

**AUTOMATING INHABITANT INTERACTIONS IN HOME AND
WORKPLACE ENVIRONMENTS THROUGH DATA-DRIVEN
GENERATION OF HIERARCHICAL PARTIALLY-
OBSERVABLE MARKOV DECISION PROCESSES**

by

GREGORY MICHAEL YOUNGBLOOD

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2005

Copyright © by Gregory Michael Youngblood 2005

All Rights Reserved

To my wife Julie and our sons, Gregory and Austin.

Thank you for your love and support.

ACKNOWLEDGEMENTS

There is an African proverb that states, “It takes a village to raise a child.” I really believe this to be true, especially when it comes to education. We are a part of everyone who has touched our lives and imparted knowledge and wisdom. I have been more fortunate than most to have been taught by and worked with the very best people on this planet. I truly appreciate the lessons I have learned in life and know that I still have much to learn.

Nine years ago I met Dr. Larry Holder at a student ACM meeting at UTA. Through a series of events I eventually came to work for Dr. Holder as a system administrator as an undergraduate. He advised my Honors Thesis on robotic map-building and exploration, and my Masters Thesis on the D’Artagnan Cognitive Architecture, as well as this work. I appreciate his guidance over nearly a decade and the hours upon hours of conversations and discussions we have had over the years. It is hard to believe so much time has passed, it does not seem like it. Words can not express how much I have enjoyed working with Dr. Holder, and I hope we can continue working together in the future. Thank you!

For nearly as long as I known Dr. Holder, I have worked for Dr. Diane Cook. She sponsored my position as the Learning and Planning Lab, as well as the Robotics Lab, system administrator as an undergraduate. She kindly let me put her robots to work for my Honors Thesis. After some time in industry in 2002, she (along with Dr. Holder) invited me back to UTA as a Research Faculty Associate to manage the AI Lab and the MavHome Project. It has been a tremendous experience working on intelligent environment work, and I have learned a great deal about many, many things through the course of the project. I really appreciate Dr. Cook’s guidance and opinion on research—she really helps bring out the best in people and their work. Thank you for the opportunity and years of inspiration and guidance.

When I look back at all of the influential conversations I have had with people at UTA the most prominent conversations have always included Dr. Manfred Huber. I remember all of the great talks we have had at several FLAIRS conferences, the times we shared with REU students, and the late night conversations in the labs at UTA. Thank you for all of your advice, dedication, and commitment to excellence.

Another great source of inspiration is always to be found with Dr. Farhad Kamangar—now you have to go to the GACB to get it though. I have enjoyed many conversations with Dr. Kamangar who always seems to be able to put things in the right perspective. His well-timed advice and questions have been a big help over the years. Thank you.

Working with the IT Lab is always interesting and still somewhat of a mystery at times. They do a lot of interesting work and are led by Dr. Sharma Chakravarthy, who is always on the go. We have shared many ideas and a common vision over the years. I look forward to working more with Dr. Chakravarthy and his students in the years to come. Thank you for being part of UTA.

Almost since my first day at UTA in August 1996, Dr. Lynn Peterson has been an influence in my life. I have had the privilege of taking her programming class, which to this day I clearly remember—I still have the passion for programming. Dr. Peterson and I worked closely for a year when I was an undergraduate and served as the JCEO President. Ever since, we have kept in touch and talked about life, the universe, and everything (rest in peace, Douglas Adams). I have enjoyed our conversations over the years and look forward to many more. Thank you for being there and being who you are—always with an open door and warm heart.

In general, all of the faculty at UTA have been very influential in the last nine years of my life. I appreciate them all, but especially thank Dr. Behrooz Shirazi, Mr. David Levine, and Mr. Ray Springston for their guidance and advice over the years.

Chris Lance has been my right-hand man over the past year and a half—thank you Chris for all of your assistance and dedication. Darin Brezeale and Michael Brownlow have helped

maintain the AI Lab and keep me sane over the past few years—thank you. I have also appreciated all of the REU students who have worked hard for me over the years. I also hold a special place in my heart for our lab rats Kien Tran, Juan Lopez, and Denis Gjoni—thank you, and I wish you all the best in life.

Finally, but really foremost, I thank the Lord, our God, for the strength, intelligence, perseverance, and wisdom he has given me to perform this work. “For thine is the kingdom, and the power, and the glory, forever.” (Matthew 6:13)

July 19, 2005

ABSTRACT

AUTOMATING INHABITANT INTERACTIONS IN HOME AND WORKPLACE ENVIRONMENTS THROUGH DATA-DRIVEN GENERATION OF HIERARCHICAL PARTIALLY- OBSERVABLE MARKOV DECISION PROCESSES

Publication No. _____

Gregory Michael Youngblood, Ph.D.

The University of Texas at Arlington, 2005

Supervising Professors: Lawrence B. Holder and Diane J. Cook

Markov models provide a useful representation of system behavioral actions and state observations, but they do not scale well. Utilizing a hierarchy and abstraction through hierarchical hidden Markov models (HHMMs) improves scalability, but these structures are usually constructed manually using knowledge engineering techniques. We introduce a new method of automatically constructing HHMMs using the output of a sequential data-mining algorithm, Episode Discovery, and apply it to solving automation problems in the intelligent environment domain. Repetitive behavioral actions in sensor rich environments such as smart homes can be observed and categorized into periodic and frequent episodes through data-mining techniques utilizing the minimum description length principle. Utilizing this approach, we provide an architecture and a set of algorithms for a pervasive computing system showing that inhabitant interactions in home and workplace environments can be accurately automated through sensor observation and intelligent control using a data-driven approach to automatically generate hi-

erarchical inhabitant interaction models in the form of HPOMDPs and these models may be modified using temporal-difference reinforcement learning techniques to continually adapt to changes in the inhabitant's patterns until a new model should be generated. We present our life-long learning system and apply this work in our MavPad and MavLab environments where we have been successful at automating up to 40% of the life of a real inhabitant and 76% of a virtual inhabitant as well as dynamically adapting to concept changes over time. Findings from several case studies are provided to show the feasibility of this approach.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	iv
ABSTRACT	vii
LIST OF FIGURES	xiv
LIST OF TABLES	xviii
Chapter	
1. INTRODUCTION	1
2. INTELLIGENT ENVIRONMENT WORK	6
2.1 Introduction	6
2.2 Intelligent Environments	7
2.2.1 Framework Projects	7
2.2.2 Application and Gadget Projects	15
2.2.3 Industry Initiatives	19
2.2.4 Healthcare Initiatives	22
2.2.5 Learning and Adapting Initiatives	24
2.2.6 Other Initiatives	27
2.3 Pervasive and Ubiquitous Computing	31
2.4 Conclusions	32
3. A LEARNING ARCHITECTURE	33
3.1 The Problem	33
3.1.1 Goals	33
3.1.2 Perception	34
3.1.3 Reasoning	35

3.1.4	Actuation	40
3.2	Solving the Problem	40
3.2.1	The Approach	41
3.2.2	Data-mining	44
3.2.3	Belief State Support	45
3.2.4	A Rules Engine	46
3.2.5	Decision-making	46
3.3	The System Framework	48
3.3.1	Abstract View	48
3.3.2	Concrete View	50
3.3.3	Implementation	51
3.4	The System Architecture	55
3.4.1	ProPHeT	55
3.4.2	Episode Discovery (ED)	55
3.4.3	Active LeZi (ALZ)	57
3.4.4	Episode Membership (Epi-M)	58
3.4.5	ARBITER	58
3.4.6	The Core	59
3.5	The Environments under Examination	63
3.5.1	The MavPad	63
3.5.2	The MavLab	64
3.5.3	ResiSim	64
4.	DETAILED METHODOLOGY	67
4.1	The Data and Flow	72
4.1.1	The Character of Intelligent Environment Streaming Data	73
4.1.2	System Data Pathways	77

4.2	Overall Control	78
4.3	Training and Learning	79
4.3.1	ProPHeT Control	80
4.3.2	Episode Discovery Learning	85
4.3.3	Active LeZi Learning	94
4.3.4	Episode Membership Learning	96
4.4	Model Generation	101
4.4.1	Automated HHMM Construction	110
4.4.2	Markov Model Learning	118
4.5	Model Extension	120
4.5.1	The HPOMDP	120
4.5.2	Action Extension	120
4.5.3	Reward Structure	121
4.6	Events to Automation	123
4.6.1	ProPHeT Streaming Control Algorithm	124
4.6.2	Prediction	124
4.6.3	Episode Membership	124
4.6.4	Environment Automation	125
4.6.5	Rule Intersection and Feedback	134
4.6.6	User Feedback	135
4.7	Adaptation	136
4.7.1	Continued Learning with Temporal-Difference	137
4.7.2	Pruning	147
4.7.3	Policy Convergence	147
4.8	Concept Transformation	148
4.8.1	Drift and Shift Identification	148

4.8.2	Shift Reboot	150
4.9	Complexity	151
4.9.1	Initialization/Learning	151
4.9.2	Operational Runtime	153
4.10	Summary	154
5.	EXPERIMENTAL FINDINGS AND DISCUSSION	156
5.1	Experimentation	156
5.1.1	Human Studies	158
5.1.2	Simulation	159
5.2	Evaluation	159
5.3	Isolated Test Case Summary	160
5.4	Case Studies	162
5.4.1	MavPad Inhabitant 1	163
5.4.2	MavPad Inhabitant 2	164
5.4.3	MavPad Inhabitant 3	171
5.4.4	MavLab Mixed Inhabitant	176
5.4.5	Long-term Virtual Inhabitant	179
5.5	Component Dependence	185
5.5.1	Core Component Relationship Requirements	185
5.5.2	Belief State Component Analysis	187
5.6	Observations	191
6.	CONCLUSIONS AND FUTURE WORK	194
6.1	Conclusions	196
6.1.1	Strengths	196
6.1.2	Challenges	197
6.1.3	Final Analysis	198

6.2	Future Work	199
6.2.1	Resource Consumption Reduction	200
6.2.2	Multiple Inhabitants	201
6.2.3	Health Care	201
6.2.4	Synergistic Interaction	202
6.2.5	Other Domains	203
6.2.6	Intelligent Neighborhoods and Beyond	203
6.2.7	Enhancement Technologies	204
6.2.8	Unprocessed Data	204
6.2.9	Security Issues	204
Appendix		
A.	Understanding X-10	206
B.	The Argus Sensor Networks	219
C.	The MavPad Environment	261
D.	The MavLab Environment	280
E.	ResiSim: A Zero Configuration Distributed Simulation	301
F.	Licenses and Terms	319
	REFERENCES	329
	BIOGRAPHICAL STATEMENT	350

LIST OF FIGURES

Figure	Page
3.1 Basic agent architecture	34
3.2 Watching television Markov chain	37
3.3 Basic hierarchical Markov model	39
3.4 Phase 1: Knowledge discovery and initial learning phase	42
3.5 Phase 2: Operational phase	43
3.6 Phase 3: Adaptation and continued learning phase	44
3.7 Abstract framework	49
3.8 Concrete framework	52
3.9 Core system architecture in initialization phase	60
3.10 Core system architecture in operation phase	61
3.11 Core system architecture in adaptation phase	62
4.1 MavHome Steve Pattern 1: Lab entry to desk	68
4.2 MavHome Steve Pattern 2: Going on break	69
4.3 MavHome Steve Pattern 3: Going off break	69
4.4 MavHome Steve Pattern 4: Go to alternate workstation	70
4.5 MavHome Steve Pattern 5: Return from alternate workstation	70
4.6 MavHome Steve Pattern 6: Leave lab from desk	71
4.7 MavHome Steve HHMM structure—abstract nodes only	71
4.8 MavHome Steve HHMM structure pattern 2 abstract node expansion showing the production nodes and the automatable node in grey-fill	72
4.9 Data flow from sensors to online and offline objects	73
4.10 Kitchen motion over a sample day in the MavPad	75

4.11 Dining room light usage over a sample day in the MavPad	76
4.12 Couch usage over a sample day in the MavPad	76
4.13 Fan usage over a sample hour in the MavPad	76
4.14 Dataflow through architectural components	78
4.15 Episode Discovery data compression over time (MavHome Steve Patterns)	82
4.16 Episode Discovery interesting episode discovery over time (MavHome Steve Patterns)	82
4.17 The effects of different window sizes upon the compression ratio using Episode Discovery	83
4.18 The effects of different window time spans upon the compression ratio using Episode Discovery	83
4.19 Episode Discovery interesting episode discovery over time (MavPad Inhabitant 2 Patterns)	89
4.20 Episode Discovery interesting episode discovery over time (MavPad Inhabitant 3 Patterns)	89
4.21 Episode Discovery processing time (MavPad Inhabitant 3 Data)	90
4.22 Effect of noise in the data stream on episode discover in MavHome Steve patterns using 5% random noise interference in the data input stream	91
4.23 Effect of noise in the data stream on episode discover in MavHome Steve patterns using 10% random noise interference in the data input stream	92
4.24 Effect of noise in the data stream on episode discover in MavHome Steve patterns using 20% random noise interference in the data input stream	93
4.25 ALZ learning rate for MavHome Steve data	95
4.26 Example circular probability of two events occurring at different times of the day	98
4.27 The Evolution of “Reinforcement Learning with a Hierarchy of Abstract Mod- els” by Satinder Singh.	102
4.28 The Evolution of “Hierarchical Solution of Markov Decision Processes using Macro-actions” by Milos Hauskrecht, Nicholas Meuleau, Craig Boutilier, Leslie Pack Kaelbling, and Thomas Dean (see Figure 4.27 for nomenclature details) . .	106
4.29 Example HHMM	113

4.30	Episode Discovery reported episode segment	115
4.31	HHMM derived from ED data for MavHome Steve pattern 5 (single abstraction layer)	116
4.32	Episode Discovery reported abstract episode segment	116
4.33	HHMM derived from ED data for MavHome Steve pattern 5 (double abstraction layer)	117
4.34	HHMM derived from ED data for MavHome Steve (complete abstract model, no production nodes)	118
4.35	HHMM derived from inhabitant 3 data	118
4.36	MavHome Steve pattern 4 belief state determination HPOMDP	133
4.37	MavHome Steve pattern 4 focused HPOMDP with contention area	140
4.38	MavHome Steve c16_OFF ARBITER rule learning	145
4.39	MavHome Steve inhabitant feedback initiated learning of c11_ON	147
4.40	MavHome Steve pattern shift example	150
4.41	ALZ processing time (MavPad Inhabitant 3 Data)	152
4.42	Epi-M processing time (MavPad Inhabitant 3 Data)	153
5.1	ALZ training convergence for inhabitant 2 data	165
5.2	ALZ accuracy for inhabitant 2 trial	166
5.3	Interesting Episode Discovery over time	167
5.4	Learned HHMM from MavPad inhabitant 2 data	168
5.5	MavPad Inhabitant 2 interaction reduction (1 day trial)	168
5.6	MavPad Inhabitant 2 interaction reduction (10 day trial)	169
5.7	Inhabitant 3 interesting episode discovery over time	172
5.8	HHMM derived from inhabitant 3 data	172
5.9	MavPad inhabitant 3 interaction reduction	174
5.10	MavPad inhabitant 3 rule violation reduction	175
5.11	ALZ Accuracy for MavLab inhabitant	177

5.12 MavLab virtual inhabitant HHMM (production nodes omitted)	178
5.13 MavLab virtual inhabitant interaction reduction	178
5.14 MavLab virtual inhabitant automation performance	181
5.15 ALZ accuracy during long-term experimentation	182
5.16 MavLab virtual inhabitant HHMM after reboot (production nodes omitted) . . .	182
5.17 MavLab virtual inhabitant interaction reduction during long-term experimentation	183
5.18 MavLab virtual inhabitant rule violations during long-term experimentation . . .	184
5.19 Interaction reduction from varying prediction accuracy	190
5.20 Correct automations versus prediction accuracy	190

LIST OF TABLES

Table	Page
4.1 Data-stream from inhabitant interaction in a MavEnvironment	75
B.1 Argus Master Parts List	226
B.2 Argus Superslave Parts List	232
B.3 Argus Dongle Parts List	237
C.1 MavPad Sensors and Actuators	271
D.1 MavLab Sensors and Actuators	290

CHAPTER 1

INTRODUCTION

Most of the computers that participate in embodied virtuality will be invisible in fact as well as in metaphor. Already computers in light switches, thermostats, stereos and ovens help to activate the world. These machines and more will be interconnected in a ubiquitous network.

—Mark Weiser, *The Computer for the 21st Century*

Imagine a future of humans interacting with machines that are able to anticipate their needs and desires and fulfill them without command. Now imagine the pervasive computing future where all of these machines are ubiquitous and integrated into the environments of the world. This seems a distant reality from the current state of things, but research in understanding human environmental interactions and systems to anticipate and automate those actions can help move society in this direction.

Automation through anticipation, learning, decision-making, and adaptation falls into the category of intelligent systems research. It can be viewed as an agent problem [42], where an agent perceives the environment through a set of percepts (e.g., sensors), reasons about the information (e.g., decision-making), and acts upon the environment (e.g., automation). The goal of this agent would be to fulfill the desires and needs of a human or group of humans. The agent must anticipate actions to automate before the human performs that action by predicting the occurrence in advance. It must also be able to learn human patterns through observation and historical precedence. Decisions to automate actions need to be correct in order to avoid creating work for humans by causing them to correct wrong actions and perform their own

desired actions. As all things undoubtedly change, so will the humans that a system is seeking to automate; therefore, an agent must be able to adapt to the changing needs and desires of the humans it seeks to assist.

Americans spend the majority of their time in either their home or workplace environments [144] so automating humans in these environments would have an immediate impact on their lives. Creating these “intelligent environments” could have many positive implications such as assisting the handicapped and elderly, improving safety, and reducing resource consumption. The overall goals of these environments should be to maintain safety and security and maximize the comfort of the inhabitants.

Work in intelligent environments is an important step in the forward progress of technology. As computing becomes more pervasive and people’s lives become busier, advances in intelligent environments can aid by automating the simple things (e.g., lighting and HVAC control), work to actively conserve resources (reducing cost), and improve safety and security. Environments that sense their own well-being and can request repair or notify inhabitants of emergencies can save property and lives. Homes that can increase their own self-sufficiency over time can augment busy or aging inhabitants allowing people to live in their homes longer (potentially alleviating some health care system burdens) and free time to allow people to focus on other aspects of their lives.

Intelligent environments with humans in the loop present a real-world domain with unique features. The domains need to be sensor rich for perception but will still not provide a fully observable environment. How could one ever know what goes on in someone’s mind? If humans are creatures of habit, interactions should follow activity patterns with some regularity. However, human behavior can also be unpredictable at times providing a level of uncertainty. In order to automate features of an intelligent environment, we must make automation decisions in these uncertain, partially observable, and individually unique environments.

The unique properties of environments containing numerous sensors and *automatable* objects (objects that have control capability and are therefore able to be automated) create a large state domain. The ability to manually design data structures and model data for tasks involving large state domains will become an intractable problem. The ability to learn structures and models from data automatically would enable an increased capacity for working with large data domains and provide fertile areas for autonomous learning without human-influenced bias on structure.

This research examines the problem of learning human inhabitant behavioral models and their interactions in intelligent environments (i.e., smart homes and offices). By observing human-subject interaction data, we can create user models for the development of intelligent automation systems.

Work in decision-making under uncertainty has popularized the use of Hierarchical Hidden Markov Models (HHMMs) [54] and Partially Observable Markov Decision Processes (POMDPs) [189, 94]. Although the Hierarchical POMDP (HPOMDP) fits well for the intelligent environment domain, current work in the field requires *a priori* construction of the HPOMDP. Given the large size of our domain, we need to seed our model with structure automatically derived from observed inhabitant activity data. As a result, we look to the data-mining community for a possible solution since work in that community has been successful at finding patterns of activity in large amounts of data.

Hierarchical structures can be built through increased abstraction of discovered patterns building patterns of patterns in a tree-like structure until no further abstractions can be made. This creates a HHMM from the observed Markov chains. Extensions adding actions and rewards to this model create a HPOMDP. A combination of this user-model with a current observation data stream, a probability of current episode membership, and a prediction of events to occur develops a belief state and a possible chain of events to follow and potentially automate.

The accuracy of this technique can be measured by observing a reduction in the required manual interactions required of an inhabitant in an environment over a consistent set of interactions. Such a system is valuable if conditions never change and inhabitant patterns remain consistent; however, few things in life remain static and any useful automation system should be able to dynamically adapt to change.

Reinforcement learning based on inhabitant and system feedback used in the HPOMDP framework can be used to create adaptation to changing environmental interaction patterns. When an inhabitant corrects an incorrect action or takes an action not predicted and automated, an automation system should be able to respond by learning that change. The use of embedded safety and security rules should also be able to be learned so that any such system will not issue contradictory action decisions just to have a safety system prevent their action. Unsafe, insecure, and incorrect actions should be avoided by the decision process.

Given this background, motivation, goals, and approach, the work in this dissertation focuses around the following central hypothesis.

Inhabitant interactions in home and workplace environments can be accurately automated through sensor observation and intelligent control using a data-driven approach to automatically generate hierarchical inhabitant interaction models in the form of HPOMDPs, and these models may be modified using reinforcement learning techniques to continually adapt to changes in the inhabitant's patterns until a new model should be generated.

The ability to adapt to a dynamic environment can be evaluated through empirical observation of adaptation scenarios in which an automation system can maintain accuracy through a minimal amount of inhabitant interactions even in the presence of change. Learning of the embedded safety and security rules can be measured by a reduction in the number of violations of those rules over time.

We seek to validate our hypothesis by conducting empirical case studies using real and virtual inhabitants in a home and workplace setting in order to minimize inhabitant-initiated

interactions and the violation of safety and security rules as well as user preference rules. Additional evaluation test cases will be performed on the various aspects of the approach in order to better understand pattern discovery, automated model construction, and adaptation learning.

This research goes beyond the home and workplace environments and encompasses all environments (even virtual ones) in which observations can be perceived through sensors, those observations can be reasoned about by the system, and actions can be taken to automate features of that environment. However, we focus on these specific environments in order to bound this work for better evaluation and understanding and to support our hypothesis.

This work is done in conjunction with the MavHome (**M**anaging an **A**daptive **V**ersatile **H**ome) Project at The University of Texas at Arlington and utilizes the facilities of the MavLab as a workplace and the MavPad as a home. The MavLab (a.k.a., the AI Lab or Learning and Planning Lab) is a research lab that incorporates offices, research space in the form of a living room and a kitchen, and a conference room area. The MavPad is an on-campus apartment that hosts a student living in the facility full-time for the benefit of research.

This dissertation begins with an overview of the current work in intelligent environments, provides a perspective of the developed architecture for learning including a discussion of the system framework and architecture specifics, covers the details of the methodology from data collection through automation and adaptation, examines the experimental findings from a series of test cases and case studies, and finishes with a set of conclusions and a glimpse of future work.

CHAPTER 2

INTELLIGENT ENVIRONMENT WORK

Nine-fifteen, sang the clock, time to clean.

Out of warrens in the wall, tiny robot mice darted. The rooms were acrawl with the small cleaning animals, all rubber and metal. They thudded against chairs, whirling their mustached runners, kneading the rug nap, sucking gently at hidden dust.

Then, like mysterious invaders, they popped into their burrows. Their pink electric eye faded. The house was clean.

—Ray Bradbury, *There Will Come Soft Rains*

2.1 Introduction

Observing, learning, automating, and adapting in the intelligent environment domain crosses several areas of active research. There are many intelligent environments projects around the world, each focusing on a different aspect of the overall problem. This chapter focuses on the work of others and how it relates to the work presented in the upcoming chapters and covers areas and techniques from which our work is inspired. Other areas of related work are covered with the material presented in subsequent chapters to provide improved context and understanding.

2.2 Intelligent Environments

The work in this dissertation focuses on the emerging domain of intelligent environments or smart homes and buildings. Generally, these environments are defined by the way in which people interact with them or in the way that these places interact with the inhabitants. The community generally recognizes value in either type of interaction. Benefits include providing comfort and productivity for inhabitants and generating cost savings for utility consumption. There are many researchers working on interesting problems in this domain. We classify these projects into six groups (framework, application and gadget, industry, healthcare, learning and adapting, and others) in order to discuss their approach and contributions.

2.2.1 Framework Projects

Middleware and framework support for building intelligent environments has been the focus of many of the initial research endeavors in this area. These projects pioneered the ideas that fuel many of the newer and ongoing projects by providing software support and ideas that are useful in developing intelligent environments and related research.

Out of the MIT Artificial Intelligence Lab comes the AIRE (Agent-based Intelligent Reactive Environments) group. The AIRE group is engaged in research involving pervasive computing designs and people-centric applications and have constructed “AIRE spaces” in the forms of an intelligent conference room, intelligent workspaces, kiosks, and “oxygenated¹” offices. To assist in their research and to integrate their research technologies, they have developed middleware called Metaglué and an extension, Hyperglué [6].

Metaglué is an extension to the Java programming language to allow the creation of intelligent environment-controlling software agents. It provides *linguistic primitives* that facilitate the interconnection and management of a large number of disparate components (hardware

¹Derived from the MIT Project Oxygen which is focused on pervasive, human-centered computing that is as abundant as oxygen. Oxygenated refers to projects that incorporate the ideas and work of the MIT Project Oxygen.

and software), real-time operations, agent management, resource allocation, dynamic configuration, dynamic components, and state capture mechanisms. Metaglué provides an inherited *agent* class to Java objects, a post-compiler to generate new Metaglué agents from Java-compiled classes, and a *Metaglué Virtual Machine* infrastructure to be run on all supporting computing equipment. The Metaglué authors claim that the infrastructure creates a negligible amount of overhead and is more efficient than CORBA or KQML (Knowledge Query and Manipulation Language) because it provides both communication and control with a lighter-weight solution [34].

Hyperglue extends Metaglué by providing a society communication model and discovery system for Metaglué agents in a new infrastructure layer. Metaglué allows for agents to be segmented into small groups called *societies*. Hyperglue provides a communication layer for societies to communicate with each other and to find needed service-providing societies through resource managers and society ambassadors [152].

In addition to infrastructure for intelligent environments, the AIRE group is focused on developing collaboration support tools to support meetings by capturing information and facilitating the meeting, using sketch understanding for information capture, developing algorithms for the arrangement of information to foster a better understanding of the data, and extending instant messaging beyond the desktop to the physical environment. They are also involved in investigating novel human-computer interfaces that utilize technologies such as gaze-aware interfaces, streaming media, and multi-modal sketching which involves capturing speech with white-board sketching and gestures in order to capture the full meaning of the intended discourse. Recent work includes the development of applications to support plan-based proactive computing which can store and manipulate knowledge about user plans, habits and needs in order to execute actions of a user plan on request [6].

The work in this dissertation shares little with the work done by the AIRE group. Their focus is more on an enabling infrastructure which is designed to support HCI and collaborative

work in their intelligent environments than in automation, thus the direction for infrastructure in this dissertation did not utilize Metaglug or Hyperglug due to the learning overhead and focus difference. The AIRE technologies could be used in conjunction with our work to provide more collaboration support in environments such as our workplace environment.

The AMBIENTE division of the Fraunhofer-IPSI Research Institute in Germany is working on a number of intelligent environment-related projects. As a partner of the fifteen member European AMIGO project [9], they are focused on creating ambient intelligence (a paradigm that promotes the empowerment of people through environments that are aware of their presence and respond to their needs [69]) for the networked home environment where home automation, consumer electronics, mobile communications, and personal computing come together under complete integration. AMIGO seeks to research and develop open, standardized, interoperable middleware and intelligent user services for these environments. The goal of AMBIENTE's i-LAND is to create "intelligent user services" which will provide experience enriching services that will make the system seem intelligent. They are focusing these services on home care and safety, home information and entertainment, and the extended home environment. AMIGO is just in the beginning stages of life, but it draws from work on Ambient Agoras [57], i-Land [10], and other related precursor projects.

Ambient Agoras² is a project involving the development of situated services, place-relevant information, and a feeling of the environment in workplaces. It does this through ambient displays and user communication services as part of smart artifacts that are available in an invisible and ubiquitous manner. The goal is to develop environments into social marketplaces or *agoras* [57].

i-LAND, an Interactive LANDscape for creativity and innovation, seeks to create cooperative buildings out of *roomware* components such as an interactive electronic wall, interactive table, and computer-enhanced chairs in order to create the offices of the future.[10] The

²Agora is the greek word for a marketplace.

research focus is on these *roomware* components, but they are connected through integration on the COAST [179] cooperative hypermedia framework. COAST is groupware and heralds from research done in the Computer Supported Cooperative Work (CSCW) field. COAST is an object-oriented (SmallTalk) toolkit that supports the creation of synchronous groupware.

In the realm of producing devices, artifacts, and environments with a “coolness” factor to them, AMBIENTE research is a clear leader [197], but most of their work involves creating gadgets and exploring issues related to human-computer interaction. The items they create provide great insight for intelligent environment researchers to see what lies ahead in the future. Their middleware components are also more HCI-oriented and are not of current use to the work in this dissertation; although, work from AMIGO made publicly available in the coming years may prove valuable to many intelligent environment researchers.

At Stanford University, the Interactive Workspaces project is exploring work collaboration technologies in technology-rich environments with a focus on task-oriented work such as design reviews or brainstorming sessions. Their experimental facility is called “iRoom” where they are investigating integration issues with multiple-device, multiple user applications, interaction technologies, deployment software, and component integration.[193] To answer the integration and interactive workspace building challenge they have developed iROS, a middleware system for interactive workspaces. iROS is comprised of three subsystems: the *EventHeap* which provides coordination, the *DataHeap* which provides data movement and transformation, and *ICrafter* which provides user resource control. Through iROS they seek to provide a middleware layer that provides true platform portability, application portability and extensibility, robustness, and simplicity [160]. Current work at Stanford has shifted focus to the area where HCI meets the system in the development, deployment, and operation of iRoom human interfaces. They are studying issues with eye contact in videoconferencing, wall-display interaction, and information fusion issues.

Interactive Workspaces is another example of a project that created some middleware and then focused on the HCI aspect in a conference room workspace. The work in this dissertation does not utilize ideas from this project because of a difference in focus; however, the work could be utilized to complement our research by enhancing the user's experience through sophisticated HCI techniques.

The Laboratory for Communication Engineering (LCE) at Cambridge University (originally in conjunction with AT&T Laboratories Cambridge) is pursuing their Sentient Computing project with the focus of simulating computer perception in computing systems that detect, interpret, and respond to facets of a user's environment [17]. Research has involved a deployed ultrasonic location system, advancements in world modeling including spatial considerations, and sentient computing applications such as world model browsing, remote desktop displays that follow users, smart posters using context-aware information retrieval, and ubiquitous user interfaces [25]. The omniORB CORBA package and VNC (Virtual Network Computing) both originated from the AT&T Labs, Cambridge, research.

The Sentient Computing group no longer uses omniORB in favor of middleware that can support context-aware multimedia applications called QoSDREAM, also under research and development at the LCE. QoSDREAM supports multiple types of sensors and provides a simple spatial model for representing *locatable entities*, real-time model and sensor data integration, an event mechanism for notifying applications of location information, a queryable location database, and an ease of extensibility. QoSDREAM is based on providing Quality of Service guarantees and takes a location-centric approach to the services it provides [136].

The work from this project as well as other projects that were originated from the AT&T Laboratories Cambridge includes some of the pioneering efforts in ubiquitous computing. Many current projects use technology and ideas that originated from these groups. The active badge location system (or BATS) was a fully developed, deployed, and tested inhabitant location system that was used in office workspaces to forward phone calls and information.

OmniORB and VNC are used throughout the world on many other projects today including some of the work in this dissertation though mostly in terms of support technologies. Many other interesting projects were also developed and the interested reader is encouraged to review the online information at www.xorl.org.

The Gaia project at the University of Illinois at Urbana-Champaign [216] involves the creation of active spaces for ubiquitous computing. Their focus has been on creating middleware to support environments that sense inhabitant actions and assist them with different tasks—referred to by this work as an “active space.” In support of this goal they have developed the Gaia OS which is a meta-operating system that has been customized for specific physical spaces in order to support application development in those domains. Gaia supports mobile user-centric active space applications by managing the resources and services of an active space and providing services for location, context, events, and storage of information. Gaia consists of the Gaia Kernel, the Gaia Application Framework, and Gaia Applications. The kernel consists of an event manager which distributes events in the active space, a context service which provides contextual information in the form of first order logic and boolean algebra, a presence service which is resource-aware and provides information, a space repository to store information, and a context file system to make personal data available to applications, organize data, and retrieve data in a format based on the context of user preferences or device characteristics. The application framework provides mechanisms to develop, execute or accommodate existing applications to active spaces and is comprised of a distributed component-based infrastructure which provides a model, view, controller, and coordinator; a mapping mechanism which customizes applications to different active spaces, and a group of policies to customize the applications [172]. Gaia applications include a presentation manager to present slideshows on one or many displays in an environment, ConChat which is a context-aware chat program, a calendar program, meeting attendance task recorder, media players, speech engines, PDA interfaces, and other presentation and workgroup centric applications [216]. The Gaia researchers

have limited their domains to spaces used for teaching such as classrooms, offices, and lecture rooms.

Gaia is mostly about infrastructure with a number of conference/lecture room applications. The work in this dissertation does not utilize any of this work mostly because of the difference in focus as is the case with projects similar to Gaia. There are a number of commonalities in the work that is presented here and Gaia, namely the concepts of object-oriented encapsulation for real objects in the virtual world representations, component-centric views, object-oriented communication (i.e., CORBA), and the focus on context to provide service value.

The United States' National Institute of Standards and Technology (NIST) has established the Smart Space Laboratory to "address the measurement, standards, and interoperability challenges" [139] for *smart spaces*. This group has established a two-phase approach to addressing their interests in pervasive computing. In the first phase they are identifying areas for standardization, creating real experiences in which to identify applicable measurements, and identifying security measures to ensure the privacy and integrity of systems. The second phase involves developing specific metrics, tests, and comparative data sets for the community, providing reference implementations of designed system ideas, collaborating with industry to form standard specifications, establishing industry and academic partnerships, and integrating the phase 1 technologies to explore distributed issues with the technologies. From these goals they are working to create a test-bed containing a defined middleware component that provides real-time data communications, a connection broker for sensors, and containers for processing data. NIST has already released their Smart Flow System that provides a data flow server, graphical interface, and control console as well as audio, video, and voice recognition interfaces [139].

Development by NIST in the Smart Spaces Laboratory has been stagnant since 2000 with only active work continuing to date with their smart flow system. Their work can be classified

with the groups focusing on conference/lecture workplaces, but their research on video and audio systems and tracking could be very beneficial to projects using these types of sensor arrays. The work in this dissertation does not use video or audio information and therefore does not use any NIST technology.

Researchers at the University of Florida are building the Gator Tech Smart House with the goal of creating assistive environments that can perceive themselves and their residents and utilize telematics to provide services. Drawing on their previous knowledge gained from experimentation in their Matilda Smart House laboratory, they have constructed a full house in Gainesville, Florida. This home features such technologies as a smart mailbox that senses mail, a smart front door to control access, a driving simulator in the garage to evaluate elderly driving abilities, smart blinds, a smart bed, a smart closet that helps with clothing choices, a smart laundry, a smart bathroom mirror to display information, a smart bathroom that controls water temperature and features bioinformatic capture, smart displays throughout the home, a smart microwave, a smart refrigerator and pantry, smart cameras for security, ultrasonic location detection, a smart floor for localization, a smart phone, smart wall plugs for electrical items, smart thermostats (i.e., HVAC control), smart leak detectors in areas with water, a smart stove, a smart display projector for information, a home security monitor, an emergency button, and cognitive assistance through visual and audio cues to help the inhabitants remember medications, appointments, and other important items. Their current key contribution is the development of a middleware architecture which includes a physical layer of devices, a sensor platform layer to convert readings into service information, a service layer to provide features and operators to components, a knowledge layer that offers ontology and semantics, a context management layer to provide context information, and an application layer to support a rich set of features for inhabitant living. The state of the project is still focused on integration and the middleware development, but they are beginning to focus on issues with eldercare and the aging in place initiatives [76].

2.2.2 Application and Gadget Projects

A number of intelligent environment projects focus on applications or gadgets that improve the experience in those environments. Often these elements are focused at developing context awareness in order to augment the inhabitant experience in that context. Location also plays an important factor in understanding inhabitant context and providing value-added services in the new breed of intelligent environments.

The Aware Home Research Initiative (AHRI) at the Georgia Institute of Technology is conducting research looking to answer the question, “Is it possible to create a home environment that is aware of its occupants whereabouts and activities?” [5] Efforts of this group are focused in the areas of interactive experience applications, technology, software engineering, and an investigation of the social implications involved with aware home living.

Their interactive experience applications have been developed in the areas of social communication, memory aids, and home assistants. The digital family portrait project has produced an application that provides well-being information in the form of sensed activity levels and home environmental conditions about a remotely monitored individual. The “dude’s magic box” project has focused on improving the connection between geographically separated people and between grandparents and grandchildren in particular. This project involves the capturing and sharing of imagery and voice-note interactive annotations about physical objects such as a child’s play item. A child would place an object in a capture facility where an image can be captured and relayed to the grandparent who can then ask questions or make remarks about the object (in essence playing with the child) while not having physical presence. In the area of memory aids, their work has been focused on the cook’s collage project which involves the image capture of cooking steps from initial observation and subsequent playback during repetitive use of the recipe in order to provide visual cues for repeatability. Other work in this area includes the memory mirror project which seeks to capture and record short repetitive events that are performed frequently (e.g., feeding a pet) and maintain a record so that users can use it

to remember if they performed an action or not. The gesture pendant is a home assistant device that a user wears around their neck and performs hand motions (i.e., gestures) in front of in order to interact with their environment. The system can also be used to monitor the user to detect hand tremors or the loss of motor skills over time.

Technology development by the AHRI has involved an indoor location service and an activity recognition algorithm. AHRI has focused on the problem of sensing location and as a by-product have produced the Location Service infrastructure which seeks to provide a robust and accurate location service, performs sensor fusion transparently for the user, and supplies reusable and extensible techniques to application programmers so that they may utilize location information in application-relevant ways [2]. The specific technologies used in this service involve RFID (Radio Frequency IDentification) tags, smart tiles that sense pressure placed upon them, and computer vision systems. Activity recognition methods are being developed by AHRI to monitor the general activities of the inhabitants and includes low-level tasks such as reading a newspaper or watching TV, and high-level tasks such as cooking or using a health monitor. Other technology projects being developed at AHRI include multi-camera eye/head tracking, audio and video sensor fusion, open-air microphone speaker identification, automated separation of sound sources, large-scale projective displays, and controlling multiple distributed displays.

In the area of software engineering, the AHRI researchers have conducted research developing two toolkits, INCA and the Context Toolkit, as well as a location service. INCA is the Infrastructure for Capture and Access, and it provides the means for creating systems that capture life experience details and preserve them for future access. INCA provides capture, storage, format-conversion, and access support [214]. The Context Toolkit provides abstractions and support for context-aware development. This Java toolkit consists of widgets as an encapsulation object, aggregators to create meta-widgets, and interpreters to translate low-level context information into higher levels all communicating via XML over HTTP [178]. Other

software construction projects being developed by AHRI include secure storage for managing personal information, space-time memory abstraction for interactive multimedia, and the impact of industry standards (e.g., UPnP) on an aware environment infrastructure.

The work done by AHRI has been pioneering in the area of intelligent environment research, but has mainly focused on infrastructure for the support and development of specific applications, most of which are geared more toward HCI and assistive technologies than home automation. Since the work in this dissertation focuses on the latter, it does not use AHRI work. The only similarity are the needs for location and context awareness. Since our work does not involve video or audio, employ location tiles, or use RFIDs, the location work does not fit. Also since the focus is different the context-awareness work does not fit; however, many of AHRI's technologies are very complementary to areas of our own active research and would help augment our environments and improve the inhabitant experience if incorporated.

MIT Media Lab's Consortia on Things That Think (TTT) and their special-interest group on Counter Intelligence are primarily focused on single applications such as an augmented reality kitchen, context-aware tables, dishmaker, food oracle, SmartSink, hand-washing compliance, InVision, living food, Mcam, Sensa/ible dishware, synaesthetic recipes, talking trivet, and waterbot. They have also been involved in developing smart architectural surfaces, an intelligent spoon, and have also produced a distributed agents platform called Hive [103]. The augmented reality kitchen projects informational displays onto existing counters, cabinets, and appliances in order to improve ease of use, efficiency, and safety. Context-aware tables are physical tables that change their purpose by height (e.g., display picture book photos when raised and food menus when lowered). Dishmaker is an appliance designed to replace cupboards and dishwashers with a home manufacturing system that produces dishes on demand, and then recycles them when you are done using them. The Food Oracle is a set of tools that combines learning and reasoning about cooking and food into a system to help people explore creative and intuitive cooking. The system reasons about recipe-sensitive ingredient substitu-

tions and the intentions of recipes, and uses a database of food and culture to create an adaptive and dynamically generated smart recipe system based upon a wealth of information about food. SmartSink is a context-aware sink that adjusts to user height as they approach and adjusts the temperature of water based on what is put in the sink. It also illuminates the water stream with color by temperature (i.e., red for hot and blue for cold). Hand-washing compliance assistance in conjunction with the smartsink has been developed in order to perform localized environmental automation (e.g., turning on lights) to facilitate maintaining clean hands in such environments as hospitals. InVision examines user visual perception and observation patterns to infer context in order to provide context-aware application assistance to the user's task. Living food is a refrigerator replacement for fresh fruits, vegetables, herbs, and spices that seeks to prolong their storage life. MCam is a milk-carton camera that is used to communicate pictures of the inside of a refrigerator to an Internet-connected appliance. Sensa/ible dishware is focused on adding sensors to everyday dishware in order to learn usage patterns, food information, and the user's eating habits. Synaesthetic recipes is a visual search program which allows users to enter imaginative textual descriptions (e.g., something light, fresh, easy to make, but not creamy), and uses these to produce recipe recommendations from a large recipe database. Talking trivet/oven mitt provide digital enhancements to existing household objects in order to communicate information such as cook time, temperature, and faults to external components. Waterbot is a persuasive technology to motivate water conservation by providing context-sensitive feedback to users in order to change user behavior to promote water conservation while using the bathroom sink. Water usage information is collected and used to initiate unobtrusive user prompts using several persuasive techniques (e.g., law of contrast, positive reinforcement, variable schedule of reinforcement, social validation, scarcity, curiosity, and challenge) in order to modify user behavior. Smart architectural surfaces focuses on the creation of modular computational elements which can be used to build sensor and display equipped intelligent spaces. Elements such as smart video tiles, advanced image projectors,

camera tracking systems, and object-based media characterize this work. The intelligent spoon is another example of a sensor laden utensil that can relay temperature, acidity, salinity, and viscosity information back to a computer. There are a dozen or more such projects as these undergoing varying levels of development, research, and testing at the MIT Media Lab [103].

