the push of a button or at a specific time. This type of control naturally leads to a tree-like view of an environment. Figure 3.1. shows an example of a hierarchical view of a home.

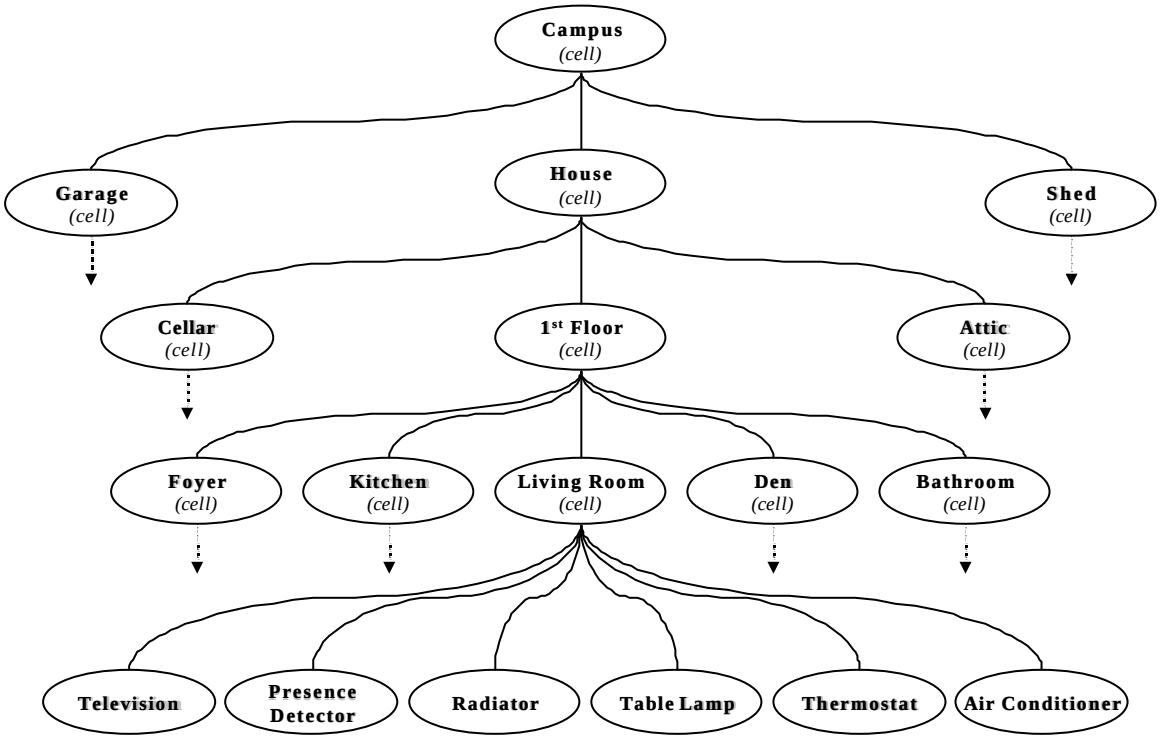

Fig 3.1: Hierarchical Organization

It is convenient, then, to think of the entire home as a large hierarchy, or tree, of buildings, rooms, devices, etc. The term **node** will be used to describe *any* object in this tree.

It can be seen that there are many contexts of control within our hierarchy. Controlling temperature can affect a room, a floor, or possibly an entire building. A type of node that groups devices within a scope of control is called a **cell**. A cell is an abstract container and can contain devices as well as other cells. It is also possible that a cell *only* contains other cells. For instance, when controlling temperature for a floor of a building, it is conceivable that the cell "1st Floor" contains the cells "Foyer", "Kitchen", "Living Room", etc. as its only immediate children, and that these child cells will be affected whenever the cell "1st Floor" is affected.

Another type of node in the hierarchy is a **device**. The leaf nodes of the hierarchy must always be devices. Consequently, devices are always contained in a cell. There can be three distinct types of devices:

- **sensors** measure one or more cell **attributes** (e.g., lux, temperature, motion, etc.)
- **effectors** affect the environment in some unique fashion, ultimately altering a state (e.g., lamps, radiators, and blinds)
- **punishers**, a form of sensor which, when activated, indicate user dissatisfaction with a specific cell state (e.g., light switches, thermostats, etc.).

Figure 3.2. shows the different types of devices.

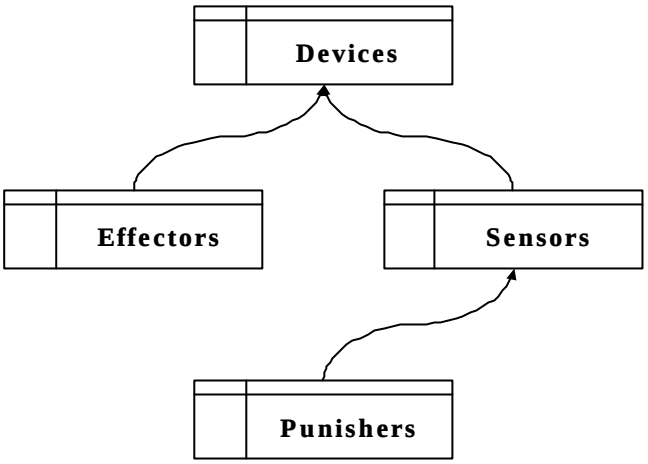

Fig 3.2: Device Types

While sensors and effectors are fairly simple to understand, the concept of a punisher is slightly more difficult. Punishers can be seen as sensors that report the user's desire. Because we want to provide control at any level of perspective in the hierarchy, even over cells, at least one punisher is required for every node in the tree whose state is to be altered by the user. Punishers can affect cells or devices.

Employing a hierarchical view of an environment provides two major advantages when attempting to control the home. First, control can be achieved on any node of the tree with a guarantee that all child nodes will change accordingly. For instance, a node representing a room can be forced into a specific state and all child nodes' states will subsequently be changed, if a state change is necessary. Secondly, generalization of state information can be quickly ascertained and delivered to parent nodes. This allows for a continuously generalized view of states at each higher level of the tree. Knowing that one attribute of a home is that it is 'secure' is sufficient information without needing to the know the specific states of the child nodes.

Figure 3.3 shows an example hierarchy with the current state of each node indicated by brackets.