Hive is a Java-based framework that provides ad-hoc agent interactions, mobility, ontologies, and a graphical user interface to the entire distributed system. Hive is composed of three elements: *cells* which exist on each machine in the distributed system and provide the infrastructure connectivity, *shadows* which are the local resource encapsulation mechanism in each cell, and *agents* which utilize local resources through shadows and exist in one cell at a time. Hive has been used to develop a smart kitchen, a jukebox, and other applications [128].

The MIT Media Lab is focused on gadget creation and specific implements of the future. Many of these projects could be incorporated into an intelligent environment to enhance the inhabitant's experience, but they probably will not be commercially available for another decade. The work in this dissertation does not incorporate any MIT Media Lab technology or gadgets primarily due to their availability and the significant amount of engineering effort that would be required to duplicate and integrate their work; however, specific ideas such as those in the augmented reality kitchen, localized context awareness, and the interactive nature of many of their projects could be incorporated into our environments.

2.2.3 Industry Initiatives

Industry is very interested in the pervasive computing future that society is moving towards, and they recognize that the home is a place where there exists fertile ground for offering products and services to improve the lives of people. To this end there are a number of industry initiatives that explore technologies and concepts in this arena.

The Pervasive Computing Lab at IBM Research in Austin, Texas, is performing work involving speculative integration to create *proof of concept* demonstrations utilizing modern

technology to create such things as an advanced media living room, a networked kitchen, and integrated automobiles. Although not developing new technologies, they are combining existing technologies in new and interesting ways to show what is possible. For the designer and integrator it should be of interest that they use IBM Websphere, Java servlet technology, and Lotus Notes [226].

The Vision Group at Microsoft Research were in the process of developing a prototype architecture and associated technologies for intelligent environments in their Easy Living project (no updates since 2001). Their research was concerned with using computer vision for inhabitant tracking and visual gesture recognition, sensor fusion, context-aware computing using geometric models, automatic sensor calibration, dynamic and adaptable user interfaces, generalized communication and data protocols, and system extensibility [65]. To provide integration for their research systems the Easy Living project used middleware called InConcert which was being developed at Microsoft Research in conjunction with other groups there. InConcert was created out of a need for middleware that can provide communication in a distributed environment, but they feel is better suited for the real-time and unusual needs of an intelligent environment than DCOM, Java, or CORBA [23].

Microsoft also has a demonstration home on their Redmond campus that features a six room home enclosed in a building where they demonstrate what they perceive to be technology in the five to ten year out window. Featured technology involves the integration of Microsoft products and services such as their digital music and video offerings. Recent demonstrations show the integration of RFIDs on clothing and a mirror with a built in display that provided care instructions as well as garment matches from the user's wardrobe [48].

Other industry initiatives include British Telecom's Telecare project which established 10 "millenium homes" for aged people where an array of sensors monitored environmental conditions, notified the inhabitants to correct potentially dangerous conditions, and notified care providers on necessity [24]. CISCO Network's Internet Home explores the impact of the

Internet on the home by showcasing Internet-dependant technologies and wireless web pad interfaces [33]. Intel Corporation's Proactive Health Lab is exploring technologies to help seniors "age in place" in order to help the increasing health care burden of the rapidly aging population of the United States by anticipating inhabitant needs through observation with wireless sensors and taking action to meet those needs through available control and interactive systems [92]. Siemens AG is promoting "living made easy" home automation products which include a full range of components for automating almost all of the common items in a home and integration systems as part of their smart home technology initiative [182]. Royal Philips is involved with exploring ambient intelligence and user experience in their HomeLab experiments involving people living in an apartment and being observed by a research team [153]. Finally, Accenture is focusing on elder care and lifestyle with their room of the future which examines activity monitoring and interactive furniture and objects that are meant to improve the inhabitant experience while allowing for aging in place [3]. Many other companies enter and exit the domain offering products and services for a time then moving on to other business opportunities. Besides the companies previously mentioned here there are also a number of commercial retail companies selling products to make people's homes "smart." The most prominent of these companies is SmartHome (www.smarthome.com). There are also a number of local and regional installers and contractors that will perform custom integration of items into homes. Many of these are involved with the home security market. In general, industry initiatives are aimed at showing how a particular group of products or services can be integrated in order to solve a particular problem such as aging in place or an immersive Internet-based information experience for the home user. Some of the materials used in the implemented environments in this dissertation use equipment purchased and/or donated from companies pursuing smart home technologies.

2.2.4 Healthcare Initiatives

Advances in technology are prolonging the life expectancy of people. Coupled with a large rapidly aging population born in the post World War II era (i.e., baby boomers), a rising problem is emerging in the increased need for eldercare. Trends in the healthcare community are placing more of a burden for this increased care on family and friends while there is a simultaneous desire by the aged to remain independent. These conflicts present a strong need for research to investigate the possibilities of a maintained independence for elders who wish to “age in place” while alleviating the healthcare and caregiver burdens. Technologies such as reminder systems, cognitive assistance, telematics (monitoring information sent to remote locations where interested parties are informed), safety monitoring, and other types of systems are under active research. There is also research underway to assist in understanding the effects of these technologies, their need, their benefit, and ways to match the needs of the aged to the appropriate technologies. This is an area of growth that will become more and more important as the aging population approaches a critical mass. Several research groups have already begun to address the myriad of issues in this emerging problem.

The Medical Automation Research Center (MARC) smart house project at the University of Virginia is focused on the issue of in-home monitoring for the elderly in order to promote the concept of aging in place. Their homes are equipped with low-cost, non-invasive sensors (they do not allow cameras or microphones), and a data logging and communications to establish telematics to authorized individuals (e.g., family and their personal physician). They have developed data analysis tools to observe general health and activity levels and have developed the metrics of Activities of Daily Life (ADL), most Instrumental Activities of Daily Life (IADL), the index of well-being, and a measure of ability decline. System feedback to prompt the individual to remain active is also being explored. The MARC In-Home Monitoring System has been deployed in several case study homes. Its key features are the low cost technology, the

ability to retrofit it into existing homes, and the data-mining health status report components [116].

The Robert Gordon University CUSTODIAN (Conceptualization for User involvement in Specification and Tools Offering the efficient Delivery of system Integration Around home Networks) project has an objective to develop technology and services for disabled and elderly people through information and communication technologies in order to improve the quality, effectiveness, and efficiency of services which support their independence in living and maintained integration in society. To this end they are focused on maintaining and restoring functional capabilities through assistive technologies, electronics products, and systems around a home network in smart houses. They have designed and established a demonstration house called the Dundee Flat to perform experimentation for the personal care services. Their current work involves case studies that observe process facilitators that try to match individual needs to technology in an attempt to improve the way recipients are fit to technology [51]. Their work addresses the issues of proper fit, deployment, and usage of the technologies of smart home with the disabled and elderly.

In the last section, many of the industry initiatives (e.g., Intel, Accenture) have focused their intelligent environment research initiatives at the aging in place problem. Growth in this area is sure to bloom as groups finishing their initial intelligent environments research turn their attention to this area of specific need and valuable impact. The growing need of a rapidly aging population in developed countries, the health care system burdens, and the lack of care providers signal a warning of troubled times ahead that research now may help curb by allowing the aged to live independently longer, healthier, and safer while maintaining a high quality of life and alleviating the burdens on society.

2.2.5 Learning and Adapting Initiatives

Beyond just context inference comes the desire to learn from the inhabitant of an intelligent environment, often in order to automate parts of their routine in order to improve the inhabitant's experience. Along with the initial learning comes the desire to provide lifelong learning, or adaptation over time, in the environments so that they can move through the phases of life with their inhabitants in order to provide a long-term improvement in the environmental living experience.

The Adaptive House project at the University of Colorado at Boulder under the direction of Michael Mozer tackles the issue of overcoming the programming problem with home automation (i.e., where someone must program the rules for automation and reprogram them over time as the inhabitant's lifestyles change). Their work involves developing a system that controls the HVAC, water heater, and interior lighting of a home, learning how to control these features based on the lifestyle and desires of the inhabitants, and adapting the control policy over time in an environment with a minimal user interface. This project uses an actual residence called the neural network house (Mozer's own home which is a converted historic old school house in Marshall, Colorado) equipped with 75 sensors that monitor temperature, ambient light levels, sound, motion, door and window openings as well as actuators that control the furnace, space heaters, water heater, lighting units, and ceiling fans [132]. The control systems in this work are based on neural network, reinforcement learning, and prediction techniques called ACHE.

The ACHE (Adaptive Control of Home Environments) system uses a combined systems approach for home control. Q-learning (a reinforcement learning technique [203]) uses event-based segmentation over clock-based in order to make the problem tractable and initiates actions based on perceived state and reward. In order to simplify the state space the automation task was decomposed into zones and a heuristic based determination of event separation factors was used to partition the experience into events for the event-based control system. The

control policy involved a mixing of inhabitant comfort and energy conservation goals. The system used a state estimator to form high-level state representations which were based upon inhabitant activities (through an occupancy model and an anticipator which was neural network based and provided a prediction of occupancy of a space) and light levels in each zone (through a natural light estimator). This information and the decomposition of spaces were utilized with multiple Q controllers to automate the home. Their work also involved some exploration in order to reduce energy consumption by occasionally testing the inhabitant by altering the control policy unless counteracted by the inhabitant. A single experiment was conducted in the adaptive house and no formal evaluation of the system was ever conducted [133].

The adaptive house involved a lot of knowledge engineered into the system in the form of event partitioning heuristics, look-up tables for Q-values, and human-directed partitioning of the system into multiple Q-learners controlling specific areas of the house. The sensors used were sparse and prone to admitted error. In relation to the work in this dissertation, the anticipator and occupancy models are not necessary in our systems due to a better hardware design and implementation. We agree with the vision and premise of the adaptive home—that an intelligent environment should adapt to the needs, lifestyle, and desires of its inhabitants [131]. We also agree with the idea of using a minimal and natural interface as well as focusing on the goal of solving the user programming problem. Many of the heuristic decisions and partitioning schemes in the adaptive house are avoided by learning the requirements for decompositional and hierarchical construction of a control policy through observation. In earlier versions of our work we utilized Q-learning with CMAC tiling in a control mechanism similar to Mozer’s work [39]; however, we also discovered similar state space abstraction problems in automation and a need for better sensor information in order to accomplish our goals. The work in this dissertation can be seen as a direct extension of this shared vision towards creating environments with control policies observed from their inhabitants, but we do not currently perform energy saving learning activities directly at this time.

Researchers from the Intelligent Inhabited Environments Group (IIEG) at the University of Essex in the United Kingdom are creating an *ambient-intelligence* environment using embedded agents called the *iDorm* [88]. Their approach involves the use of a fuzzy logic based *Incremental Synchronous Learning* (ISL) system to learn and predict inhabitant needs. Their testbed environment, a dorm room, involves the access of 11 environmental parameters and nine effectors (mostly lights). The use of parallel fuzzy logic controllers (FLC) in a hierarchy is used to learn and encode rules. Each FLC has a single modifiable parameter and is used to learn a particular aspect of the environmental control. The FLCs are either static (i.e., pre-seeded with knowledge) or dynamic (i.e., observed from the inhabitant). In combination all of the FLCs form the ISL system and encode the desired control behavior of the environment. Other management systems prune down the number of FLCs by observing factors of redundancy and low usage to keep the system computationally manageable. The researchers have presented evidence via empirical evaluation of *iDorm* inhabitants that the system can perform initial and lifelong learning of inhabitant needs over a 132 hour experiment [69].

The *iDorm* project has many similarities to the work presented in this dissertation. The notion that there first needs to be an observation period followed by usage and learning has been a part of our systems. The general focus on learned automation is also the same. Besides the obvious difference in approach we seek to learn the hierarchy as well and not engineer it through heuristics. We agree that it is an important feature of intelligent environment control systems to have specific rules for automation and not generalized ones which tend to not automate the environments correctly, this issue is on how to deal with environments with large state spaces in order to perform real-time calculations and decision tasks with the correct level of particularization. *iDorm* has only been tested on environments a fraction of the size of the ones presented in this dissertation; however, as our respective work continues we may find commonalities that may prove to be better in combination in order to satisfy our shared vision.

2.2.6 Other Initiatives

Architectural focus, the psychological aspects of smart home living, integrated device and appliance based smart home projects, emerging consortiums, and other unique aspects characterize a number of intelligent environment projects focusing on specific niches in the areas of research. They all contribute to the collective knowledge we are learning about these new breeds of environments.

The Department of Architecture's Changing Places/House_n project at MIT is focused on how technology, materials, and design strategies can create dynamic, evolving places that respond to the lives of their inhabitants. Using the one bedroom condominium called the PlaceLab, researchers use sensors spread throughout the environment to observe the environment and develop innovative user interfaces for control of the spaces, resource management, and to maintain their health and activity levels. Current projects also include participation in the Open Source Building Alliance (OSBA) where key components of a more responsive model for creating living spaces are being developed: home chassis design to make building a home similar to industry standards for building vehicles or electronics, integrated interior infills which replace interior walls with customizable cabinetry-like components, design and configuration tools to promote the ease of design of these new types of structures, just-in-time persuasive user interfaces for promoting healthy behaviors by encouraging a more active lifestyle, context-sensitive measurement of physical and sedentary activities, context aware experience sampling which attempts to learn inhabitant activities from observed sensor readings, portable wireless sensors for studying behavior in natural settings, proactive health displays for health assessment and self reflection, idle moment detection for proactive health activities using personal and environmental sensors and interfaces, and many others [129].

Much of the work done on the House_n project focuses on architecture and measurement. The work in this dissertation also uses environmental measurement, but we are also concerned with automation so the sensed information focus differs from their approach. There are still

many insightful pieces of information coming from their project in the area of context awareness especially in the area of health and activity monitoring which may be useful for future work.

The Australian Commonwealth Scientific and Industrial Research Organization (CSIRO) and the University of Technology Sydney NanoHouse project is working on a demonstration of nanotechnology applied to advanced building materials and their incorporation into the building of homes. Their focus is on energy conservation by developing transparent materials that allow light to pass but reflect heat, building materials that are more environmentally friendly, and other building technologies in order to create an ultra-efficient house [41].

The Digital World Research Centre (DWRC) at the University of Surrey is involved with psychological impact studies and cover how people communicate, live, play, shop and work [49]. DWRC has studied how families react to living in an intelligent environment and have documented the lessons learned for the future development of smart homes and smart home technologies [70]. Much of their work is confidential and not available to the public.

Tampere University of Technology Institute of Electronics in Finland started the eHome research project in 1999 and created the smart living room in 2002. Their research is focused on the design and construction of ubiquitous electronic devices in a smart home environment. In their environment they have developed a wired and wireless sensor network, lighting control, speech control, and user interfaces. Unique features include a plant monitoring system that senses water and light needs, user identification with infrared tags, air quality measuring, and floor sensors for occupancy [207].

In Japan, efforts by the National Institute of Information and Communications Technology (NiCT) and their Keihanna Human Info-Communications Research Center are focused on the development and testing of the Ubiquitous Home. Their research goal is to support and optimize the usage of information appliances in the home across the users regardless of age or lifestyles. They are developing many technologies which include middleware in the

form of a distributed collaborative infrastructure for the home appliances to interact, an interactive field model for context-sensitive services to enhance the user-appliance interaction, a distributed environment action database to store and recall information about interactions, and event interactive robotics that provide context-sensitive interaction. In league with many Japanese technology companies, their research aims to improve the relationship between humans and household appliances. Efforts for evaluation of case studies involving human family participation in their Ubiquitous Home are just beginning [138].

Besides AMIGO on the European continent, the United Kingdom Equator Interdisciplinary Research Collaboration comprised of eight members (The University of Bristol, The Lancaster University, The Royal College of Art, The University of Sussex, The University of Glasgow, The University of Nottingham, The University of Southampton, and The University College London) and supported by the UK Engineering and Physical Sciences Research Council (EPSRC) is another supergroup of researchers working on pervasive/ubiquitous computing and intelligent environment related work. They are focused on the integration of physical and digital interaction in order to bridge the gap between reality and virtual reality. This combined effort aims to search for a better understanding of what it means to live in an age when digital and physical activities not only coexist but co-operate and interoperate. Their list of publications cover almost all areas of related research (e.g., HCI, location systems, sensors, and so forth), but do not seem to currently be focused on developing specific intelligent environments or automation and inhabitant context learning [53].

Two other examples of integrated components and HCI work for interaction in intelligent classrooms are the Northwestern intelligent classroom [56] and the smart classroom at Tsinghua University in China [181].

In a more specific use of smart rooms, the UCLA HyperMedia Studio is investigating the use of intelligent environment sensors and other ubiquitous computing technologies to provide services for film production in those environments [215].

The Multi-Agent Systems Lab at UMASS has investigated the use of a multi-agent system for the control of a simulated intelligent home called IHome. Their research is focused on applying a multi-agent system to the domain and examining issues of resource coordination and temporally shared resource activities among agents [109].

The PRIMA project at INRIA is concerned with the scientific foundation for interactive environments. Their mission is to develop and integrate systems with the ability to perceive and model an environment and its contents, to act in/upon this environment, and to interact with the occupants. Their research focus is on multi-modal observation and tracking of people, new forms of man-machine integration, control and integration of perceptual processes, and context guided learning and recognition [89]. They have an augmented meeting room testbed for their experimentation that is equipped with cameras and a microphone array. This group is a member of the AMIGO project.

There are many other intelligent environment projects in academia, government, industry, and even amongst the enthusiastic general population and home hobbyists. We have covered many of the current and historical efforts in this area. The current trend is that many groups are combining their efforts to eliminate redundancy of effort and focus on the research challenges. These super efforts from groups such as Equator in the UK and AMIGO on mainland Europe are beginning to produce large quantities of results and forward knowledge in intelligent environment research. If the research community in the United States does not form such research cooperatives in the very near future, we will soon lose our standing as a research leader in this area [77]. New projects such as the Gator Tech House host promising new research facilities, but will researchers in this area start large-scale collaborations or remain islands to themselves?

2.3 Pervasive and Ubiquitous Computing

The creation and development of intelligent environments involves the integration of computation into the environment, a notion that is referred to today as ubiquitous computing or pervasive computing. Ubiquitous computing owes its origins to Mark Weiser who was the Chief Technologist at Xerox PARC until his untimely death from cancer in 1999. His 1991 *Scientific American* paper entitled “The Computer for the 21st Century” [221] is considered the seminal paper in this emerging area of research. In that paper, Weiser forwards the idea of “calm computing” or “ubiquitous computing” where the technologies that have the most impact on our lives are an integral part of the fabric of our lives, disappear as technology itself, and become the things that we as humans take for granted to work in a way that enriches our lives and makes us productive, satisfied, or entertained. His original vision contains many of the technology paradigms employed in intelligent offices, meeting rooms, and classrooms (e.g., interactive white boards, PDAs, gesture devices, and so forth) and that explains why so many projects focus on this environment. The idea of ubiquitous computing is still far off in our future. The road to that future is unclear and is why there are so many ideas currently under investigation. This is a healthy start though to a promising future.

This area of research is so new that there are still a number of dual-monikers being used to describe the same ideas. One point of confusion is the very name of the area of research. Is it ubiquitous or pervasive computing? Reviewing the literature published in the proceedings of the UbiComp, Pervasive, and PerCom conferences yields no clear direction either since the same general areas are included across all of them. An idea forwarded by the 2005 PerCom conference keynote speaker, Roy Campbell (project director for the UIUC Gaia project), was that based on the definition of ubiquitous “being present everywhere at once” [163] and pervasive “spreading throughout,” [163] ubiquitous computing is the end result (and the goal of) pervasive computing—the journey to ubiquity is through pervasive technologies. In this dis-

sertation, we will use the term pervasive over ubiquitous as we feel that we are contributing to the spread of computing with a goal of being ubiquitous.

Research in pervasive computing includes sensor networks, location management, energy efficiency, device creation, wireless networks, service discovery, security, middleware services and platforms, context and system support, mobile services and protocols, applications, and so forth. Many of these technologies are used in this research, and we have made some contributions in those areas (especially in sensor networks, service discovery, middleware, and context services) which are or will be covered in other avenues of publication, but the core work of this dissertation focuses on our work in decision-making and areas related to user modeling and context understanding.

2.4 Conclusions

Many of the intelligent environment projects offer complementary areas of research that are all a piece of a large puzzle that can be assembled in order to create a complete intelligent environment, building, neighborhood, city, and so forth. The work in this dissertation differs from the related work presented in this chapter by providing a completely data-driven approach to learning a model of user activity. The HPOMDPs created by this technique are unique compared to the current practices of hand-generation or heuristic creation. This work differs from most intelligent environment work in the adaptability over time of our models. The MavHome project also hosts the longest running human experiments and captured information to date of any intelligent environment project. We hope that our findings and the approaches taken in this dissertation provide useful scientific information to the community researching and developing intelligent environments.

CHAPTER 3

A LEARNING ARCHITECTURE

We can only see a short distance ahead, but we can see plenty there that needs to be done.

—Alan Turing, *Turing Test*

The work in this dissertation focuses on learning to automate the intelligent environment. In this chapter we introduce the problem, present how the problem will be solved, provide an overview of the system framework, and then introduce the core system architecture. This chapter will end with a presentation of the environments used for testing and evaluation. The purpose of this overview is to familiarize the reader with the system we have developed, the names and functionalities of the major components, and the domain to which we are applying this work.

3.1 The Problem

The motivation for this work is the development of systems to automate intelligent environments in an accurate and efficient manner. There are a number of significant challenges in this work in order to meet our goals.

3.1.1 Goals

The goals of our system are to learn a model of the inhabitants of the intelligent environment, automate devices to the fullest extent possible using this model in order to maximize the comfort of the inhabitant while maintaining safety and security, and adapt this model over

time to maintain these requirements. In order to accomplish these goals, we must first learn a model of inhabitant activities, and then incorporate this into an adaptive system for continued learning and control.

Our development goal is to create an agent based system as shown in Figure 3.1. The very essence of this system is to perceive the environment through sensors, reason about this information in order to make decisions on whether or not an action should be taken to change the state of the environment in which the agent is situated, and then perform this action through actuators which will affect the perceived state continuing the cycle *ad infinitum*. This work focuses on an agent based system centered around a known single inhabitant in our environment.

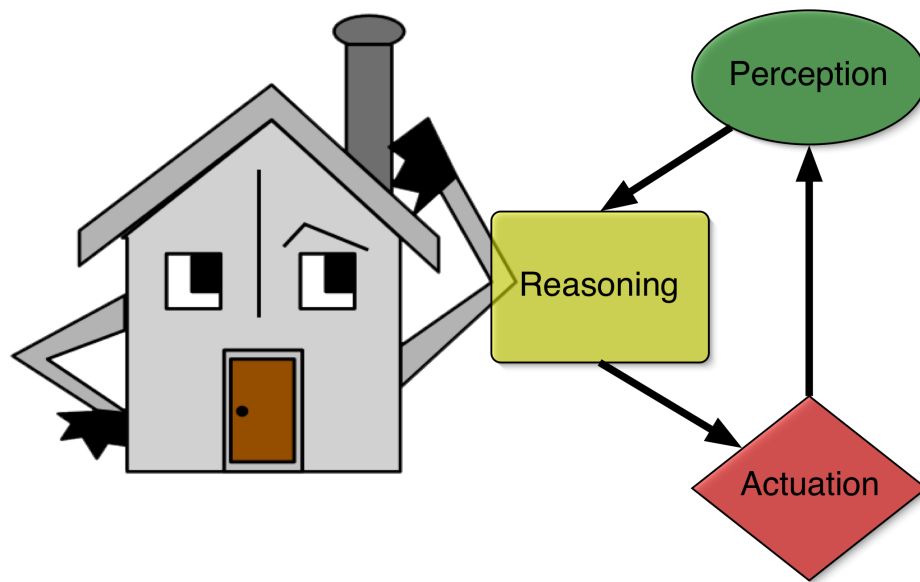


Figure 3.1. Basic agent architecture.

3.1.2 Perception

This work explores beyond simulated worlds and into real home and workplace environments. In order to develop correct theories and algorithms, we must be able to perceive the real

and virtual testing environments in the same manner. The challenge lies in being able to observe accurately enough of the real world to have sufficient information from which to observe and reason about acting. It must also be sufficient enough to model the real world with enough precision in the simulation environments in order to conduct longer term and more controlled experimentation. The type, number, and placement of sensors as well as data collection issues present an important layer of information. This work will focus on the perception of appliance state (e.g., lights, fans, mini-blind position, and so forth) as well as environmental data such as humidity. Perception in our environments comes from the X-10 system as discussed in Appendix A and the Argus sensor networks discussed in Appendix B.

3.1.3 Reasoning

Information regarding the current environmental state space and an understanding of the character of the observed data perceived coupled with a good representation of the behavioral activities of an inhabitant in an environment can be used for making decisions about future actions that follow the inhabitant model. These future actions may be automated by the system. The inhabitant model is difficult to hand generate, so automatic methods for deriving the model from observed behavior can provide a valuable contribution. A model focused on dynamic inhabitants must also be able to adapt to model changes over time to maintain the value of the model. A good model and representation can support proper reasoning and action execution in an environment.

3.1.3.1 Character of an Intelligent Environment

The sensing and control capabilities of our intelligent environments fit into the generalized models of any sensed and controlled system. The sensors, and for that matter all objects, in our environments are designated with a zone-number combination (e.g., A1) for uniqueness. In our environments, there is a one-to-one correspondence between state and action (e.g., an

action such as turning on a light produces the state change of that light being on). The state of objects in these environments is determined by the last action performed upon the object such as the state of a lamp being on corresponding directly to the action of turning it on and not some other mechanism. In some domains, states and actions do not correspond directly (e.g., in a robot domain a specific positional state is related to the action of motion through motor control which is not always consistent with achieving the same state since motor control can lead to other robot positions). This one-to-one correspondence between state and action is an attribute shared by many systems such as the click-stream of a computer graphical management environment and interaction of users with desktop elements, but certainly not all. Inhabitant interaction in our environments produces a constant stream of time-ordered data.

Environments of this nature provide significant challenges. The largest involves the curse of dimensionality [94]. The state space of an intelligent environment is enormous. For example, if we were to examine a very small environment with ten motion sensors and five lights for a total of fifteen objects and each of these objects has only two states (they are binary) that would give us $2^{15} = 32,768$ unique states. If we can reason about each one for 0.01 seconds it would take 5.46 minutes to make a decision. The environments in this dissertation have state spaces closer to the size of 2^{150} or 1.43×10^{45} unique states. The size of the problem space makes it difficult to develop real-time reasoning for intelligent environments.

The second largest problem is the curse of generalization [156]. Most approaches to state space reduction involve generalization techniques that reduce the state spaces into similar groups; however, in the intelligent environment domain where inhabitants are involved in specific local activities generalizing will often produce undesirable results. For example, if an inhabitant reads, listens to music, and watches television all in the same room, and the commonality between these events is that the same light is on in the room, all that a generalized approach will provide is an automation of that light, missing the desired automation of the CD player and television as appropriate. The challenge is to develop a solution that can maintain

a small state space for macro reasoning, but still maintains the details for micro reasoning and automation.

3.1.3.2 Representing the Inhabitant

The assumptions we make are that people *are* creatures of habit and will provide some periodicity and/or frequency to a number of activities they perform in any given environment, that these patterns can be observed through sensor perception, and that these patterns can be represented as Markov chains. We support this as a valid assumption since any and all observed events occurring in our environments appear as a time-ordered sequence of observable state changes which can always be represented as a Markov chain. The underlying mechanisms may not be understood or complete since we may not observe all possible patterns, but this representation can capture all observed state changes and is sufficient for our work. The base pattern representation of a Markov chain in our work represents a certain identifiable pattern of activity or *episode*.

Episodes may be abstracted into higher-level episodes that represent a grouping of related episode activity. It should be noted that we will violate the Markov property in this work by adding in a limited amount of history. The Markov property states that “the transition probabilities from any given state depend only on the state and not on the previous history [175].” In order to distinguish pattern permutations when building hierarchies we add history to the transitions to determine the correct transition probabilities—this will be discussed in further detail in upcoming sections.



Figure 3.2. Watching television Markov chain.

As a basic example, Figure 3.2 shows a typical Markov chain of inhabitant activity, namely the pattern of watching television. Patterns similar to watching television such as listening to music and reading a book can be grouped together because they occur in the same space in an environment. Furthermore, activities that occur on that side of the house could be grouped together and eventually all activities in a house fall under the root node as shown in Figure 3.3.

This location-based hierarchical decomposition of activities illustrates the type of information we are trying to learn about the inhabitants of an environment—how they utilize the environment. The model influenced by location is typical of human partitioned state spaces, but in this work we seek to learn the structure automatically through observation. Hierarchical decomposition of the Markov activity model will be guided by how the inhabitant interacts in the environment. In other words, the hierarchies we learn are based upon observed patterns so that if the inhabitant eats then watches television followed by a period of sleep then those activities are more likely to be grouped together because at a higher level they form a pattern. Regardless of the decomposition policy, in the intelligent environment domain the root node indicates events that occur in the purview of that particular intelligent environment.

In the real world the current state of our environments is never fully understood. We can make observations and infer about the general state of the environment, but the environment is still only partially observable—we cannot observe what takes place in peoples' minds, in the ductwork, behind the couch, inside the television, and so forth. In our environments what we actually learn are Hidden Markov Models (HMMs). HMMs still describe a process that goes through a series of states, but each state has a probability distribution of possible transitions [175]. Each state also represents a perceived observation that encapsulates many possibly unseen events that are *hidden* from the observer. In our work, we also depart from the traditional state-based chains of the Markov model which typically represent the entire world state in favor of an event-based chain, one in which the world state is represented only by the

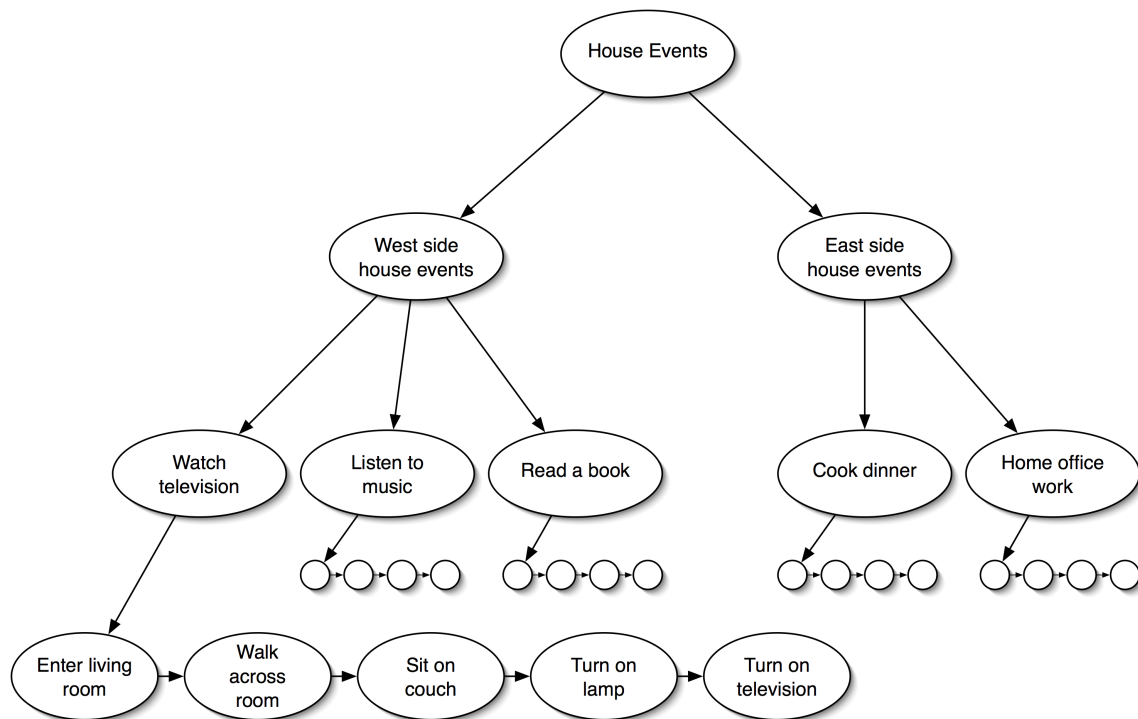


Figure 3.3. Basic hierarchical Markov model.

single change observed at that point in time. Since our model focus is on a single inhabitant, we concentrate on the changes made to the world state by that individual and assume that the rest of the world has not changed. Our model as such represents a chain of events where each event represents the observation that we make at a given point. Each event encapsulates all of the hidden acts that may occur as well. Our approach will also build HMMs into a hierarchy, called a Hierarchical Hidden Markov Model (HHMM). If you tie actions and rewards to the transitions between states this model becomes known as a Hierarchical Partially-observable Markov Decision Process (HPOMDP) which will be discussed in further detail in upcoming sections. This work will focus on single inhabitants or multiple inhabitants treated as a single grouped inhabitant—this is a single inhabitant based system.

3.1.3.3 Learning from Observation

The core challenge of this work is to learn an inhabitant model solely from observation. The learned model can only utilize data from the perception of the environment and designed mechanisms for converting that data into a useful knowledge representation. The model should be computationally tractable, accurately reflect the interactivity patterns of the inhabitant, and provide for the accurate and efficient automation of the environment.

3.1.3.4 Interactive Life-long Learning

Learning past the initial model is the secondary challenge. Automation systems for intelligent environments are only useful in the real world if they can adapt to the ever changing lifestyles of the inhabitants to whom they cater. The system should accommodate for both a slow drift in patterns and for dramatic shifts. The system should adapt quickly while minimizing the loss of accuracy and efficiency. The goal is to provide for the life-long adaptation of the system with the inhabitant of the environment.

3.1.4 Actuation

Automation can only occur when the systems can actuate objects in the real world in order to affect state and therefore affect perception. The type and number will be determined by off-the-shelf availability and environmental needs. This work will focus on lights, fans, limited appliances, and mini-blind control actuation systems mostly through X-10 based control as discussed in Appendix A.

3.2 Solving the Problem

Data streams from the sensors of the environment into software components which contain algorithms that learn, predict, and generate knowledge. These components are controlled

by a single entity that seeks to organize structures that model the inhabitant of the target environment in an effort to maintain their safety and security and to cater to their needs by automating the mundane tasks in their lives. These structures must also be modified over time to adjust for the lifestyle changes of the inhabitant they model in the given environment.

3.2.1 The Approach

There are three distinct phases to our approach. The first phase as shown in Figure 3.4 is the *Knowledge Discovery and Initial Learning* phase which involves the decision-maker utilizing the data-miner to produce hierarchical knowledge of inhabitant activity patterns, creating a model, and training the prediction and episode membership algorithms. In that flow diagram, solid lines represent the flow of processing control and data in the direction of the arrow, dashed lines represent the transformation of data into different forms from the source to the product following the direction of the arrow. The second phase as shown in Figure 3.5 is the *Operational* phase which involves observing the event stream and providing current observation data and then receiving next observation and membership probability information from the predictor and episode membership algorithms in order to form a belief state in the inhabitant model. This information is used to potentially make an automation decision. The rules engine is constantly running during this phase. The third phase as shown in Figure 3.6 is the *Adaptation and Continued Learning* phase which involves feedback from the rules engine to adjust the transition probabilities in the model to improve performance, monitoring of system performance, and the monitoring of data-mined inhabitant activity patterns to observe shifts in the inhabitant's activities. Together this system is designed to learn a model of the inhabitants of the intelligent environment, automate devices to the fullest extent possible using this model in order to maximize the comfort of the inhabitant while maintaining safety and security, and adapt this model over time to accommodate shifts and drifts in the inhabitant's life patterns.

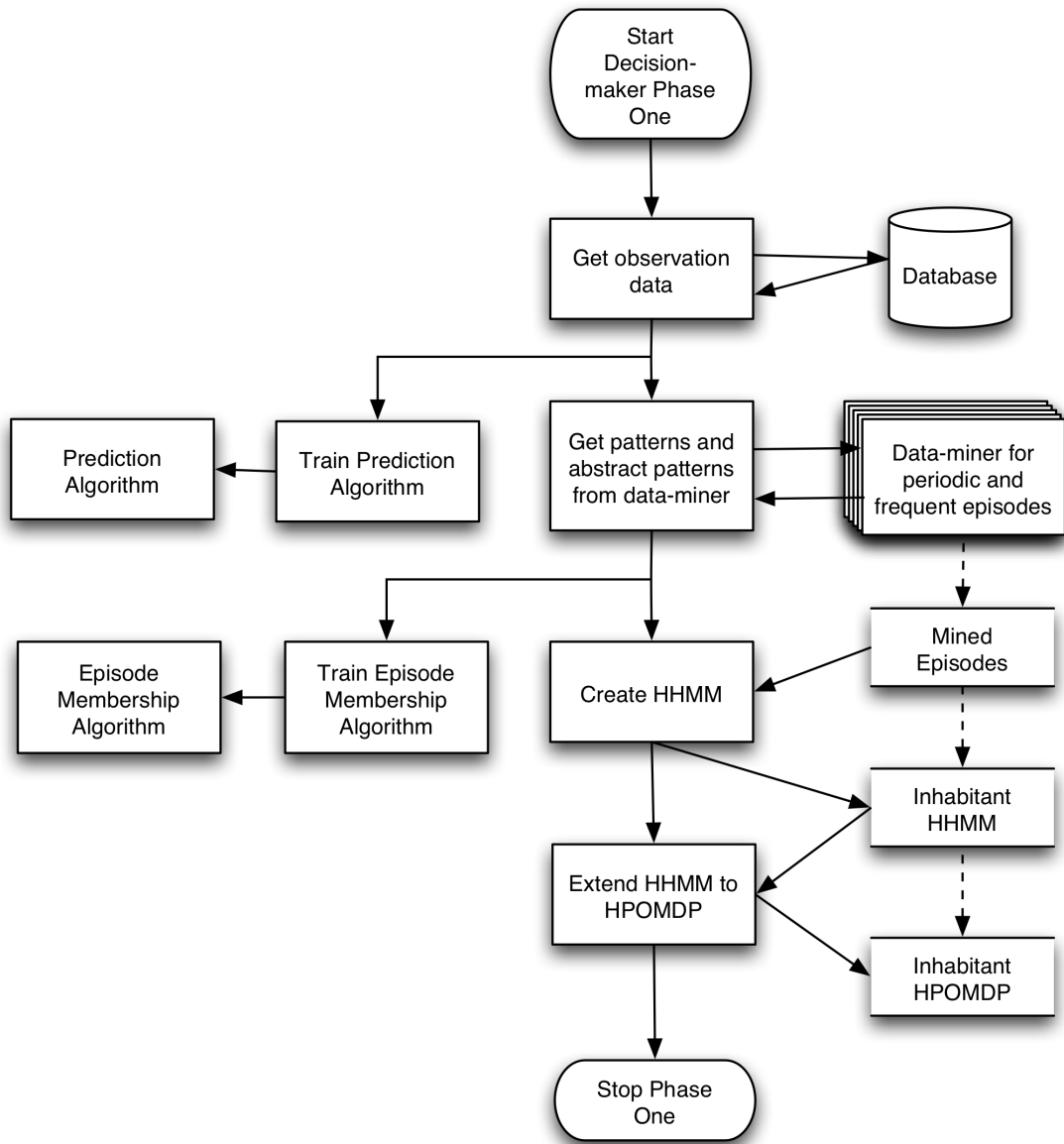


Figure 3.4. Phase 1: Knowledge discovery and initial learning phase.

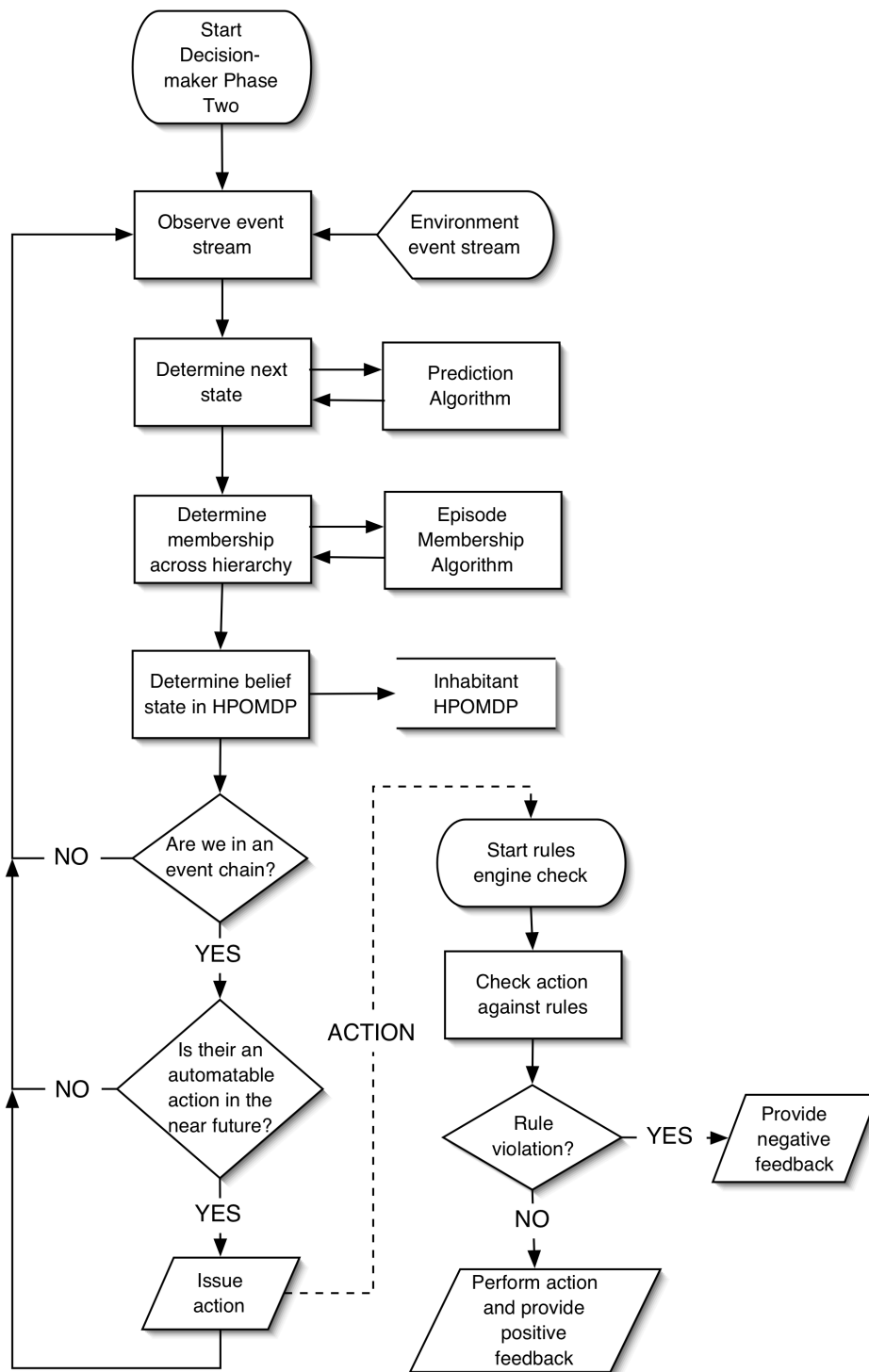


Figure 3.5. Phase 2: Operational phase.

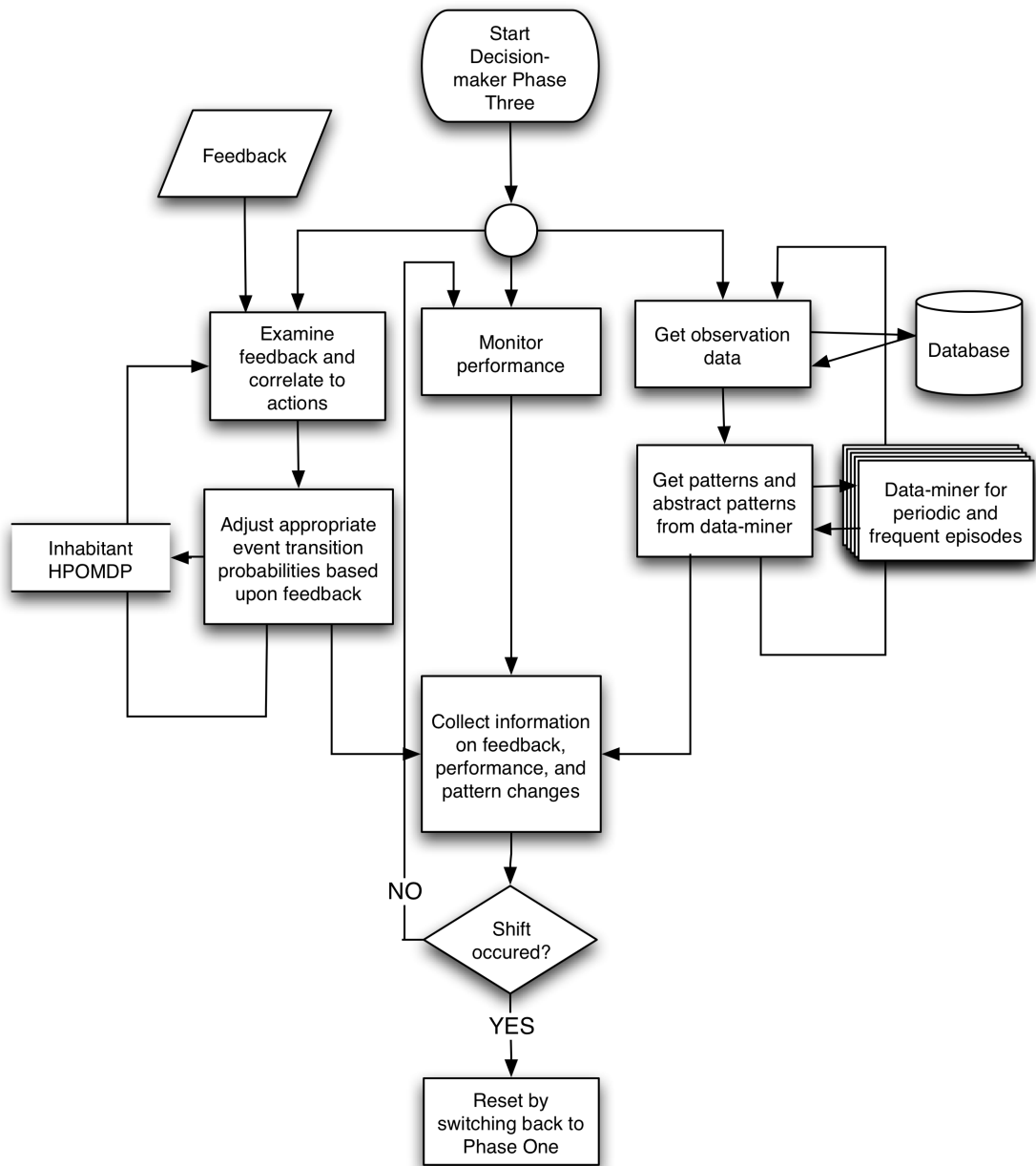


Figure 3.6. Phase 3: Adaptation and continued learning phase.

3.2.2 Data-mining

The better the quality of information, the better the model, and the better the control policy. Central to our approach is the necessity to recognize the Markov chain patterns of the

life of an inhabitant in one of our environments and to recognize the patterns of the abstract patterns, comprising a sequence of the low-level patterns—all from observation data. The large quantities of observed data and the desire to extract the patterns from it have led us to the data-mining community. If we could employ a data-mining technique to discover the periodic and frequent episodes of behavioral patterns in the data, we could use that knowledge to build our inhabitant models. We utilize the work by Ed Heierman in his Episode Discovery (ED) technique [75] as a tool for extracting the desired knowledge from the data stream.

3.2.3 Belief State Support

If a data-mining technique can generate knowledge to create a hierarchical model, then in order to be able to use it for automation we will require information that will provide a mapping from the real world observations to the specific location within our model. The event stream coming into the system provides one clue as to which pattern we are currently observing, but may be insufficient to truly pinpoint the exact chain of current activity. What we need to develop is a belief in which state the current world is engaged in order to utilize our learned model to automate future events. Understanding what is the most probable next event to occur would assist in this belief. A prediction algorithm trained on our observation data sets and with reasonable accuracy could be used to provide this type of information. In addition, since our approach is to create hierarchical layers that are labeled as groups of events, there will always be a probability of membership given an observation data stream to these groupings. An algorithm that produces a probability of membership given the current event stream to learned groups would provide information that could narrow the choices of specific patterns in current practice and improve the belief of which state the system is currently engaged. The combination of the current event stream, a membership probability across the hierarchical layers, and a prediction of the next event to occur yield a belief state of where in the derived model the current inhabitant is interacting. If we look ahead in the model we can determine events that

will occur in the near future, and if these events are within the control of the system it can issue actions to automate them.

3.2.4 A Rules Engine

Invariably, there will be events that escape periodic or frequent patterns, but are desired items for automation. The notion of encoding safety, security, and user preferences into the system is also important to our system goals. In order to accommodate these needs, we are employing the use of a rules engine that will maintain a knowledge base of user preference, safety, and security rules and constraints. These rules would incorporate knowledge such as not opening the mini-blinds at night or turning on exhaust fans at high humidity levels. It can also accommodate user preference that could specify rules such as to not automate particular items perhaps because it is their favorite lamp or they just do not feel comfortable with the automation for a particular device. These rules can also be used to incorporate desired events outside the realm of normal observation by the system. For example, patterns that cannot be performed by the inhabitants such as turning off all of the lights when the inhabitant leaves the environment can be encoded as a rule.

Since our goal is to learn how to automate the intelligent environment, the rules engine can also serve as a feedback mechanism. Whenever a rule is violated or fires, feedback can be given to the learning mechanisms of the decision-making component to incorporate into its knowledge for the future. Ideally, the decision-maker would learn not to violate the safety and security rules and automate the inhabitant-designed rules as well.

3.2.5 Decision-making

The decision-maker is the core control policy component. Our approach is to develop an overall control algorithm in a three-phase system. The first phase will extract the appropriate observation data from a database and control the data-mining algorithm in order to find

patterns and patterns of patterns to build a hierarchical hidden Markov model. This HHMM will be extended with actions and rewards to form a HPOMDP model of the inhabitant for the environment under evaluation. The observation data will also be used to train a prediction algorithm. The observation data and data-mined patterns will be used to train an episode membership algorithm. After the initial information is processed, the model is derived, and useful components are trained, we can move into the next phase.

The second phase involves the operational use of the components under the direction of the decision-maker to automate the environment. The decision-maker takes the incoming data stream and provides the information to the predictor and the episode membership algorithms to receive a predicted state and membership probabilities. Based upon the current event, the recent history, the next state prediction, and the probabilities of membership the decision-maker will develop a belief state of where in the learned HPOMDP model the inhabitant's activities are currently engaged. If the belief is strong enough and exists in a series of non-abstract events (i.e., there is sufficient evidence and probability that current observations are part of a known low-level Markov chain), then the decision-maker will look ahead and make an action decision if one exists. These action decisions automate the environment. While the second phase continues to automate the environment, as feedback is returned from the rules engine and the inhabitant interacting in the environment, we enter the third phase.

The third phase involves adaptation and learning by the decision-maker altering the transition probabilities between events based on feedback in order improve automation performance. These local changes to the model accommodate for minor changes in the activity patterns of the inhabitant over time. The decision-maker will also continue to periodically reexamine the historical data using the data-mining tool to determine if new patterns are emerging with the goal of detecting shifts in the patterns. Large lifestyle changes in the inhabitant may lead to a breaking of the current model. In order to accommodate such shifts the decision-maker must evaluate performance and pattern change information in order to contemplate potential

reset of the entire system in order to accommodate a major change in the inhabitant's patterns. These three phases are designed to initiate, operate, and maintain a system for the automation of the intelligent environment.

3.3 The System Framework

Given the problem and our chosen approach, it is important to develop a framework in which to support our work. Our system framework is designed of modular components and open source software. Modularity is chosen over a monolithic system to promote ease of maintenance and replacement. The architecture is designed to allow components to be swappable, potentially even hot-swappable, in order to create a robust and adaptive system. We present the framework first in a functional abstract view and then in a detailed concrete form.

3.3.1 Abstract View

The system framework shown in Figure 3.7 consists of four cooperating layers. Starting at the bottom, the *Physical* layer contains the hardware within the environment. This includes all physical components such as sensors, actuators, network equipment, and computers. The *Communication* layer is available to all layers to facilitate communication and service discovery between components. The communication layer includes the operating system, device drivers, low-level component interfaces, device proxies, and middleware. The *Information* layer gathers, stores, and generates knowledge useful for decision making. The information layer contains prediction components, databases, user interfaces, data mining components, resource utilization information providers, and high-level aggregators of low-level interfaces (e.g., combined sensor or actuator interfaces). The *Decision* layer takes in information, learns from stored information, makes decisions on actions to automate in the environment, deter-

mines if faults occur and correlates them back to responsible components, and develops policies while checking for safety and security.

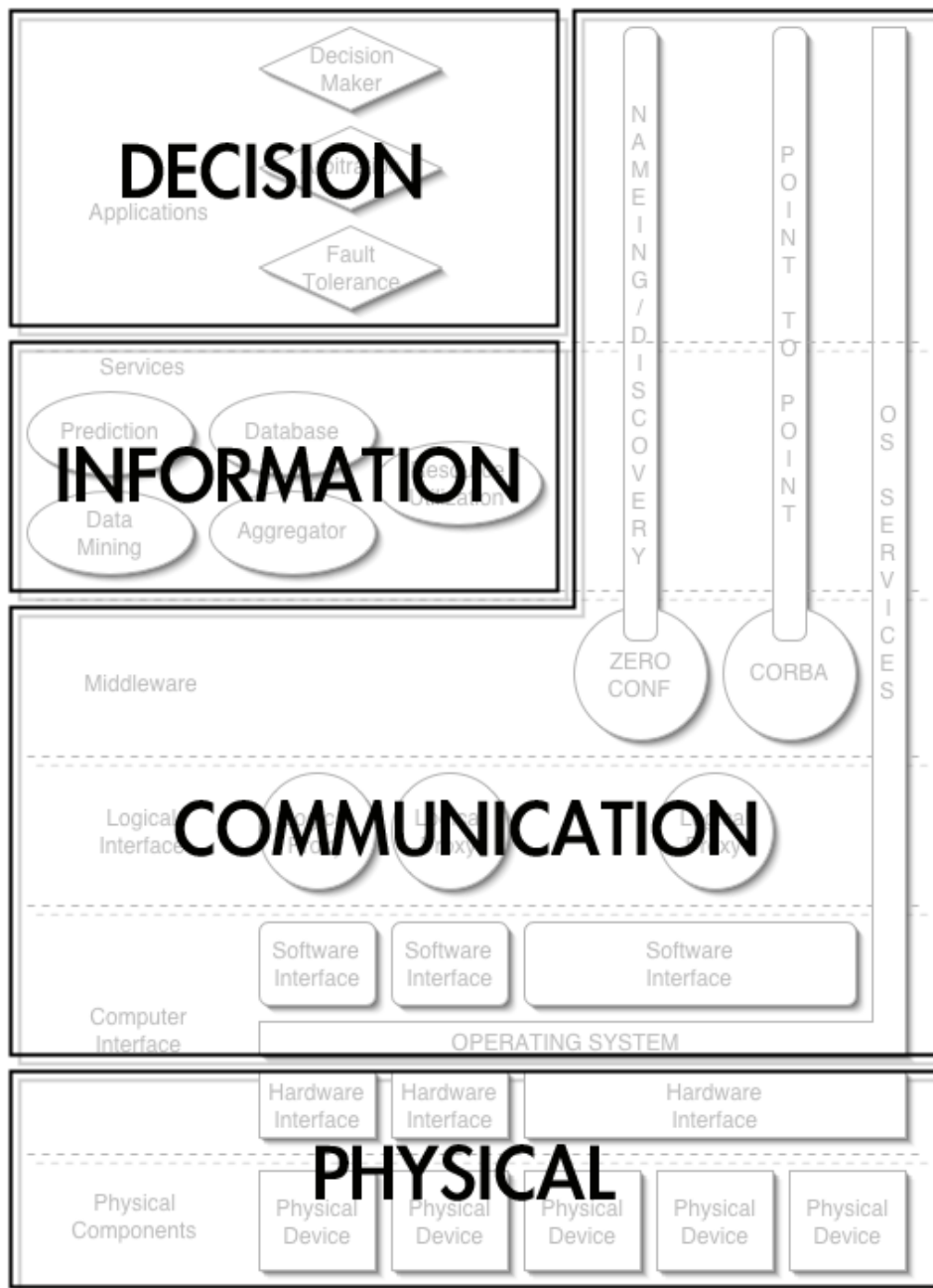


Figure 3.7. Abstract framework.

Perception is a bottom-up process. Sensors monitor the environment and make information available through the communication layer to information layer components. The database stores this information while other information components process the raw information into more useful knowledge (e.g., predictions, abstractions). New information is presented to the decision layer components upon request or arrangement. The decision layer uses learned experience, observations, and derived knowledge to select an action (which may be empty). The decision is checked for safety and security concerns and, if allowed, signals the beginning of action execution. Action execution flows top-down. The decision action is communicated to the information layer which records the action and communicates it to the physical layer. The physical layer performs the action, thus changing the state of the world and triggering a new perception. The process repeats *ad infinitum* with periodic retraining of the decision layer components, policy development, database archiving, and component maintenance.

3.3.2 Concrete View

The abstract layers of the system framework are realized through a set of concrete functional layers. These concrete layers are shown with components in Figure 3.8. The base layer is the *Physical Components* layer which consists of all real devices utilized in the system. These devices include powerline control interface hardware, sensor networks, input devices, cameras, and so forth, with the exception of the computer with which equipment is interfaced. The physical computer(s) and associated network this system resides on is considered the host of all layers above the physical. The *Computer Interface* layer contains the hardware interfaces to physical devices (e.g., PCI card interfaces, USB, Firewire), device drivers to utilize the hardware, the operating system of the computer, and all software interfaces that provide services or APIs for hardware access. It should be noted that since all components of above layers reside and utilize operating system services, these services are shown to extend to all layers. In the *Logical Interface* layer, the hardware device services and APIs are utilized to create simple,

light-weight programs that create a series of atomic services around each sensor and effector in the system. These *logical proxies* provide information and control via socket and shared memory based interfaces in a modular design. All of the lower layers are based on simple single application components, but in higher layers the components become more complex. The *Middleware* layer provides valuable services to the upper layers of the architecture to facilitate communication and service discovery. The MavHome architecture specifies middleware that provides both point-to-point (done through CORBA) and publish-subscribe (done through multicast messaging utilizing OS socket services and the IP stack) types of communication and naming/service discovery provisions. The *Services* layer utilizes the middleware layer to gather information from lower layers and provide information to system applications above. Services either store information, generate knowledge, aggregate lower-level components, or provide some value-added non-decision making computational function or feature (e.g., user interfaces). The *Applications* layer is where learning and decision-making components operate. Not all components that appear in Figure 3.8 are utilized in this research, but are used throughout the MavHome project in general.

3.3.3 Implementation

To provide the reader with a better understanding of the system framework we employ, we will discuss some implementation-specific details. More information on our systems and environments can be found in the appendices.

Lighting control is the most prominent effector in most intelligent environments. We currently use X-10-based devices in the form of lamp and appliance modules to control all lights and appliances. The CM-11A interface is used to connect computers to the power system to control the devices. Radio-frequency based transmitters (in remote control form factor) and receivers are also used for device interaction. X-10 was chosen because of its availability and

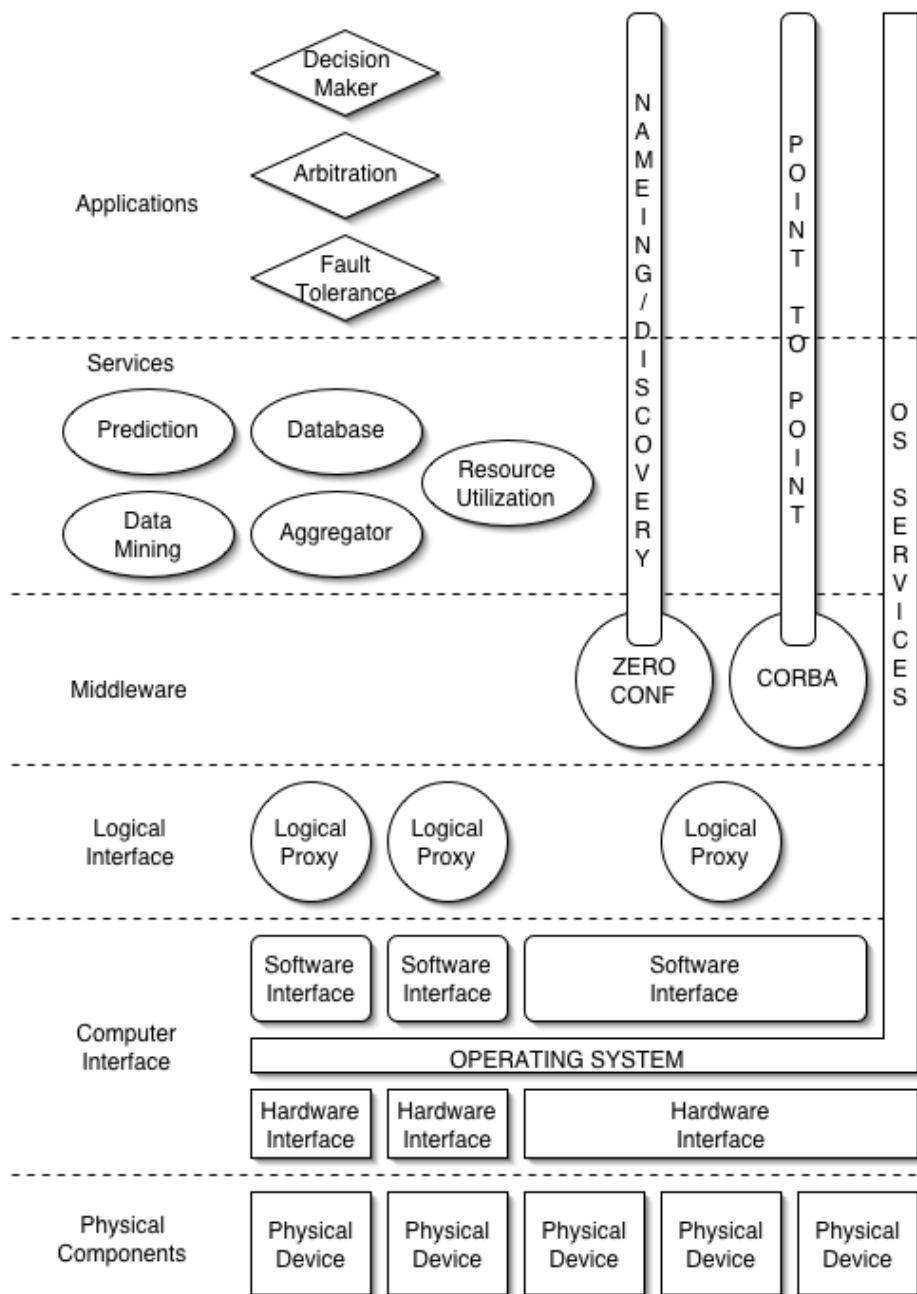


Figure 3.8. Concrete framework.

low price. Many home users also utilize X-10 technology, so immediate benefits to the current home user are possible. Refer to Appendix A for more information on X-10.

Perception through light, humidity, temperature, smoke, gas, motion, and switches is performed through a sensor network we developed. The Argus network system is a PIC16F877-based system comprised of a master board that interfaces to the computer via a serial interface and connects up to 100 slave boards that host up to 64 sensors each, ganged in groups of four on a sensor dongle. Special masters have also been developed for high speed digital and mixed digital/analog sensing applications. A stepper-motor master has also been developed to control up to four mini-blinds. Refer to Appendix B for more information on the Argus networks.

A key element in perception is inhabitant localization. The Argus Digital Master is used in conjunction with passive infrared (PIR) sensors placed on the ceiling in traffic areas to detect motion. The sensors have a 60° field of view and are placed between eight and ten feet from the ground depending on the height of the ceiling. In order to reduce the sensing area, tubes are placed over the sensors to reduce the floor footprint to a three to four foot sensing circle. Tests in our environments show a consistent single inhabitant location detection rate of 95% or better accuracy. Multiple inhabitant studies will require augmenting technology, so our focus is on single inhabitants.

All system framework components interface through either serial, USB, or firewire interfaces. The system framework and components have been developed on Intel-based PCs (Pentium 4) and use the Linux operating system (SuSE 9.1).

The logical interfaces for all X-10 and Argus-based components have been written as light-weight configurable modules. The proxies maintain the current state of each device and provide a mechanism for reading and, if applicable, control. The communication protocols for X-10 devices and Argus components are well defined and interface availability is advertised through zero configuration (ZeroConf [87]) technology.

Components desiring to find X-10 or Argus components merely need to perform a link-local query for devices that follow the defined MavHome X-10 and Argus protocols and a list of available devices will be presented to the requester. Contact information is returned to the

requester to allow connection to the logical proxy. Through this mechanism no configuration is required and the system is very adaptive and dynamic. New proxies advertise their availability and older ones remove theirs before they shut down. We have had a high level of success using ZeroConf technology with very few problems once the components were developed. When we were using a CORBA name server we had close to a 50% component communication or discovery failure rate at any given time.

The system framework uses two main middleware packages. Communication between high level components is performed using the Common Object Request Broker Architecture (CORBA) due to the clarity of interface design provided by the Interface Description Language (IDL), ease of integration, maturity and stability of the technology, and object-oriented design compatible with our C++ implemented components. Zero configuration technologies are used for replacing the CORBA naming service and providing service discovery. They are provided by the Apple Multicast DNS responder and adherence to the ZeroConf standard.

Implemented services include a PostgreSQL database that stores information, user interfaces, prediction components, data mining components, and logical proxy aggregators (e.g., the projector screen aggregator that takes simple “up” or “down” commands to coordinate the efforts of a timed control of three switches to place the screen in the proper position). Resource utilization services monitor current utility consumption rates and provide usage estimates and consumption queries, but are not used in this work.

The core of this work resides at the application layer which along with some of the services comprise the core system architecture of this approach. Other applications such as a fault tolerance correlation application as shown in Figure 3.8 exist, but are not currently used in this work.

3.4 The System Architecture

Inside the system framework exists the core system architecture for our approach. In section 3.2 we outlined the components we utilize in our work and place those in a framework in section 3.3. At this point the exact mechanisms have remained nameless in order to provide a better understanding through an overview path that will continue to present the ideas, approach, and design in increasing detail. We will now present the specific core architectural elements we utilize that will be completely analyzed in the next chapter.

3.4.1 ProPHeT

Decision making is performed in the ProPHeT (**P**roviding **P**artially-observable **H**ierarchical (HMM/POMDP) based **d**ecision **T**asks) component. The world representation at this level is the Hierarchical Hidden Markov Model (HHMM) [54] based upon a hierarchy of episodes of activity mined from stored observations. Episode Discovery (ED) is used to generate low-level episode Markov chains and build the hierarchy of abstract episodes under the direction of ProPHeT. Learning is performed by extending the HHMM to a hierarchical Partially Observable Markov Decision Process (HPOMDP) and applying temporal-difference learning. Constant feedback from ARBITER is used for continuous learning using TD(0) reinforcement learning [203]. Action decisions are made by using the incoming event stream, recent history, the stream episode membership features of Episode Membership (Epi-M) to provide input into the current belief state in the model, and the Active LeZi (ALZ) prediction of the next event to chose the appropriate transitional action.

3.4.2 Episode Discovery (ED)

The Episode Discovery (ED) data-mining algorithm [73] discovers interesting patterns in a time-ordered data stream. ED processes a time-ordered sequence, discovers the interesting episodes that exist within the sequence as an unordered collection, and records the unique

occurrences of the discovered patterns. Because the framework is capable of processing the interactions incrementally, it can be used as part of a real-time system. These features make ED a suitable algorithm for mining an intelligent environment data stream.

Our approach to state space reduction from the large number of potential environment observations is to abstract inhabitant activity to episodes that represent the current task of involvement. Given the inhabitant task episode, observations not related to the task can be pruned. A difficult problem is how to discover these episodes. After discovering the episodes, it is also desirable to be able to classify streamed observations to episodes in real time.

We use ED to find inhabitant episodes in the collected data and for episode classification of streamed observations. ED mines device activity streams trying to discover clusters of interactions that are closely related in time. Significance testing is performed on discovered clusters to generate sets of significant episodes based on the frequency of occurrence, length, and regularity. Further processing using the Minimum Description Length (MDL) principle [171] and greedy selection produces sets of significant episodes. These are labeled and directly correspond to an inhabitant task.

When an inhabitant is first introduced to an intelligent environment, no automation should occur for an initial observation period. This allows the building of a database of potential episodes of normal task activity. This is inhabitant centric and the observation period duration is determined by data compressibility which is used to determine the stability of the data with relation to episode discovery. A stable, consistent data compression as reported by ED indicates an end to the initial observation period. Identification of concept drift and shift is performed by continued monitoring of streaming data and compressibility. Changes in compressibility indicate a need to reevaluate the discovered episodes.

Episode discovery, classification, and identification are utilized to reduce the state space of an intelligent environment to a set of inhabitant-centric tasks. Thus, the MavHome architecture is inhabitant-centric.

3.4.3 Active LeZi (ALZ)

An intelligent environment must be able to acquire and apply knowledge about its inhabitants in order to adapt to the inhabitants and meet the goals of comfort and efficiency. These capabilities rely upon effective prediction algorithms. Given a prediction of inhabitant activities, MavHome can decide whether or not to automate the activity or even find a way to improve the activity to meet the system goals.

Specifically, the MavHome system needs to predict the inhabitant's next action in order to automate selected repetitive tasks for the inhabitant. The system will need to make this prediction based only on previously-seen inhabitant interaction with various devices. It is essential that the number of prediction errors be kept to a minimum—not only would it be annoying for the inhabitant to reverse system decisions, but prediction errors can lead to excessive resource consumption. Another desirable characteristic of a prediction algorithm is that predictions be delivered in real time without resorting to an offline prediction scheme.

MavHome uses the *Active-LeZi* algorithm (ALZ) [61] to meet our prediction requirements. ALZ is a predictor based on text compression methods. It has been well-investigated that good compression methods are also good predictors, and according to Information Theory, a predictor that builds a model whose entropy approaches that of the source achieves greater predictive accuracy. It has also been shown that a predictor with an order that grows at a rate approximating the entropy rate of the source is an optimal predictor. ALZ is based upon these characteristics. By characterizing inhabitant-device interaction as a Markov chain of events, we utilize a sequential prediction scheme that has been shown to be optimal in terms of predictive accuracy. Active-LeZi is also inherently an online algorithm, since it is based on the incremental LZ78 data compression algorithm.

3.4.4 Episode Membership (Epi-M)

Effective utilization of the derived HHMM/HPOMDP-based inhabitant model requires an understanding of how to map the current observation stream into the derived abstractions. Episode Membership (Epi-M) performs this function by using the information learned from Episode Discovery to build internal correlation tables and further augment those tables with time-based occurrence information based on circular probability capture from the same data stream. Data stream observation over the specified time window supplied to ED can be used to generate match probabilities with the episode sets over each layer of abstraction. Augmenting the probability with the likelihood of occurrence based on the observed occurrence time distribution for each of the discovered episodes with relation to the current time further improves the accuracy of possible episode membership reporting. For example, if the current observation stream matches with 90% probability either reading a book or the pattern of sleeping on the couch, but the inhabitant has never slept on the couch at this time of day, then the probability of sleeping can be discounted to promote reading as the most probable episode of membership. Epi-M output is used by ProPHeT to determine belief state in the operational phase for the current event observations.

3.4.5 ARBITER

When issues of safety and security are of the highest importance in a system there is the need for an enforcer of rules before actions are made. This system works by using a knowledge base of rules and evaluating each action event against these rules to determine if the action violates them. Actions in violation will be prevented from occurring and feedback will be sent back to the originating system (i.e., the decision-maker). Rules are not required to be just of a safety and security type, any type of rule can be used in order to guide the behavior of the system. Cases where system behaviors are desired but will never be trained by streaming

data or interactions can be handled by the addition of rules to provide feedback and facilitate learning of the desired behavior.

Before an action is executed it is checked against the policies in the policy engine, ARBITER (**A Rule-Based InitialTor of Efficient Resolutions**). These policies contain designed safety and security knowledge and inhabitant standing rules. Through the policy engine the system is prevented from engaging in erroneous actions that may perform such activities as turning the heater to 120° F or from violating the inhabitant's stated wishes (e.g., a standing rule to never turn off the inhabitant's night light).

ProPHeT performance depends on ARBITER preventing the violation of rules and relaying feedback information so ProPHeT can learn. When ARBITER performs correctly it can improve ProPHeT performance through proper feedback and prevent incorrect ProPHeT decisions from being executed. Conversely, if ARBITER performs incorrectly, despite correct actions from ProPHeT, bad rules could prevent helpful automations and degrade system performance.

These components work in concert to learn, adapt, and automate the inhabitants' lives in an intelligent environment.

3.4.6 The Core

The core of this work lies in the data-mining–decision-making–belief–rule/feedback chain or in what we call the EPBA chain comprised of the Episode Discovery (ED), ProPHeT, belief through Active LeZi and Episode Membership (Epi-M), and the ARBITER components.

Information and action flow through the system according to the three main system phases. These phases as previously stated can be restated as *Initialization*, *Operation*, and *Adaptation*.

The operation and adaptation phases occur simultaneously until interrupted by the adaptation phase, usually to return through the initialization phase. All components in blue with a

bold, solid border in the architecture Figures 3.9, 3.10, and 3.11 represent components that are developed as part of this work, yellow items with a dashed border are components developed on the MavHome project but were developed by others, and components in other colors with thin borders represent data or storage.

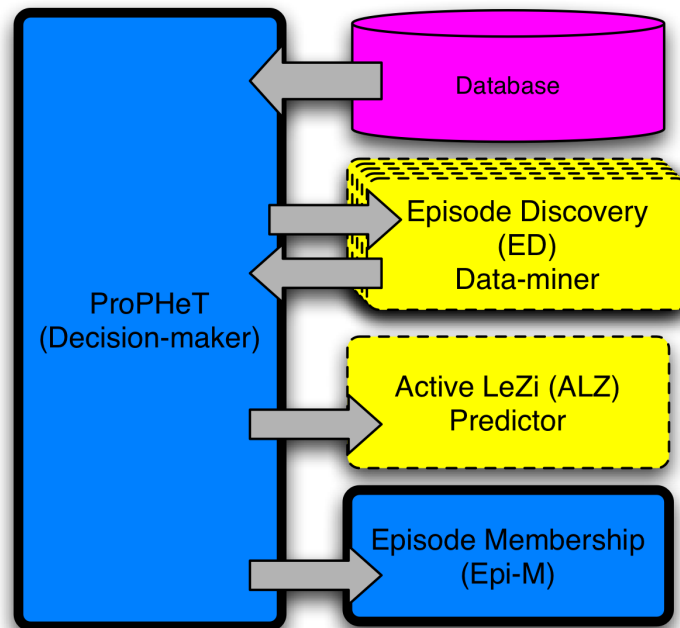


Figure 3.9. Core system architecture in initialization phase.

The structure of the core components in the initialization phase and the flow of information are shown in Figure 3.9. During initialization information flows from the database through ProPHeT into ED. As many instances of ED as are necessary to process the data into a hierarchy of discovered patterns are started by ProPHeT. An instance of ED is needed for each successive layer in the hierarchy from the lowest productions nodes through each layer of higher abstraction until reaching the root node when no further abstract patterns can be found. The multiple instances of ED return information on the discovered patterns to ProPHeT. ProPHeT then trains

ALZ over the same observation data from the database. ProPHeT determines what information to pull from the database based on available observation data and an observation of episode discovery and compression over time (discussed in Section 4.3.1) and performs the necessary data conversions and filtering for each component to accept the data and perform computation within a reasonable amount of time. After ALZ, ProPHeT trains Epi-M with the same observation data and the hierarchical data returned from ED. After deriving the data from the target observation data set and training the belief supporting components, ProPHeT generates the HHMM and subsequent HPOMDP models.

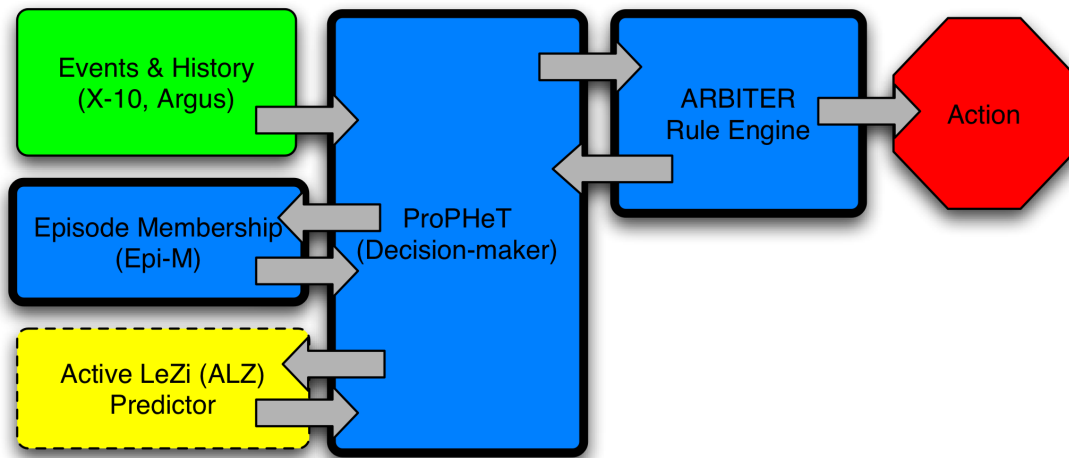


Figure 3.10. Core system architecture in operation phase.

The structure of the core components in the operation phase and the flow of information are shown in Figure 3.10. During operation information flows from events generated in the environment and perceived by the logical proxies and presented to ProPHeT. ProPHeT relays these events to ALZ and Epi-M and receives prediction and membership information. Based upon the incoming event, history, prediction, membership probabilities, and the HPOMDP model an action decision may be made. Any action decisions, or at a minimum the current

event, will flow through to ARBITER to be checked for rule violations. If a satisfactory action is to be performed, ARBITER will contact the appropriate logical proxy to initiate the action.

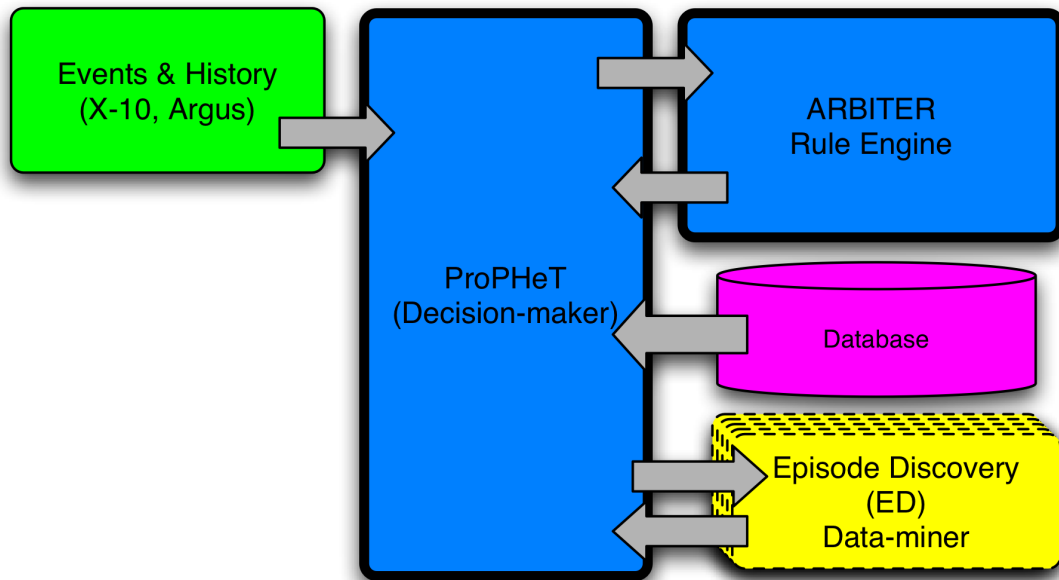


Figure 3.11. Core system architecture in adaptation phase.

The structure of the core components in the adaptation phase and the flow of information are shown in Figure 3.11. During adaptation information flows from the event stream through ProPHeT to ARBITER, often accompanied with an action. Rule violations and any other feedback are relayed back to ProPHeT from ARBITER including inhabitant feedback correlation to countermanded automation. Action correlation is performed through observation in a tunable time window. Each system action is recorded with the label of the node within the HPOMDP that initiated it. If the system performs an action and the action is undone by observing the incoming environmental data stream, then feedback is correlated to the label of the node within the HPOMDP that caused the incorrect action. ProPHeT uses feedback from ARBITER to adjust the HPOMDP structure to improve performance and accommodate for pattern drift. In-

ternally ProPHeT is evaluating performance based on feedback and usage. Information also flows into and out of ED as ProPHet will periodically evaluate the continuously-growing observation database—it should be noted that in our system the database component has an event listener that logs all events into the database—for changes in patterns and hierarchy (the mechanism is to be discussed in the next chapter) in order to detect pattern shift. ProPHeT will decide based upon performance as well as indications of pattern shift and drift whether a reset of the decision-maker is required. The reset is essentially a reboot of the system using a new set of observations—one that will hopefully better fit the inhabitant.

The framework and architecture presented in this chapter are fully expanded and explained in detail in the next chapter. This approach and system is applied to the MavPad and MavLab environments in order to automate inhabitant activities.

This is only one possible approach to developing a solution given the problem and goals. There are many other possible approaches, some of which we discuss in chapter 2, but others remain to be explored. We are exploring this method because it had not been previously explored in this domain. We present this work to the scientific community in order to further knowledge and understanding.

3.5 The Environments under Examination

This work uses two real environments and their simulated counterparts. The MavPad is an on-campus apartment, and the MavLab is the workplace of the researchers of this project. ResiSim is an in-house developed “residential simulator” for interactive simulation of intelligent environments.

3.5.1 The MavPad

The MavPad is an on-campus apartment located at the University Village on The University of Texas at Arlington campus and sponsored by the UTA College of Engineering that hosts

a full-time student occupant that participates in the project allowing us to learn about them and automate their life in accordance with approved UTA IRB Protocol # 04.136. The MavPad hosts automation capability through 25 X-10 controllers (three fans, thirteen lights, one HVAC unit, and seven electrical outlets) and two ArgusM mini-blind control systems. Sensing capability is provided by the ArgusMS and ArgusD systems that provide eighteen light, eleven temperature, four humidity, four leak detection, four door open/close, three window open/close, two seat occupancy, four HVAC vent position, two smoke detectors, two CO detectors, and 36 motion sensors. The MavPad has been operational for over a year and has hosted three inhabitants. For specific information on the MavPad refer to Appendix C. Information on X-10 can be found in Appendix A and Argus sensor network information can be found in Appendix B.