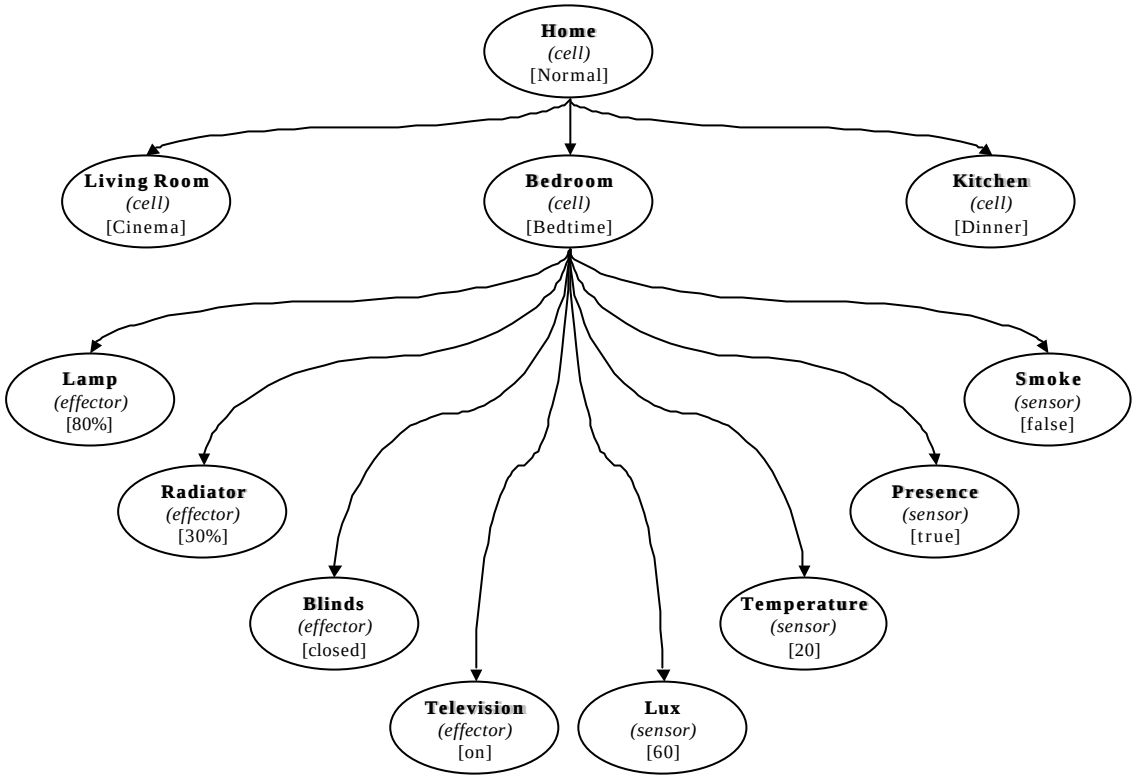

Fig 3.3: An Example Hierarchy

Now recall from the discussion of the subsumption architecture that subsumption is concerned with levels of competence within a system. Basic levels of competence must be implemented before higher levels of competence have any meaning. For instance, before one can effectively regulate the temperature in the room, the base competency to turn the radiator on and off is required. Every layer of the architecture has complete access to all sensors and all effectors, even if the higher layers would probably not take advantage of them as much as the lower layers. In this way, every layer can directly interact with the environment, if needed, without going through lower levels [Pfeifer & Scheier, 1999].

Our hierarchical view of a home can also be compared loosely with the subsumption architecture. Every agent can be considered a layer of the system that uses the lower layers to accomplish its task. In addition, every agent has access to the states of all other agents through subscription, and can even affect the states of the agents below. In this way, we can see that

the hierarchical tree of agents is already on the way to being a fully functioning control system.

3.2. Universal Agent Attributes

We have briefly described how state information can be passed down to child nodes in order to control environments from a high level of perspective. However, a scheme for *generalizing* specific information about child nodes must also be developed.

We will now discuss how we can categorize the specifics of a device's functionality. We will also investigate how we might be able to use state generalizations for simplifying control as opposed to just observing the states of specific devices. This section describes a framework for providing such categorization and generalization.

3.2.1. Static Device Binding

One of the largest problems in home automation systems today is the **static binding** of devices to one another. A wall switch always controls the same light, or a lux sensor always alerts the same blind that it needs to open. These static device bindings often hinder the ability of a home automation system to solve some fundamental problems.

To begin discussing these problems, we must first analyze what a user actually desires when he punishes the system. For example, when a user turns on a light, what are his actual intentions? Does he want a specific lamp to turn on or does he want to raise the overall lux in a cell? Often he may just want the lux in the room to increase regardless of what effectors are used to achieve the goal. We must keep in mind, however, that he may also want to turn on a specific effector.

The first problem is that of using the cheapest effector to alter a cell state. The system should choose the cheapest method for altering a cell state while still keeping the user satisfied. The second problem is that static device binding creates the need for explicit configuration of new devices as they are added to the system. Allowing plug-and-play functionality for all devices is optimal as it reduces the amount of time for installation and configuration. Plug-and-play functionality should also allow devices to be added to the system without any specific knowledge of what the device actually *does*. For example, we may want to add a sensor, effector, and punisher for a new UAA "foo". We need only to know how the "foo" effector changes the sensor readings of "foo" and then optimize to a set-point or to specific punisher requests. This functionality has been greatly simplified here, and will not be explained in full in this paper. It is one of the design concepts that has yet to be fully researched.

Returning to the lux example above, it may be easier to talk about sensors that measure lux, effectors that affect lux, and punishers that assert a lack of user satisfaction with lux. By speaking about lux in an abstract fashion, we can begin to address some of the fundamental problems of static device binding.

One solution for solving the static binding problems is to implement Universal Agent Attributes (UAA's). A **Universal Agent Attribute** is an abstract attribute for classifying device functionality as well as states. A device has the option to *subscribe* to one or more UAA's depending upon its function. For example light switches are punishers of the UAA "lux" and lamps are effectors of the UAA "lux". Blinds are effectors subscribed to both "temperature" and "lux", as their state can have an affect on either UAA, depending on the outside lux. Figure 3.4 illustrates this UAA subscription.

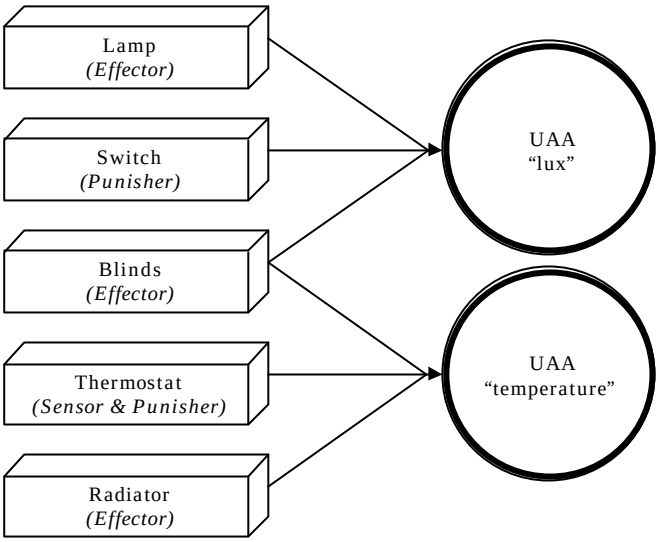

Fig 3.4: Device UAA Subscription

One can now imagine punishers that punish UAA's as opposed to punishing a specific device. Often a user is merely unhappy with the temperature in a cell, so giving him the ability to punish the UAA "temperature" for the cell, as opposed to a specific temperature effector, allows the system to choose the optimal method for altering the UAA.

It should be noted that current heating systems operate in a manner similar to this. A punisher, the thermostat, is provided to alter the UAA "temperature". However, in almost every home, the heating system does not communicate with any other systems, such as windows, blinds, etc. Heating is still performed using only one effector.

It is clear, however, that static device bindings must also exist in the architecture to account for a user actually desiring a specific device to be employed.

The following is a list of potential UAA's for the system:

- presence
- lux
- volume
- smoke
- time
- temperature
- power usage
- intrusion
- CO_2

3.2.2. UAA Composition

Often, valuable information used to describe a cell can be generated by examining and combining archived UAA information. For instance, one method for determining presence in a cell, is to look at the current motion in the room as well as the motion for the last few minutes. Or in a cell with multiple motion detection sensors, a composed "motion" UAA could be created by the logical 'or' of all device states subscribed to the UAA "motion". For the UAA "lux", this composition may be more complex. For example, it might be calculated by taking the mean of all "lux" values in the cell. Figure 3.5 and Fig 3.6 illustrate this composition.