3.5.2 The MavLab

The MavLab is the project name for the Artificial Intelligence Lab (a.k.a., Learning and Planning Lab) located in Nedderman Hall at The University of Texas at Arlington which is base of operations for this research. The MavLab is a workspace setting with offices, cubicles, a break area (MavKitchen), a lounge (MavDen), and a conference room. The MavLab hosts automation capability through 54 X-10 controllers (49 light, five appliances, a projection screen) and fourteen ArgusM mini-blind control systems. Sensing capability is provided by the ArgusMS and ArgusD systems that provide 36 light, ten temperature, three humidity, two door open/close, six seat occupancy, and 25 motion sensors. MavLab has been in various operational states for the last two years. For specific information on the MavLab refer to Appendix D.

3.5.3 ResiSim

It is important that this work be well grounded in reality and in dealing with the real world, especially with the goals being directed to the intelligent environment domain. This

research work is aimed at real-time decision-making in the real-world. However, it is not always feasible to study interactions solely in the real world. A tool for interactive simulation and data visualization was needed. Since this is a relatively new area of study there are not any available simulation tools designed for intelligent environments or home simulation, so the development of a tool became an important issue for performing this type of work.

ResiSim is a residential simulation environment tool. It is designed to provide a simulation environment for any indoor environment where people would typically spend time. It features Markov model-based virtual inhabitants that interact with the environment and can react to changes in that simulation. It can also provide real inhabitant data playback and limited interaction. As an evaluation tool it can track automation and its source whether from an external system, playback, or elsewhere.

ResiSim is comprised of three basic parts. The first part is the logical proxies of the environment itself. ResiSim environments are comprised of individual Zeroconf-enabled objects that represent either real-world, simulated, or hybrid objects that include the actual environment itself as an object. It is based upon the theory that in the future all objects will have some level of self-awareness and controllability (even if just to provide limited self-information). The second part is the server that collects all of the individual objects and assembles the environment. The server controls the time and interactions of the simulation and coordinates and reports all changes to attached clients. The server provides a 2D overview display and an interface GUI. The third part is the client which provides a 3D interactive interface with the environment. This application aims to provide a photo-realistic experience in either interactive mode where the user can interact with the environment or in observer mode where the user can watch the interactions of others (either live or playback). ResiSim uses information learned from the development, utilization, and interaction with our real environments to accurately simulate or emulate the real world. Simulation of this fidelity is invaluable to understanding

the environments, sensors, inhabitant interactions, and ability of systems to interact with an intelligent environment. For more information on ResiSim refer to Appendix E.

CHAPTER 4

DETAILED METHODOLOGY

I believe I can see the future, because I repeat the same routine.

—Trent Reznor, *Every Day is Exactly the Same*

In the last chapter, we covered the architectural view of this work leaving out many details. The purpose of this chapter is to walk through those details of the core system architecture in a data flow driven manner. Throughout this chapter, we will use a running example as we describe and discuss the specific methods employed in our approach to intelligent environment automation. This example involves a virtual inhabitant, MavHome Steve, who works in the MavLab.

MavHome Steve, the virtual researcher, has a routine for his work in the MavLab. This routine contains six unique patterns. These patterns include 1) *lab entry to desk* as shown in Figure 4.1, 2) *going on break* as shown in Figure 4.2, 3) *going off break* as shown in Figure 4.3, 4) *go to alternate workstation* as shown in Figure 4.4, 5) *return from alternate workstation* as shown in Figure 4.5, and 6) *leave lab from desk* as shown in Figure 4.6. These patterns occur with a probability defined by the Hierarchical Hidden Markov Model shown in Figure 4.7 in which no production nodes are presented to simplify this figure and to allow it to be viewed on a single page. The production nodes are presented in the individual pattern figures. In the HHMM figure, each oval represents an abstract node and the square an end node. Probabilistic transitions are shown as solid arrows with the probability of transition labeled near the head of each arrow. Automatic transitions are shown as dashed arrows and show a transition that is always taken once reaching a node—always an end node in HHMMs. Using ResiSim (see

Appendix E), we simulate MavHome Steve's actions in the MavLab over time to produce data and to interact with Steve with our system to automate his routine. Steve's patterns are designed to be simple and straightforward to provide a baseline understanding and measure of performance for our approach.

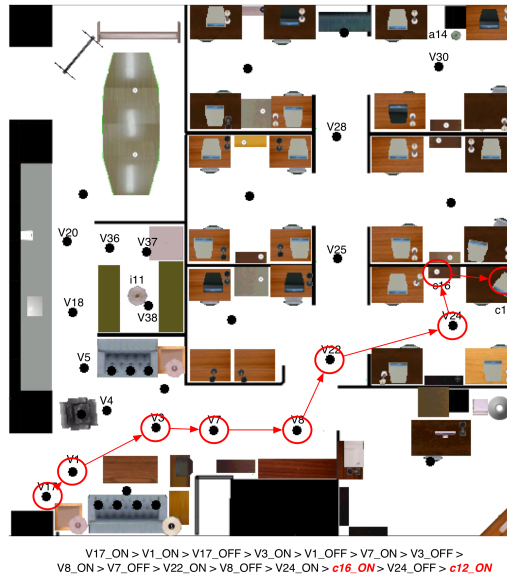


Figure 4.1. MavHome Steve Pattern 1: Lab entry to desk.

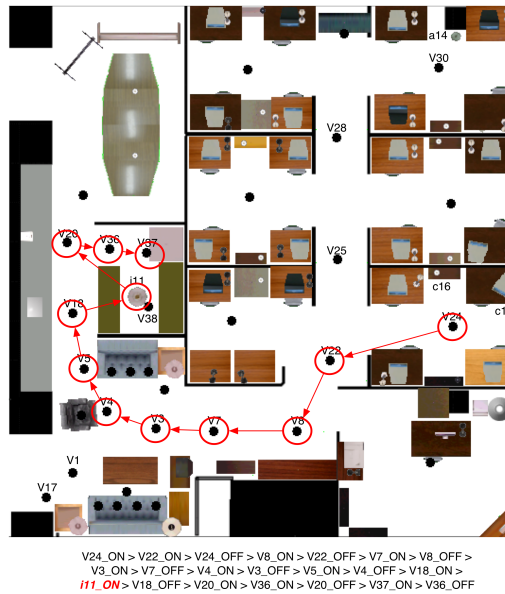


Figure 4.2. MavHome Steve Pattern 2: Going on break.

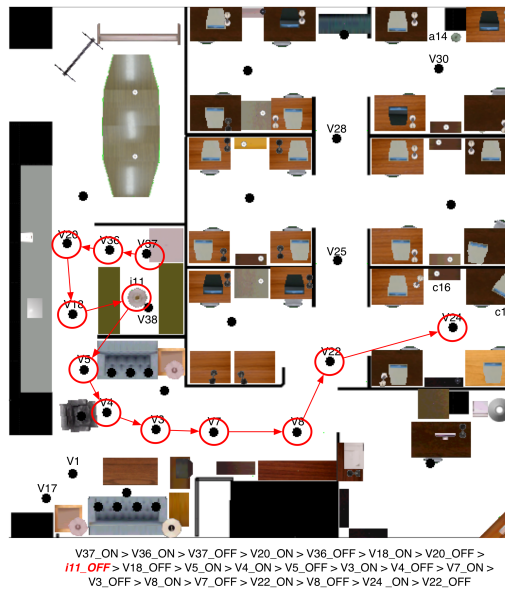


Figure 4.3. MavHome Steve Pattern 3: Going off break.

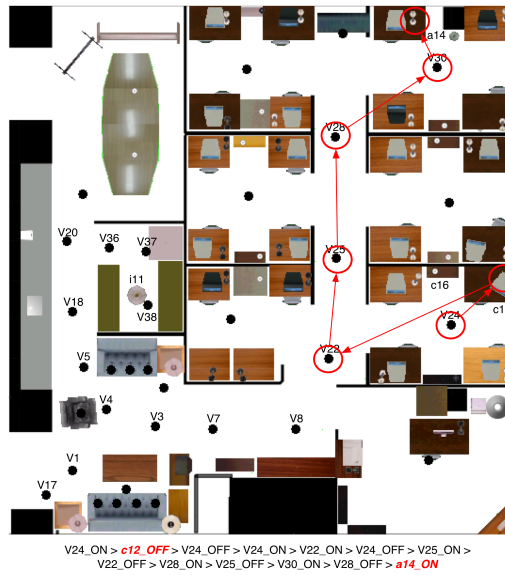


Figure 4.4. MavHome Steve Pattern 4: Go to alternate workstation.

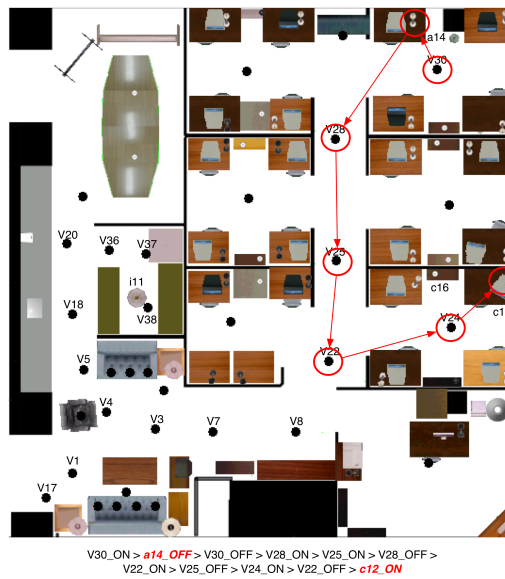


Figure 4.5. MavHome Steve Pattern 5: Return from alternate workstation.

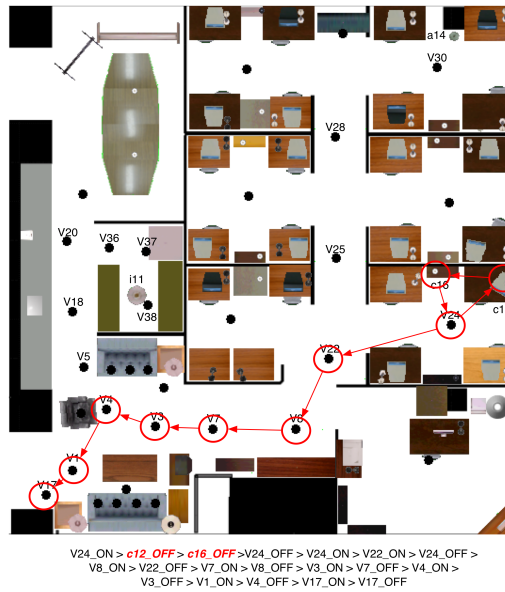


Figure 4.6. MavHome Steve Pattern 6: Leave lab from desk.

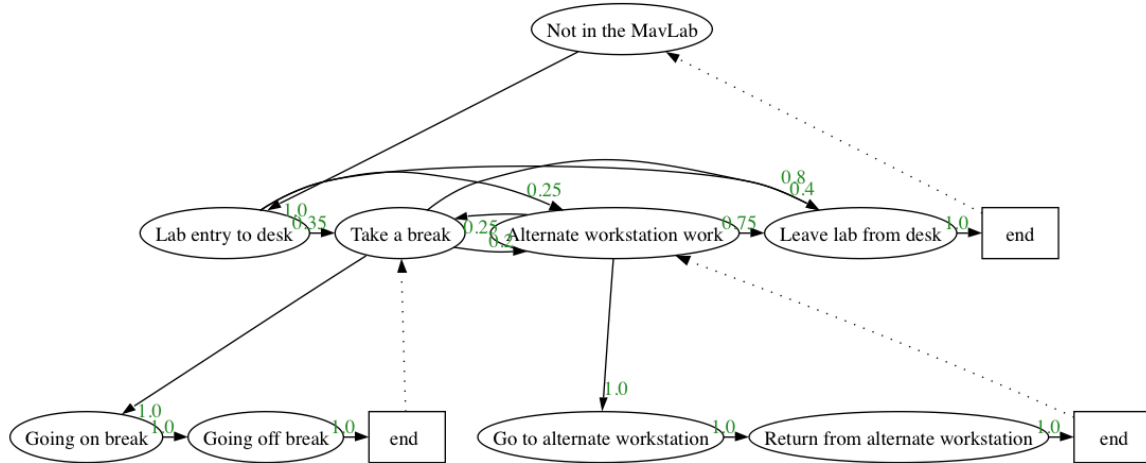


Figure 4.7. MavHome Steve HHMM structure—abstract nodes only.

The sensors starting with ‘V’ are motion sensors or door sensors. The other sensors are X-10 units. A description of the sensors can be found in Appendices A, B, and D. Items

listed in the Markov chains that are in *red bold italics* are *automatable* actions (i.e., the system can control those items and set them in the state indicated). Figure 4.8 presents an expanded abstract node showing the production node chain including an automatable production node. Each object is designated with a unique letter-number identification (e.g., v24) followed by an underscore and a state (e.g., ‘_OFF’). The automatable actions are the ones that should be controlled by our system. The Markov chains represented in these example patterns are representative of those in our sensed environments and are based on actual observed Markov chains for these actions in the MavLab. The only differences between simulated and actual patterns is the variability of the timings between events, the purity of the pattern (i.e., sometimes humans will slightly deviate from an exact pattern, but are intent on the same goal), and sensor noise. We do introduce some variability in timings, but order is preserved. Noise can be injected, but is specifically mentioned when applied.

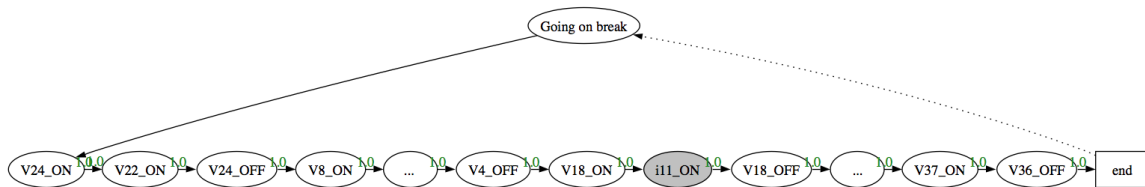


Figure 4.8. MavHome Steve HHMM structure pattern 2 abstract node expansion showing the production nodes and the automatable node in grey-fill.

4.1 The Data and Flow

The sensing and control capabilities of our intelligent environments fit into the generalized models of any sensed and controlled system. In our environments, there is a one-to-one correspondence between state and action (e.g., an action such as turning on a light produces the state change of that light being on). Inhabitant interaction in these environments produces a constant stream of time-ordered data. This data flows from the perception sensor logical prox-

ies as described in the last chapter to online objects (e.g., ProPHeT and ARBITER) listening to the data stream and indirectly to offline objects through storage objects (i.e., a database) as shown in Figure 4.9.

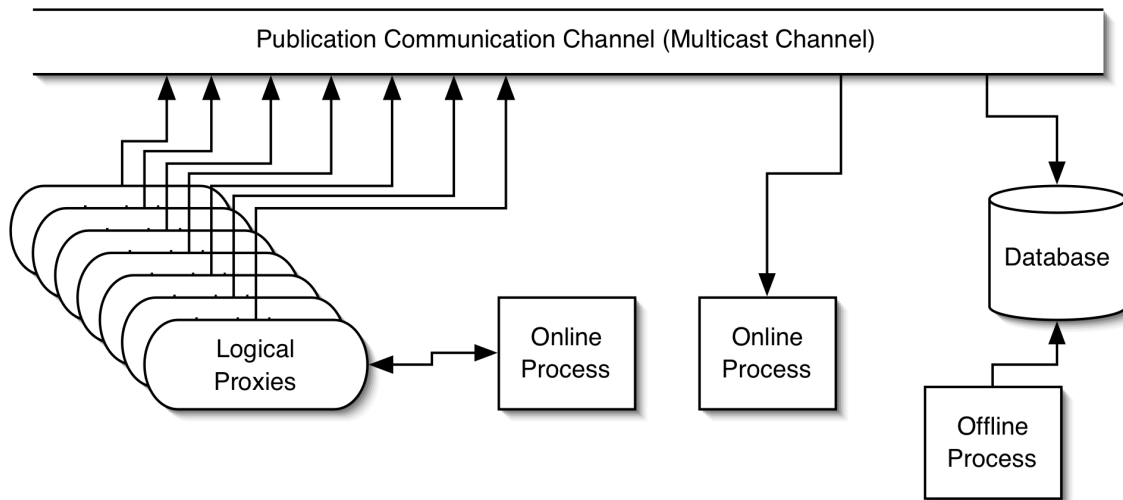


Figure 4.9. Data flow from sensors to online and offline objects.

4.1.1 The Character of Intelligent Environment Streaming Data

Data streams from the logical proxies reflecting changes in the state of the objects they represent. Our intelligent environment data streams are formally defined as a 6-tuple $\langle t, z, n, l, r, p \rangle$ where

- t is the timestamp of the event (DD-MM-YYYY HH:MM:SS)
- z is the device or sensor zone
- n is the device or sensor number
- l is the level or value of the new state
- r is the source of the event (e.g., sensor network, powerline control)
- p is the specific inhabitant initiating the event (if identifiable)

An example of streaming data captured in our database can be seen in Table 4.1. The data corresponds to the 6-tuple definition, but it also hosts an additional field *state* which is a Boolean representation of *level* where 100 = 1 and anything else is 0. We use this for convenience and that is why it appears in our data streams. Also note that the *inhabitant tag* merely states “inhabitant.” This is because we do not employ technology to distinguish inhabitants, but this field is left in the data stream for future separation of activities by inhabitant.

In analyzing common data streams in our intelligent environments, the most frequently occurring data is the sensed light, temperature, and humidity levels that are not used specifically by the focus of this research, but for other systems in the intelligent environment. However, we do use some of those sensors in combination with simple reasoning software to create other simple sensors. For example, by observing the average relative humidity over time we can determine the mean and standard deviation—a simple sensor that would indicate ‘ON’ for levels above two standard deviations could be easily employed as an indicator to trigger exhaust fans. The next frequent sensor information comes from the motion sensors followed by the reed sensors (i.e., door openings and sitting on furniture detection) which are both of interest. Interspersed in the data stream are the inhabitant interactions with objects that can be automated—objects such as lights, blinds, and appliances. Figures 4.10, 4.11, 4.12, and 4.13 provide examples of the distribution of sensor readings from inhabitant interaction. Repetition of these patterns over time (days, weeks, months, and years) makes them easier to identify, but also corresponds to the repetitive nature of human life. The inhabitant’s interactions in the environment over time create temporal patterns of their activity in the database.

Table 4.1. Data-stream from inhabitant interaction in a MavEnvironment

Timestamp	Zone	Number	State	Level	Source	Inhabitant Tag
2005-01-04 01:51:26	V	20	0	0	ArgusD	inhabitant
2005-01-04 01:51:26	i	4	0	0	X-10	inhabitant
2005-01-04 01:51:27	V	24	1	100	ArgusD	inhabitant
2005-01-04 01:51:28	V	24	0	0	ArgusD	inhabitant
2005-01-04 01:51:29	V	24	1	100	ArgusD	inhabitant
2005-01-04 01:51:31	V	24	0	0	ArgusD	inhabitant
2005-01-04 01:51:31	a	1	0	0	X-10	inhabitant
2005-01-04 01:51:31	V	20	1	100	ArgusD	inhabitant
2005-01-04 01:51:32	V	19	1	100	ArgusD	inhabitant
2005-01-04 01:51:32	V	20	0	0	ArgusD	inhabitant
2005-01-04 01:51:35	V	20	1	100	ArgusD	inhabitant

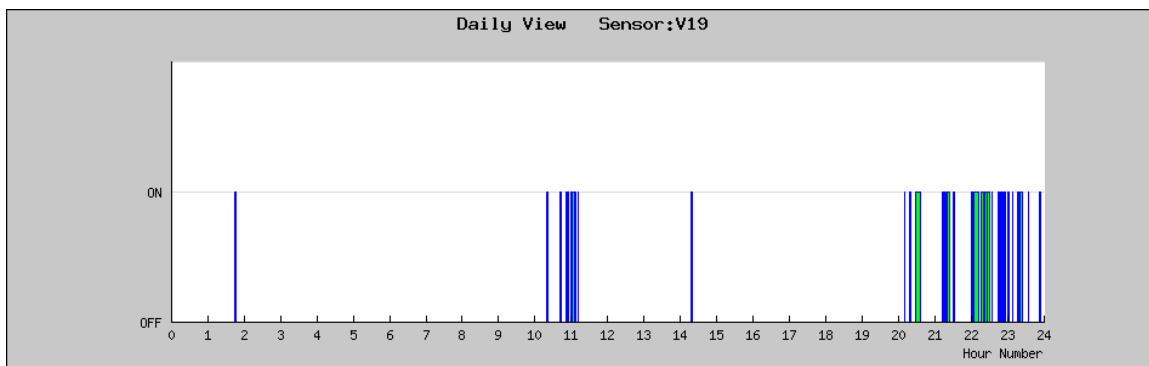


Figure 4.10. Kitchen motion over a sample day in the MavPad.

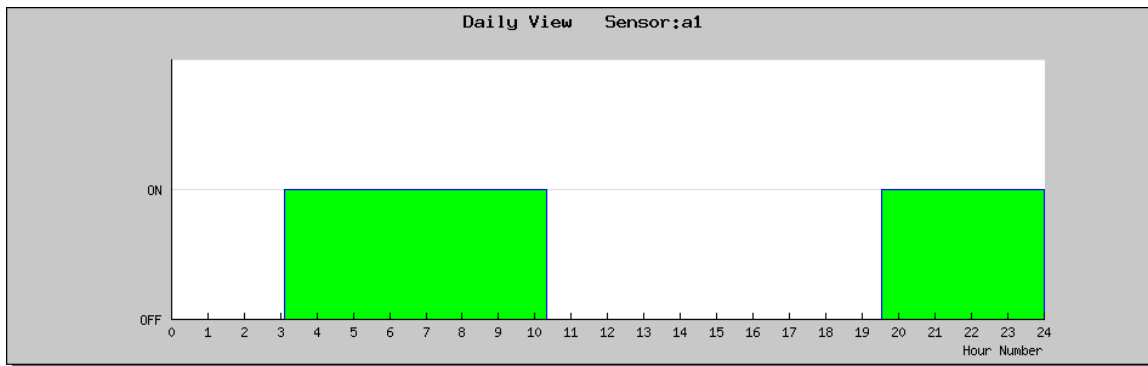


Figure 4.11. Dining room light usage over a sample day in the MavPad.

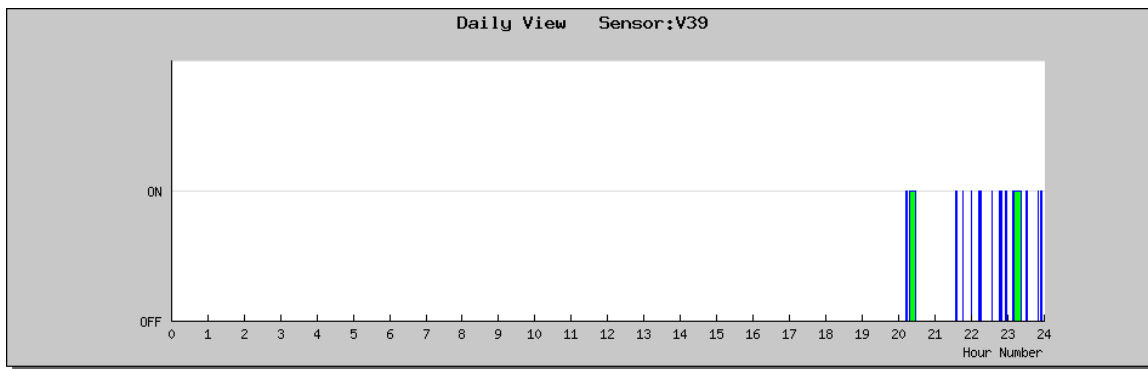


Figure 4.12. Couch usage over a sample day in the MavPad.

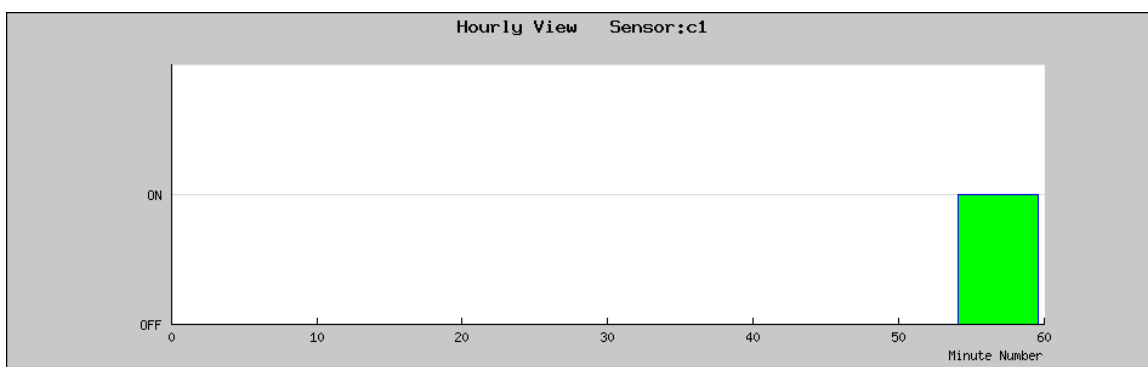


Figure 4.13. Fan usage over a sample hour in the MavPad.

4.1.2 System Data Pathways

Data flows from the observing sensors through the sensor-specific hardware into a computer using an interface driver and eventually becomes represented as state in the logical proxies of our architecture. Data flows freely through multicast broadcast from the logical proxies into ProPHeT which will pass the information on to other components and a database logger which captures all events for later recall, use, and analysis. This data flow is illustrated in Figure 4.9. ProPHeT provides information to ActiveLeZi (ALZ), Episode Discovery (ED), and Episode Membership (Epi-M) and also accesses the database for historical information. Software objects may query the state of the logical proxies for all environmental objects (i.e., sensors, switches, and so forth) as well. This is useful for determining the initial state of available objects in an environment.

ProPHeT consumes data from the online data stream and from the database. It will filter and convert data as necessary for each component to which it provides information—namely ALZ, ED, and Epi-M. ProPHeT through ARBITER will issue commands to alter the state of the environment in order to satisfy the system goals. Thus, ProPHeT-ARBITER is a producer of data. Other producers include ALZ (which generates predictions), ED (which generates significant episodes), and Epi-M (which generates episode membership probabilities).

Actions taken by the system flow back in through the observing sensors. Internally, feedback is provided by ARBITER to ProPHeT for rule violations. Externally, feedback is provided by the inhabitant through their interactions with the environment. Environmental interactions make the inhabitant a producer of information. Figure 4.14 illustrates the flow of data through the architectural components of our approach.

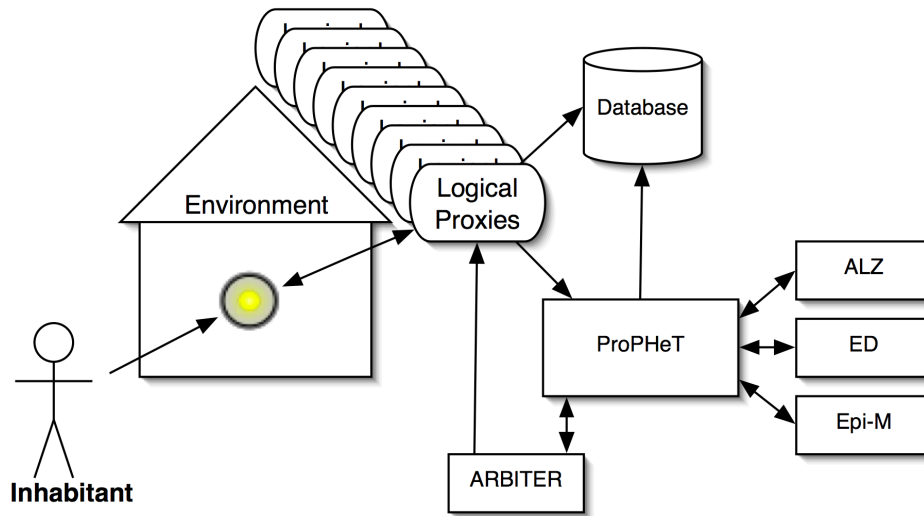


Figure 4.14. Dataflow through architectural components.

4.2 Overall Control

In the last chapter, Figures 3.4, 3.5, and 3.6 presented the knowledge discovery and initial learning phase (Phase 1), operational phase (Phase 2), and adaption and continued learning phases (Phase 3) of the control algorithm. These comprise the modes of operation for our approach in which ProPHeT is the central broker for data flow and system and information control. The algorithm for overall control flow is presented in Algorithm 1.

After the logical proxies for all perception and actuation objects in an environment have been established and the database is available for SQL interaction, ProPHeT is executed. ProPHeT begins by loading all configuration items, initializes all variables and necessary memory, and then uses Zeroconf to locate, configure, and connect to the logical proxies and database. While intelligent environmental automation is desired, ProPHeT starts with knowledge discovery and initial learning bringing ALZ, ED, and Epi-M online, training them, and preparing them for system operations. After Phase 1, Phase 2 and 3 are started concurrently. Phase 2 uses the learned HPOMDP model from ED, ALZ, and Epi-M with the incoming

stream of data to make environmental automation decisions. Simultaneously, Phase 3 is concerned with examining feedback to improve the HPOMDP, monitoring system performance, and communicating with ED to explore the conditions for a concept shift from the current set of learned concepts. Phase 2 and 3 continue until Phase 3 determines that performance has degraded below a defined threshold and that clear indications of a concept shift have occurred in which Phase 1 must be initiated (i.e., a *reboot* conducted), the system must be trained on new data, and the process starts again. The rest of this chapter will explore the three phases of operation in more detail.

Algorithm 1 ProPHeT Overall Control Algorithm

Require: MavHome logical proxies operational, database loaded and available

Ensure: Satisfaction of system goals

- 1: Load configuration file
 - 2: Initialize all variables
 - 3: Discover all logical proxies using ZEROCONF
 - 4: Discover database using ZEROCONF
 - 5: **while** environmental automation desired **do**
 - 6: Perform Phase 1: Knowledge Discovery and Initial Learning (see Algorithm 2)
 - 7: **while** reboot not required **do** {Perform these operations in parallel threads}
 - 8: Thread Perform Phase 2: Automation Decision-making Operations (see Algorithm 7)
 - 9: Thread Perform Phase 3: Adaptation and Continued Learning (see Algorithm 12)
 - 10: **end while**
 - 11: **end while**
 - 12: Cleanup and restore utilized resources
-

4.3 Training and Learning

The first step in automating our intelligent environment is to discover the inhabitant's activity patterns in order to derive a working model and to train the prediction and membership components. The results of this process affect the performance of the system, so it is important

that the best possible set of episodes be derived from the data and that the supporting systems be properly trained in order to develop as accurate a model of the inhabitant as possible.

4.3.1 ProPHeT Control

Phase 1, knowledge discovery and initial learning control, follows Algorithm 2. This algorithm begins by retrieving observation data from the database and filtering out unnecessary information. The data set to extract from the database is determined by observing the compression of the data over time as shown in Figure 4.15 and the number of interesting episodes reported by ED, as shown in Figure 4.16. The compression ratio represents the minimum description length compression compared to the original size of the data achieved by the ED-discovered model, and reflects how well the model describes the input sequence [75].

Algorithm 2 Phase 1: Knowledge Discovery and Initial Learning

Require: Database is online and available for SQL queries

Ensure: HPOMDP derived, Predictor and Episode Membership trained

- 1: Initialize all variables
 - 2: Retrieve observation data from database
 - 3: Filter data as necessary
 - 4: Format data as necessary for components
 - 5: Train prediction algorithm (Active LeZi) using data
 - 6: Perform knowledge discovery in data for patterns and abstract patterns of inhabitant activity (Episode Discovery)
 - 7: Filter interesting discovered episodes
 - 8: Format interesting discovered episodes
 - 9: Train episode membership algorithm (Episode membership) using data
 - 10: Create HHMM using interesting discovered episodes
 - 11: Extend HHMM to HPOMDP
 - 12: Cleanup and restore utilized resources
-

When ProPHeT observes a relatively consistent compression of the data (i.e., $\pm 3\%$) and number of discovered interesting episodes (i.e., episodes of activity that exhibit a periodicity

or frequency of appearance in the data stream), the duration time is established for database observation data. This evaluation is performed with a consistent window capacity c_w and time span t_w which are the two main configurable parameters of ED. Window capacity is the maximum length of the longest pattern that can be identified as an episode. Window time span is the maximum duration in seconds of the longest pattern that can be identified as an episode. These tunable parameters are used to limit the length of the patterns both in events and temporal duration. The other two configurable parameters in ED are the autocorrelation threshold β_{AC} and the interval tolerance β_t which provide control over the preciseness of the periodic and frequent information discovered [75]. Since these values are independent of the input sequence, and we achieve reasonable results from an unmodified ED, we left them set to the default values. After a level of consistent compressions and discovered episodes, the selection of proper window size and time span is determined by evaluating first the maximum compression ratio for a set of window sizes as shown in Figure 4.17 and then the maximum compression ratio for the maximum window size from the first step for a set of window time spans as shown in Figure 4.18. The window size/span (e.g., 15/300) with the maximum compression ratio indicates the settings that best represent the data and therefore will most likely yield the best set of episodes.

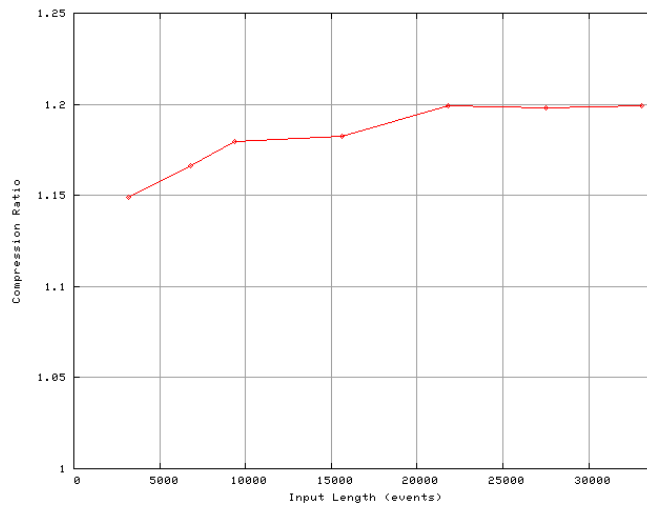


Figure 4.15. Episode Discovery data compression over time (MavHome Steve Patterns).

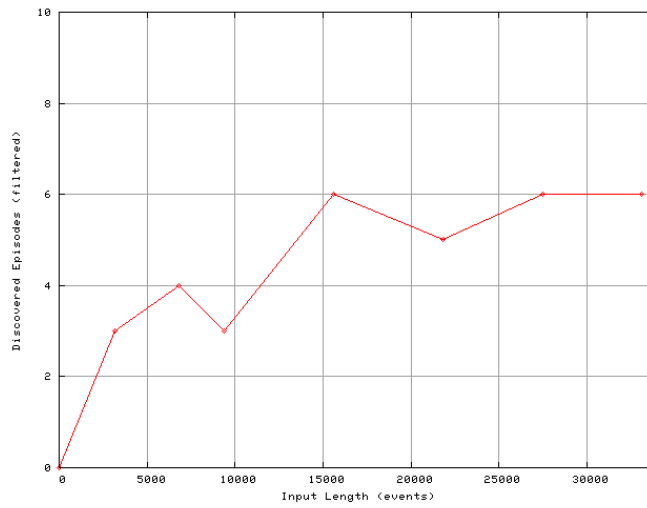


Figure 4.16. Episode Discovery interesting episode discovery over time (MavHome Steve Patterns).

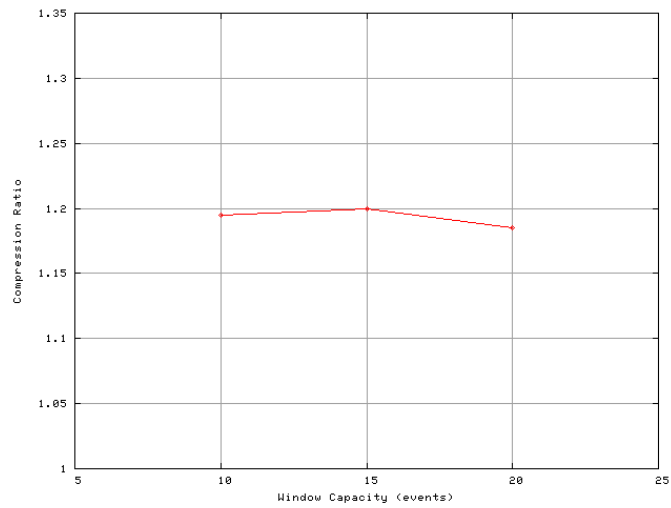


Figure 4.17. The effects of different window sizes upon the compression ratio using Episode Discovery.

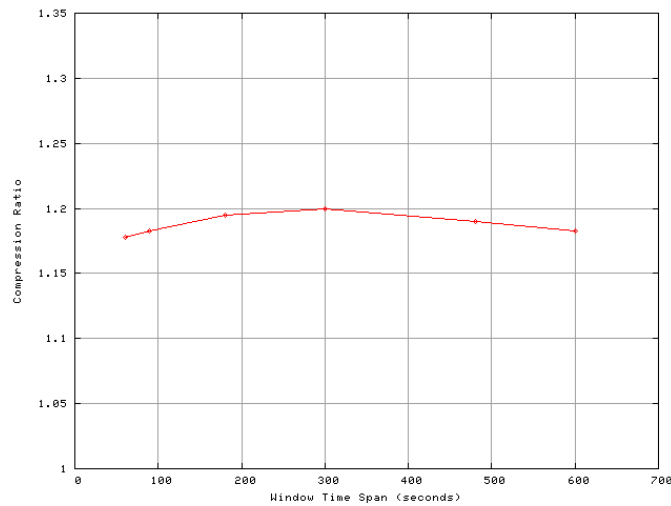


Figure 4.18. The effects of different window time spans upon the compression ratio using Episode Discovery.

Data from the database is filtered by ProPHeT to remove sensor data that is not useful for automation decisions. Data from the temperature, light level, and other such sensors that

are used in fault correlation and for other systems is removed. The data is also cleaned to remove duplicates and other anomalies if present. Despite this basic pruning, data sets in our environments average tens of thousands of sensor events daily. Since ED's complexity in the worst case is exponential, we further filter the data to isolate the patterns of interest—namely those that include automatable actions. This filtering is performed by using the learned appropriate window capacity and time span to move a sliding window over the data, removing all data that does not fall into a window which contains an automatable action. In this manner, pockets of useful information are isolated and maintained while large amounts of data not useful for automation are discarded. Note that this information may still have value, but not to our automation system. In application, the most appropriate window capacity and time span are not known *a priori*. We use a standard window capacity of 20/500 which is the maximum capacity we have found to find episodes without excessive overfit—most environment data sets peak at 15/300 as shown in Figures 4.17 and 4.18 which were taken by running a simulation based upon our example MavHome Steve activities over a period of a month. The general approach is to filter first with a less restrictive filter to prevent important information loss, and when the best window values are learned by observation of the resulting data from ED, we can narrow the filtering.

After the data set has been extracted from the database, filtered, and formatted as necessary for each component, Active LeZi is actually trained first with the data (this order is not important though), then ED processes the data to find interesting episodes. As ED discovers episodes in the data set it produces another data set of abstracted patterns with a timestamp of mean time that the pattern occurred. This data set is processed by ED to find the usage statistics of the abstract patterns. With each subsequent processing of a data set by ED another data set at a higher level of abstraction is produced. The process is repeated until no further interesting episodes are discovered. In empirical data, we have not seen more than five layers of depth to any of our learned models, but more complex data may yield deeper results. At

the production of each set of interesting episodes they are filtered to remove patterns without automatable actions since there is no value in retaining them—this improves performance by having fewer patterns to match. Upon completion of ED processing, ProPHeT combines the discovered episodes output from each layer of abstraction into a single episode discovery file. The data source file and the discovered episodes file are used to train the Episode Membership algorithm. After all the components have processed the data and have been trained, then the hierarchical models can be built which will be discussed in the Section 4.4; however, first we will explore ED, ALZ, and Epi-M in more detail.

4.3.2 Episode Discovery Learning

Inhabitant interactions can be viewed as a time-ordered sequence, and several works address the problem of discovering patterns in such sequences. Variations of the Apriori property can be used for mining sequential patterns from time-ordered transactions [4]. When the data is not transactional in nature, frequent parallel and serial episodes can be discovered by sliding a window of user-defined size over an event stream [115]. However, the important patterns in a time-ordered data stream captured from an intelligent environment may not be the ones that occur with the most frequency, but rather the ones that occur with the most regularity. Therefore, we employ a technique that evaluates patterns for frequency, size, and regularity [73]. This data-mining technique is used to find frequent and periodic repetitious patterns in the inhabitant data.

This work utilizes the dissertation work of Ed Heierman [75] as a tool for extracting the desired knowledge from a data stream. The rest of this section goes over a few details of Episode Discovery for the interested reader, for detailed information about the Episode Discovery algorithm please refer to the referenced source material.

The Episode Discovery (ED) data-mining algorithm discovers interesting patterns in a time-ordered data stream. ED processes a time-ordered sequence, discovers the interesting

episodes that exist within the sequence as an unordered collection, and records the unique occurrences of the discovered patterns. Because the framework is capable of processing the interactions incrementally, it can be used as part of a real-time system. These features make ED a suitable algorithm for mining an intelligent environment data stream.

An input stream O processed by ED consists of a time-ordered sequence of events $\{s, t\}$, where s is a symbol representing an interaction and t is a time stamp. Given an input stream O , ED follows Algorithm 3.

Algorithm 3 Episode Discovery (high-level)

Require: An input stream O of a time-ordered sequence of events $\{s, t\}$

- 1: Partition O into Maximal Episodes E_{max}
 - 2: Create Candidates C from the Maximal Episodes.
 - 3: Compute a Compression Value V for each candidate.
 - 4: Identify Interesting Episodes P by evaluating the Compression Value of the candidates
-

Each 6-tuple (see 4.1.1) of the intelligent environment data stream is transformed into an event by constructing a symbol s from the concatenation of z , n , and l , and using t as the timestamp. The other information items (i.e., source and inhabitant) are not used by ED since they provide information for filtering activities by other [future] components. Each atomic device interaction is represented by a unique symbol with this approach.