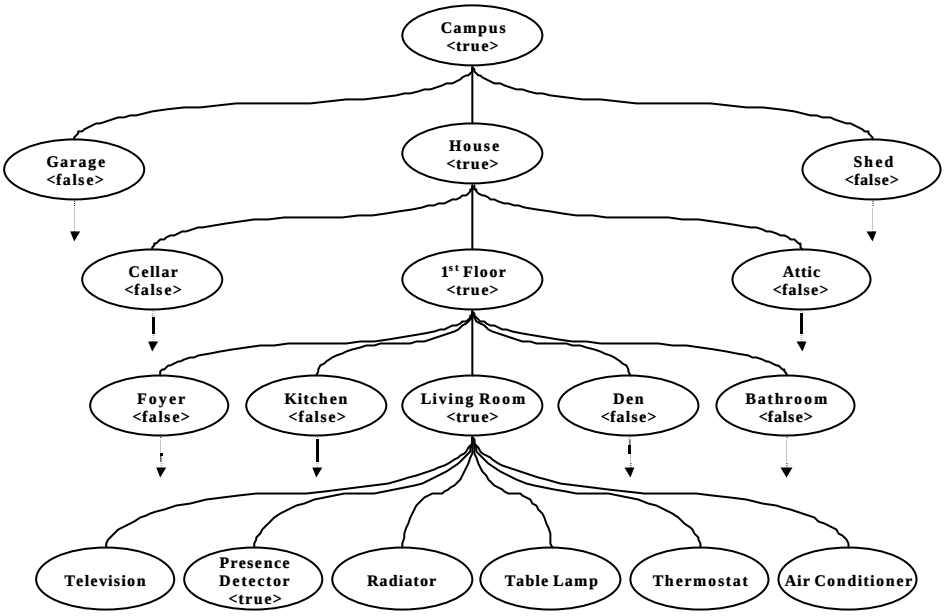

Fig 3.5: Composition of UAA "presence"

As one can see, UAA sensor information can also be generalized just as states are. In Fig 3.5 we are able to quickly ascertain the presence of the entire campus by merely referencing one attribute of a node. Presence is very easily composed, merely by using the logical 'or' of the

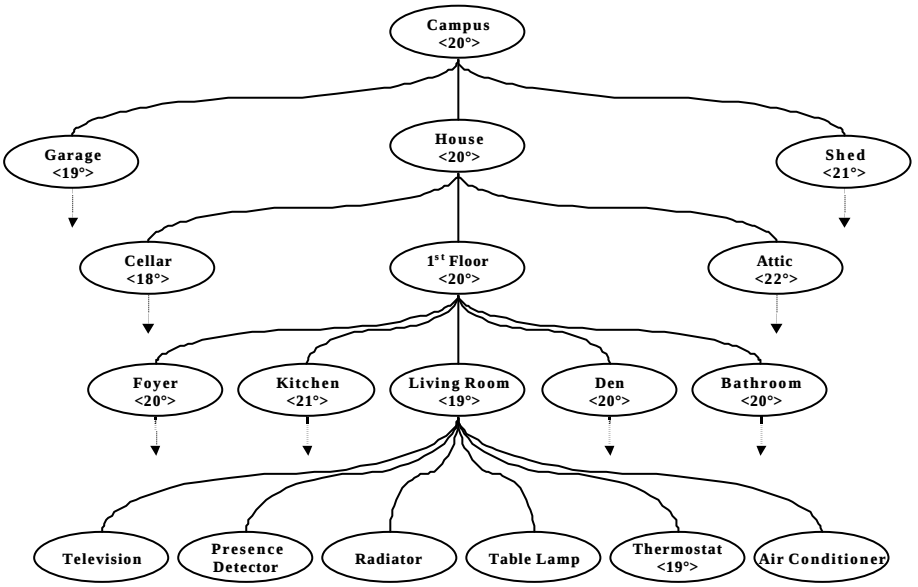

Fig 3.6: Composition of UAA "temperature"

UAA presence for all child nodes, moving upwards from the leaf nodes. In Fig 3.6, the UAA "temperature" is calculated by taking an average of all child nodes.

To implement such a scheme, very specific information must be delivered from the devices which want to participate in the UAA model. First, a **UAA enabled** device must declare what type of device it is (e.g., sensor or effector) as well as which UAA's it senses or affects. For example, a lamp must declare that it modifies the UAA "lux".

Secondly, the device must declare *how* it modifies it's respective UAA's. For instance, a UAA enabled agent must provide the ability to query exactly how a specific UAA will be affected if the agent were put into a certain state. This decision is naturally dependent upon the current state of the environment. This allows the system to compare the potential action of a UAA enabled device with the desired state for the UAA and decide how the device should be used, or whether it should be used at all to accomplish the desired control.

UAA composition provides the necessary functionality to complement the state changes forced by parent agents. While parent's can force child agents to change state, UAA's allow a non-intrusive method for communicating agent attributes upwards, providing an overview of certain aspects of an environment.

3.3. Additional Control Mechanisms

Now that we have defined a hierarchy of devices and cells in an environment, we must begin to discuss how control decisions will be made over this hierarchy.

We already know that the agent hierarchy provides some basic control through the use of a subsumption-like scheme. However, control mechanisms must also be provided directly to device manufacturers, service providers, system installers, and the user themselves.

Figure 3.7 shows a suggested set of control mechanisms and their priority within the system.

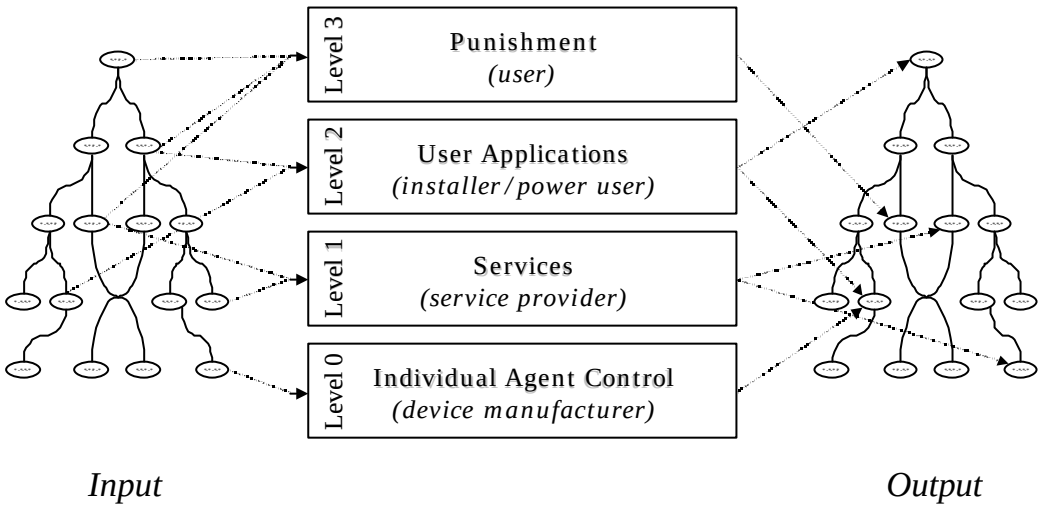

Fig 3.7: Control Mechanisms and their Priority

The four levels of control mechanisms were chosen to provide direct access to the system to each of the user groups listed in parentheses. Priority of the mechanism moves upwards from level 0 to level 3. The effect that a mechanism has on the agent hierarchy is more specialized and unusual as one moves up in the levels.

3.3.1. Individual Agent Control

Beginning with the most basic mechanism, individual agent control describes the default behavior for any of the nodes in the hierarchy.

Every node is outfitted with some base description of how it should behave. For instance, a radiator might turn on during the day and shut off for the evening as a default behavior. The behavior for a lamp may be to remain off at all times. *Every* node has some basic behavior, even if it is to do nothing all the time.

Definitions of individual agent control are provided by the manufacturer of the device and can be considered a form of device driver. The information provided by the manufacturer describes the default behavior for the device.

It is interesting to note that most home automation systems can be seen as only implementing this first level of competence. There are currently many systems and devices in a home that function completely independent of one another. In this way, after implementing this first layer, we can fairly say that we have implemented a basic home automation system (although in this case, not meeting our goals). The implementation of this mechanism will be discussed in more detail in Section 3.4.

3.3.2. Services

The next type of control mechanism, **services**, implements the core behaviors in a home. Examples of services would be HVAC (Heating, Ventilation, and Air Conditioning), security, lighting control, user comfort, and energy management. Services use multiple devices to achieve their goals. Because of this, services, and all higher levels of control, are not contained within the agent. For instance, optimally heating a house may involve using more than one specific effector. Much sensor information, as well as many different effectors may be used to achieve optimal heating.

Services are practically realized as software plug-ins to the system that describe how a certain set of devices, or cells, should act. They are able to observe the state of the agent hierarchy and make decisions that then effect the hierarchy. Services are written and delivered by service providers.

3.3.3. Applications

In home automation installations, users tend to expect very specific behavior from their systems, most often very tailored to their specific needs. **Applications**, our third level of control mechanism, are intended to implement custom user-defined behavior. Custom user applications can be anything from recording a specific television program once a week, to announcing over an intercom that the back door has been opened. Allowing users to schedule activity based on certain criteria will allow the creation of the "killer applications" that people expect from home automation systems.