The algorithm processes the sequence of events as follows. First, ED partitions the input sequence into overlapping clusters, or episodes, of symbols that are closely related in time by collecting the events in a window. Once the window accumulates the maximum number of interactions that occur within the window time frame, the contents are converted to a maximal episode. These maximal episodes cannot currently overlap in the data stream. The maximum number of interactions accumulated by the window, as well as the time span of the window, is determined by analyzing the compression statistics computed by the algorithm. A candidate

is created to represent each unique maximal episode, and additional candidates are generated from the subsets of these candidates [74].

Next, a periodic compression value is computed for each candidate by the use of an evaluation function, based on Minimum Description Length (MDL) principles [171], that incorporates regularity (predictable periodicity, such as daily, weekly, or monthly), in addition to frequency and size. Using autocorrelation techniques, the algorithm identifies the best periodic or frequent pattern that describes the episodes represented by the candidate. A compression value is then computed based on this pattern. The larger the potential amount of compression a pattern provides, the greater the regularity demonstrated by the pattern, and the greater the impact that results from automating the pattern.

Once all of the candidates have been evaluated and sorted in descending order, the algorithm greedily identifies the episodes by selecting from the sorted list the candidate with the largest compression value. After selecting a candidate, the algorithm marks the events represented by the interesting episode. To avoid selecting overlapping candidates, the second and subsequent candidates are rejected if any of the interactions represented by the candidate are already marked. These steps are repeated until all candidates have been processed, resulting in a collection of interesting episodes. Because periodicity is evaluated, the interesting episodes that are periodic can be identified. Frequent interesting episodes are also identified.

The k uniquely-identified interesting episodes P output by ED are formally defined as a 3-tuple

$\langle \Phi, \rho, \Psi \rangle$:

- Φ is the unordered set of symbols represented by P
- ρ is the number of instances of P in the dataset
- Ψ is the set of unique ordered occurrences of P in the dataset, and is defined as a 2-tuple

$\langle \Upsilon, \varphi \rangle$:

- Υ is an ordered set of symbols

- φ is the number of occurrences of this unique pattern out of the total number of instances, ρ

The performance of ED is dependent upon the input stream quality. Input streams that contain data with little noise and clearly identifiable patterns will result in faster learning rates and better discovery of periodic and frequent patterns. In the cases shown in Figures 4.16, 4.19, and 4.20, we observe the varying learning rates of ED over similar data, but with different environments and/or inhabitants. Episode discovery performance in the intelligent environment domain appears to exhibit consistent behavior. As more input data is processed, more episodes are discovered until a plateau is reached. In the process of episode discovery, patterns are first discovered by frequency and through repetition over time they usually become identified as periodic, but because this is a consistent trend we have observed in our data processing we consider both frequent and periodic episodes to be interesting for our work. Continued processing over time will yield some variances in the numbers of discovered episodes, but after a plateau is reached we have observed the deviation to be consistently within $\pm 15\%$ for our evaluated environments. The gradual discovery as illustrated is inherent in the process, but often discovered episodes will peak and then decay. This phenomenon is due to the data processed at that point indicating frequent patterns above the reportable compression threshold of ED, but as more data is processed they disappear because they are often short lived repetitious patterns. This is advantageous in our work, because we do not desire to learn localized, short-term behaviors, but would rather focus on the consistent behaviors of inhabitant life.

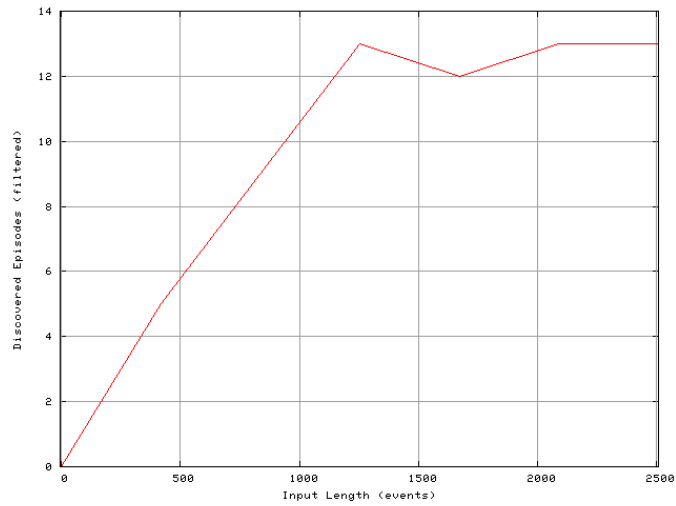


Figure 4.19. Episode Discovery interesting episode discovery over time (MavPad Inhabitant 2 Patterns).

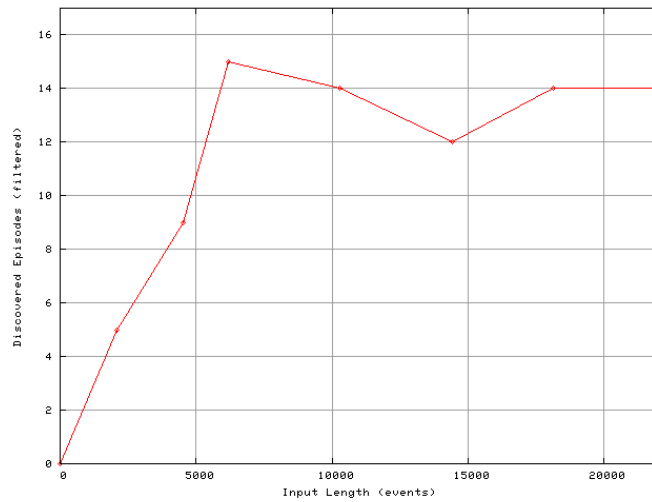


Figure 4.20. Episode Discovery interesting episode discovery over time (MavPad Inhabitant 3 Patterns).

Processing time is an issue with ED. Since it runs in polynomial time for most test cases (and is anticipated to stay within this bound for most datasets) or exponential time in the

worst case [75], ED can take an intractable amount of time to discover episodes. Figure 4.21 illustrates the impact that the size and complexity of data have upon runtime. While exhibiting polynomial runtime behavior, the amount of data and unique characteristics caused ED to take three days (255,325 seconds or 70 hours 55 minutes 25 seconds) to process a filtered seven weeks of inhabitant data. Without the filtering of ProPHeT, ED's processing time requirements would make it too costly to use. Given this information, it is important to accommodate and plan for sufficient resources and time when discovering episodes with ED.

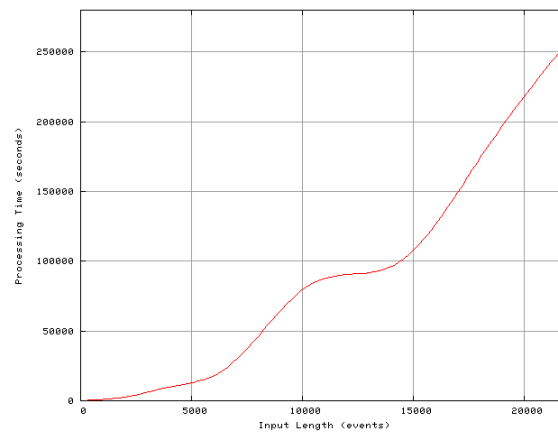


Figure 4.21. Episode Discovery processing time (MavPad Inhabitant 3 Data).

Noise in the data input stream also causes a significant challenge to ED. Figures 4.22, 4.23, and 4.24 show the effects of noise on episode discovery. The baseline discovery in a noiseless environment is shown in Figure 4.16. Noise between patterns in the data stream does not interfere with the patterns; however, noise in the patterns themselves causes the patterns to look different and thus appear inconsistent, preventing them from being frequent or periodic. As illustrated, noise as low as 20% in the data stream can completely prevent useful episode discovery. It should be noted that previously six episodes were discovered in the clean data, but with 5% noise only three were found (a 50% reduction), two with 10% noise, and only

one with 20% noise. Noise causes severe problems with the data preventing the discovery of episodes. Noise injected into the patterns causes the patterns to be split into parts where consistent sections are discovered, but the noise in the middle prevents them from being identified as complete episodes. Even when patterns are discovered in noisy sensor environments, they differ from those found in clean data environments—particularly, the patterns have a tendency to be shorter and will often force the automatable actions to the beginning of a pattern which causes a problem for actually automating the pattern. Problems with episode discovery at the production node level will affect the abstract episode discovery since the data for discovery is generated from the source data—in observation of test cases, noise usually causes a drastic decrease in depth of the layers of abstraction due to the decrease in finding episodes and the breaking of the chains of consistent observed activity on all levels. In our work we utilize ED as our data mining tool, other sequential data mining tools could be utilized such as Significant Interval Discovery (SID) [192]. However, we have found that the current version of SID incurs longer runtimes and does not find as many patterns as ED. Sequential data-mining is still a developing area of research and new discoveries in this area could benefit our work.

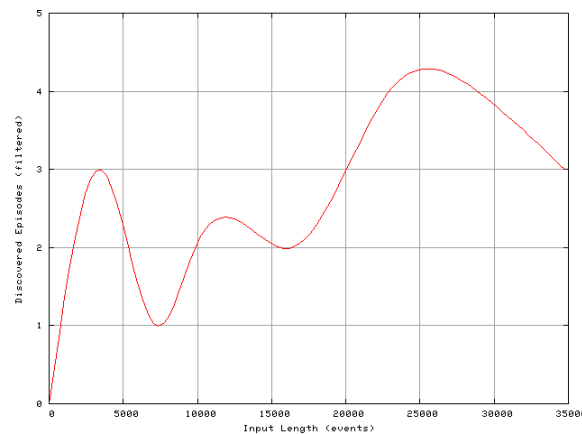


Figure 4.22. Effect of noise in the data stream on episode discover in MavHome Steve patterns using 5% random noise interference in the data input stream.

Unfortunately, the episode discovery algorithm is detrimentally susceptible to noise in the input data stream. It is important to design sensor networks and the natural flow of interaction in the environment to minimize the introduction of noise. Filtering and conditioning of the data stream can also be an effective means of preparing the data for episode discovery. The cleaner the data and the more frequent or periodic the patterns within, the better the results of episode discovery. ProPHeT applies filters to the data in order to reduce the amount of data to improve processing time and to remove noise which improves the data quality. In addition, ED's ability to abstract away noise in the discovered instances could improve performance, although this would represent lossy compression. Advanced filtering techniques and ED's ability to handle noise is an area of future work.

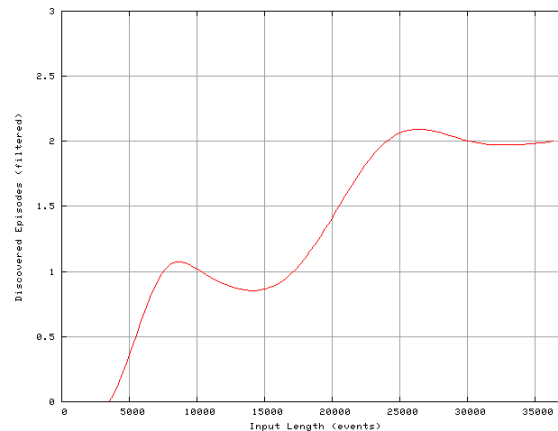


Figure 4.23. Effect of noise in the data stream on episode discover in MavHome Steve patterns using 10% random noise interference in the data input stream.

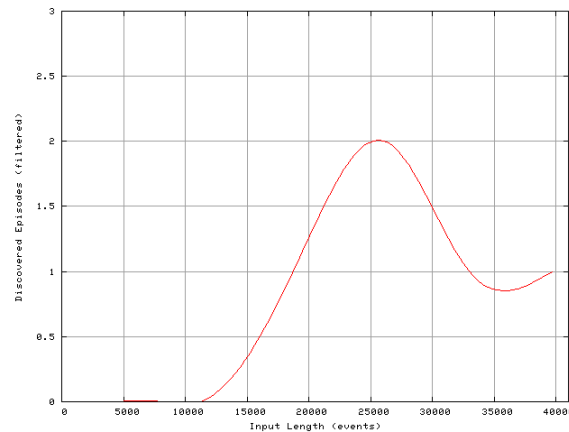


Figure 4.24. Effect of noise in the data stream on episode discover in MavHome Steve patterns using 20% random noise interference in the data input stream.

A well-designed environment with a sensor network logging inhabitant data that is filtered to reduce complexity and improve data quality can produce a data set suitable for use with the Episode Discovery algorithm. Controlled usage of ED by ProPHeT provides this data, ensures its timely processing, and the results are filtered to remove episodes without automatable actions or that contain other anomalies (e.g., too short episodes). A useful set of learned interesting episodes is then gained. Multiple iterations of data from each level of abstraction is then used to find episodes at each level of abstraction until no further episodes are discovered. All resultant episodes discovered are then combined by ProPHeT into a single file describing the episodes associated with a specific inhabitant in a specific environment over a defined observation period.

Episode Discovery was developed by Ed Heierman and Dr. Diane Cook at The University of Texas at Arlington. Interested readers should refer to the referenced material for more information on ED [73, 74, 75].

4.3.3 Active LeZi Learning

In an event-driven system there is a need to predict the next action in order provide a clear picture of the current belief state. The system will need to make this prediction based only on previously-seen interaction data. It is essential that the number of prediction errors be kept to a minimum in order to maximize the belief probability of being in the correct state. Another desirable characteristic of a prediction algorithm is that predictions be delivered in real time without resorting to an offline prediction scheme.

We use the *Active LeZi* (ALZ) algorithm [61] to meet our prediction requirements because it is a compression-based algorithm that, in accordance with Information Theory, should converge on an optimal solution as it is a predictor with an order that grows at a rate approximating the entropy rate of the source. By characterizing inhabitant-device interaction as a Markov chain of events, we utilize a sequential prediction scheme that has been shown to be optimal in terms of predictive accuracy. Active-LeZi is also inherently an online algorithm, since it is based on the incremental LZ78 data compression algorithm.

Active LeZi is an enhancement of LZ78 and *LeZi Update* [20] that incorporates a sliding window approach to address the drawbacks of these approaches. LZ78 suffers from a loss of information crossing phase boundaries in LZ parsings of an input string (possibly incurring the loss of significant patterns that cross the phase boundaries) and a slow convergence rate to the optimal predictability. *LeZi Update* addressed slow convergence by keeping track of all possible contexts within a given phrase, and not just the prefixes—leaving the phase crossing drawback open. The ALZ solution to the phase crossing drawback involves maintaining a variable length window of previously-seen symbols. They gather statistics on all possible contexts within this window and use them to build a better approximation to the order- k Markov model. ALZ uses prediction by partial-match (PPM) to combine data from different contexts sizes to make the final prediction. ALZ gains a better convergence rate to optimal predictability as well

as greater predictive accuracy because it has captured information about contexts in the input sequence that cross phrase boundaries [61].

An intelligent environment must be able to acquire and apply knowledge about its inhabitants in order to adapt to the inhabitants and meet the goals of comfort and efficiency. These capabilities rely upon effective prediction algorithms. Given a prediction of inhabitant activities, we can decide whether or not to automate the activity or even find a way to improve the activity to meet the system goals.

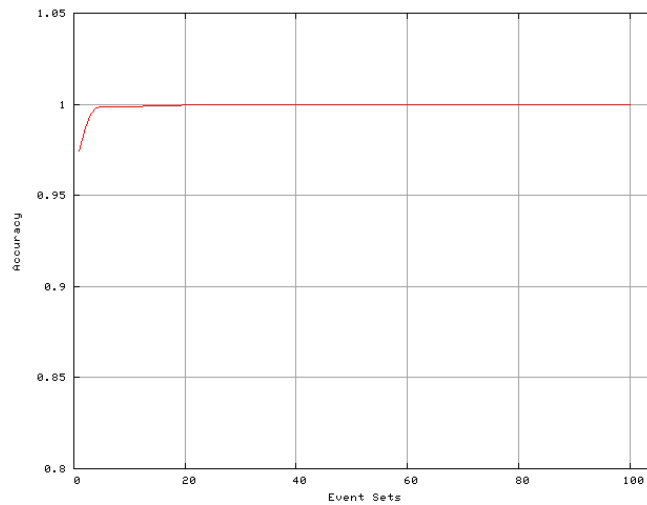


Figure 4.25. ALZ learning rate for MavHome Steve data.

Each 6-tuple (see 4.1.1) of the intelligent environment data stream is transformed into a two-part event, consisting of an identifier and state, by constructing a symbol s from the concatenation of z and n for the identifier and using l for the state. This event is provided as input to ALZ. In the training phase, a data set of these two-part events is used to create the trie and learn the data. During prediction operations, a single two-part event (i.e., the currently observed event) is provided and the prediction of the next event is returned in the same two-part

event format. Time and other information elements are not used because they are not currently used by ALZ.

ProPHeT first trains the ALZ algorithm and then uses the predictions in real time. A typical learning rate for ALZ is shown in Figure 4.25. ALZ processed the MavHome Steve simulation generated and filtered observation data and converged to 99.99% accuracy on data repeated 10 times to convergence—results are similar for real data in the MavLab and MavPad environments. Each training event set contains 33,088 events for the MavHome Steve data set. These are typical training results for our intelligent environment data.

ALZ is not used directly for decision-making because of it does not perform with sufficient accuracy to automate an intelligent environment alone. The primary problem with training a predictor in an intelligent environment is that there is not enough training data for it to see enough activity cases in the environment to gain acceptable accuracy. Coupled with an ever-changing inhabitant model, a predictor may never observe enough data for successful automation. We utilize information from the predictor to assist in belief state because when limited to specific Markov chains the performance of ALZ is sufficient to aid in belief state determination.

Active LeZi was developed by Karthik Gopalratnam and Dr. Diane Cook at The University of Texas at Arlington. Interested readers should refer to the referenced material for more information on ALZ [61].

4.3.4 Episode Membership Learning

Due to the hierarchical abstractions that contain sequences of observations in the environment, another useful tool to assist with decision-making is one that provides the knowledge of what is the most likely episode the system is currently observing in each layer of abstraction. The Episode Membership algorithm (Epi-M) provides this data. Epi-M takes the current observation event and the recent observation history and returns the probability of membership for

the top number of specified episodes for each layer of abstraction. It performs this by matching the current event stream against the learned patterns from ED and ordering the results by highest percentage of pattern match. Because our systems are not temporally grounded, we take advantage of time in Epi-M to provide a method of temporal differentiation between patterns. We do this by employing the technique of storing and utilizing circular statistics of episode occurrence.

The same data that is mined by ED and used to train ALZ is also used by Epi-M to build a probability of episode occurrence around a circle using circular probabilities [15]. This circle represents the most probable time of occurrence in a 24-hour period. First, the patterns are matched to their respective episode from the training data and the mean time of the events that occur in that episode is stored. Time in seconds t is converted to degrees ϕ , based on 360° per day, by Equation 4.1.

$$\phi = t \cdot 0.004167 \quad (4.1)$$

The time in degrees or *observed angles* for each occurrence of the episode is used to derive the the rectangular coordinates of the center of mass using Equations 4.2 and 4.3.

$$\bar{x} = \frac{1}{n} \sum_i \cos\phi_i \quad (4.2)$$

$$\bar{y} = \frac{1}{n} \sum_i \sin\phi_i \quad (4.3)$$

The length of the *mean vector* is found by applying Equation 4.4, and the *mean angle* is determined from Equation 4.5 which describes the normal and exception cases.

$$r = \frac{1}{n} (\bar{x}^2 + \bar{y}^2) \quad (4.4)$$

$$\bar{\phi} = \begin{cases} \arctan(\frac{\bar{y}}{\bar{x}}), & \text{if } \bar{x} > 0 \\ 180 + \arctan(\frac{\bar{y}}{\bar{x}}), & \text{if } \bar{x} < 0 \\ 90, & \text{if } \bar{x} = 0 \text{ and } \bar{y} > 0 \\ 270, & \text{if } \bar{x} = 0 \text{ and } \bar{y} < 0 \\ \text{undetermined,} & \text{if } \bar{x} = 0 \text{ and } \bar{y} = 0 \end{cases} \quad (4.5)$$

Through these equations we can understand the mean time of occurrence for a specific episode, and by using Equation 4.6 we can derive the *angular deviation* of the occurrences.

$$s(\text{degrees}) = \frac{180}{\pi} \sqrt{2(1-r)} \quad (4.6)$$

Equations 4.1-4.6 and the circular probability technique are derived from the work of Batschelet [15]. This information can be used to understand when episodes normally occur by reporting the mean and angular deviations of the prior observations of the episodes.

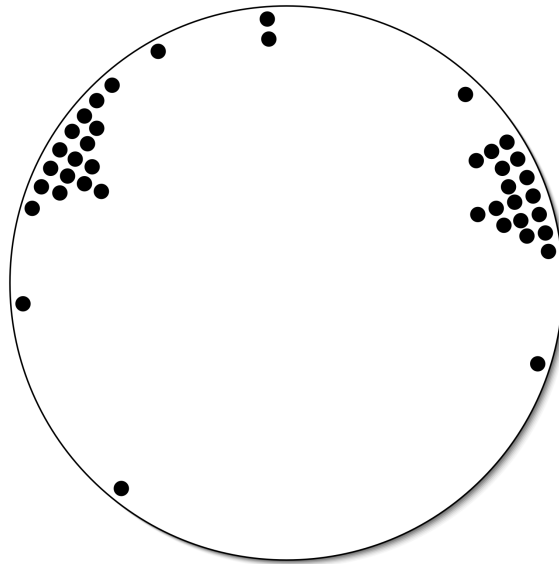


Figure 4.26. Example circular probability of two events occurring at different times of the day.

The use of circular statistics to differentiate between similar episodes separated by minor pattern differences but major temporal differences is valuable in proper episode membership identification. Figure 4.26 provides a generic example of two episodes that occur at different times. If the patterns in the episodes were similar, perhaps only differing by as few as one event, the current event stream prior to that singular change may yield similar probabilities for both episodes; however, if there were a temporal difference this could be used to weight the probability of match higher for the one with a temporal match (i.e., a matching episode that occurs at the same clock time as previous episodes of that type). Thus, a higher accuracy in proper episode membership reporting can be gained through the addition of temporal information. The use of circular probability is a mechanism for the efficient tracking and representation of time.

Algorithm 4 defines the training process for storing the episodes from ED and learning the temporal occurrences (i.e., when in time each episode occurs).

ED contains a mechanism for episode membership that simply uses percentage of pattern match without considering any temporal issues, but its usage was discontinued because of software reliability issues. We performed a test case of the difference in performance between ED-Episode Membership (ED-EM) and Epi-M in which we used the MavHome Steve patterns in simulation and modified pattern 4 to turn on light a_14 if after midnight and before noon (pattern 4a); otherwise, the pattern substituted light a_16 for light a_14 (pattern 4b). Ten patterns were performed during the course of a simulated 24 hour period including pattern 4a in the morning and pattern 4b in the afternoon. ED-EM classified eight patterns with the proper probability ranking after at least 50% of the pattern had been observed, but patterns 4a and 4b both indicated the same probability when 4a should have been more probable in the morning and 4b in the afternoon. Epi-M classified all ten patterns with the proper probability ranking, again after at least 50% of the pattern had been observed. This minor improvement by Epi-M

Algorithm 4 Episode Membership Training**Require:** Derived episodes file from ED and the input data stream used for ED and ALZ

```

1:  $\Lambda_n \leftarrow$  episodes from file
2:  $\kappa \leftarrow$  empty {History of events}
3:  $n \leftarrow$  number of episodes
4: {Store mean time (in seconds) for each episode instance in the episode}
5: while events  $\varepsilon$  in input file do
6:    $\kappa \leftarrow \kappa + \varepsilon$  {Add current event to history}
7:   for all  $n$  do
8:     if  $\Lambda_n = \kappa$  then
9:       {Episode match! Add time to list of times}
10:       $\Lambda_n[\phi_{i+1}] \leftarrow \text{mean}(\kappa(t)) * 0.004167$ 
11:     end if
12:   end for
13: end while
14: {Store circular statistics with each episode}
15: for all  $n$  do
16:    $\bar{x} \leftarrow \frac{1}{n} \sum_i \cos \Lambda_n[\phi_i]$ 
17:    $\bar{y} \leftarrow \frac{1}{n} \sum_i \sin \Lambda_n[\phi_i]$ 
18:    $r = \frac{1}{n} (\bar{x}^2 + \bar{y}^2)$ 
19:   if  $\bar{x} > 0$  then
20:      $\Lambda_n[\bar{\phi}] \leftarrow \arctan(\frac{\bar{y}}{\bar{x}})$ 
21:   else if  $\bar{x} < 0$  then
22:      $\Lambda_n[\bar{\phi}] \leftarrow 180 + \arctan(\frac{\bar{y}}{\bar{x}})$ 
23:   else if  $\bar{y} > 0$  then
24:      $\Lambda_n[\bar{\phi}] \leftarrow 90$ 
25:   else if  $\bar{y} < 0$  then
26:      $\Lambda_n[\bar{\phi}] \leftarrow 270$ 
27:   else if  $\bar{x} = 0$  and  $\bar{y} = 0$  then
28:      $\Lambda_n[\bar{\phi}] \leftarrow$  undetermined
29:   else
30:     Trap error!
31:   end if
32:    $\Lambda_n[s] \leftarrow \frac{180}{\pi} \sqrt{2(1-r)}$ 
33: end for

```

in proper probability of membership reporting provides a better belief state to ProPHeT for HPOMDP automation. Details of the process for providing membership probabilities and how the circular probability information is used are presented in Section 4.6.3.

4.4 Model Generation

Research into building hierarchical models has been an ongoing effort since the early 1990's [185]. In order to provide the reader with an overview of the work in this area and how it has evolved, we present two notable and heavily cited papers. The first is "Reinforcement Learning with a Hierarchy of Abstract Models" by Satinder Singh [185] and the second "Hierarchical Solution of Markov Decision Processes using Macro-actions" by Milos Hauskrecht, Nicholas Meuleau, Craig Boutilier, Leslie Pack Kaelbling, and Thomas Dean [72]. We provide an overview of the evolution of the approaches presented in these papers in order to provide some background for our work and to provide useful information for the interested reader wishing to pursue a deeper understanding. We will discuss more relevant work after this exploration, but first provide a historical perspective.

In "Reinforcement Learning with a Hierarchy of Abstract Models" by Satinder Singh [185], he presents a hierarchical approach to reinforcement learning based on a common TD approach at each level. A two-level, hand-generated hierarchy is presented and evaluated in a typical robot grid world. Abstract nodes are created as experimenter-selected aggregates of lower-level production states. The world is order 0 Markov and completely observable. All actions and states are finite and the policy at each level is defined by $\pi : S \rightarrow A$. The system controls all action decisions and causes all state transitions [184, 186]. This work has become the basis and reference point for many other pieces of work. Figure 4.27 shows the evolution diagram for this paper.

There are a number of referential papers. Sutton [202] presents an extension through equation improvements and generalization for representing temporal differences throughout the same structure (VRTM). Thrun and Schwartz [213] present the SKILLS algorithm providing a use of the temporal abstraction mechanisms. Donnart and Meyer [47] extend the hierarchal approach by mixing reactive and deliberative components. Koenig and Simmons [100] perform a complexity analysis of these techniques and make suggestions for enhanced

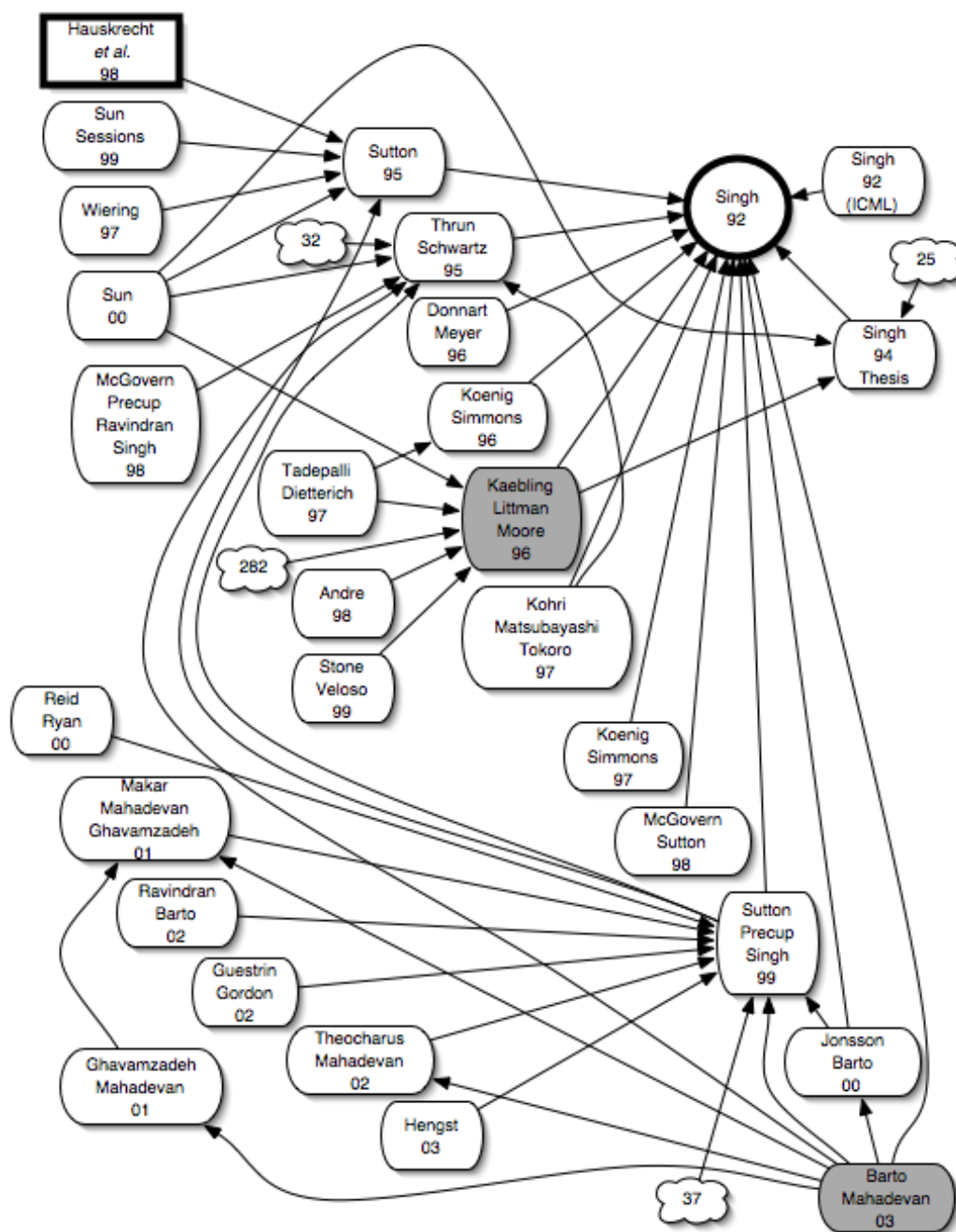


Figure 4.27. The Evolution of “Reinforcement Learning with a Hierarchy of Abstract Models” by Satinder Singh. The source paper appears as a heavy circle, other papers are in a bowed rounded rectangle, papers in a heavy square are other evolved papers expanded in our discussion. Each paper representation graphic includes the author names and year of publication. An arrow is drawn from a paper to another with the arrow head pointing to the referred to paper and the tail at the referring paper. Clouds with internal numbers indicate a large number of citations for the paper that the arrow from it points to. Time flows historically from the upper right corner to the lower left corner. Self referential papers and direct derivative works by the same author appear to the right of the source paper. A shaded paper indicates a survey paper.

performance tuning by better reward structures and strategies. Kaelbling, Littman, and Moore [95] present a survey of reinforcement learning techniques at that point in time. Kohri, Matsubayashi, and Tokoro [102] forward an automated extension to modular Q-learning. Koenig and Simmons [101] present some additional complexity analysis. McGovern and Sutton [123] perform some additional supporting empirical analysis of this approach. Sutton, Precup, and Singh [204] present a major evolution of VRTMs into an MDP with closed-loop policies called options to form Semi-Markov Decision Processes (SMDPs). Jonsson and Barto [93] adapt the U-Tree algorithm to build option-specific state abstractions improving hierarchical learning systems. Barto and Mahadevan [14] present an updated survey of recent advances in hierarchical reinforcement learning.

Many papers reference Sutton [202]. Hauskrecht *et al.* present the work that is examined in detail next. Sun and Sessions [198, 199] explore self-segmentation—building the reinforcement hierarchies autonomously through segmentation based on reinforcement received during task execution. They present an algorithm for segmenting sequences in order to reduce non-Markovian temporal dependencies. This is accomplished by segmenting around Q-learning control policies. Self-segmentation is provided generic structures (specified number of levels) in which to develop the hierarchies. They apply this to robot maze problems. Wiering [222] extends Q-Learning into Hierarchical Q-Learning (HQ-Learning) and uses it on POMDPs.

Many papers also reference Thrun and Schwartz [213]. Sun [198] is working on self-segmentation. McGovern, Precup, Ravindran, Singh [124] explore the extension of options which are closed-loop policies over a temporal partition in an MDP.

A multitude of papers reference Kaelbling, Littman, and Moore [95]. Tadepalli and Dietterich [206] forward work on Hierarchical Explanation-Based Reinforcement Learning which are the first hierarchical extensions to EBRL. Andre [11] discusses how to learn systems of hierarchical behavior. Stone and Veloso [195] explore layer learning, which explores mapping inputs to outputs in problems of intractable size by performing hierarchical task decomposi-

tion, performing learning on each layer, and integrating the learning across layers. The task decomposition mechanism is provided by experimenter-selection. They apply this to robotic soccer problems.

The new basis extension work of Sutton, Precup, and Singh [204] is often referenced. Reid and Ryan [170] present the RL-TOPS Hierarchical Reinforcement Learning System and an extension that uses Inductive Logic Programming (ILP) to bridge the gap between temporal abstractions to improve performance. Makar, Mahadevan, and Ghavamzaadeh [114] extend MAXQ (a hierarchical reinforcement learning approach) to the multi-agent case. Ravindran and Barto [169] work on minimizing MDP by taking advantages of redundancies and extend this model minimization to options. Guestrin and Gordon [67] explore distributed planning in HMDPs. Theocharus and Mahadevan [209] perform exploratory work in Hierarchical POMDPs in robot navigation tasks. Hengst [78] examines safer aggregation of reusable sub-task states through a supporting decomposed discount function. He also forwards an HRL modification to help solve infinite horizon problems in which a sub-task must persist in order to achieve an optimal policy.

Ghavamzadeh and Mahadevan [60] extend MAXQ to handle continuous-time reward SMDPs.

The evolution of Singh's work can be summarized in the following steps:

1. Singh presents idea of hierarchy of abstract models
2. Sutton formalizes and generalizes
3. Kaelbling includes in a survey
4. Minor improvements, application, and empirical performance work
5. Sutton, Precup, and Singh take VRTMs \rightarrow SMDPs (Big Event)
6. New work focuses some auto segmenting, more minor incremental improvements, more applications
7. MAXQ comes out and unifies this through the work of Sutton [202] and Hauskrecht [72]

8. HPOMDP and SMDP work in active research

In “Hierarchical Solution of Markov Decision Processes using Macro-actions” by Milos Hauskrecht, Nicholas Meuleau, Craig Boutilier, Leslie Pack Kaelbling, and Thomas Dean [72], the authors present a series of approaches for solving complex MDPs. Through the use of temporally abstracted macro-actions, a hierarchical MDP can be constructed by augmenting an MDP action space with macro-actions, abstracting an MDP by replacing all of the actions with the macro-actions and removing all but the boundary states to those actions, or by creating a hybrid MDP using an abstract MDP with an expansion of the original MDP in areas of recent dynamic change. Automated generation and construction techniques for macros are discussed. Empirical performance on robot navigation tasks is presented. This work has become the basis and reference point for many other pieces of work. Figure 4.28 shows the evolution diagram for this paper.

There are a number of referential papers. McGovern and Sutton [123] perform some additional supporting empirical analysis of this approach. McGovern [122] proposes the acQUIRE-macros algorithm for automatically finding macro-actions online in a reinforcement learning framework.

Laroche, Charpillet, and Schott [107] use a directed graph to automatically generate information that is utilized to improve the communication between regions. Parr [145] discusses hierarchical control and learning for Markov decision processes and the use of an abstract probabilistic plan called the Hierarchical Abstract Machines (HAM). Singer and Veloso [183] present bias techniques to improve the speed of macro-action learning. Sutton, Precup, and Singh [204] present a major evolution of VRTMs into an MDP with closed-loop policies called options to form Semi-Markov Decision Processes (SMDPs). Dietterich [45, 44] presents a new approach to HRL (hierarchical reinforcement learning) based on the MAXQ decomposition of the value function. MAXQ unifies the previous work of Singh, Kaelbling, and Dayan and Hinton. Huber [82] presents a hybrid architecture for adaptive robot control which considers

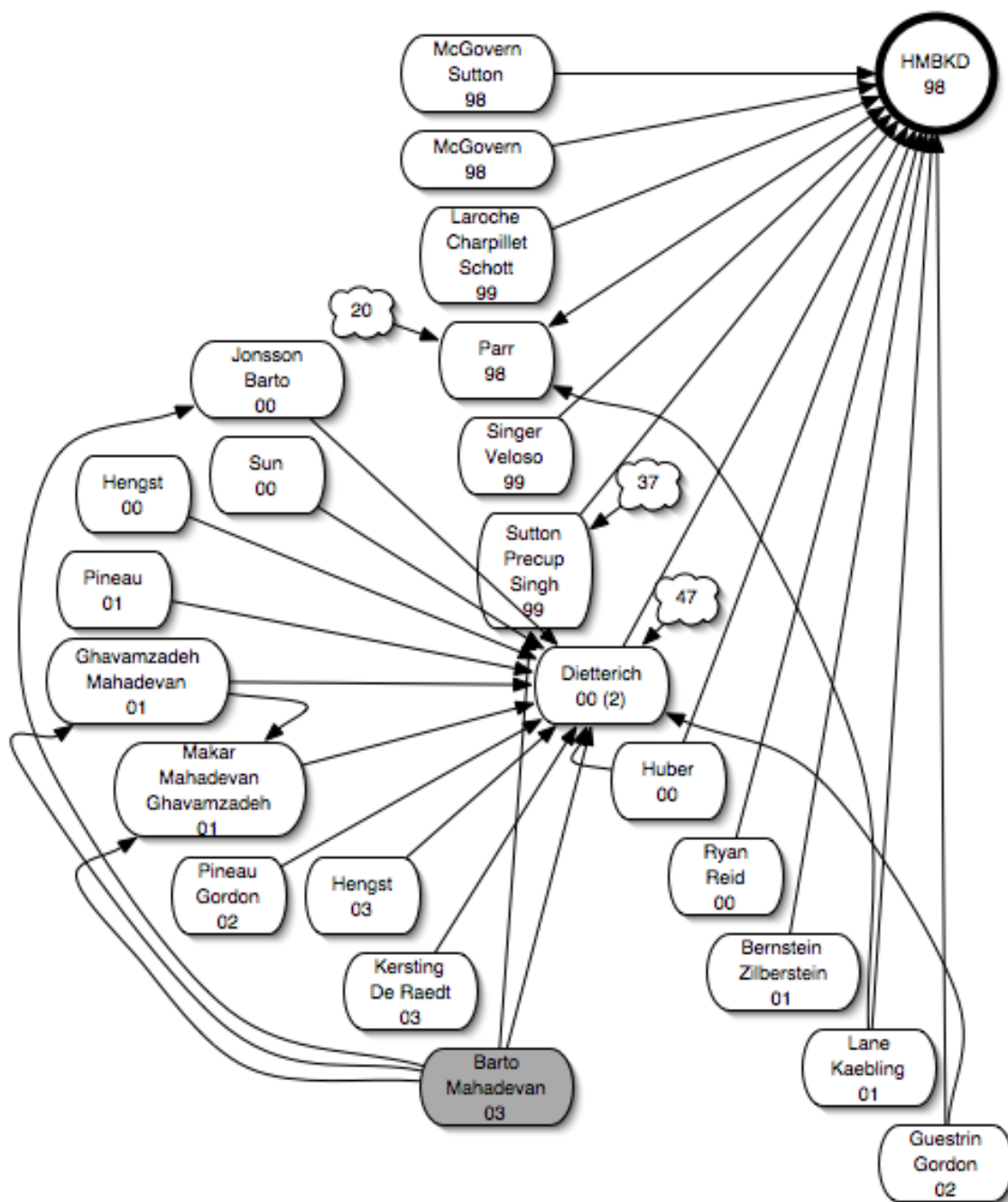


Figure 4.28. The Evolution of “Hierarchical Solution of Markov Decision Processes using Macro-actions” by Milos Hauskrecht, Nicholas Meuleau, Craig Boutilier, Leslie Pack Kaebing, and Thomas Dean (see Figure 4.27 for nomenclature details).

hierarchical action spaces. Ryan and Reid [170] present the RL-TOPS Hierarchical Reinforcement Learning System and an extension that uses Inductive Logic Programming (ILP) to bridge the gap between temporal abstractions to improve performance. Bernstein and Zilberstein [18] present work in applying micro-actions to weakly-coupled MDPs in a planetary rover scenario. Lane and Laebling [104] explore the application of well-understood deterministic optimization techniques that can benefit MDP solutions techniques. Guestrin and Gordon [67] explore distributed planning in HMDPs.