Applications will also exist as software plug-ins. However, they will be visible to users, so that they may change application preferences, or add their own system functionality.

3.3.4. Punishment

At the highest level of our control mechanisms is **punishment**. Punishment of the system is the employment of light switches, thermostats, handheld devices, or the like to change the state of a cell or device. Punishment indicates that the current state is unacceptable and must be changed, regardless of the desire of all agents, services, and applications. In this way, user desire is maintained as the most important factor in the system.

Punishment is implemented by every punishment device subscribing to one node or node attribute. Punishment should then invoke an

immediate change in the state of the node. Punishment is not only important to implement immediate state changes, but punishment is also the aspect of the system that permits learning to occur. Punishment information will be delivered to all three sub-levels of our subsumption architecture. Services and applications may choose to do nothing with the punishment information. However, nodes will use the punishment signal to alter their behavior to match the patterns of the user.

An example of punishment would be the following: a user enters a dark room and the lights remain off. Because the lights did not automatically turn on, the user employs the wall switch (punisher) to indicate his dissatisfaction with the system's decision. The punisher informs the overhead lighting fixture agent that it's state should now be on. Because punishment is the highest level control mechanism, this new state will override all other lower level decisions for some established time period (unless the user decides to turn off the light again).

When the user punished the system, the node's intelligence recognizes that the required state and the existing state were different. This difference in states, along with the data about the situation in which this punishment occurred, is valuable training data that the node now uses to alter it's behavior. Depending on the learning scheme, after a certain number of iterations of similar punishment, the system should alter it's behavior to match the desire of the user.

Now, consider a situation where there are two services or two applications that wish to change the state of the same device. For example, the HVAC service requests that the blinds open and the lighting service requests that the blinds close. Which action is to be carried out? One action ultimately has to be performed. These types of overlaps do not actually occur often.

However, it is clear that a method for managing them is required. To solve this problem, each of the levels themselves, must also have a concrete order of preference. For instance, security is more important than lighting, lighting is more important than heating, etc. Alternatively, rules of combination might be employed, resulting in a blind that is half-open. However, this solution could be quite complex and should be a topic of future research.

3.4. The Agent

Now that we have seen an overview of the architecture, we come to the next level of system design. How will individual agent control be implemented?

To answer this question, we must begin by speaking of control at the smallest level. The smallest unit of functionality within the system architecture is an *agent*. Everything in the environment that can have a state (buildings, rooms, devices, etc.), is represented in the system by an agent. Every agent, except at the highest level of control, has a parent agent and may also have child agents. More easily described, every one of the nodes in the aforementioned hierarchy is represented by an agent. This means that the hierarchy is concrete and tangible, comprised of agents. Every device and every cell is represented by it's own agent.

In keeping with our basic AI design principles, it is important that we speak about *complete agents*. The agents in the system should be autonomous, self-sufficient, embodied, and situated. We want to assure that the agent will always perform it's task, even if it were to be cut off from the rest of the system (note that the system may or may not be physically distributed).

Complete agents naturally lead to agents that run asynchronously, loosely coupled to one another, only interacting with one another if the interaction could somehow improve the agent's task.

Knowing these facts about our agent, we can now begin to draw a picture about how it may look. Figure 3.8 illustrates our potential agent.

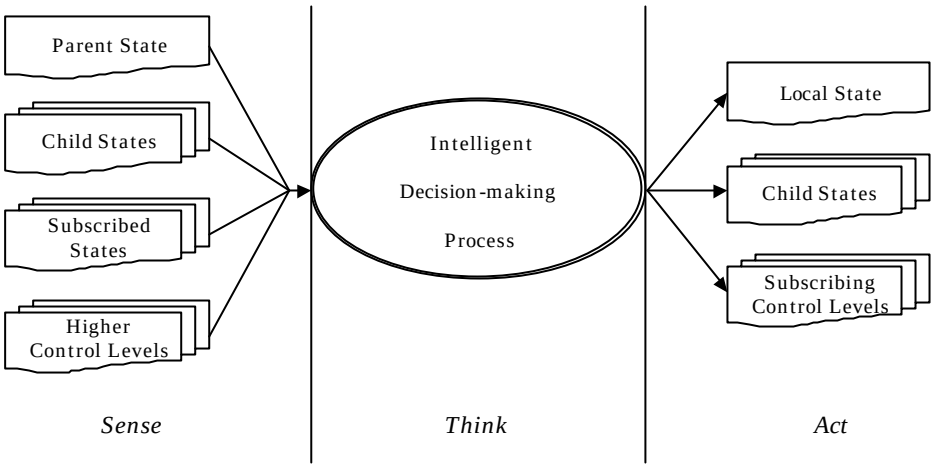

Fig 3.8: The Agent Proposed

An agent's decision-making process may be very simple and not require any external information in order to operate. However, an agent also has the ability to subscribe to the states of other agents whose states can have an effect on what the local state will be. For example, an agent that represents a lamp may subscribe to the presence detector agent's state, as well as the system time agent. When an agent subscribes to another agent's state, it is requesting that when that subscribed state changes, it will be notified. An agent automatically receives state change information from it's parent and child nodes, but an explicit subscription can be made to the state of any agent state within the entire tree. This subscription can be made during the original configuration of the system, or dynamically at runtime.

Our agent currently implements a basic "sense-think-act" cycle. It first receives state information from it's parent, children, and optionally subscribed states. It, then, makes some decision about what it's own state should be, and then alters it's own state and consequently the state of it's children.

Because agents also represent leaf nodes of the tree, i.e., sensors and effectors, we most also provide access to device drivers so that agents' state changes can ultimately translate into environment changes.

The decision-making process must also be examined. The "value principle" states that every agent should be equipped with mechanisms for self-supervised learning. Therefore, we want to guarantee that our agent is able to individually learn about it's behavior and adapt to operate optimally. However, we also need to provide a standard set of rules by which the agent can determine it's default functionality. The decision-making process, therefore, has been broken into two separate subsystems, the procedural subsystem for default functionality, and the intelligence subsystem for optimization of this functionality to the behavioral patterns of the user.

Figure 3.9 illustrates the proposed architecture of a single agent.