Dietterich's [45, 44] work has been widely accepted and taken in as part of active research because of its unification of existing HRL work. Work referencing Dietterich follows. Josson and Barto [93] adapt the U-Tree algorithm to build option-specific state abstractions improving hierarchical learning systems. Sun [198] is working on self-segmentation. Hengst [79] examines the addition of heuristics to HMDPs to improve performance. Pineau [157] explores the application of HPOMDPs to solving robotic dialog problems. Ghavamzadeh and Mahadevan [60] extend MAXQ to handle continuous-time reward SMDPs. Makar, Mahadevan, and Ghavamzadeh [114] extend MAXQ (a hierarchical reinforcement learning approach) to the multi-agent case. Pineau and Gordon [158] propose an automatic, lazy algorithm which plans in a bottom-up approach and finds good abstractions for high-level planners. Hengst [78] examines safer aggregation of reusable sub-task states through a supporting decomposed discount function. He also forwards a HRL modification to help solve infinite horizon problems in which a sub-task must persist in order to achieve an optimal policy. Kersting and De Raedt [97] explore improved abstractions by integrating MDPs with Logic Programs (LOMDPs). Barto and Mahadevan [14] present an updated survey of recent advances in hierarchical reinforcement learning.

The evolution of the work by Hauskrecht *et al.* can be summarized in the following steps:

1. Minor refinements in theory, generalization, and improvements in speed.

2. Work on auto segmentation and auto learning of macro-actions
3. Sutton, Precup, and Singh develop SMDPs
4. Dietterichs MAXQ unifies hierarchical MDP work
5. Further refinements and applications apply to MAXQ
6. HPOMDP and SMDP work emerges

The growth in research exploring the utilization of hierarchy and abstraction in stochastic models has led to the current explorations in HHMM and HPOMDP work. None of the work presented in the model generation section thus far involves data-driven learning of hierarchical structure.

Hierarchical Hidden Markov Models (HHMMs) [54] and Partially Observable Markov Decision Processes (POMDP) [189, 94] have been popular techniques for making decisions under uncertainty. These are natural extensions of a long chain of HMM and MDP work that proceeds them [164, 117, 16] and evolved from the work previously discussed. The expansion of these models with features for abstraction and hierarchy is allowing for more application potential over the traditional flat, uniform scale models which perform poorly in large state space domains.

Recently there have been some published hierarchical extensions that allow for the partitioning of large domains into a tree of manageable POMDPs. Joelle Pineau while at Carnegie-Mellon University investigated hierarchical approaches to POMDP planning and execution by taking advantage of the structure inherent in the problem domain and finding modular policies for complex tasks. Her work partitioned the action space into groups of related actions and created a HPOMDP. This approach was applied to Sondik's parts manufacturing problem, Dietterich's taxi task, and a robot dialogue interface and proven to significantly improve time performance while maintaining accuracy [156]. Georgios Theodorou while at Michigan State University introduced the Hierarchical POMDP as an extension to the Hierarchical HMM with the addition of actions and rewards. His work involved the planning and learning for robot

navigation using the HPOMDP which was updated using a hierarchical Baum-Welch algorithm (a.k.a, the Forward-Backward algorithm). His work converged on a policy for successful robot navigation, but used a pre-constructed HPOMDP model [210]. In follow-on work he continued work in robot navigation of spatial environments and investigation of manual modeling of these environments and the improvements by using hierarchical over flat models [211]. In work that takes a similar approach to ours, Lühr *et al.* at the Curtin University of Technology in Australia used a HHMM to represent kitchen activities from observed sensor data by hand-coding the model. Evaluation of the model proved to correctly classify observed human activity in the same domain [113].

The work performed by these researchers has developed and explored the powerful representation model of the HHMM/HPOMDP, although, there is a noticeable problem—current work in the field assumes sufficient *a priori* knowledge to pre-construct the HHMM/HPOMDP where the work focuses on learning the model and the appropriate transition probabilities between states. Given the potential size of the state space in intelligent environments, the uniqueness of each, and the desire to minimize model contamination from knowledge engineering, we will need to seed our decision-making systems with structure derived through domain knowledge gained from collected data in these given environments. In this data rich domain we need to discover this knowledge in our database of inhabitant observations.

There are some examples of current work in automating the generation of HHMMs. Lexing Xie *et al.* from Columbia University/Mitsubishi Electric Research Labs are involved in the learning of HHMMs through unsupervised Bayesian learning techniques in combination with filter and wrapper methods for feature selection in video structures. Differences in activity during a recorded soccer game were discovered [225]. Skounakis, Craven, and Ray of the University of Wisconsin have used information extraction techniques to automatically extract instances of specified relations or classes from text and then construct a HHMM from the extracted data. They train their derived model using the forward-backward algorithm and use

the Viterbi algorithm to predict tuples in the test sentences. The authors also extend the HHMM to a *Context hierarchical HMM* to represent additional information about sentence structure within phrases. They have applied this work in empirical studies using biomedical literature text and have extracted several instances of biomedical relations [187].

Our research supports previous work in harnessing the strength of the HHMM/HPOMDP as an appropriate model for human activity and extends previous approaches for automated building of these models. We will show that a data mining technique can automatically construct such a model, and we will demonstrate that our data-driven technique can be used to automate facets of an intelligent environment.

4.4.1 Automated HHMM Construction

An HHMM is defined as a set of states S , which include *production* states (leaves) that produce observations¹, *abstract* states which are hidden states (unobservable) representing entire stochastic processes, and *end* (child) states e^S which return control to a parent node; a horizontal transition matrix T , mapping the probability of transition between child nodes of the same parent; a vertical transition vector Π , that assigns the probability of transition from an internal node to its child nodes; a set of observations Z ; and a mapping of a distribution probability set of observations in each product state O . Thus, it is formally defined as a 5-tuple: $\langle S, T, \Pi, Z, O \rangle$. The j^{th} child of state s is found using the function $c(s, j)$ which is used in the horizontal transition function $T^s(c(s, i), c(s, j))$ to denote the transition probability between the i^{th} and j^{th} child of state s . We utilize a common set of nomenclature for better understanding and consistency in the field, for detailed discussions of the HHMM the interested reader should refer to work reported by Fine and Theodorou [54, 210].

From a collection of data that contains many repetitive patterns, data-mining techniques can automatically discover these patterns and provide statistical data on pattern permutations

¹In the intelligent environment domain there is a one-to-one correspondence between action and observation.

within a given set of members and over the entire data set. This information can be utilized to create abstract states from the identified patterns and production states from the pattern sequences. Repeating the data-mining process to discover episodes of episodes can be used to create a hierarchical organization. As previously described, we utilize the Episode Discovery algorithm to perform the knowledge discovery process.

The 3-tuple data provided by ED is converted to a 5-tuple HHMM by taking each periodic episode k and performing Algorithm 5.

Algorithm 5 ProPHeT HHMM Construction

Require: 3-tuple data provided by ED for the hierarchy

```

1: for all abstract nodes do
2:   Create an episode labeled abstract node  $s_n$  if one does not exist
3:   for all sets in  $\Psi$  do
4:     if  $\Psi$  contains production states then
5:       Create an observation labeled product node  $s_m$  for each observation in  $\Upsilon$ 
6:     else { $\Psi$  contains abstract states}
7:       if abstract node does not exist then
8:         Create an episode labeled abstract node  $s_m$ 
9:       else
10:        Insert  $s_n$  into the location of  $s_m$  {All nodes connected to  $s_m$  should be reconnected
        to  $s_n$  as this is an abstracted replacement representation}
11:      end if
12:    end if
13:     $T^s(c(s, i), c(s, j)) \leftarrow \varphi/\rho$  {Assign the horizontal transition matrix value between each
    node}
14:     $T^s_\epsilon(c(s, i), c(s, j)) \leftarrow \Psi_0 \times \Psi_m$  {Store history with each transition}
15:    Create an end node  $e_n$ 
16:    Assign the last node in the sequence from  $\Psi$  a horizontal transition value to the end
    node  $e_n$  of
17:    if another transition exists from this node then
18:       $T^s(c(s, i), c(end)) \leftarrow \varphi/\rho$ 
19:    else
20:       $T^s(c(s, i), c(end)) \leftarrow 1$ 
21:    end if
22:  end for
23:  Assign the vertical transition vector value
24:  if  $s_m = \Psi^i(\Upsilon^0)$  then {Appears as a first node in  $\Psi$ }
25:     $\Pi^s \leftarrow \Psi^i(\varphi/\rho)$ 
26:  else
27:     $\Pi^s \leftarrow 0$ 
28:  end if
29:  Connect the abstract node  $s_n$  to the root node  $s_1$  {Special Case : If  $\Psi_{s_n}$  contains all
  lower tier abstract nodes then  $s_n = s_1$  }
30:  Connect all abstract nodes not connected to another abstract node to an end node di-
  rectly connected to the root node {This ensures complete transition paths for all nodes}
31: end for

```

intelligent environment work, a dominant Markov chain does not promote good automation. We extend the HHMM with a bounded violation of the Markov assumption by adding ε -deep history to each horizontal transition as shown in equation 4.7 and bounded by equation 4.8. The first equation states that the number of history elements in a node is determined by the number of preceding nodes in the event chain per permutation up to a maximum length, as set in the second equation, of the permutation event chain. Moving away from an order 0 to a mixed-order Markov model allows for the ability to differentiate from among alternative Markov chains making automation decisions clear while maintaining the strengths of this representation.

$$\varepsilon_i^s = \text{instances}(s, \Psi) \quad (4.7)$$

$$\text{length}(\varepsilon^s) \leq \text{max_length}(\Psi) \quad (4.8)$$

This technique accounts for the automatic construction of the HHMM S , T , and Π . Z is the set of all observations in Φ^k . From the last paragraph discussion on abstract state representations of observations, it can be seen how O can be easily calculated (where $O(s, z) \rightarrow (0, 1)$: the probability of observing z in state s). This process creates a HHMM representation of the observed patterns of an inhabitant in an intelligent environment.

As a test case we will provide a walk-through of the generation of a section of the HHMM created from observation data of our simulated inhabitant, MavHome Steve, and his six patterns as defined at the beginning of this chapter. Eight weeks of observation data were generated using ResiSim and the MavHome Steve model from Figure 4.7. Steve was modeled to enter and leave the MavLab 3–5 times a day (i.e., morning, lunch, and two classes) and while in the lab take 4–6 breaks and work from the alternate workstation 1–3 times per day. Random motion noise was injected into the data, but was not inserted between the events in the specific patterns—this was done to better simulate the environment. What resulted is a data set

of 33,088 events which when filtered contained 19,880 pattern related events (≈ 355 per day). ED was used to process the data into episodes and found all six MavHome Steve episodes (it also found five other episodes, but they did not contain automatable events and were filtered) and two layers of abstraction.

```

ID = 67348
11 v_30_1 a_14_0 v_30_0 v_28_1 v_25_1 v_28_0 v_22_1 v_25_0 v_24_1
v_22_0 c_12_1
NumberOccurrences = 112
NumPatterns = 1
v_30_1 a_14_0 v_30_0 v_28_1 v_25_1 v_28_0 v_22_1 v_25_0 v_24_1 v_22_0
c_12_1 112

```

Figure 4.30. Episode Discovery reported episode segment.

A single episode of data (extracted from the complete set) produced by ED is shown in Figure 4.30. Using Algorithm 5 an abstract node 67348 is created, which decomposes into eleven production nodes and one end node. There are no permutations, so the vertical transition function Π only contains a single entry $C^{v_{30.1}} \rightarrow 1$. The horizontal transition matrix contains transitions between each node connected in the episode sequence each with a transition probability of 1 also due to no permutations. The resultant HHMM from adding this node is shown in Figure 4.31 and corresponds to MavHome Steve pattern 5 (i.e., return from alternate workstation).

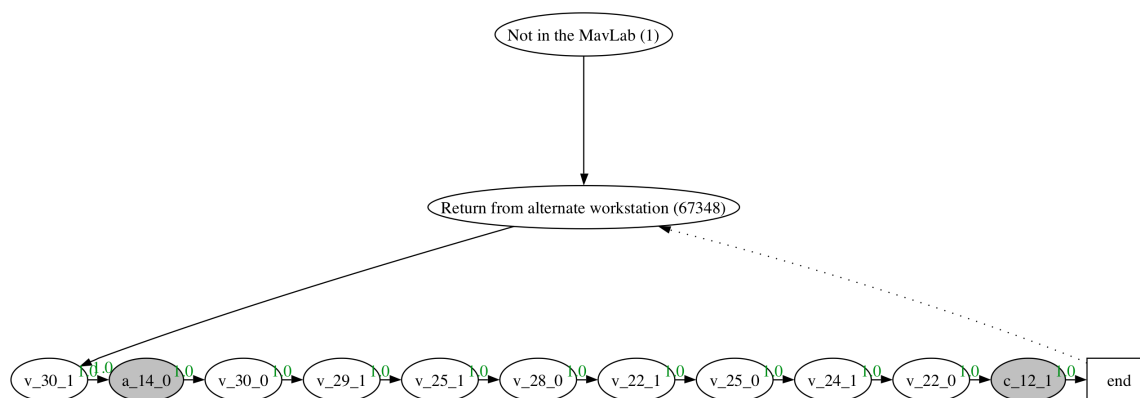


Figure 4.31. HHMM derived from ED data for MavHome Steve pattern 5 (single abstraction layer).

```

ID = 237845
2 pe_62435 pe_67348
NumberOccurrences = 112
NumPatterns = 1
pe_62435 pe_67348 98
pe_62435 pe_67348 pe_62435 pe_67348 14
  
```

Figure 4.32. Episode Discovery reported abstract episode segment.

To build a hierarchy, the input data is compressed by replacing each discovered episode with a timestamp for the episode, then is processed again by ED to discover sequences of episodes. Figure 4.32 shows a single episode of data (extracted from the complete set) produced by ED in the second pass. The “pe_” prefix is used by ED to denote a *periodic episode* versus a *frequent episode* (see [75]), but what is important to note is the trailing episode number. The 67348 should look familiar, it is MavHome Steve pattern 5. Episode 62435 is pattern 4. Thus, ED has discovered abstractly the pattern of *alternate workstation work* (i.e., 237845) which consists of the two abstract episodes *go to alternate workstation* (i.e., 62435) and *return*

from alternate workstation (i.e., 67348). In the data there were 112 occurrences of this pattern, but of interest is that ED discovered a permutation of this pattern that involved 14 occurrences of back-to-back repetition. The probability of transitioning back to repeat the pattern instead of ending is 0.125 and of just ending after one execution of the sequence is 0.875. The resultant HHMM in combination with the previously-discovered episode in Figure 4.31 is shown in Figure 4.33.

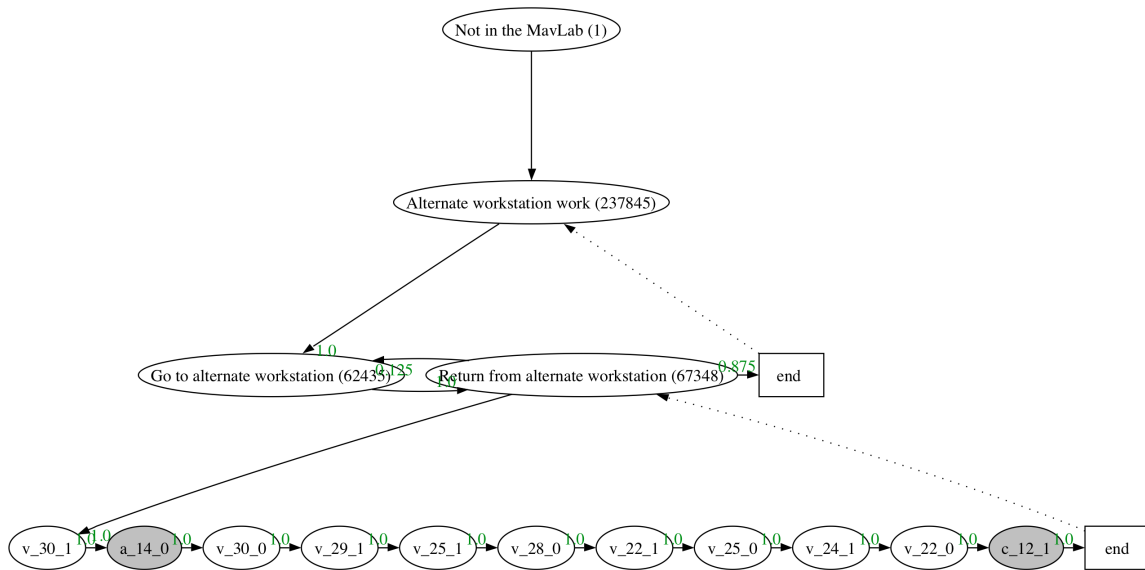


Figure 4.33. HHMM derived from ED data for MavHome Steve pattern 5 (double abstraction layer).

The learned HHMM from the MavHome Steve simulated inhabitant data is shown in Figure 4.34. For comparison against a real data-derived HHMM Figure 5.8 presents the HHMM from the MavPad inhabitant 3. The slight differences in nomenclature are due to hand editing of the Figure 4.34 visualization file to include readable names—Figure 5.8 is the raw graphical output from ProPHeT. Both figures present only abstract nodes due to space considerations and readability issues with presenting the full models with production nodes. It can be seen from

observed model accurately reflects the transition probabilities observed in the data and indeed generated by the simulated inhabitant. The design model was human created and based upon the desired transition probabilities in a stochastic environment. Inspection of the transition values does reveal similar probabilities in most cases except where a deviation in transition structure was discovered. The data-derived model found a number of pattern loops, particularly in the *take a break*, *alternate workstation work*, and between the *lab entry to desk* and *leave lab from desk* episodes. These loops are explained by the fact that they were actually observed and did generate those patterns and fall within the design model—trace a route through the *end* node through the abstraction and back to the episode beginning. Self-referential loops in the *take a break* and *alternate workstation work* episodes were also discovered and reflect an accurate model of the simulation behavior. ED often discovers and identifies loops in episodes that commonly repeat themselves. The use of history aids in maintaining an understanding of where in the event chain the current state exists and what transitions are available. Overall, the design and data-derived models are essentially the same, supporting the hypothesis that inhabitant models can be learned through our process.

Evaluating the correctness of real human models is much more difficult because the design model is not available. The proper automation of an inhabitant's environment is the only measure we can use to validate the correct user model. The better the automation accuracy the better the model. For inhabitants with a high degree of observable regularity, modeling and automation should be more accurate than for individuals who lead an unstructured lifestyle. If all humans are creatures of habit and those habits are regular, we should be able to build an accurate model of an observed human.

Several sections of this chapter have touched on the aspect of modeling inhabitants. There is an abundant amount of research in the user modeling area. An entire research field of user modeling exists with a dedicated conference, *International Conference on User Modeling* which is in its tenth year in 2005, and journals such as *User Modeling and User-Adapted In-*

teraction (UMUAI) published by Springer and the *International Journal of Human-Computer Studies* by Academic Press Ltd. UM'05 is co-located with IJCAI'05 showing the ties to the AI community. In fact, AAAI, IJCAI, and other AI conferences usually have user modeling sections in their conferences and proceedings. Related work in user modeling was previously presented in this chapter.

4.5 Model Extension

Creating a HHMM is only the first step to developing a useful model in our process. In order to perform automation actions in an environment we need to extend the HHMM into a HPOMDP.

4.5.1 The HPOMDP

A HPOMDP follows the same base definition as an HHMM with the addition of a set of actions A that is used to transition between states, and a reward function R defined on the product states. Thus, it is formally defined as a 7-tuple: $\langle S, A, T, \Pi, Z, O, R \rangle$. For detailed discussions on the HPOMDP interested readers should refer to [210].

4.5.2 Action Extension

Due to the distinguishing characteristic of a one-to-one correspondence between action and observation in our domain (i.e., intelligent environments), the transition action is merely the action of that observation (e.g., if the observation is that the light came on then the action is to turn the light on). In the class of domains these techniques are designed for there are two types of observations: those that have corresponding actions and those that do not. For example, in the intelligent environment domain we may control lights and appliances, but we cannot control motion sensing or opening doors (yet). For those transitions that do not have a corresponding action, the action is to do nothing (in practice we use zone 'z', number 0,

and state OFF to represent this). The derived HHMM from the last section is extended to a HPOMDP by adding the obvious actions to the state transitions and then by establishing a reward (or cost) for transition using Algorithm 6.

The production nodes in the HPOMDP have two varieties: automatable and non-automatable. Automatable production nodes are those in which an action can be performed by the system prior to entering the state. Non-automatable nodes are those which can only be entered into by environmental observation. It is important to set the action A for each automatable production node properly so that the correct action can be performed by the system; likewise, it is equally important to set a non-action for all non-automatable production and abstract nodes. The utilization of our HPOMDP model differs from traditional planning and control usage of POMDPs due to the unique features of our domain. We will discuss how automation occurs and how we use the HPOMDP in upcoming sections.

4.5.3 Reward Structure

The reward for a HPOMDP can be fixed, heuristic-based, or function-based (e.g., a function of utility cost changes for an action). HPOMDP policies generally follow a path with the least cost or maximum reward. In our usage of the HPOMDP we take a more passive role in that we must observe a sequence to occur, reason about a belief of where this observation stream corresponds in our model, and look ahead to possible future observations. Our influence over the environment is limited and serves as augmented control to an inhabitant who is in control. Unlike decision-making for control of an entity in an environment, we must mostly observe, reason about the future, and automate with certainty during a narrow window of opportunity. To this end, we have found it useful to tie the reward to the transition probabilities between states and use a function that seeks to utilize the maximum transition (reward) when traversing between states. Initially all reward values are set to the transition probability values

and then updated based upon feedback from ARBITER as described in Section 4.7.1. Algorithm 6 describes the reward assignment process.

Algorithm 6 ProPHeT HHMM Conversion to HPOMDP

```

1: for all Nodes in HHMM do
2:    $\{s_n \rightarrow s_{n+1}$  denotes an edge from the current node to the next node $\}$ 
3:   if Node is an abstract node then
4:     for all Edges  $s_n \rightarrow s_{n+1}$  do
5:        $A_{s_n \rightarrow s_{n+1}} \leftarrow z_{0_0}$  {non-action zone_number_state}
6:       if Transition from  $s_n \rightarrow s_{n+1}$  is horizontal then
7:          $R_{s_n \rightarrow s_{n+1}} \leftarrow T^s(s_n, s_{n+1})$ 
8:       else {Transition from  $s_n \rightarrow s_{n+1}$  is vertical}
9:          $R_{s_n \rightarrow s_{n+1}} \leftarrow \Pi^s(s_n)$ 
10:      end if
11:    end for
12:  else if Node is a production node then
13:    for all Edges  $s_n \rightarrow s_{n+1}$  do
14:      if Node  $s_{n+1}$  is automatable then
15:         $A_{s_n \rightarrow s_{n+1}} \leftarrow \text{zone}(s_{n+1})\text{-number}(s_{n+1})\text{-state}(s_{n+1})$ 
16:      else
17:         $A_{s_n \rightarrow s_{n+1}} \leftarrow z_{0_0}$ 
18:      end if
19:      if Transition from  $s_n \rightarrow s_{n+1}$  is horizontal then
20:         $R_{s_n \rightarrow s_{n+1}} \leftarrow T^s(s_n, s_{n+1})$ 
21:      else {Transition from  $s_n \rightarrow s_{n+1}$  is vertical}
22:         $R_{s_n \rightarrow s_{n+1}} \leftarrow \Pi^s(s_n)$ 
23:      end if
24:    end for
25:  else {No action or reward for end nodes}
26:    for all Edges  $s_n \rightarrow s_{n+1}$  do
27:       $A_{s_n \rightarrow s_{n+1}} \leftarrow \emptyset$ 
28:       $R_{s_n \rightarrow s_{n+1}} \leftarrow \emptyset$ 
29:    end for
30:  end if
31: end for

```

Our system seeks to automate action transitions between states. The reward for performing an automation is related to reducing the number of user interactions. Therefore, if

the reward for automating the action in a situation where the inhabitant intended to manually perform the action is 1 and there is no penalty (i.e., a reward of 0) to automating it and if the inhabitant does not intend to issue them but does not mind them being issued (countermanding action handling is introduced in section 4.7.1), then the expected reward for automating the action is equal to the transition probability. It is for this reason that we initially seed reward with the transition probabilities.

4.6 Events to Automation

In the intelligent environment domain, which is similar to a click-stream desktop application interaction domain and other such domains where a human-in-the-loop interacts with the environment in a partially observable fashion through interactive feature points (places where interaction occurs, such as a lamp in an intelligent environment or a widget in a GUI application), the unique features create the need for a different approach in decisions regarding control. Most of the actions that cause observations and subsequent changes in environmental state are beyond system control, but the system must follow the observations in its model. The decision to be made is opportunistic. It is a decision to perform an action to change the state of the environment at the correct time in order to satisfy the goals of the system. In our environments this must occur in real-time and in a narrow span of opportune time.

Our system is characterized by its staccato observation and control of our environments. Similar to the Viterbi algorithm in using a HHMM for speech recognition by observing a phoneme stream and providing the highest probable word match [175], we provide a set of algorithms for the non-traditional use of the HPOMDP to observe the event stream. We determine a belief state of where in the model the current environmental state corresponds based upon episode membership, history, the current observation, and a prediction of the next observation in a model that was automatically created through a data-driven knowledge discovery

process and algorithmic construction. By performing a look-ahead to determine if a goal satisfying action can be made, we can issue a control command and observe feedback which will be used to refine the HPOMDP to better converge on a policy π that will best satisfy our goals.

4.6.1 ProPHeT Streaming Control Algorithm

Algorithm 7 shows how events and our derived HPOMDP are used to make automation decisions in the intelligent environment. Events stream into the system where they are observed and passed to ALZ and Epi-M to produce a next observation and a list of membership probabilities for each layer in the HPOMDP hierarchy. A belief state is then calculated using all available information in order to determine where in the HPOMDP the current observation exists. If the current observation stream is identifiable within the HPOMDP, we look ahead within a defined horizon to determine if an automation event should occur in the near future. If automation should/can occur, a control action is initiated and checked for safety, security, and inhabitant preference rule violations before initiating or declining the control event. Feedback is issued and the process repeats *ad infinitum*.

4.6.2 Prediction

Prediction is a black box endeavor for our systems since we utilize ALZ. The trained ALZ algorithm is provided with the current observation o and returns the predicted next observation χ . χ will be used for the belief state calculations.

4.6.3 Episode Membership

A trained Epi-M is used to determine the set of most probable memberships Ξ given the current event observation o and event observation history ε . Ξ contains a specified top number of probabilities of membership from each layer of the HPOMDP and is generated through

Algorithm 7 Phase 2: Automation Decision-making Operations

Require: HPOMDP model, event stream connection to logical proxies

- 1: Initialize all variables
 - 2: **loop**
 - 3: Observe event stream
 - 4: Determine next state \leftarrow prediction algorithm (ALZ)
 - 5: Determine membership across hierarchy \leftarrow episode membership algorithm (Epi-M)
 - 6: Determine belief state in HPOMDP
 - 7: **if** Incoming stream is in a recognizable event chain **then**
 - 8: **if** Automatable action within lookahead horizon ζ **then**
 - 9: Issue Action
 - 10: Initiate ARBITER check
 - 11: **end if**
 - 12: **end if**
 - 13: **end loop**
 - 14: Cleanup and restore utilized resources
-

Algorithm 8. Hierarchical layers above the first abstraction of *production* nodes generate observations o_{i+1} from the highest probability-matched member of the next lower level. Combined with the prediction χ , current observation o , and ε observation history, the membership probabilities in Ξ can be used to provide a belief of the current HPOMDP state. In Algorithm 8, the tunable parameters k for angular deviation match and the positive match bias α_+ and negative match bias α_- values are determined experimentally by test case observation and hand manipulation. The values presented perform well with our intelligent environment data to promote correct temporal matches, but they may need alteration for different environments and domains.

4.6.4 Environment Automation

Context is defined as the “set of facts or circumstances that surround a situation or event [163].” It is environmental information that provides clues as to what is occurring in a place by the situated actors. The role of user context is an important aspect in developing systems

Algorithm 8 Episode Membership Determination

Require: ε event observation history, o current observation event, trained Epi-M, number n of desired top membership probabilities per hierarchical level

- 1: $\Xi \leftarrow \emptyset$ {Membership set containing ξ (hierarchy level, episode id, match probability)}
- 2: $C \leftarrow \emptyset$ {Working candidate set}
- 3: $o_0 \leftarrow o$ {0 becomes the bottom level observation}
- 4: $k \leftarrow 2$ {Tunable number of angular deviations for match}
- 5: **for all** *Abstract* nodes that contain *production* nodes **do**
- 6: **if** o is contained in s_n **then**
- 7: $v \leftarrow$ number of ε matches from o backwards in s_n
- 8: $\delta \leftarrow$ number of nodes in s_n
- 9: **if** o occurs within k angular deviations of the angular mean **then** {Bias by circular probability of occurrence at this time}
- 10: $\alpha_+ \leftarrow 1.10$ {Tunable positive match bias}
- 11: **else**
- 12: $\alpha_- \leftarrow 0.90$ {Tunable negative match bias}
- 13: **end if**
- 14: $\Xi \leftarrow \xi$ (hierarchy level, $s_n.id$, $\alpha_{\frac{v}{\delta}}$)
- 15: **end if**
- 16: **end for**
- 17: **for all** Hierarchical levels above the *production* nodes **do**
- 18: $o_{i+1} \leftarrow \max \Xi_i$ {Next level higher o_{i+1} is the highest current probability o }
- 19: **for all** *Abstract* nodes **do**
- 20: $v \leftarrow$ number of ε_{i+1} matches from o_{i+1} backwards in s_n
- 21: $\delta \leftarrow$ number of nodes in s_n
- 22: $\Xi \leftarrow \xi$ (hierarchy level, $s_n.id$, $\frac{v}{\delta}$)
- 23: **end for**
- 24: **end for**
- 25: Sort top n membership probabilities in Ξ per hierarchical level
- 26: Return Ξ

that reason about human activity. Knowing the context of the user limits the scope of activities in which they may engage or currently be participating and thus reduces the searchable state space making prediction more tractable and yielding a higher probability of success. Context is the key to understanding what services to offer or actions to automate [40].

At its core, the work presented in this dissertation provides a learned user context model embedded in the automation system. Work in this area is very active and there are others work-

ing on the problem of determining context based on learned, encoded, or inferred information. As previously mentioned in Chapter 2, as part of the Georgia Tech Aware Home they developed a context toolkit to provide context widgets that identify user context by sensor readings and convey this information to context-aware applications that need the information [178]. The context toolkit for the Aware Home is a low-level tool and provides the same functionality as the logical proxies and aggregators in our work—they simply provide a level of encapsulation around sensor and actuator features and provide information about that object. Context in this view is defined around single objects and is a view held by other projects such as iDorm (see Chapter 2). The view of context taken by the work in this dissertation and shared by others [27] is focused at a higher level, one around the tasks of everyday life that include many sensor readings. Many of the projects described in Chapter 2 use some form of context-aware computing. Many of the research activities at the former AT&T Cambridge Labs are good examples (i.e., call forwarding, teleporting, and so forth) [17, 27].

The area of context-aware computing encompasses many areas of active research. Application information forwarding, active mapping, shopping assistants, advanced web browsers, cyberguides, augmented reality systems, office assistants, conference assistants, location-aware services, and the associated supporting architectures are just some examples. Current models of context modeling for these systems include key-value models where context is provided as a value to a key variable, markup scheme models where markup language based profiles are provided for local objects, graphical models such as the Unified Modeling Language (UML) provide visual representation of context, object-oriented models which break down context into encapsulated models usually centered around sensors and build them up through aggregation and abstraction, logic-based models which utilize formal logic to describe the context, and ontology-based models which use ontologies (i.e., “explicit formal specifications of how to represent the objects, concepts, and other entities that are assumed to exist in some area of interest and the relationships that hold among them” [81]) to specify concepts and inter-

relations that define context. The work in this dissertation uses the object-oriented model of context modeling. The survey papers by Guanling Chen and David Kotz at Dartmouth College [27] and Thomas Strang and Claudia Linnhoff-Popien at the German Aerospace Center (DLR) and Ludwig Maximilians University respectively [196] provide good starting places for more information in context-awareness and current work in context learning.

The work in this dissertation is initially about learning user activity patterns and predicting the most likely event to occur next as we observe the modeled user. Related work in this area includes that of Cielniak, Bennewitz, and Burgard who utilized the Expectation-Maximization (EM) algorithm to learn and utilize the motion behaviors of people in a typical office environment from clustered trajectories using laser range finders to collect motion patterns. They also derived HMMs from the motion pattern data and compared the approaches. Their EM models were empirically proven to be better predictors of motion [32] than the HMMs. Similiar work performed outdoors using GPS information was conducted by Patterson *et al.* at the University of Washington. Particle filters, a variant of Bayes filters, and the EM algorithm were used to learn a traveler's current mode of transportation as well as his most likely route [150]. Liao, Fox, and Kautz continued that work and developed a hierarchical Markov model to learn and infer the movements of people through the community. The hierarchy was used to create multiple levels of abstraction that bridged the distance between raw GPS readings and injected high-level knowledge. Rao-Blackwellized particle filters were used for efficient inference. The generated user models were used to detect abnormal behaviors (e.g., getting off at the wrong stop) [110]. There is much work in user activity prediction, but the focus is more on feature learning, modeling, or anomaly detection. Our work focuses on the automation of environmental objects in the activity stream of our modeled users.

Our approach entails using belief states based on current environmental observations that map to states in the HPOMDP. We are using Epi-M to provide a set of membership belief states and ALZ to provide additional belief through a next state prediction. From the given current

belief state, the action with the largest reward, which in our case relates to the path of most probable occurrence, should be chosen [94]. In our intelligent environment work, we look for the nearest automatable action within a look-ahead horizon. Additional learning from feedback or interaction data may be used to update the transition probabilities of the model with the goal of improving convergence to the desired model [210].

Details provided in Algorithm 9 describe the belief state determination process. Different from the traditional POMDP belief state view, which usually refers to a system's belief about its state by a probability distribution over states indicating a likelihood with which the system is in each one of the states, we take a definition where our belief is based on the most likely state of the system narrowed to a single state. There is a constant stream of observations o coming into the system that become ε observation event history. ALZ provides a prediction χ of the next event to occur based on the current stream of events. Epi-M provides an ordered set Ξ containing lists of n current membership probabilities ξ_i (consisting of a probability, p , and episode identification, id), where ξ_1 is the most probable current episode and ξ_n the least for each layer in the hierarchy. Belief state is initially assigned a zero pointer and strength value. The strength value β_{value} is a numerical indicator of the strength of belief $\beta_{value} \rightarrow [0, 1]$ for a given match. For each of the top n membership probabilities in Ξ , a match attempt is made to align the horizontal transitions across ζ_i —there must be a valid path from the root of the HPOMDP to the lowest level of abstraction. Path transition matches that provide a clear path to a set of production nodes are evaluated on a first-find basis for a production node match to o , ε , and χ . If an alignment match is made, then belief state β is set to point at the current observed state. In order to filter out poor matches, the belief strength is assigned a value that accounts for the percentage of horizontal and vertical match as shown in Equation 4.9.

$$\beta_{value} \leftarrow \frac{\left(\frac{match(\varepsilon+o+\chi)}{pattern_length} + \sum_{i=0}^n \frac{prob(\xi_i)}{i} \right)}{2} \quad (4.9)$$

In implementation, we have found that the first-find is the best match. If β_{value} is greater than or equal to the defined belief threshold ζ then the belief state is not filtered and a valid pointer is returned. The belief threshold ζ is experimentally determined by hand tuning. It could also be given control of by the inhabitant. Low ζ values promote more risky decisions to be initiated since the belief state would be low. High values of ζ promote a more conservative system which will not issue decisions without a high level of belief. ζ should be set to the desired comfort level of the inhabitant which can only be determined experimentally with each individual inhabitant. The belief state is used to make an automation decision.

Algorithm 9 ProPHeT Belief State Determination

Require: o current event observation, ε event history, Ξ membership sets, χ next observation prediction, HPOMDP model

- 1: $\beta \leftarrow \emptyset$ {Belief state}
 - 2: $\beta_{value} \leftarrow 0$ {Numerical strength value of belief}
 - 3: $\zeta \leftarrow x$ where x is a real and $0.0 \leq x \leq 1.0$ {Tunable belief threshold}
 - 4: {Find belief state in HPOMDP}
 - 5: **for all** Ξ_i **do** {Find valid path and state match in HPOMDP from top 1 to n until first match}
 - 6: Match ξ_i for each level in HPOMDP
 - 7: **if** Ξ horizontal transitions valid **then** {Must be a valid vertical function of the learned HPOMDP}
 - 8: **if** Match o with ε history and χ next observation **then** {Find match state}
 - 9: $\beta \leftarrow s_i$ {Assign pointer to belief state}
 - 10: $\beta_{value} \leftarrow \frac{(\frac{match(\varepsilon+o+\chi)}{pattern.Length} + \sum_{i=0}^n \frac{prob(\xi_i)}{i})}{2}$ {Strength of belief calculation}
 - 11: **break** {Found first match}
 - 12: **end if**
 - 13: **end if**
 - 14: **end for**
 - 15: {Found belief state strength value must exceed the cut-off threshold}
 - 16: **if** $\beta_{value} \geq \zeta$ **then**
 - 17: return β
 - 18: **else**
 - 19: return $\beta \leftarrow \emptyset$
 - 20: **end if**
-

Automation decisions are made by the process detailed in Algorithm 10. For action determination, action transitions are examined based on 1) ε history and 2) β belief state. If β is not a valid pointer (i.e., empty) then there is no decision to be made because there is no current belief state. Given the state β an action decision needs to be made. First β is found in the HPOMDP by simply following the pointer. Then, the edges leading from β to other state observations are examined and the path of highest utility, given ε history, is followed until an automatable action is incurred or the look-ahead horizon is exceeded. The look-ahead horizon is a tunable parameter that is experimentally determined by observing the desired results of automation in the target environment. We employ different horizons for “ON” (or not-off for more than binary state objects) and “OFF” actions from experimentation in the intelligent environments domain. Specifically, we found that it is more acceptable to turn on an object (e.g., a light) slightly before the normal sequential occurrence so that the object appears to be in the correct state when desired (i.e., the light is on in the room just prior to entry). Conversely, it is not acceptable to turn off objects prematurely, users preference tends to require off automations to occur more closely to the observed sequence (e.g., do not turn the light in the room off until the inhabitant has just about left the room). A vector of different look-ahead horizons could be established—one for each potential state type. When the path is followed and finds the first automatable action, it is returned as an action decision. If no automatable action is found, then no action will be taken (i.e., the action returned is empty).

In order to facilitate discussion and a better understanding, the following scenario and example HPOMDP are presented from our MavHome Steve scenario. An example of the generated section of the HPOMDP is presented in Figure 4.36. MavHome Steve is currently engaged in pattern 4. o is v25_OFF, $\varepsilon = \{v28_ON, v22_OFF, v25_ON, v24_OFF, v22_ON, v24_ON, v24_OFF, c12_OFF, v24_ON\}$, $\chi = v30_ON$, and $\Xi = \{\xi_1:0.769(62435), 0.0(67348), \dots; \xi_2:1.0(237845), \dots; \xi_3: 0.0(1), \dots\}$. Using Algorithm 9 with a $\zeta \leftarrow 0.70$ (a typical value

Algorithm 10 ProPHeT Automation Decision

Require: β belief state, ε event history, HPOMDP model

- 1: $\zeta_{ON} \leftarrow x$ where x is an integer {Tunable horizon for “ON” automations}
 - 2: $\zeta_{OFF} \leftarrow x$ where x is an integer {Tunable horizon for “OFF” automations}
 - 3: $a \leftarrow \emptyset$ {Action to automate}
 - 4: **if** $\beta = \emptyset$ **then**
 - 5: return a
 - 6: **else**
 - 7: Find state β in HPOMDP
 - 8: **while** Following highest probability edges given current state β and ε history up to $\max(\zeta)$ look-ahead horizon steps **do**
 - 9: **if** next state is an automatable action **then**
 - 10: $a \leftarrow$ automatable action
 - 11: break
 - 12: **end if**
 - 13: **end while**
 - 14: **end if**
 - 15: return a
-

for our environments and data stream), we can find a match in episode 62435 (Go to alternate workstation) with a direct trace from episode 237845 and none back to episode 1 (i.e., not tracing back to root is valid for this pattern) through episode membership. A match of o , ε , and χ fits to a belief state three observations before the end of the pattern. The strength of belief is shown in Equation 4.10 which indicates that $\beta_{value} \geq \zeta$; therefore, $\beta \leftarrow o_i$ which is returned as the belief state. Given the evidence, it should be obvious that this is correct. However, not all outcomes are as clear, especially when patterns appear very similar so the ζ parameter must be tuned to minimize false belief states. Also, the match function for ε may be relaxed slightly from exact match in order to accommodate noise and slight imperfections in the observation data stream.

$$\beta_{value} \leftarrow 0.94225 \leftarrow \frac{\left(\frac{9+1+1}{11} + \frac{0.769+1.0}{2}\right)}{2} \quad (4.10)$$

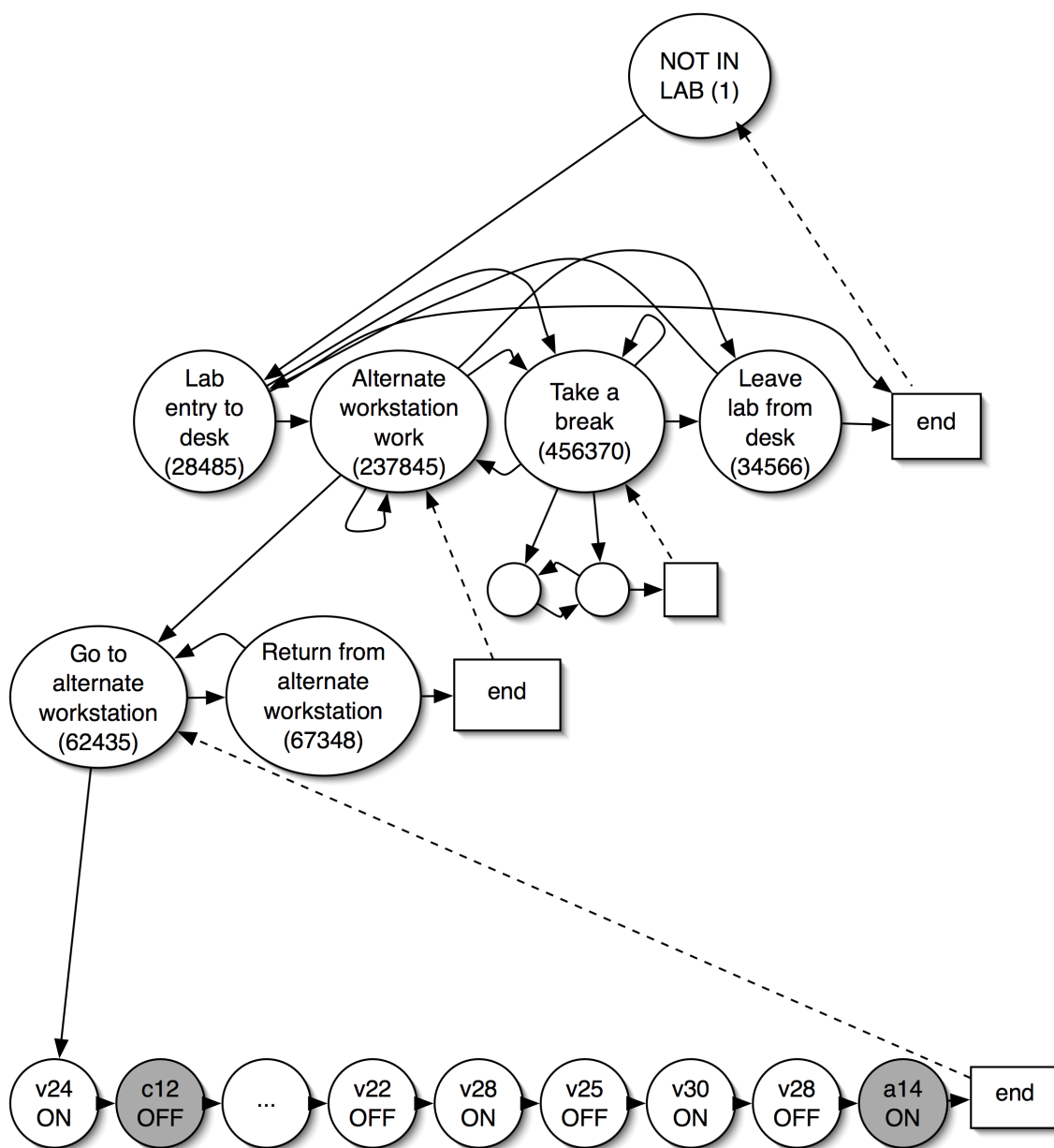


Figure 4.36. MavHome Steve pattern 4 belief state determination HPOMDP.