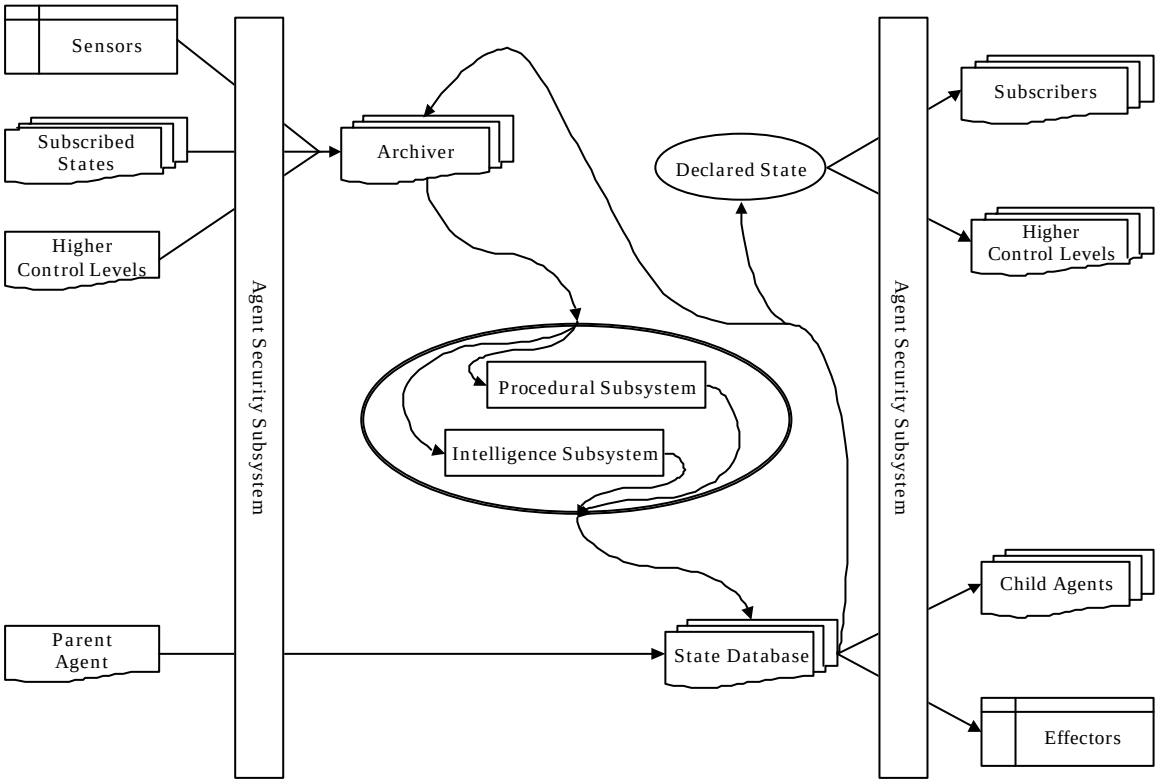

Fig 3.9: The Agent Complete

The **Procedural Subsystem** (PS) makes decisions regarding the current state of the agent by describing and implementing a default behavior. These decisions may or may not be dependent upon input states. Procedural control is necessary to implement both default behavior as well as decision-making during high risk situations.

High risk home automation applications, such as home security, cannot afford to be learned. For example, it would be unacceptable for the system to have to learn that presence in a secure cell is actually a burglary. Likewise, it is also unacceptable to have to learn that a detection of smoke in a cell actually means that there could be a fire and that an alarm should sound. Behaviors such as these should be procedural.

The PS is implemented by a custom scripting language providing logic operations performed over available subscribed state information to determine the desired output state. The language also provides for a neutral output state, meaning that the procedural subsystem has no state preference.

In many situations, the default behavior of the agent may not be sufficient in an environment. In fact, the actual behavior of an agent may need to be exactly the opposite of what the PS defines. The procedural subsystem can also be employed as a template of basic initial behavior for the Intelligence Subsystem upon startup.

The **Intelligence Subsystem** (IS) is the optional portion of the architecture that "intelligently" decides what the current state should be, if it has been activated. If the intelligence subsystem has been activated, the procedural subsystem no longer has control of the agent's state. It can be implemented using variations of the following methods, but is not limited to this list:

1. No IS. Procedural control only.
2. Decision Trees
3. Neural Networks

4. Hidden Markov Models
5. Fuzzy Inference System

All of the above listed implementations, and most learning algorithms in general, require *many* iterations of training before they approach an optimal solution. Because of this, the IS must be sufficiently trained to behave in a specific manner before it is allowed to control the state of the agent. To provide the required default behavior, the first task of the intelligence subsystem, before making any decisions, is to optimize itself to emulate the exact behavior described by the procedural subsystem. By providing the intelligence subsystem, we have enabled the individual agents to dynamically alter their behavior to match that of services, applications, and punishment.

Once the IS has undergone it's initial training, subsequent state changes requested by higher level control mechanisms (services, applications, and punishment) are employed as reinforcement training data. As the upper levels request state changes from the agent that are not predicted by the IS, the IS will recognize it's fault and alter it's behavior slightly. After many iterations of the same mistake being encountered, the combined effect of many small changes in the IS becomes an adaptation to account for behavioral patterns.

The **Archiver** is a system responsible for saving, compacting and correlating the state information of the agent for later reference. There is one archiver per agent. The archiver essentially provides a complete statistical profile of the agent's states, allowing the procedural and intelligent subsystems to perform their tasks. This profile information is made available to the agent itself, as well as to all agents that have subscribed to the local agent's state. The profile provides both customized statistical data as well information required to predict the agent state to allow for more efficient control from all levels of the our subsumption architecture. Information about a state is

saved if it was encountered for more than a time period longer than some threshold to avoid saving arbitrarily encountered states.

After the procedural or intelligent subsystem decides on the state of an agent, this state must be specialized into the corresponding states of all child agents. The **State Database** contains the corresponding child states for all previously encountered states of the agent. The details of both user-defined states as well as dynamic states that have been created by the intelligence subsystem are stored here. As the desired state is asserted by the PS or IS, the state database converts a state name back into the corresponding child states. In this way, the database provides the bridge between what a state represents locally to an agent and the states of child agents (and/or actual device settings). In this way, state changes will eventually trigger effectors to actually alter the environment.

For example, a control decision to set the "Living Room" agent to a state of "romance" will cause the state database to inform all child nodes of a state change. Child agents of the living room will then be forced to change their states to those saved by the state database of the "Living Room" agent. For example, the lamp agents may be required to change to a state of "30% dimmed" and the stereo may change to a state that softly plays a little Barry White. These child agents, representing physical devices, alter the physical environment by communicating with the devices via device drivers.

The **Security Subsystem** (SS) is responsible for protecting the agent from undesired controlling agents, services, applications, and punishers. Agent security is required to guarantee that users will not attempt to read agent states and activate effectors that are not within their jurisdiction. In an average home, this is most likely not a problem. However, consider a larger automation project, such as an office building. It may be very possible that in a certain lab that the lab operator would want to guarantee it's security. Nobody should be allowed to accidentally change his environment and maybe

even not be allowed to know what was happening in the lab (as sensors could practically give a comprehensive view of a cell).

The architecture for a single agent described above is the fundamental building block for creating an intelligent, automated home. It is an agent that can be scaled up or down to control any level of the home. There exists, for example, an instantiation of this agent for the entire home (i.e., the highest level of control), dictating the individual states of agents by informing child agents of a desired state. In addition, every room is being controlled by an instantiation of this agent architecture. And to go even further, every television, lamp, VCR, or other *device* is also being controlled by yet other agents to provide intelligent control.

3.4.1. A Note On Training Data

Recognizing the intelligence subsystem as the core learning mechanism of our agent, it is now important to detail how this learning may occur.

First, we must determine the form of the training data necessary for our machine learning. Selecting and formatting data can be very important to increasing the accuracy of a machine learning method. Because the agent learning method is not yet definite, we will speak now of training data in it's most basic form – tables of agent relevant information.

We know that every agent is informed of the state changes of it's parent, children, and the agent states and UAA's to which it has subscribed. It also receives directions from services, applications, and user punishment as to what it's state should be. With this information, the agent can begin to build tables of information, each instance of the

table representing a state change. Figure 3.10 illustrates a very abridged example of this table construction for a lamp in a kitchen.

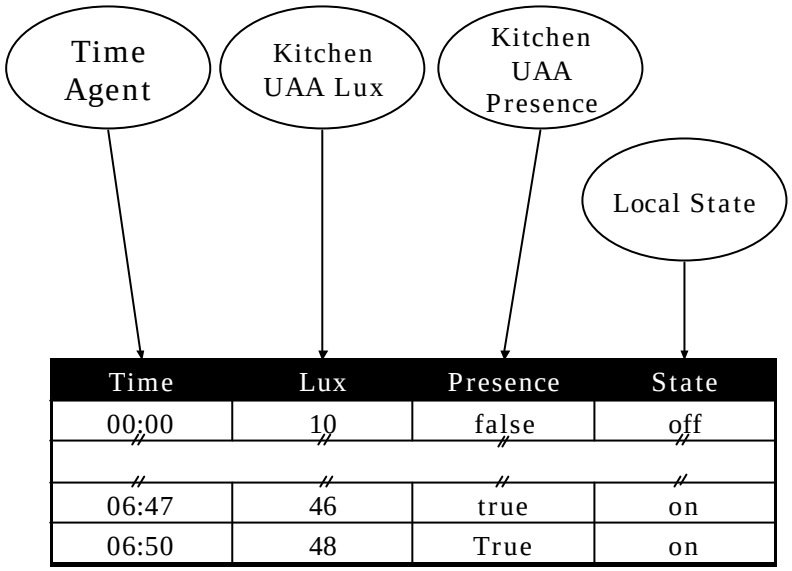

Fig 3.10: Table Construction for a Lamp

It is important to note that tables are constructed on demand as to avoid saving redundant data. Because the Archiver stores a record of the state for only one agent, it need not store the entire table for the agent. When various state data needs to be accessed by the IS to perform a training iteration, the appropriate agent's archiver is referenced.