The belief state, history, and HPOMDP are used in Algorithm 10 to determine an automation. We typically use $\zeta_{ON} \leftarrow 3$ and $\zeta_{OFF} \leftarrow 1$, so these will be used for this example. β is not empty, so if we examine the transitions from β in episode 62435 (Go to alternate workstation), we see a first transition to v30_ON which has no automation, a second step to v28_OFF which has no automation, and a third step to a14_ON which is automatable and within the ON look-ahead horizon; therefore, $a \leftarrow a14_ON$. A decision has been made and will be submitted for action.

These examples should make it clear how this system behaves in a similar fashion to Viterbi in determining phonemes for speech recognition, but with a data-driven learning of the context-providing episodes. Given the previous empirical performance of ALZ [61], ED [73], and Epi-M as well as the derived HPOMDP structure, this is sufficient for automation on the merits of a match from the ε history for the automated action a from the belief state β . However, this action is currently unchecked and may violate safety, security, or user preference and also may be wholly insufficient if the inhabitant begins to change their patterns which is inevitable in the real world. The HPOMDP structure requires a mechanism that can check for validity and furthermore accommodate change for the growth of the system.

4.6.5 Rule Intersection and Feedback

Validating ProPHeT-generated actions is performed by the ARBITER component. ARBITER (**A Rules-Based Initiator of Efficient Resolutions**) contains three sets of rules: system-defined safety and security rules, \mathfrak{R}_{SDSS} , which are coded into the system by the developer to provide a base set of these types of rules for the system; user-defined safety and security rules, \mathfrak{R}_{UDSS} , which the user specifies; and user-defined activity rules, \mathfrak{R}_{UDA} , which simply stated provide a *do as I say and not as I do* mechanism for the injection of behaviors into the system that are put in the event stream from rule-firings rather than by inhabitant interaction. The former types of rules prevent the system from operating objects in an unsafe, insecure, or

undesired manner while the latter rules provide a mechanism for training the system for desired behavior which may not be produced by the inhabitant (but obviously desired)—this type of rule came from our experimentation and the problem of how to train the system for behavior that is often difficult for an inhabitant to perform. In our case, we wanted to train the MavPad to turn off the lights when the apartment was unoccupied, but once we left we could not turn off the lights ourselves.

Algorithm 11 provides the basic operation of ARBITER rule checking. Rules encoded in ARBITER are simple conditional rules that check for the existence of a condition. In the case of \mathfrak{R}_{SDSS} and \mathfrak{R}_{UDSS} rules, a matched condition would prevent the ProPHeT action from being executed. For example, the MavPad contains a halogen light b3 which is never to be automated since if it is left on too long it may become a fire hazard. If ProPHeT issues an action to turn on light b3, the system will prevent the action and issue a negative feedback. In the case of \mathfrak{R}_{UDA} rules, actions are initiated by ARBITER from a set of conditions that when met fire an automation rule. For example, if there is no motion in the MavPad then turn off all lights. \mathfrak{R}_{SDSS} and \mathfrak{R}_{UDSS} rules provide positive and negative feedback in order to be used for HPOMDP adaptation and policy convergence; whereas, \mathfrak{R}_{UDA} rules are anticipated to be learned as patterns from Episode Discovery and added to the HPOMDP at the next reboot, but may be learned through perceived inhabitant-induced feedback. ProPHeT does not consider time, but instead relies upon events in order to progress decision-making, which facilitates the need for an alternate system (e.g., a rule system) for handling non-event based or non-observable event trigger conditions.

4.6.6 User Feedback

Another source of system feedback comes directly from the inhabitant and potentially from the \mathfrak{R}_{UDA} rule firings. ProPHeT stores a queue of executed actions, A . When an incoming

Algorithm 11 ARBITER Rule Check

Require: o current event observation, a action request, \mathfrak{R}_{SDSS} system-defined safety and security rules, \mathfrak{R}_{UDSS} user-defined safety and security rules, \mathfrak{R}_{UDA} user-defined activity rules

- 1: $\mathcal{E}_{ARBITER}$: ARBITER-maintained o history
- 2: $f \leftarrow 0$ {Feedback variable}
- 3: **if** a violates \mathfrak{R}_{SDSS} **then**
- 4: $f \leftarrow$ negative feedback
- 5: **else if** a violates \mathfrak{R}_{UDSS} **then**
- 6: $f \leftarrow$ negative feedback
- 7: **else**
- 8: Perform action through logical proxies
- 9: $f \leftarrow$ positive feedback
- 10: **end if**
- 11: **if** o with $\mathcal{E}_{ARBITER}$ meet \mathfrak{R}_{UDA} firing criteria **then**
- 12: Perform action through logical proxies
- 13: **end if**
- 14: $\mathcal{E}_{ARBITER} \leftarrow o$
- 15: return f

observation o appears to contradict one of the recent actions in A , a negative feedback inversely proportional to the time lapsed since the automation, up to a defined maximum time span, is applied to the part of the HPOMDP that initiated the action. A small positive feedback is applied to actions not contradicted (or, as we say, countermanded). In addition to countermands, we also observe substitutions. For example, our system turns on light a1 after which we observe that a1 is immediately turned off and light a2 is turned on. The countermand causes a negative feedback to the system, but the substituted action a2_ON is also noted. These feedback mechanisms play an important role in adapting the HPOMDP inhabitant model for life-long learning.

4.7 Adaptation

Human inhabitants may be creatures of habit, but those habits are likely to change. As people progress through life they undergo many changes in lifestyle. As they undertake their

journey through life, it is important that any lasting system that seeks to automate a part of their dynamic journey must also change with the inhabitant. Adaptation is the key to life-long learning and assisting the inhabitant to age-in-place. Our system has been designed to accommodate adaptation, both in a near-term and a long-term manner.

The three pillars of adaptation in our architecture center around feedback for near-term adjustment of the current HPOMDP model, monitoring system performance, and continual mining of the data stream to detect new and changing patterns. Algorithm 12 is the main control sequence for the adaptation and continued learning phase. Feedback is used with temporal-difference learning to adjust to drift (i.e., a continual change at a slow rate) in the data patterns. Performance monitoring and continual data-mining using ED are used to adjust to a shift (i.e., a sudden dramatic change) in the data pattern.

4.7.1 Continued Learning with Temporal-Difference

The work in this dissertation involves the use of a reinforcement learning technique called temporal difference (TD) learning and in particular we use TD(0)—temporal difference learning without an eligibility trace. Tabular TD(0) first appeared in work by Ian Witten [223] and later in its more commonly used form by Sutton [201]. We use the Sutton form and do not offer an extension to the work, but use it in combination with the other techniques in our system to create a more useful combinational approach. For more information on reinforcement learning the interested reader should consult Richard Sutton and Andrew Barto’s *Reinforcement Learning: An Introduction* [203].

The goal of the system is to learn the control policy π that 1) minimizes user interactions, 2) eliminates safety and security rule violations, and 3) reduces user preference rule violations. The former two are determined by critic feedback, while the latter is determined by inhabitant system interaction. Resource consumption control learning could be addressed by influencing

Algorithm 12 Phase 3: Adaptation and Continued Learning

```

1: reboot  $\leftarrow$  false
2: repeat
3:   {Concurrent loop 1}
4:   loop
5:     Receive feedback
6:     Examine feedback
7:     Correlate feedback to recent actions
8:     Adjust appropriate event transition based upon feedback
9:   end loop
10:  {Concurrent loop 2}
11:  loop
12:    Monitor performance
13:    Collect information on feedback, performance, and inhabitant pattern changes
14:    Determine if concept shift has occurred
15:    if concept shift has occurred then
16:      Reboot the system (see Algorithm 1)
17:    end if
18:  end loop
19:  {Concurrent loop 3}
20:  loop
21:    Get observation data
22:    Run sliding window of data through ED to look for concept shift
23:    Set warning in the case of a concept shift
24:  end loop
25: until reboot = true

```

proper behavior of the environment by the users and rule influences but will not specifically be explored here.

The learning issues are characterized by Figure 4.37. In this example there is an area of contention in the learned model from ED that must begin and eventually be fully resolved in the HPOMDP because ED will not recognize changes in data patterns until they begin to occur with sufficient regularity. Waiting for these changes would be intolerable to the inhabitant of such an environment and would thus cause an increased amount of interaction. The adaptive learning mechanism in the HPOMDP must be able to handle the following situations:

1. **Countermand (Deletion):** Adapt to a feedback-generated countermanded automation. For example, if the pattern normally turns light a14 on but one day the inhabitant decides that they no longer desire that light to be on and begin turning it off every time it is automatically turned on, the system should recognize this feedback and learn the new preference. This is a change in an existing pattern.
2. **Insertion (Replacement):** Adapt to pattern insertion candidates which include the class of substitution (replacement) candidates (alternate actions). For example, the inhabitant may decide that they prefer light a16 on instead of a14 while working at the alternate workstation. Replacement often, but not always, follows a countermanded action.

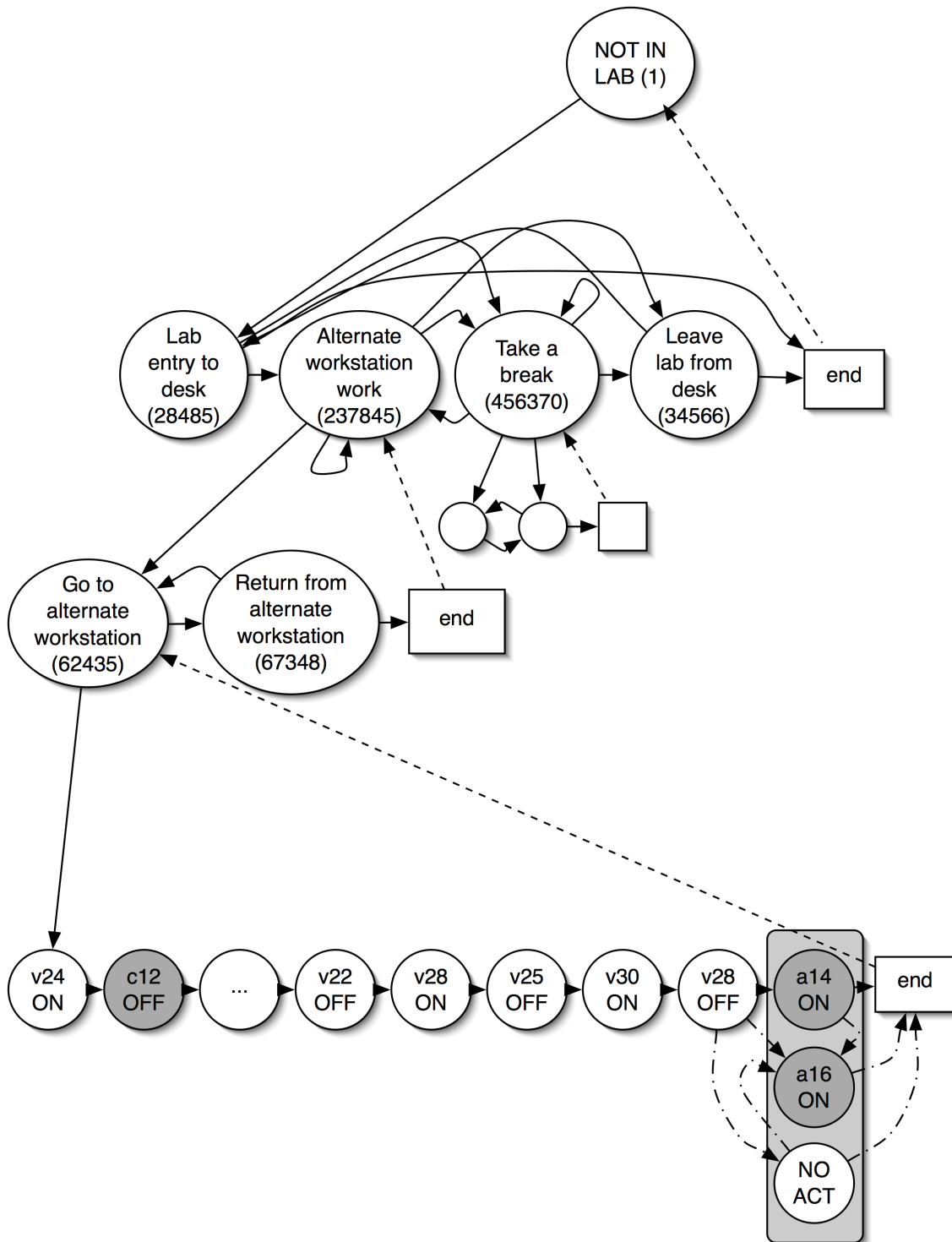


Figure 4.37. MavHome Steve pattern 4 focused HPOMDP with contention area.

In Figure 4.37, the light a14 is subject to a replacement by a16 in the *go to alternate workstation* episode (episode # 62435). The approach for adaptation is to create two parallel states: a16_ON and NO_ACT[ION] (i.e., do nothing). The NO_ACT[ION] state is introduced in both situations listed above. When an event is in contention (i.e., it is either being countermanded or inserted) the first action is to no longer automate it—so the NO_ACT[ION] state acts as a shunt around the area of contention. In the countermand (deletion) situation this state should become the new replacement state thus preventing a consistently countermanded state. In the insertion (replacement) situation, the NO_ACT[ION] acts as an alternative to the new automation until sufficient reinforced observation feedback causes the inserted state to become the dominant state and issue an automation. The new states reconnect back into the chain to all possible downstream states in the chain and through observation learn what the appropriate connection should become.

What the system needs to do is to learn a policy π that takes into account the current observation o , the probability of episode membership Ξ , ϵ history, and the current prediction χ in order to select the best action to take, if any. Since o corresponds to a state in this environment $o = s_i$. So, the learning task can be summarized as the need to learn $\pi : (S, \epsilon, \chi, \Xi) \rightarrow A$.

The principle mechanism for learning in this system is based on evaluative feedback from the environment inhabitant and ARBITER rule feedback. π is determined by user feedback and the arbitration rules. π is guided by the current belief state using Epi-M episode membership probabilities and the current observation as well as recent history, the predicted next event, and the utility for performing an action. Temporal-difference learning [203], TD(0), is used to update state-action utility using equation 4.11 after each action feedback. $V(s)$ is the update to the estimated value for current state, α is the step-size which influences the rate of learning, r is the reward, γ is the discount factor (typical value is 1, no discount), and $V(s')$ is the update to the estimated value for the next state. A simplified formula based on a step-size and discount factor of one is shown in Equation 4.12.

$$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)] \quad (4.11)$$

$$V(s) \leftarrow r + \gamma V(s') \quad (4.12)$$

Feedback learning in ProPHeT is guided by Algorithm 13. Feedback is either returned from ARBITER after submitting an action or through observing an inhabitant or ARBITER-based action in the environment. Direct ARBITER feedback provides either a negative feedback (we commonly use -0.1) for prevented actions or a small positive reward (we commonly use 0.05) for allowed actions. This feedback is applied to the reward transition that issued the action. A NO_ACTION node is also inserted around the action to allow an alternative action if negative feedback is provided—this allows for preventing inappropriate or undesired actions. Proper actions are promoted.

Observed action feedback is determined from a countermanded action (i.e., overturning an action by external initiation of the opposite command action) or an action initiated by an inhabitant in a currently observed event stream (e.g., a new user action is performed in an already observed and recognized pattern of activity). In these cases, a NO_ACTION node is inserted either around the countermanded (or deleted) action or in conjunction with a new action node. Feedback is applied to support the proper path by decreasing the value on paths not taken and increasing the value of the one taken. In these cases, the amount of feedback is dependent upon the time the feedback occurs t_f since the action t_a within a maximum time allowed t_{max} such that $f \leftarrow \frac{t_{max} - t_f}{t_{max}} \times r$ —this provides a time weighted reward so that quick feedback is weighted more heavily than late feedback (i.e., quick feedback is more important and thus given more consideration). The time-weighted feedback is motivated by inhabitant behavior in which when an inhabitant disagrees with an automation they are usually quick

to correct the system and if they do not care they eventually may correct the system. The t_{max} bound is established so that the system does not have to wait indefinitely to resolve any feedback. We typically use values of $t_{max} \leftarrow 60$ seconds (this minimizes difficulties in determining which action to associate with the feedback since automations are usually observed to be spaced more than 60 seconds apart), $r_{positive} \leftarrow 0.05$, and $r_{negative} \leftarrow 0.10$. Through this system of feedback and utility adjustment the system alters the HPOMDP transition values to align with the desired system behavioral policy as guided by the ARBITER rules and the inhabitant.

Algorithm 13 ProPHET Feedback Learning

Require: f feedback, t_f time feedback occurred

```

1: if  $f$  was returned from ARBITER then
2:   {Feedback directly related to action just issued to ARBITER}
3:   if  $f < 0$  then
4:     Insert NO_ACTION node around  $s_a$  {If one does not exist}
5:     if New NO_ACTION node inserted then
6:       Assign  $V(s_{NO\_ACTION}) \leftarrow 0.5 \times V(s_a)$ 
7:     end if
8:   end if
9:    $r \leftarrow f$ 
10:   $V(s_{a-1}) \leftarrow V(s_{a-1}) + \alpha[r + \gamma V(s_a) - V(s_{a-1})]$ 
11: else {Feedback came from inhabitant or  $\mathfrak{R}_{UDA}$  user-defined activity rules}
12:   Correlate feedback to action recently taken
13:   if associated action  $a$  found AND  $t_f - t_a \leq t_{max}$  then
14:     if  $f$  is a countermands (deletion) at  $a$  then
15:       Insert NO_ACTION node around  $s_a$  {If one does not exist}
16:       if New NO_ACTION node inserted then
17:         Assign  $V(s_{NO\_ACTION}) \leftarrow 0.5 \times V(s_a)$ 
18:       end if
19:       {Negative reward to countermanded action}
20:        $r \leftarrow f_{negative}$ 
21:        $V(s_{a-1}) \leftarrow V(s_{a-1}) + \alpha[r + \gamma V(s_a) - V(s_{a-1})]$ 
22:       {Small positive reward to NO_ACTION}
23:        $r \leftarrow f_{positive}$ 
24:        $V(s_{a-1}) \leftarrow V(s_{a-1}) + \alpha[r + \gamma V(s_{NO\_ACTION}) - V(s_{a-1})]$ 
25:     else if  $f$  is an insertion(replacement) at  $a$  then
26:       Insert NO_ACTION node after  $s_a$  {If one does not exist}
27:       if New NO_ACTION node inserted then
28:         Assign  $V(s_{NO\_ACTION}) \leftarrow 0.5 \times \max(V(s))$ 
29:       end if
30:       Insert  $f$  node after  $s_a$  {If one does not exist}
31:       if New  $f$  node inserted then
32:         Assign  $V(s_f) \leftarrow 0.5 \times \max(V(s))$  {Assign the new node the value of half of the
           amount of the current highest transition}
33:       end if
34:       {Small positive reward to inserted(replacement) action}
35:        $r \leftarrow f_{positive}$ 
36:        $V(s_{a-1}) \leftarrow V(s_{a-1}) + \alpha[r + \gamma V(s_f) - V(s_{a-1})]$ 
37:     end if
38:   end if
39: end if

```

As a test case, we ran ProPHeT in simulation using MavHome Steve data and employing an ARBITER rule that will not allow light c16 to turn on (for not-so-obvious safety reasons). Pattern 1 *Lab entry to desk* turns this light on as a learned part of the HPOMDP. When ProPHeT submits the c16_ON action to ARBITER, ARBITER will prevent the action and provide a negative feedback of -0.1. Given the initial HPOMDP production node transitions of v24_ON to c16_ON with a value of 1.0 and a transition of c16_ON to v24_OFF with a value of 1.0, the first rule violation inserts a NO_ACTION node between v24_ON and v24_OFF in parallel with c16_ON and transition values of 0.5 and 1.0 respectively. The transition value of v24_ON to c16_ON is lowered by negative feedback to 0.9. With each rule violation, which normally occurs 3-5 times per day, the transition value from v24_ON to c16_ON is lowered until after five violations the transition utility is lower than NO_ACTION as shown in Figure 4.38. After five violations, the HPOMDP has been altered to not turn c16 ON anymore, unless subsequent operations positively reinforce turning c16 ON. It takes approximately two days to learn the rule due to the frequency of pattern occurrence.

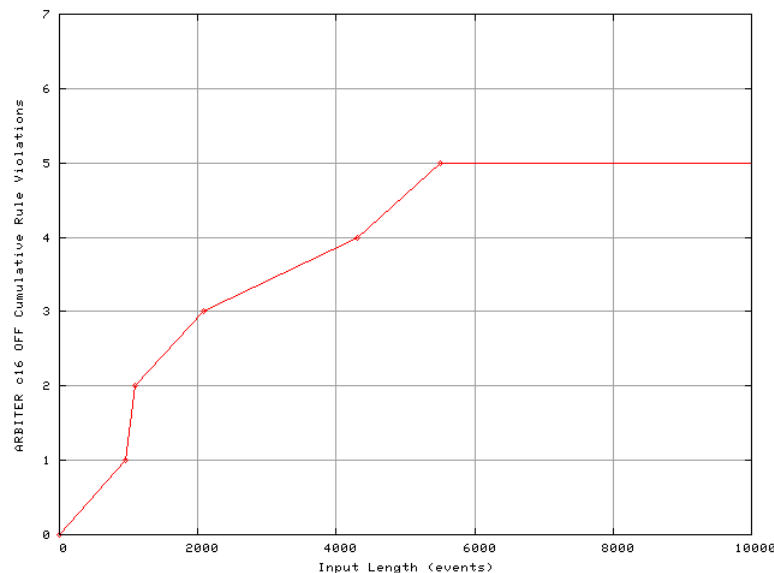


Figure 4.38. MavHome Steve c16_OFF ARBITER rule learning.

In a slightly more complex test case, we used the virtual inhabitant MavHome Steve to turn on light c11 which is located on the desk behind Steve, but controlled from Steve's desk, whenever light c12 is turned on. This should affect patterns 1 and 5 which involve a c12_ON action. Steve is programmed to turn the c11 light on between 1 to 15 seconds after c12 is turned on. When Steve first turns on c11 seven seconds after c12 is automatically turned on, the system associates the c11 with the c12 action and inserts a NO_ACTION node between c12_ON and the *end* node with value 0.5 (since the value of c12_ON → *end* was 1.0). A c11_ON node is also inserted at the same location with value 0.25 plus $\frac{(60-7)}{60} \times 0.05 = 0.044$ for a value of 0.294. Both new nodes are connected to the *end* node with a value of 1.0. Steve's average response time of 7.5 seconds will cause the change for automation to take effect after six interactions—longer if the response is slower as the reinforcement learning takes longer to overcome the NO_ACTION value. This occurs for patterns 1 and 5, both requiring around six inhabitant interactions associated with the particular pattern in order to insert and reinforce the node for automated action. Figure 4.39 shows the inhabitant c11_ON interactions and the associated rule in which the action is being added/reinforced. It takes two days to add the action to pattern 1 and four days to add it to pattern 5 due to frequency of pattern occurrence. The added actions in this process must always be automatable actions. This process can be used in the limit since there is no limit on the growth of patterns; however, it is highly unlikely (based upon observation of real environmental actions) of this growing without bound—inhabitants will not constantly interact with objects. The regularity in learning is somewhat biased by the regularity of the simulated inhabitant, real inhabitants have been observed to be much more inconsistent as seen in the next chapter.

The astute reader may notice an apparent flaw in the function of the adaptation learning mechanism for inserting new action observations—the chances for mismatch may insert nodes in various locations in the HPOMDP. This may indeed occur, but due to the consistency required to reinforce the action in order for it to be automated means that either the nodes are

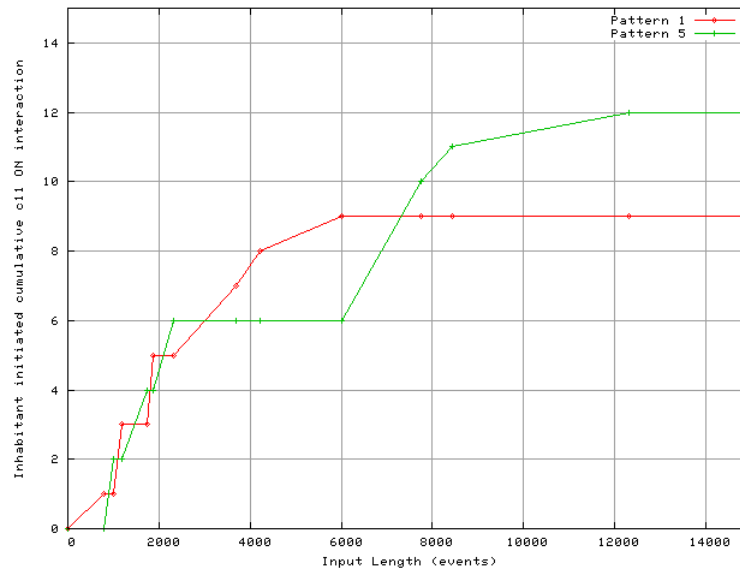


Figure 4.39. MavHome Steve inhabitant feedback initiated learning of c11_ON.

placed in the correctly perceived location to become automated actions or they become clutter in the HPOMDP. Clutter is handled by pruning.

4.7.2 Pruning

In order for state space reduction to occur, episodes and individual states are given a *time-to-live* counter that is refreshed with each traversal of that section of the HPOMDP. Unused sections of the HPOMDP will be pruned for increased efficiency. This assists in removing unused episodes and clutter inserted from feedback learning that may have been placed without sufficient continued reinforcement. Maintaining as clean and dynamic a structured HPOMDP as possible helps promote policy convergence.

4.7.3 Policy Convergence

Our current learning goal is to develop the HPOMDP policy π such that it converges with the policy $\pi_{Arbiter}$ established by the ARBITER rules and the policy $\pi_{inhabitant}$ of the inhabitant and how they live their life—remember that until augmenting technology makes specific

inhabitant identification and pattern separation possible, this is just a single-inhabitant technique. The system rule violations and inhabitant feedback should be minimized as π converges with $\pi_{Arbiter} + \pi_{inhabitant}$. If either of these policies change, then the system will adapt while incurring violations and feedback as necessary to converge with the policy changes.

The learning task in this work is divided into two areas: learning within episodes that have production states and learning within episodes that have abstract states. The learning requirements for production states are described in this section. Learning between abstract states is determined by re-evaluation of discovered episodes through data-mining due to performance and data-compression changes.

4.8 Concept Transformation

The compression of continually-observed data is monitored by ProPHeT in order to detect concept drift and shift. Upon significant shift in data the system will be directed to relearn (i.e., reboot), a new HPOMDP will be generated, and it will replace the current working one. This will accommodate for new learned patterns and overall corrections and cleaning of the structure.

4.8.1 Drift and Shift Identification

Drift is a slow change in the patterns of observations in the data stream over time. Shift is a sudden change in the patterns of observations. Drift occurs naturally in a continually-changing system such as an intelligent environment where the inhabitant gradually forms new habits, stops old ones, interacts differently with their environment, and partakes in the journey of life. Shift occurs when there is a significant departure from the previous patterns of life—change of job, position, stature, injury, and so forth may cause a shift. The key indicators for detecting drift and shift are the compression of the observed data and the performance of

the system. Drift is compensated for by the learning mechanisms discussed previously in this chapter, but shift indicates a departure from the existing model and a need to learn a new one.

System performance measured by the number of correct automations over the total number of device interactions (both by the system and by the inhabitant) when combined with compression observation can be used to determine drift and shift. Dramatic changes in compression but a steady system performance is an indicator of drift. In this case, the system should be allowed to continue functioning without interruption. Counter-intuitively, a sudden change in the compression of the observation data is not an indication of shift, but rather of drift since minor changes in the observed patterns tend to unravel the patterns backward in time reducing compression by expanding their description. Minor changes in compression due to the addition of new, emerging patterns (the shift) are not noticed since their compression will not become prevalent while the old patterns exist and continue to be compressed. If compression remains consistent and performance begins to drop significantly, this is an indicator of shift—actions are not being automated correctly, so the model is incorrect. However, system performance should not be judged too quickly, but should be evaluated over a span of time that will allow for a day or so of differing inhabitant behavior. The threshold for shift detection duration is tunable by the willingness of the inhabitant to be subjected to incorrect automations. After a specified period, if system performance continues to degrade then a reboot should be initiated in order to relearn a new model.

Figure 4.40 illustrates the appearance of shift created using the MavHome Steve virtual inhabitant. At event 3000, half of the six patterns (patterns 4–6) were replaced with their reverse. The result yielded three new patterns that did not significantly change compression, but executed foreign patterns that could not be automated which caused an increase in inhabitant interactions. In this case, over half of the model no longer fits the data patterns. If this incurs prolonged poor performance, then a new model should be learned (i.e., reboot initiated). It is useful to note that in our experience such massive changes to the model are difficult to correct

for through the temporal-difference learning and feedback mechanisms—a reboot and relearn is more efficient.

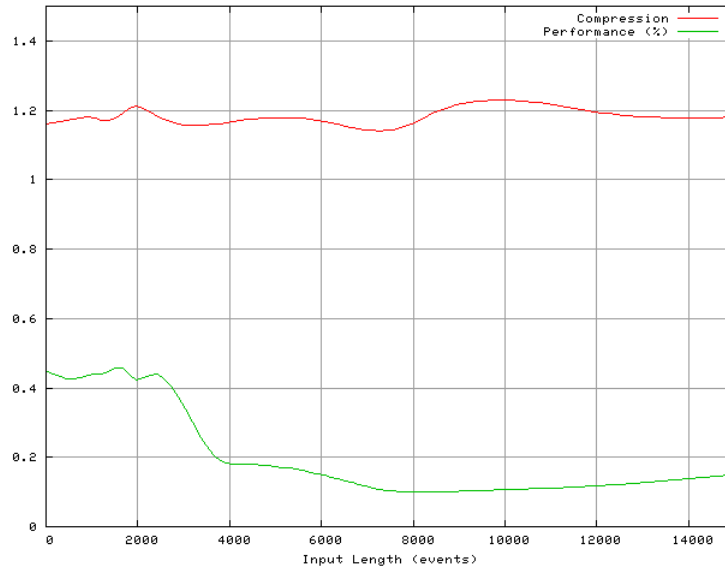


Figure 4.40. MavHome Steve pattern shift example.

4.8.2 Shift Reboot

Complicated measures could be employed to correct a model that no longer provides sufficient automation performance for the currently observed data stream and reflective inhabitant patterns. However, when the model no longer fits the inhabitant it is best to learn a new model by resetting the system and returning to the observation phase to collect more data—new activity pattern data. After the observation period, the system can begin again with Phase 1. Periodic reboot and relearn provides a mechanism for continual life-long learning beyond simple drift compensation and policy convergence through feedback-based reinforcement learning techniques. When performance suffers from improper model fit, the system can relearn and retry—a good human learning concept.

4.9 Complexity

Algorithm performance is an important issue. The computational complexity must be tractable, especially for the components that must run in real-time or near real-time. We provide an analysis of the theoretical complexity of the three main algorithmic phases of our approach and discuss the performance influences associated in each phase.

4.9.1 Initialization/Learning

The *Knowledge Discovery and Initial Learning* phase (Phase 1) starts with retrieving observation data from the database, then filtering and formatting it. If d items were retrieved, then the worst case complexity would be $O(d)$. ALZ is then trained and is reported to have complexity of $O(n^{\frac{3}{2}})$ [62] in the training phase where n is the number of training events. ED is then used to discover the episodes in the data. ED reports a complexity as shown in the polynomial Equation 4.13 for all six phases of the ED algorithm, but with an overall worst case of $O(2^m)$ [75].

$$O(l) + O(n(\log q)m) + O(mq^2) + O(m(\log m)qn^2) + O(q^2(\log q)l) + O(q) \quad (4.13)$$

l : number of events in the input sequence

m : maximum length of all of the symbolic patterns

n : number of episodes generated by the event-folding window

q : number of candidates evaluated by the algorithm

Next, the i interesting episodes are filtered and formatted for a worst case complexity of $O(i)$.

Epi-M is then trained with the i episodes over the input d for a worst case complexity of $O(id)$.

The HHMM is created from the i discovered episodes with a maximum window size of w and maximum number of pattern permutations p for a worst case complexity of $O(iwp)$ and extended to a HPOMDP with complexity $O(iwp)$ for a combined worst-case complexity of $O(iwp)$. The Phase 1 overall complexity is shown in Equation 4.14 and is mostly affected by ED followed by ALZ.

$$O(d) + O(n^{\frac{3}{2}}) + O(2^m) + O(i) + O(id) + O(iwp) \Rightarrow O(2^m) \quad (4.14)$$

Actual runtime performance is characterized in Figures 4.21, 4.41, and 4.42 which shows the runtime costs for Phase 1 on MavPad inhabitant 3 data. This data is representative of typical real-world environment data and some of the more complex data for the system to process. Evaluation of seven weeks of filtered data takes nearly 3 days (70 hours 55 minutes 25 seconds) for ED, 8 seconds for ALZ (note that this performance is usually multiplied by the number of times the data set is repeated to converge to an acceptable training level which is usually near 5 making the actual training time closer to 40 seconds for complex data), and 6 seconds for Epi-M. The discovered episodes (14) take less than a second to process into a HPOMDP by ProPHeT. ED is the largest time consuming component.

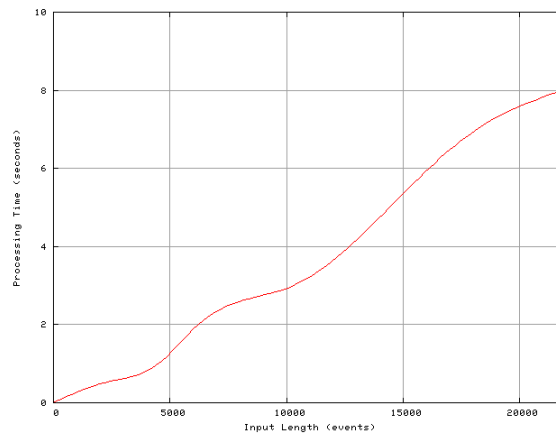


Figure 4.41. ALZ processing time (MavPad Inhabitant 3 Data).

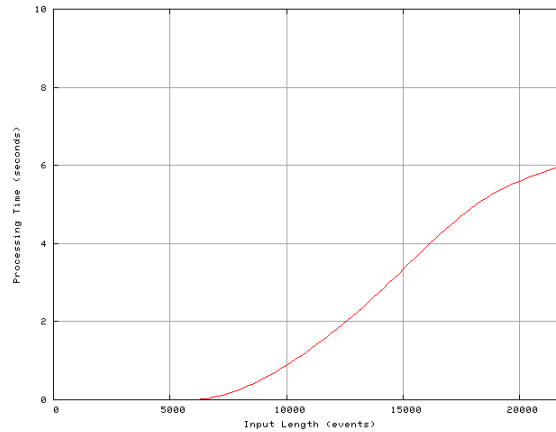


Figure 4.42. Epi-M processing time (MavPad Inhabitant 3 Data).

4.9.2 Operational Runtime

In operation, ProPHeT employs two parallel phases. The *Automation Decision-making Operations* phase (Phase 2) runs *ad infinitum* until the process is stopped. Processing occurs with each received observation event which is used to first determine the prediction of the next observation from ALZ with complexity $O(n)$ [62]. Epi-M membership is then calculated with a worst case complexity of $O(iwph)$ where i is the discovered episodes with a maximum window size of w and maximum number of pattern permutations p and maximum history length h . Belief state is determined with worst case complexity of $O(mw)$ where m is the number of membership matches. The automation decision is made in $O(1)$ time and the ARBITER checks require time $O(r + c)$, where r is the number of rules in the ARBITER rule base and c is the conditions per rule. The Phase 2 complexity as shown in Equation 4.15 is mostly affected by ALZ, Epi-M, and the Belief-state determination.

$$O(n) + O(iwph) + O(mw) + O(1) + O(r + c) \Rightarrow O(n) \quad (4.15)$$

The *Adaptation and Continued Learning* phase (Phase 3) also runs *ad infinitum* until the process is stopped. This phase is characterized by three concurrent loops. The first receives and handles feedback which operates with a worst case complexity of $O(a)$ where a is the action history that must be searched to correlate the feedback—the rest of the algorithm runs in constant time. The second monitors performance to observe and identify shift and reboot the system based on encode heuristics which run with a constant complexity $O(1)$. The third loop continually runs ED over the constantly updated stored observations to observe signs of shift. Due to utilizing ED, this loop runs with a worst case complexity of $O(2^m)$. The Phase 3 overall complexity is shown in Equation 4.16, but is mostly affected by ED.

$$O(a) + O(1) + O(2^m) \Rightarrow O(2^m) \quad (4.16)$$

Actual runtime performance for Phase 2 and 3 computation take less than a second to process each incoming event through all steps on our deployed equipment (Intel 3 Ghz Pentium IV and IBM 2.5 GHz PPC 970) with one exception—ED. Additional ED processing is performed in the background with runtime performance as shown in Figure 4.21.

Slowdowns in the system occur due to the processing requirements of ED to discover episodes, but this cost is initially only paid at system startup (which can take several days). Once ALZ and Epi-M are trained their performance is suitable for real-time operations. Search engages most of the computational complexity, but the compact representation and use of the hierarchy reduce search time to accommodate real-time operations.

4.10 Summary

The detailed methodology of the three-phase ProPHeT algorithm is presented in this chapter with supporting examples and selected test case and case study information. Data flows from sensor and actuator objects in the intelligent environment into a storage database of

observation data. In Phase 1, *Knowledge Discovery and Initial Learning*, ProPHeT selects data from the observation database and trains a predictor, Active LeZi (ALZ), in order to provide next-observation predictions in the next phase; sets the Episode Discovery (ED) algorithm in motion to employ a minimum description length (MDL) data-mining technique to find interesting episodes in the data (those that exhibit either frequent or periodic regularity); trains the Episode Membership (Epi-M) algorithm to identify episode membership of observed events for the next phase; creates a Hierarchical Hidden Markov Model (HHMM) user model from the discovered episodes; and extends this model to a Hierarchical Partially-observable Markov Decision Process (HPOMDP). In Phase 2, *Automation Decision-making Operations*, ProPHeT receives streaming event observation data and develops a belief state of the current observation in the HPOMDP model based upon the current ALZ prediction, the Epi-M indicated memberships across the hierarchy, and the observation history. This belief state is used to look-ahead in the model following the path of highest utility within a look-ahead horizon to determine if an automation action should be issued. Issued actions are checked for safety and security, as well as user preference, by the ARBITER rule engine which executes valid actions and provides appropriate feedback. In Phase 3, *Adaptation and Continued Learning*, the ARBITER and inhabitant feedback is used with temporal-difference learning and expansion algorithms to update transition utilities and pathways in the HPOMDP in order to converge on a policy to satisfy the inhabitant's desires and ARBITER rules which define the system goals. Concurrent monitoring of observation data compression and system performance is used to monitor concept drift and detect concept shift which will lead to a system reboot and relearning of a new inhabitant model. Complexity of this system is dominated by the ED component, but can operate in real-time once knowledge is generated. In the next chapter, we will examine case studies involving the use of this methodology in home and workplace environments with real and virtual inhabitants.

CHAPTER 5

EXPERIMENTAL FINDINGS AND DISCUSSION

The true method of knowledge is experiment.

—William Blake, *All Religions are One*

Through focused experimentation and usage case studies we seek to validate our approach as established in Chapter 3 with the methodology presented in Chapter 4 of this dissertation. Exploratory test cases shall evaluate each component, illuminate an understanding of dependencies, and provide insight into the integrated system. Case studies will establish an applied understanding of system performance. Our goal is to provide an evaluation of our work that validates our approach and provides an understanding of what we have observed.