These tables of data can now be used by the intelligence subsystem to find patterns of data and more accurately determine the most probable output state for the agent. Again, this process is specific to the selected machine learning method.

4. System Implementation

4.1. Introduction

Now that the complete system design has been presented, a proposed project implementation will be explained. Due to time constraints, the system presented here has not yet been implemented. However, it is planned that a version of this implementation will be used as a guideline for the first prototype to be installed.

As many of the devices needed to optimally achieve home automation goals currently do not exist, some of the hardware solutions are slightly less elegant than desired. Fortunately, as more devices become available, and home automation standards improve, the system design discussed in the previous sections will still provide a compatible architecture. In the future, however, the physical implementation should simply become easier to install.

4.2. Hardware Implementation

The computational portion of the system will be comprised of a desktop server running Windows 2000. Three Compaq iPaq PocketPC's will be employed as universal punishment devices. All four of the machines should be networked using an industry standard 802.11b wireless network.

All effectors are accessible using an X10 power line bus. For testing purposes, this bus is optimal as it is easy to install and there are Java libraries already available for quick coding. The sensors will be a modified version of the HTS Compact Office presence detector. They deliver motion, presence, *and* lux allowing for a compactly installed, adequate view of a cell in this prototype.

The server acts as a central hub of communication. It provides access to the X10 bus via a serial connection as well as access to the HTS sensors in all

cells, also via serial connections. Figure 4.1 shows a simplified version of the potential hardware configuration.

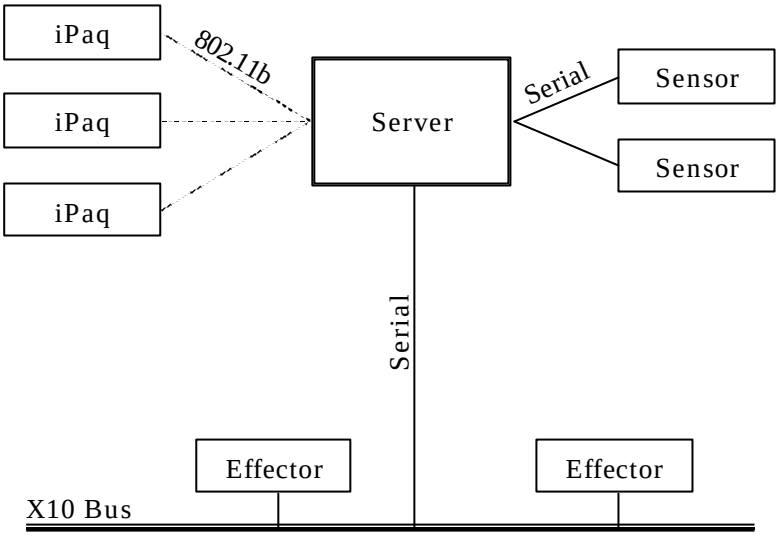

Fig 4.1: Hardware Configuration

Care must be taken when placing the sensors in the different cells as misplaced sensors can cause faulty data to be delivered. For instance, a sensor placed in the center of a room may register movement in the hallway as being presence in the room. These physical error would clearly have a negative effect on the functioning of the system.

4.3. Software Implementation

The entire software portion of the project is to be implemented in Java 1.3. As the hardware platform for the end product is not yet defined, implementing the skeletal version of the software in a platform independent environment is desired. The object-oriented nature of Java also greatly simplifies the implementation of agents.

Using a client-server model, the system can be built in two distinct portions. The “server” portion is responsible for all sensor and effector communication

as well as housing all agents, services, and applications. The "client" portion is responsible for handling all incoming punisher information from the iPaq handheld PC's via Java Servlets. The client portion will be developed using the Apache Web Server and Java Servlets.

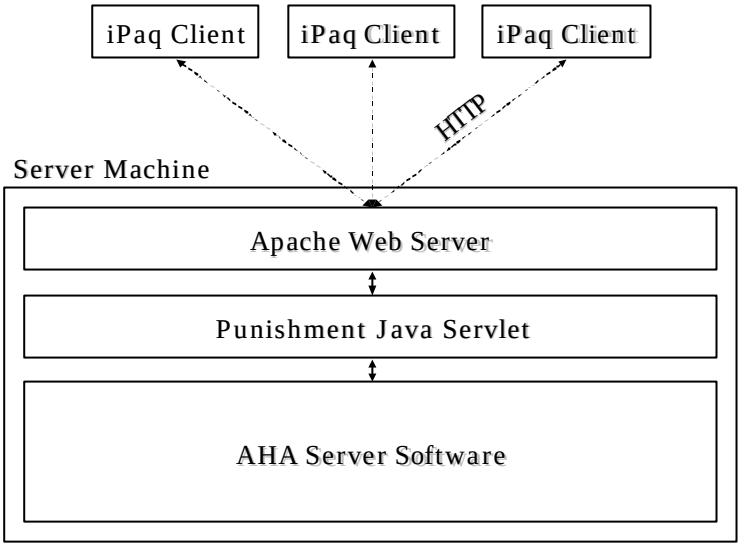

Fig 4.2: Client-Server Model

In this scheme, the iPaq handheld PC's will be used as the only punishers for the system.

5. Design Evaluation

5.1. Introduction

In attempting to evaluate the system design, answers to the following questions will be discussed:

1. Does the system comply with the original design goals?
2. What are the anticipated hardware requirements? Are they realistic?
3. Has the system been designed in such a way that it can support learning processes?
4. How well would the system scale up?
5. Using a defined set of natural use cases, how would the system perform?

The evaluation presented in this section is the evaluation of an architecture for a potential system. The arguments presented are theoretical and are based only upon anticipated system behavior. Until an implementation has been completed, and unforeseen design issues have been exposed and solved, an accurate evaluation is difficult to perform.

5.2. Project Goals

With regard to the original project goals, we have designed a system architecture that will most probably:

1. Eliminate the need for static device binding for control by providing Universal Agent Attributes. However, by not participating in the UAA model, agents still employ static device binding for use in high-risk situations.
2. Learn and adapt to an inhabitant's behavioral patterns on every level of perspective through use of the intelligent subsystem. Behavioral patterns can be learned for any node in the agent hierarchy.

3. Unite all devices and functionality into one control architecture using the individual agents, services, applications, and user punishment.
4. Allow devices to function intelligently, even if they are physically separated from the system (assuming physical distribution). The intelligent subsystem in every agent is designed to dynamically optimize its state output to match that of services, applications, and user punishments.
5. Provide procedural device control for high-risk applications through the use of the agent's procedural subsystem, services, and applications. Applications and services put procedural control into the hands of users by providing a programming interface to the system.
6. Allow for plug-and-play operation of newly added devices to the system, assuming they subscribe to the UAA model.

It is then important to note that we did not yet provide a method for maintaining security or minimizing wasted energy. The system, as it currently stands, is designed to optimize itself only to the user behavioral pattern. This could be remedied by assuring that the intelligence subsystem was biased towards providing security and energy management. See Appendix A for more information regarding optimization of decision factors.

With respect to the original project goals, it is clear that the research seems to have definitely brought adaptive home automation closer to solving some of its fundamental problems.

5.3. Projected Hardware Requirements

Clearly, without an available implementation of the system, both hardware and software, defining hardware requirements can be difficult. However, knowing our target environment, we should be able to make some educated guesses.

We know that the installed system must have a life expectancy of at least ten to fifteen years. A user does not want to have to replace an entire automation system on a regular basis. We also know that for a system installed for such a long period of time, a high tolerance to upgrades as new technology becomes available is required.

These two requirements naturally lead to the vision of an embedded PC functioning as the server at the heart of the system. As the hard drive is the most error-prone component of a system, a flash memory card is used to help extend the life of the system. Using a PC as the system base also provides the advantage of established upgrade routes. An embedded system outfitted with USB and Firewire, for instance, should provide sufficient, user-friendly, upgrade paths to our system.

Obviously, processor, memory, and data storage requirements are dependent upon the size of the installation. Making some assumption about our environment and our agents, we can begin to estimate the actual requirement of our system.

- Assumption 1: The size of the Java object in memory is most likely small enough to be considered negligible (on the order of 50 kilobytes).
- Assumption 2: The Archiver saves only a timestamp and the state of the agent no more than every ten minutes. The timestamp and state together take no more than 30 bytes, making the required space for one year of data no more than approximately 1.5 megabytes.

A home with 6 automated cells and an average of 10 automated devices within each cell would then yield a requirement of 6 * 10 * 1.5, or 90 megabytes of memory for one full year of data.

Processor requirements are slightly more difficult to predict than memory. If an agent is located physically within a device, then the processor

requirements on the device will probably not be very high at all. The device only needs to support its local behavior and communication. However, if all the agents for a home are located on the same system, the processor requirement may very well approach that of an x86-based machine. The server must support communication, a Java virtual machine and all the processing of agents, services, applications, and punishment.

5.4. Learning

In the proposed architecture, learning, and therefore adaptation, can occur at many different levels. At the most basic level, we have enabled our agent to learn about it's own local state and adapt accordingly using the intelligence subsystem. The behavior provided by the intelligence subsystem optimizes to that of the services, applications, and user punishment, independent of what level of the subsumption architecture actually requested the behavior.

In a physically distributed system, this agent level of learning is crucial in providing intelligent control during times of a system crash. Because the agent exists and is being executed within the device itself, it can emulate connected behavior even when it has no access to information from other agents, or to services, applications, and punishment from the server.

Learning can also occur within services and applications. However, the responsibility to implement this learning lies with the service provider or application author. It would be impossible to predict what type of learning would be best for custom services and applications. Ultimately the learning method to be chosen for services and applications *needs* to lie outside of the realm of the definition of the architecture to allow for different functionality.

Overall, the learning methods provided in the system are sufficient to implement a first attempt at intelligence in the home.

5.5. Scaling

The distributed nature of the architecture greatly lessens the effort of scaling the system up. As we have mentioned earlier, the requirements of the system hardware are dependent on the size of the installation. In this way, as a system's load becomes too great, it should be possible to simply add another server system on to distribute the load. Instantiations of the agents could merely be separated onto many machines.

To accomplish this, more research would have to be done in distributed computing and how this load could be distributed. However, the multi-agent architecture should make the move to a distributed system quite easily.

5.6. Areas of Concern

Again, without an actual implementation of the system, it would take a great amount of confidence, or perhaps overconfidence, to believe that the system would function as we hope. It would be wonderful if this paper described the answer to adaptive home automation, however, there are a few areas of uncertainty in the system design.

One concern is knowing the approximate amount of time it will take for the system to learn the behavioral pattern of the user. Depending on the intelligence subsystem implementation, will the number of required learning iterations be so large that the entire concept of learning in this context becomes worthless?

Another point of uncertainty is how the UAA model will actually affect plug-and-play functionality of devices. What impact will adding new UAA enabled devices have to the existing device functionality? How exactly can the system decide if a UAA device should be used or not? Does the device have certain preferences as to how it is used?

It is also still unclear how the intelligence subsystem might learn behavioral patterns from the statistical information provided by the archiver. Data mining techniques as well as traditional learning systems are possibilities, but the actual solution is not yet clear.

The ultimate complexity of the system is also still unclear. With all of the components of the architecture implemented, how easy will it be to debug, modify, and upgrade such a system?

6. Further Research and Conclusion

6.1. Further Research

The research that has been presented in this MQP regarding adaptive home automation is clearly not comprehensive and could be supplemented by various other topics.

One topic that was not addressed is that of streaming media. All of the devices that are addressed in the current architecture only have one state at any time. It would be interesting to see how streaming media content could be freely transmitted about an environment independent of I/O hardware. For instance, one could have a telephone conversation over the living room stereo speakers and the closest microphone, or watch the same security camera on the PC and on the television, even though they both use different display technologies. For audio/video devices, a possible solution to this problem would be to require all device drivers to produce and receive media content in a standard format, e.g., AVI for video and AU for audio. Naturally, research would also have to be done to determine bandwidth requirements for such functionality.

Another possible area of research involves investigating how the archiver could best store state information. Is pure statistical data sufficient for adequately intelligent device control? Instead of saving statistical data, it is possible that saving successful sensor-effector pairings may provide a more accurate model of the environment. Statistical data tends to wash out important aspects of an environment model that sensor-actor pairing may just retain.

Intention recognition, or predicting what the user actually desired when he or she interacts with the room could also be a valid branch of research. Universal Agent Attributes can concretely categorize certain abstract

characteristics of an environment and allow the user to manipulate them, but how can the system recognizes what the user actually *wants*?

Yet another related area of research is associated with ubiquitous computing and punishment. Earlier we mentioned that voice recognition is a growing technology that has the potential of heavily influencing the home automation arena. How could such technology be implemented into the architecture presented here? Because voice recognition currently operates more efficiently with a constrained vocabulary, could the states of agents actually provide context for more efficient voice recognition. Would such information aid in integrating voice recognition into the system right now?

In regards to services, applications, and punishment, how could these concepts be also integrated into the architecture as agents as opposed to being a separate type of entity? Should they even be an agent? What will be gained by having one type of agent for every component of the architecture?

Lastly, what kind of combinatory rules could be employed to avoid conflicts between different services and applications? Consider the situation of a service requesting that a lamp be turned on and another service simultaneously requesting that the same lamp be turned off. How could this apparent contradiction be handled in a way that delivers an acceptable output state from the lamp? Would such a combinatory rule even be feasible?

6.2. Conclusion

In this report, we began by presenting some of the problems associated with current home automation systems. Through the use of multiple artificial intelligence concepts, we have attempted to address some of these problems. A hierarchical view of the home, universal agent attributes, and adaptive learning techniques all contribute to the proposed a software architecture that begins to solve these problems.

However, the architecture suggested here is clearly not yet "ready for primetime". There are still many unsolved issues that need to be addressed before future implementation can occur. It is our hope that this research will serve to encourage a new assessment of home automation techniques, moving from a strict engineering approach, to a more abstract, dynamic, artificial intelligence approach.

6.3. The MQP Experience

This MQP has definitely had its challenges as well as rewards. The overall lack of existing background research in adaptive home automation demanded the development of a significant number of new ways to solve some critical problems that have not yet been addressed. The chance to be creative has definitely been a pleasure.

As the MQP was completely researched and written in Switzerland, it's international nature has also made it interesting, to say the least. Being 6000 kilometers away and attempting to do an "on-campus" MQP is quite a challenge. However, thanks to the information age and a patient advisor, turn-over on corrections went smoothly and the challenge was overcome.

In regards to the content of the research, the MQP experience has truly been rewarding. At the beginning of the project, my knowledge of artificial intelligence as well as home automation was basically nil. After what seems likes thousands of pages of reading and a whole lot of advice from various interested parties, my theoretical understanding of both subjects has improved immensely. On a more practical scale, I really enjoyed merging the theoretical research being done in universities around the world with the real-world aspects of home automation. The idea of bringing "cool" topics out of the lab and into the home was a big motivation for me during the entire project.

The MQP has forced me, for the first time in my life, to organize massive amounts of information on paper. Writing was definitely something that I feared before this project, and while my writing is clearly still not perfect, I feel that it has improved immensely. Already working for two years in the field has very quickly shown me that ability to communicate is one of the primary skills, if not the primary skill, required to actually succeed in my career.

Appendix A: Decision Factor Optimization

When considering adaptive control over a defined control set, many factors become involved in the optimization of system performance. Each factor is to be optimized for a specific behavior, often with the optimization goals conflicting directly with those of other factors. If one were to optimize control decisions upon only one factor, it is clear that the others might possibly suffer. For example, in building automation, three standard (and clearly conflicting) factors to be optimized are total energy expenditure, security and occupant comfort.

To solve the problem of conflicting factor optimization, it is proposed that a standard comparison framework would be introduced. By converting the *effect* of all decisions based on optimization factors for a system to a common "currency", the ability, then, to compare all available decision sequences is greatly simplified. A **decision sequence** represents a theoretical sequence of control decisions that a system could make. An example of a decision sequence is the sequence of states for a light into the future in *x* minute intervals (e.g., 0100000111).

The general form of the cost calculation can be found is Eq. B.1.

$$J_u = \sum_{t=t_0+1}^{t_0+k} \sum_{j=1}^{n} \text{cost}(F_j, u, t), \qquad \text{[Eq B.1]}$$

where J_u is the expected total cost for the decision sequence u. For every decision in the sequence from t_0+1 to $t_0+\kappa$, the cost is calculated for all optimization factors $F_1 \ldots F_n$, within the context of that sequence.

The total number of sequences is equal to s^{κ}, where s is the number of different states in the decision space. Each decision sequence consists of κ decisions separated by a time interval σ, giving a total iteration time of $\kappa\sigma$.

For every optimization factor, a method for calculating the cost of a decision is necessary. Every factor is dependent upon certain variables to calculate it's cost. For example, in the context of a room lighting application, the "energy conservation optimization factor" is dependent upon the energy usage of the lighting elements to be engaged, and the "occupant misery optimization factor" is dependent upon occupant presence, lux, and time of day within the control space. The cost function only calculates over the single time interval of length σ.

Most possible decision sequences tend to be unlikely. The number of calculations required to iterate over all decisions s^{κ} can be reduced drastically by limiting the number of allowed state changes within a sequence and only iterating over these sequences.

The following process is proposed for decision cost optimization in conjunction with this framework:

1. Choose appropriate optimization factors (e.g., energy usage, misery, security, etc.)
2. Select:
    a. σ – decision time interval
    b. κ – number of decisions in a sequence
    c. c – the number of allowed state changes in a decision sequence
3. At the end of every σ, calculate J_u for every valid decision sequence.
4. Execute the first decision of the lowest cost decision sequence.

Execution of this algorithm shows that it's calculation becomes very processor intense, very quickly. For example, the number of decision sequences for a light with two possible states (on or off) predicted three time intervals into the future requires $3 * 2^3 = 24$ separate cost calculations. Take now a light with 16 different dimmable states (0-15). The number of separate cost

calculations for the same prediction interval is $3 * 16^3 = 12288$, an unfortunate and unwieldy number.

Annotated Bibliography

Beale, R., Jackson, T. *Neural Computing: An Introduction*. London: IOP Publishing Ltd. 1990.

A short but informative crash course in neural network theory. It provided adequate pseudo-code to begin implementing basic neural networks.

Brooks, Rodney A., *A Robust Layered Control System for a Mobile Robot*. IEEE Journal of Robotics and Automation. RA-2, pp. 14-23. 1986.

The de-facto standard when researching the subsumption architecture. Written by the man who first conceptualized subsumption oriented control, it is a great starting point for a first attempt at subsumption.

Coen, Michael H. *Building Brains for Rooms*. MIT AI Lab, Cambridge, MA. 1997.

Describes Michael Coen's Scatterbrain system in the Intelligent Room. The paper explains the highly distributed nature of the architecture, it's relation to subsumption, agents, agent interaction, and multimodal resolution. The content is very multi-media oriented, often using an office presentation situation as it's base example.

_____. *Design Principles for Intelligent Environments*. MIT AI Lab, Cambridge, MA. 1998.

Also using the Intelligent Room as a springboard for conversation, the paper describes how intelligent environments should be designed. It focuses on that fact that intelligent environments should not be equated with ubiquitous computing – UC suggests the "computer everywhere" approach, intelligent environments should lean toward a less intrusive attempt at control.

_____. *The Future Of Human-Computer Interaction or How I learned to stop worrying and love My Intelligent Room*. IEEE Intelligent Systems. March/April 1999.

A visionary essay spurring excitement about the potential for intelligent environments. Also gives a brief non-technical overview of the predecessor to HAL, the Intelligent Room.

_____, B. Phillips, N. Warshawsky, L. Weisman, S. Peters, P. Finin. *The Metaglue System*. MIT AI Lab, Cambridge, MA. 1999.

Describes a highly distributed, multi-agent system for controlling massive numbers of agents in an intelligent environment. This paper was somewhat of a springboard for the concept of Universal Agent Attributes as it speaks heavily on categorizing agent functionality. It is very multimedia oriented, and is the same system that was used for MIT's HAL project.

Dudek, Gregory, Jenkin, M. *Computational Principles of Mobile Robotics*. Cambridge, UK: Cambridge University Press, 2000.

This resource began to change my perspective on intelligent homes in general, convincing me that a home could be viewed as a merely a robot, turned inside out. It is very robotics oriented, but really helped to shed light on implementing intelligence in practical ways.

IBM, The Official Deep Blue Website, *www.research.ibm.com/deepblue/*, 2001.

The official resource for IBM's computer chess champion. It's an interesting example of a modern "intelligent" computer.

Krauss, Jens. *Lernfähiger Heizregler*. Gebäudetechnik, Oktober 2000.

A practical example of neural networks being used in HVAC systems. The neural network technology is actually not very new, but the fact that "intelligence" is starting to be used in the home helps to validate the research presented in our project.

Mitchell, Tom M. *Machine Learning*. New York: McGraw-Hill, 1997.

A standard resource for machine learning. It covers all of the core topics in ML. Definitely required background reading.

Mozer, Michael C., R. Dodier, D. Lukianow, J. Ries. *A Comparison of Neural Net and Conventional Techniques for Lighting Control*. 1994.

Describes one of Mozer's intelligent lighting control attempts, comparing conventional control systems and one using a neural network. The results for neural networks are appealing and definitely helped to encourage the author that this intelligence "stuff" just may be a plausible idea.

_____, L. Vidmar, R. H. Dodier. *The Neurothermostat*. 1997.

Mozer describes his implementation of an intelligent thermostat that optimizes to two conflicting decision factors, energy and user misery. Results seem to be robust and provided some of the basis for the discussion in Appendix A.

_____, D. Miller. *Parsing the Stream of Time: The Value of Event-Based Segmentation in a Complex Real-World Control Problem*. 1998.

An intriguing essay encouraging the reader to deviate from his natural perspective of time and view landmark events in an environment as the new time standard. The primary argument is that learning algorithms can actually be implmented more easily with this new perspective.

Pfeifer, Rolph, Scheier, C. *Understanding Intelligence*. Cambridge, MA: MIT Press. 1999.

A fairly biased artificial intelligence textbook written with a heavy cognitive science bent. Regardless of the bias, the author did a great job at blending older and newer artificial intelligence paradigms along with providing numerous practical examples. Pfeifer and Scheier's presentation of embodied cognitive science is convincing to say the least, as easily seen through the multiple references in this paper. Commendations on making an almost 700 page textbook a pleasure to read from cover to cover.

Russell, Stuart, Norvig, P. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice-Hall. 1995.

THE artificial intelligence resource. Heavy into the classical approach and it would be impossible to say that the famous "sense-think-act" cycle didn't somehow affect this project too!

Simon, H. A. *The Sciences of the Artificial,* 2nd Edition. Cambridge, MA: MIT Press. 1969.

Simon presented the ant example in Section 2.4.1. – great example for illustrating the frame-of-reference problem (as well as emergence).

Stork, David G. *Hal's Legacy: 2001's Computer as Dream and Reality.* Cambridge, MA: MIT Press. 1996.

An interesting read for anyone getting involved in intelligent environments. Partially inspired by Michael Coen's HAL project at MIT, this compilation of essays gives some potential insight as to the future of our homes and businesses, as well as how we may be interacting with them.