5.1 Experimentation

Experimentation for this work occurs in isolated test cases and case studies involving both real and virtual environments and inhabitants. All experiments have been conducted in the intelligent environment domain and in either the MavPad (see Appendix C) or MavLab (see Appendix D) real or virtual environments.

Our studies involving our system architecture and associated algorithms and integrated approach take a three phase approach. In general, we are in search of exploratory information on the overall strength of our approach, and where this strength comes from in the different components, as well as the computational and runtime complexity involved. Strength of approach is being proven pervasively through this dissertation to be summarized in Chapter 6 and complexity was presented in Section 4.9. Evaluation of isolated test cases was embedded

in the methodology discussions of Chapter 4 so that concepts are more easily understood by providing contextual evidence of operation. We summarize these findings in Section 5.3. Our case studies provide some of the best insight into our work as well as our biggest challenges. We will examine our case study findings in-depth in this chapter.

In our case study experimentation, we discovered that in order for our system to work we needed to deploy in three distinct phases. The first phase is the *observation phase* where we simply observe the inhabitant and their normal activities in the environment—capturing their interactions over time. This phase lasts as long as necessary to collect sufficient data for data-mining the inhabitant patterns and varies from inhabitant to inhabitant. In Chapter 4, we discussed that this phase persists until there is a consistent compression rate of the data and the number of mined episodes. A good observation phase provides a good inhabitant model. The more consistent the inhabitant, the shorter the observation phase and the better the model fits to the inhabitant lifestyle.

The second deployment phase is called the *ARBITER rule phase* and involves the establishment of the ARBITER safety, security, inhabitant, and behavioral rules. In this phase, some behavioral-driven automation begins—automation that may or may not be learned by the system at a later time. The ED, ALZ, Epi-M, and ProPHeT components are not activated. The purpose of this phase is to check ARBITER performance and ensure that the rules engine is working properly in the environment. Once all issues are resolved and the rules engine is stable, we can enter phase three. Historically, phase two takes between 2-4 weeks depending on the inhabitant and their desires.

The third deployment phase is the *automation phase*. This involves full system activation and operation of all components as described in the previous chapters. This phase continues until a reboot is required in which the system will revert back to phase one. We revert back to phase one because if a reboot is required the inhabitant model was incorrect and needs to be rebuilt from new observed data—phase one is where the new inhabitant model observation

data is collected. After phase three has been initiated, phase two should not have to be entered unless the ARBITER rules are changed in which case a repeat of phase two would be optional. Life-long usage should only require moving between phases one and three with occasional uses of phase two. We have also observed that prior to entering phase one that it is helpful to allow the inhabitant a week or two of acclimation to the environment and the control interfaces before beginning phase one; otherwise, the observation phase will capture some of the inhabitant learning of the new environment which does not reflect continued normal environmental interaction—think about when people first move into a new place how they fumble with the light switches for a while before they find the correct one, after a while they eventually learn all of the environmental idiosyncrasies and can turn on the right switch despite often strange wiring and switch positions.

5.1.1 Human Studies

The MavPad has hosted three real inhabitants who have lived in the apartment on a full-time basis. The first inhabitant helped establish and test the perception capabilities of the environment providing valuable insight into how to collect data, sensor requirements (type, number, and placement), and automation control capabilities. They also helped to capture data which was mined to find the first sets of consistent patterns of life in our intelligent environment. The second inhabitant provided the sets of solid patterns from observing their life, correcting sensor and system shortcomings from our first inhabitant. Good data sets were collected from the observation phase of inhabitant two. They were also the first to participate in the second phase by providing and living under some ARBITER rules. We have run simulation experiments based on the collected data from inhabitant two. The third inhabitant has undergone all three deployment phases, and was the first to be involved in full automation and subsequent adaptation. Currently the observation phase is lasting 2-3 months, the ARBITER phase 1-2 weeks.

The MavLab hosts 4 to 20 researchers on any given day, but we have concentrated on single real and virtual inhabitant studies. This environment serves mostly as our test case proving ground and experimental focal point for new ideas and equipment, as well as our base of operations. We experiment with the real/virtual bridge in this environment more than in the MavPad and thus most of our experimentation involves either hybrid interaction (e.g., virtual inhabitants mimicking real inhabitants or vice versa) or strictly simulation—sometimes we have the virtual inhabitants affect real objects turning them on and off throughout the lab. The real inhabitant in the MavLab is the author.

All human studies in this dissertation were conducted following approved protocols from The University of Texas at Arlington Office of Research.

5.1.2 Simulation

Virtual inhabitants augment the human studies and are used to isolate specific events, change the speed of events departing from real time to scaled faster or slower time, and understand the human trial data. Simulation will be used to first perform isolated test cases before they are explored in actual environments. Simulation will also be used to perform a longer term case study based on patterns observed in the real environments. The Markov model basis for the virtual inhabitants can also be used to validate the learned patterns. For all of our virtual simulations we utilize the ResiSim residential simulation engine which is described in Appendix E.

5.2 Evaluation

MavHome systems have the goal of maximizing the comfort of the inhabitants and maintaining safety and security. We have established metrics to measure the accomplishment of these goals. Inhabitant comfort is measured by the number of inhabitant interactions performed with devices that can be automated. The system should continually strive to reduce the number

of inhabitant initiated interactions. Safety and security are measured by the number of safety interventions from the policy engine, or in other words, the number of rule violations. The system should minimize the number of policy engine interventions (i.e., rule violations should decrease over time). Where applicable we also provide a comparison to other systems in our testing and evaluation.

5.3 Isolated Test Case Summary

The isolated test cases presented in the last chapter help in analyzing the viability and character of our approach. Most of these tests used the MavHome Steve scenarios presented at the beginning of Chapter 4. The Episode Discovery learning rate was explored in Section 4.3.2, illustrated in Figures 4.15, 4.16, 4.19, and 4.20, and revealed that over time the number of discovered episodes will plateau with compression rate as the existing patterns in data are discovered. We have observed that this is tied to the consistency of the inhabitant and subsequent observation data as to the number of events before reaching a somewhat steady state. The more consistent the inhabitant the quicker to convergence and usually the higher the number of discovered episodes. However, in all examined cases, both real and virtual, we have observed ED to eventually find the frequent and periodic patterns in the data. In virtual trials where the number of embedded patterns is known, ED has found 100% of the patterns as illustrated in Section 4.4.2. ED finds patterns that exhibit frequency and/or periodicity and is not limited in the number of patterns it can find. In all cases ED finds patterns in the observation data that do not contain automatable events or patterns of interest to an automation system and they must be filtered to avoid adding unnecessary patterns to the inhabitant model. This filtering is performed with knowledge of the automatable objects in the environment and removes any patterns that do not contain any of those objects.

Automated HHMM construction was presented in Section 4.4.1 which described the process in detail and provided examples of generated HHMMs in Figures 4.29, 4.31, 4.33, 4.34, and 5.8. This description also provided a walkthrough of a generated HHMM from MavHome Steve scenario data as well as providing an example from our case studies. The ED-ProPHeT combination of mining and model construction has been effective at automatically generating HHMMs from observation data in our experimentation.

Virtual inhabitant Markov model learning evaluation in Section 4.4.2 discussed how the MavHome Steve Markov model was learned through observation, data-mining, and HHMM generation using ED-ProPHeT. It was noted that the internal model and the learned model were nearly identical (with minor exceptions due to the stochastic nature of the model and some extra edges discovered that directly related to how the model was actually executed by our simulator), thus showing that our system can learn correct models of inhabitant behavior and that they can be captured in the HHMM structure.

Adaptation learning was combined with rule learning in a MavHome Steve example in Section 4.7.1 that showed the system learning a rule initiated by ARBITER and not part of the originally learned model. As shown in Figure 4.38 our system was able to adapt and learn this rule. Another example in that section on adaptation learning from inhabitant feedback as shown in Figure 4.39 further illustrates our system's adaptive learning capabilities. Concerning the addition of behavioral rule patterns from ARBITER we have noticed that these actions are largely ignored by the system since they do not usually correspond to a local action pattern and are therefore discarded as feedback. However, we have observed that behavioral ARBITER actions appear in the learned episodes when ARBITER is allowed to run during the observation phase. On subsequent system reboots, behavioral ARBITER episodes may be learned and incorporated into the desired inhabitant model.

The issue of reboot due to changes in concept shift and drift is covered in Section 4.8.1. Concept drift is compensated by our adaptive learning mechanisms using temporal-difference

learning. Concept shift presents a significant problem to our model, such that it may render it useless as system performance degrades rapidly. Examination on the tracking and discovery of shift and drift is presented in Figure 4.40. With an understanding of how to identify concept shift or under cases of poor system performance, we can initiate a reboot of the system rolling back to an observation phase in order to learn a better model of the inhabitant—hopefully one with a better fit and higher performance. A smoother transition may be preferred, but since the model is generated from observation data an incorrect model can only be currently replaced by learning a new one. During the learning of the new inhabitant model, systems should not interfere with the inhabitant in order to prevent model contamination. Exploration for alternatives to provide a smoother transition from phase three to phase one is left to future work.

The positive results of these test cases illustrate how our system can learn interesting patterns of activity through Episode Discovery, automatically create a HHMM that closely relates to the inhabitant, can adapt to changes in the desired environmental interaction through inhabitant and rule engine feedback, and identify and adapt the system for concept shift and drift in order to participate in life-long learning.

5.4 Case Studies

A great amount of effort has been expended in reaching the current state of the ideas, theories, and infrastructure components for this work. In our case studies we have had the chance to test our hypothesis in real and simulated environments. These case studies center around three real inhabitants that participated in lengthy studies in the MavPad and the author combined with our virtual inhabitant, MavHome Steve, in the MavLab.

5.4.1 MavPad Inhabitant 1

Our work began with a number of ideas based upon the notion that humans are creatures of habit, that they would generate consistent patterns of activity if observed in an environment, and these patterns could be sensed and recorded. We also hypothesized that these patterns could be mined from the observed data.

Work on the MavPad began in November 2003, and our first inhabitant took occupancy in January 2004. Fortunately, our first inhabitant was also one of the main designers of our Argus sensor network which was wrapping up development at that time. By February 2004, we had installed a full Argus network complete with an overhead, down-looking passive infra-red motion sensor array, temperature, light, humidity, and reed sensors, and mini-blind controllers. We also hosted a complete X-10 deployment which controls lights and fans as well as the HVAC unit. Additionally, we installed HVAC outlet dampers and a water leak detection unit around area with water (e.g., under sinks, by toilet, and so forth). Four computers were installed to interface with the sensors and controls, provide computational capacity, and storage capacity for logging data.

In March 2004 we began the process of collecting data from the sensors, but ran into issues with stability in hardware and software. Work continued through April to improve the system through software and hardware changes and testing. In late April 2004, we were able to start reliable data collection and proceeded to collect observation data from our first inhabitant. In May 2004, the first set of data from the MavPad processed by Episode Discovery revealed that inhabitant activity patterns could be mined from the observation data—a technique that we had only previously believed worked because it worked on synthetically-generated inhabitant data. However, real data appears to be quite different from the synthetically generated data we created at that time (you could almost humanly read the patterns from that data, but not from real data). That first set of discovered episodes revealed motion patterns between the rooms of the apartment, the turning on of the entry light when the inhabitant entered the apartment,

the turning on of the closet light when in the closet, and a half-dozen other motion-light interactions. The important discovery was the characterization of inhabitant activity related to movement, place, and object interaction that could be observed and when it exhibited a level of periodic or frequent occurrence in the data set, it could be discovered as an interesting episode through a data-mining technique. From these initial discovered episodes, the idea of incorporation into a Markov-based framework took root which eventually led to the use of HHMMs.

Initial experimentation with reactive controlled automation based on observation rules began at this point which eventually led to our work on ARBITER. It became obvious at this point that a phased implementation of any developed system would be necessary and also that there would always be automations that probably could not be demonstrated, so that any system that essentially learned by observation would not be able to learn those desired behaviors. This solidified our position on needing a component such as ARBITER which would be able to augment our observation-taught system.

Since our first inhabitant was graduating, his time to move on ended this case study. At the end, we had achieved a complete environment with sensors and controls, had the computing and logging infrastructure to support our work, gained a better understanding of the character of the data in this type of environment, and validated our ideas that a data-mining technique could discover inhabitant activity episodes in real data.

5.4.2 MavPad Inhabitant 2

In June 2004, our next inhabitant arrived. Our second inhabitant would only be a summer participant. After a short acclimation phase (something we learned would be important at the time), we began collecting observation data. Not soon after we started on our second case study, we discovered more stability issues (or instability issues really) with our X-10 sensing and control systems, logging processes (the process kept stopping), and minor issues with the ArgusMS network. Additional engineering effort corrected most of these or made them stable

enough to continue. In July 2004, we were able to collect 16 days of data which was reduced to 10 due to minor sensor network dropout problems and days where the inhabitant was away.

During this time in 2004, our systems were still under development in our lab and were not in deployable condition. From the inhabitant 2 data collected, we evaluated a typical day in the inhabitant's life with the goal of reducing the inhabitant's interactions with the lighting in the MavPad. The data was restricted to just motion and lighting interactions which account for an average of 10,310 events per day. The number of events posed a problem for timely processing by ED, so instead of just using all of the data we made a decision to filter out periods of motion data where no lighting interaction existed. This reduced the data down to approximately 420 events/day. Filtering has now become an important part of the process in order to make the data computationally tractable by ED. There are on average 18 lighting device interactions a day in this filtered data with the remainder being motion information. Using our ResiSim tool which exactly replicates the real MavPad, we trained ALZ and ED on real data and then repeated that typical MavPad inhabitant day in the simulator to determine if the system could automate the lights throughout the day in real-time.

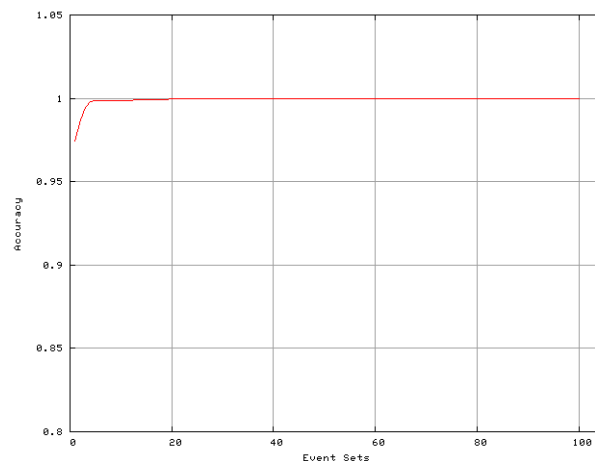


Figure 5.1. ALZ training convergence for inhabitant 2 data.

ALZ processed the entire unfiltered data set and converged to 99.99% accuracy on test data from the data set that was repeated five times for consistency with the ED training set and then repeated to convergence as shown in Figure 5.1. When the system was run with automation decisions being made by ALZ alone, it was able to reduce interactions by one event as shown in Figure 5.5. ALZ performance on streaming data maintained between 40-60% accuracy as shown in Figure 5.2. Performance on streaming data is significantly lower than on training data because the state space is very large and the examples in the observation data that are used to train ALZ initially are only a small fraction of the possible combinations of events. Coupled with some noise and a stochastic inhabitant, performance on streaming data is adversely affected. However, given the difficulty of predicting possible next events, ALZ performs well.determination.

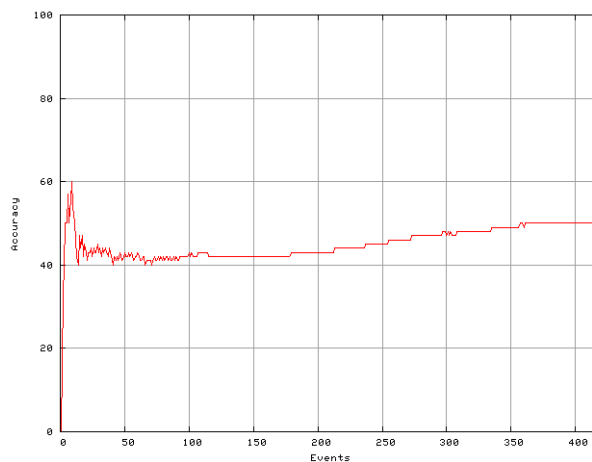


Figure 5.2. ALZ accuracy for inhabitant 2 trial.

ED processed the data and found 10 interesting episodes that correspond to automatable actions as shown in Figure 5.3—three episodes were filtered and the data set was repeated five times in order to enhance frequency for Episode Discovery. It was found that repeating the data five times was the minimum necessary to allow ED to discover the episodes. From this

experiment we learned to filter episodes that did not contain any automatable actions and to ensure that there is sufficient data in the observation period to discover interesting episodes without having to artificially increase frequency.

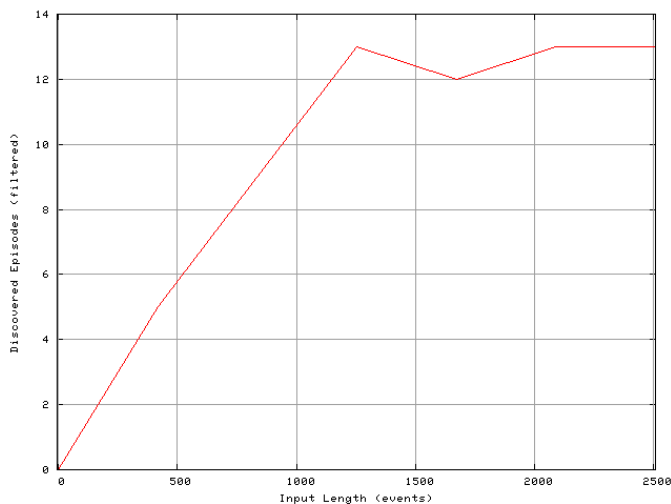


Figure 5.3. Interesting Episode Discovery over time.

The discovered episodes were abstracted through ED to three abstract nodes. A HHMM was constructed in ProPHeT. Figure 5.4 shows this model without the production nodes (it is difficult to show the full models because even the simple models when printed take over seven feet of paper in order to be legible). This system was able to reduce interactions by 72.2% to five interactions. As a comparison, the HHMM produced was flattened and the abstract nodes removed to produce a flat HMM. This HMM was still able to reduce interactions by 33.3% to 12. Comparative results are shown in Figure 5.5.

We also expanded the experiment beyond the automation of just a single day of real inhabitant data to a full ten days under the same conditions. ED was used to process the data and found 12 interesting episodes after filtering. ALZ was retrained and used with similar results. Performance dropped to a 54.9% reduction (83 automations out of 184) for ProPHeT, 26.6%

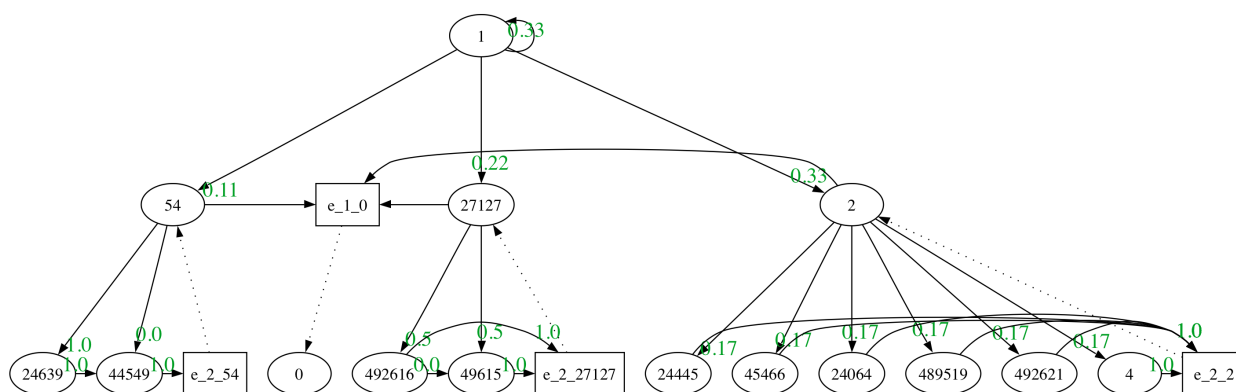


Figure 5.4. Learned HHMM from MavPad inhabitant 2 data.

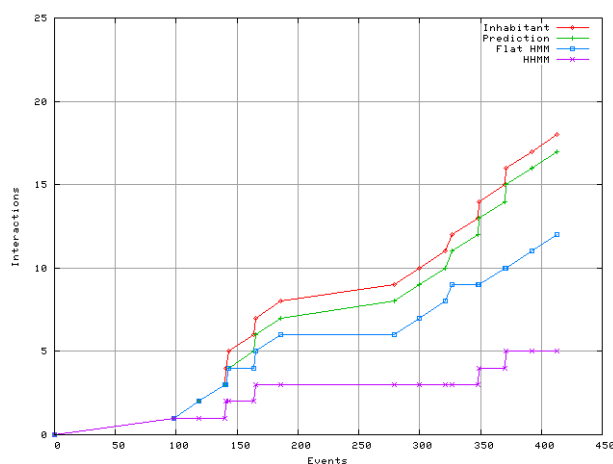


Figure 5.5. MavPad Inhabitant 2 interaction reduction (1 day trial).

(49/184) for a flat HMM, and 10.9% (20/184) for ALZ as shown in Figure 5.6. Missed automations by ProPHeT were due to insufficient belief state for automation due to some pattern performance inconsistencies and two patterns that began with an automatable action. Pattern inconsistencies, as well as inhabitant activity patterns that were not discovered originally because they lacked sufficient frequency or periodicity, are a feature of the real data .

The improvement in performance from the hierarchical model over the flat comes from the enhanced contextual clarity of the hierarchical model which allows replication of nodes

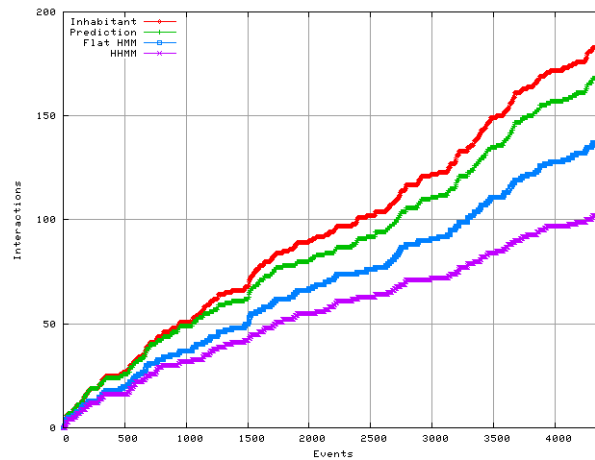


Figure 5.6. MavPad Inhabitant 2 interaction reduction (10 day trial).

that produce the same observation (and if automatable, the same related action) in the HHMM, where each node represents a unique context of inhabitant activity. In the flat HMM, there is only a single node representation for each observation regardless of the context. In a flat HMM, we must rely solely on the probabilistic framework for action transitions taking the route of the most likely next action-observation. In our experimentation, we do apply the same governing lookahead restrictions to the flat HMM as with the HPOMDP in order to provide similar limitations and control without completely following the probabilistic pathways to completion—this is required since our systems are not the primary controller of the environment. Using a hierarchy, we can partition the state space so that context information is encoded in the abstract levels. Using belief state determinations, we can move around the model more freely instead of being locked into the probabilistic transitions. This also aids in temporal handling, since our system can simply wait for observations as they occur without being forced down a probabilistic path too early—we govern the traversals by limiting the lookahead range from the current belief state. As described in Chapter 4, this staccato approach works well in an event-based environment where a system is not the primary controller of the environment, but rather a selective assistant.

The experimentation and analysis presented here is mostly hybrid in nature using real data from the MavPad and integrating it with ResiSim to perform automation in simulation. It was mostly conducted in October and November 2004 after the second inhabitant had left. This was important work in being able to process the data and perform initial automations in simulation. It also validated our hypothesis that using a HPOMDP with lookahead would be able to produce automation decisions. Due to our close connection between the real and virtual worlds, only a slight configuration change is needed to automate the real environment. After the data observation phase, we did perform some additional experimentation with our second inhabitant.

Inhabitant 2 was the first not only to provide a data set that we used for automation, but also field tested the first version of an ARBITER engine. Seven behavioral automation rules were deployed that included turning on the entry light when opening the front door from the outside, turning on the bathroom light upon bathroom entry, and other similar patterns. It was during this time that we discovered that initial programming of the behavioral rules did not always yield the desired results and needed to be tuned. This discovery led us to establish the deployment phase two strategy of an ARBITER rule break-in period. We were also considering ways to incorporate safety and security concerns both from a system and a user standpoint into the system. Our work with ARBITER made this an obvious choice for implementation of these concerns as rules where they could also be easily configured and exist even if the decision-making systems were offline.

At the conclusion of our inhabitant 2 trials, we had learned that our system could mine the observation data for interesting episodes and abstract episodes, these could be converted to a HHMM, and using ALZ prediction and episode membership (from ED during these trials) a belief state could be generated and automation actions issued that reduced inhabitant interactions. We also advanced our work in developing ARBITER, and the MavPad environment became more stable.

5.4.3 MavPad Inhabitant 3

Moving into the MavPad during late August 2004, our third inhabitant has currently lived in the apartment for ten months undergoing various levels of observation and automation. During the course of this time we have worked closely with the inhabitant to develop good ARBITER rules from a safety and security standpoint, user preference, and some general desired behavioral automations not captured by Episode Discovery. In comparison to our first two inhabitants, inhabitant 3 is a very busy person with an erratic schedule that made experimentation very interesting.

We allowed inhabitant 3 some time to acclimate to life in the MavPad and began their observation period in mid-September 2004. We observed the inhabitant through the holiday season until the end of January 2005. We used this extended observation period to collect a good set of data. During this period we also upgraded our Argus sensor network software and completed conversion of all temporary systems over to the architecture described in Chapter 3.

Deployment phase two began with the start of February 2005. Inhabitant 3 specified an initial set of 8 rules that grew to 13 and over the course of two months of tweaking ended up at seven behavioral ARBITER rules. During the ARBITER break-in phase, some of the operating systems that had been in operation for nearly a year started posing problems with the new programs we had developed. A system-wide operating system upgrade and a number of stability improvements had to be made. Firmware updates, improved drivers, and the addition of many fault-tolerant algorithms to our system greatly improved system responsiveness (e.g., the new X-10 interface driver improved response times by 100%) and stability.

We mention these engineering issues to capture our experience during our work and to provide information to those interested or currently involved in this kind of research. In applied science, there is usually a significant amount of time dedicated to strictly engineering issues—our work has not been an exception. In a 24-7 real-time system, stability is very important. We have effectively deployed a system with a high availability requirement using free software

After completing the deployment phase two and starting in late-April 2005, a three week full automation experiment was conducted in the MavPad with inhabitant 3. Seven weeks of data from our observation period was used to train ED which in the raw form consisted of 4,371,179 events (approximately 89,000 events/day, approximately 6,800 events/day excluding temperature, light, and humidity sensors) with 2,163 automatable events and was reduced to 21,863 events (approximately 446 events/day) with 1952 automatable events by filtering unnecessary and noisy data—our filters key on the automation areas and filter out large spans of data where no automations occur, we also filter some duplicate data and data inconsistencies. We learned to filter from our inhabitant 2 case study and have become adept at filtering out unnecessary information. There are on average 40 device interactions a day in this data which include lights, fans, and mini-blinds (tri-state) with the remainder being motion information. ED processed the data for over 96 hours to find 14 production node-filled abstract episodes and three hierarchical abstractions of those resulting in a three-tier HHMM. The model was extended to a HPOMDP, and ARBITER was loaded with two safety and security rules, one inhabitant preference rule, and seven general behavioral automation rules. ALZ and Epi-M were trained—ALZ trained to within 99% as previously observed (the curve is nearly identical to that in Figure 5.1). ALZ performed with between 16% and 67% accuracy over the 4,371,179 events with results similar to those shown before. The performance difference in training and streaming data by ALZ is due to some initial overfit in ALZ and the limited training data compared to possible patterns of streaming activity.

The full system was allowed to automate the environment and was able to automate 39.98% (345/863) of the inhabitant's life as shown in Figure 5.9. Note that over the course of the 21 day case study that nearly two million raw events occurred (1,842,336 events to be exact) with 143,543 events occurring excluding temperature, light, and humidity sensors with many spans of motion activity that involved no automation events. The unnecessary sensors and spans of motion activity were filtered to accentuate the data and provide a better understanding

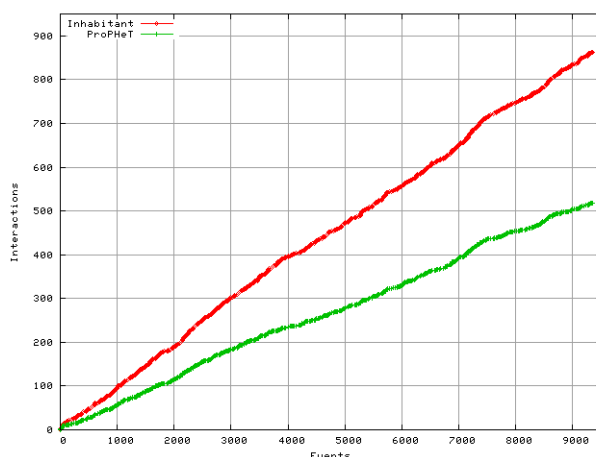


Figure 5.9. MavPad inhabitant 3 interaction reduction.

of the performance difference. Figure 5.9 shows a reduction in the number of interactions that the inhabitant would have performed.

The system learned the two programmed ARBITER safety and security rules and the inhabitant preference rule within six violations with ARBITER feedback. Those rules were all based on preventing specific lights from being automated. The system learned two of the seven general automation rules because they were modifications of existing episodes, but the others were not learned because they did not correspond to existing episodes. With regularity those rules should be found by ED and eventually added to the system on a reboot—we did observe one of the seven show up as a frequent episode in the continuous ED checks for shift and drift. Figure 5.10 shows the reduction of rule firing during the study. Note that in that figure the rule violation curve slope decreases over time as the system learns rules and the number of violations decreases over time. At the beginning of the study a consistent average of five rules per day were fired for what would have been a total of 112 over the experiment, but due to the reduction in violations from the learned rules only 81 actually fired yielding a 27.7% reduction in rule violations.

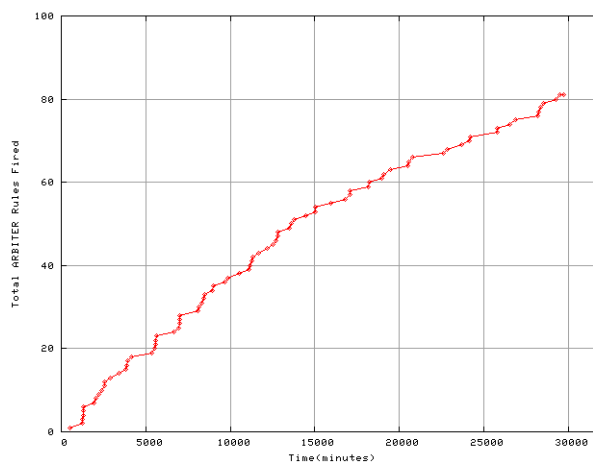


Figure 5.10. MavPad inhabitant 3 rule violation reduction.

This study brought forth some interesting observations. Inhabitant patterns for turning off household items (particularly lights) are much more consistent than turning them on. Object off patterns make up approximately 60% of the episodes and are also more easily learned by the system—all of the adaptations involved turning objects off. Inhabitants are slow to correct the system, and are willing to accept incorrect actions. There is a definite need to work more in a partnership with the inhabitant to meet system goals.

With inhabitant 3, we noticed a new phenomenon in the course of our experimentation—the system did more training of the inhabitant than the inhabitant did to the system. There seemed to be a reluctance to give prompt feedback on the inhabitant end. On interview, the inhabitant said that they were learning to “live in the dark” because it was too bothersome to correct the system. This is probably human nature. We also observed a few fights between the system and the inhabitant over control that ultimately were won by the inhabitant by changing the ARBITER rule causing the problem, but for a short duration the system caused some duress to the inhabitant—not a desired effect.

At the conclusion of our inhabitant 3 trials, in addition to what we had learned in the inhabitant 2 trials we had learned that our system could automate the MavPad environment in

accordance with the desires of the inhabitant and adapt to feedback to learn rules. Inhabitant feedback was insufficient to change system behavior mostly due to timeliness, consistency, and interference from ARBITER rules (which were the automations mostly fought by the inhabitant but are governed by a different set of rules—namely they are set by the inhabitant themselves) which indicates a need for an improved mechanism for communication between the inhabitant and the system. In general, our system works as designed and can successfully automate a single inhabitant intelligent environment as shown by the 40% reduction of inhabitant initiated actions due to system automations.

Future inhabitant studies involve improvements in our ability to log information across all components and to provide a better means of communication with the inhabitant. The ability for the inhabitant to provide system feedback needs to be improved through better means of human-computer interaction so that additional learning can take place and system response can be better evaluated. We need to utilize more of the sensed information and expand control over the HVAC system and other controllable objects in the environment. In general, future inhabitant studies need to help provide a better understanding of how people live and ways in which automation can improve their experience in an intelligent environment.

5.4.4 MavLab Mixed Inhabitant

In order to evaluate our approach in an environment other than a home, we have evaluated a simulated typical week (seven days) in an inhabitant's life with the goal of reducing the inhabitant's interactions in the MavLab, a workplace. Our goal is to show a generalization of our work across multiple environments—at least, home and workplaces for this dissertation. The data was generated from a ResiSim virtual inhabitant using captured data from a real inhabitant (i.e., the author) in the MavLab that was encoded into a virtual inhabitant model. The patterns consist of daily activity patterns involving moving around the lab and interacting with light similar to the MavHome Steve patterns in Chapter 4. Noise was injected between

patterns, but not during the course of pattern activity. The activities were restricted to just motion and lighting interactions which account for an average of 1400 events per day (filtered to 250/day). There are on average 25 lighting device interactions a day with the remainder being motion information. Using our ResiSim tool which exactly replicates the real MavLab, we trained ALZ and ED on the simulation generated data and then repeated a typical week in the simulator to determine if the system could automate the lights throughout the day in real-time. A simulated five week observation period provides the training data set.

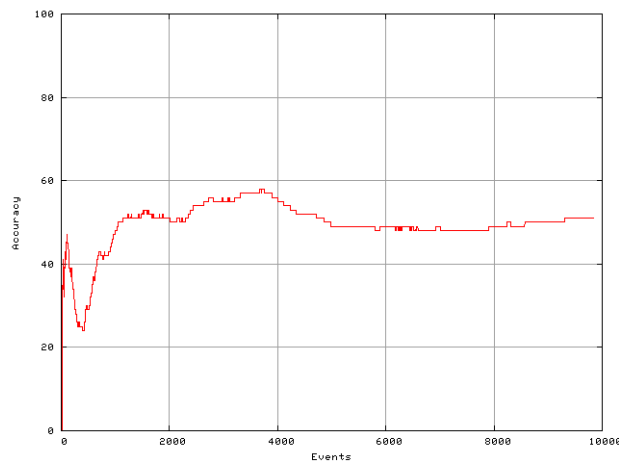


Figure 5.11. ALZ Accuracy for MavLab inhabitant.

ALZ processed the data and converged to 99.99% accuracy during training on test data from the data set in 10 iterations. When the system was run with automation decisions being made by ALZ alone, it was able to reduce interactions by 9.7% as shown in Figure 5.13. ALZ performance on streaming data maintained between 24–56% accuracy converging to around 54% as illustrated in Figure 5.11. ALZ showed consistent performance as with previous studies.

ED processed the data and found 10 interesting episodes, which filter to 8 that correspond to automatable actions (which are similar to those of MavHome Steve presented in Chapter 4).

This was abstracted through ED to two abstract nodes. A four-tier HPOMDP was constructed in ProPHeT and is shown in Figure 5.12. This system was able to reduce interactions by 76%. As a comparison, the HHMM produced was flattened and the abstract nodes removed to produce a flat HMM. This HMM was still able to reduce interactions by 38.3%. Comparative results are shown in Figure 5.13.

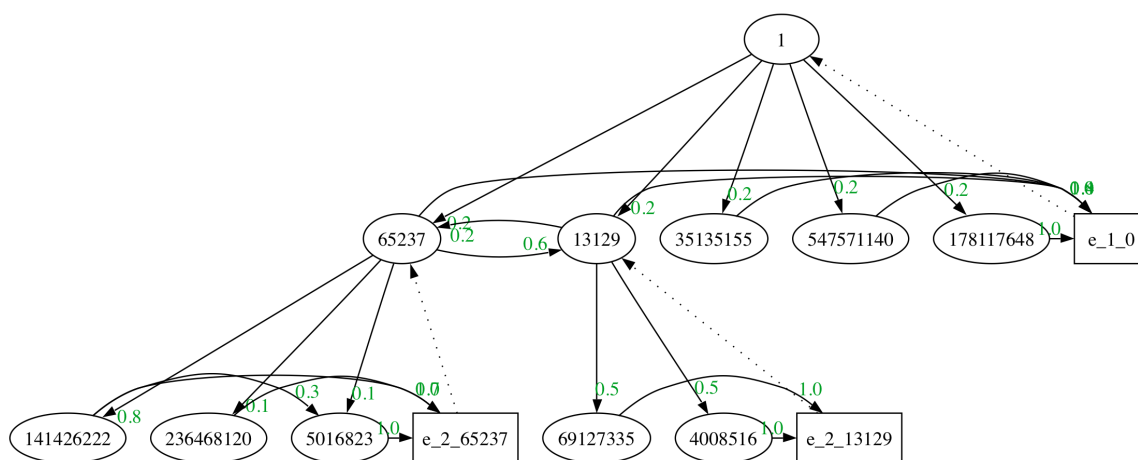


Figure 5.12. MavLab virtual inhabitant HHMM (production nodes omitted).

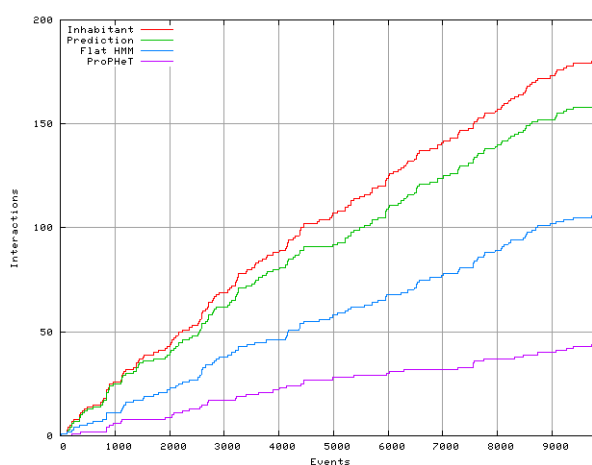


Figure 5.13. MavLab virtual inhabitant interaction reduction.

The additional abstractions in the hierarchy coupled with a next state produced by ALZ and a probability of membership from Epi-M provide input to the belief state to create a system that improves automation performance over a flat model or prediction alone. This simulation provided very clean data with consistent patterns which made for good episode discovery results. Some of the patterns started with automatable events which make them almost impossible to automate because when ProPHeT believes that the episode is being observed, the automatable event opportunity has already passed. This experiment represents some of the best performance of this technique, but it does not include any ARBITER rules—only automation through the learned model.

This MavLab study verifies the work previously discovered in the MavLab studies. What is needed is a more in-depth case that incorporates the full system in a longer term experiment.

5.4.5 Long-term Virtual Inhabitant

There have been a number of our system design features that have not been exercised by our case studies. In particular inhabitant-based feedback and shift detection followed by a reboot has not been explored or tested. In order to provide a more comprehensive analysis of our approach, we have established an extension of the reality-based MavLab virtual inhabitant experiment described in the last section. This study will involve a term of 1 year (52 weeks, 364 days) with approximately 250 events/day after filtering (1400 events/day before), 1 safety rule preventing the operation of an object (a countermand eliciting an action deletion), 1 behavioral pattern rule that initiates an additional object activation when a target object is activated (eliciting an insertion), 1 feedback change initiated by the virtual inhabitant reversing the activation of an object (a user initiated countermand), and a complete model switch at the 6 month point that includes an episode with the event that triggers the behavioral pattern rule, but otherwise contains a completely different set of inhabitant activities. Deployment phases

will transition from one to three with no break-in period for the ARBITER rules (the advantage of a virtual inhabitant—they cannot complain).

The study was initiated by loading the virtual inhabitant model in ResiSim and commencing deployment phase one in simulation to generate observation data. The data was restricted to motion, lighting, switches, reed sensors, and mini-blind control interactions which account for an average of 1400 events per day (filtered to 250/day)—we expanded the sensors, but maintained the same parameters as our previous experiment. These are the same patterns as used in the last MavLab case study with the exception of two patterns being replaced, one with a sensed couch interaction and another with a mini-blind interaction. Again, noise was injected between patterns, but not during pattern activity. This helps to separate periods of activity by the virtual inhabitant, but also more accurately reflects the data stream of real environments providing a more accurate simulation. If we do not inject noise, the time window in ED will usually prevent grouping the activity patterns into single episodes, but patterns that frequently occur consecutively will usually be discovered as a new combined episode by ED. Two different sets of activity patterns were generated for two complete models. The first inhabitant model contains eight activities and the second seven. There are on average 25 device interactions a day with the remainder being motion/switch information. We trained ALZ and ED on the simulation generated data from a simulated five week observation period (49,084 events) and then established a year-long simulation (509,623 events total).

The first inhabitant model was used at the beginning of the study. At the six-month point, we switched to the second inhabitant model after event 254,800. A four percent threshold over a one week period was set as the performance indicator for initiating a system reboot. Figures 5.14, 5.15, 5.17, and 5.18 reflect the impact of the reboot. Figure 5.14 shows the 4.5% reduction in performance over a one week period that triggered the reboot. Due to the runtime requirements in comparison with the simulation speed (we ran this experiment over six hours), ED was not continuously executed in the background to check for compression

changes. We did test data over a two week window before, during, and after the model change and discovered consistent compression values of 1.25 ± 0.03 . Upon the initiation of reboot, the system returned to the observation phase for an additional five weeks, but incorporated the week of poor performance for a total of six weeks (58,807 events).

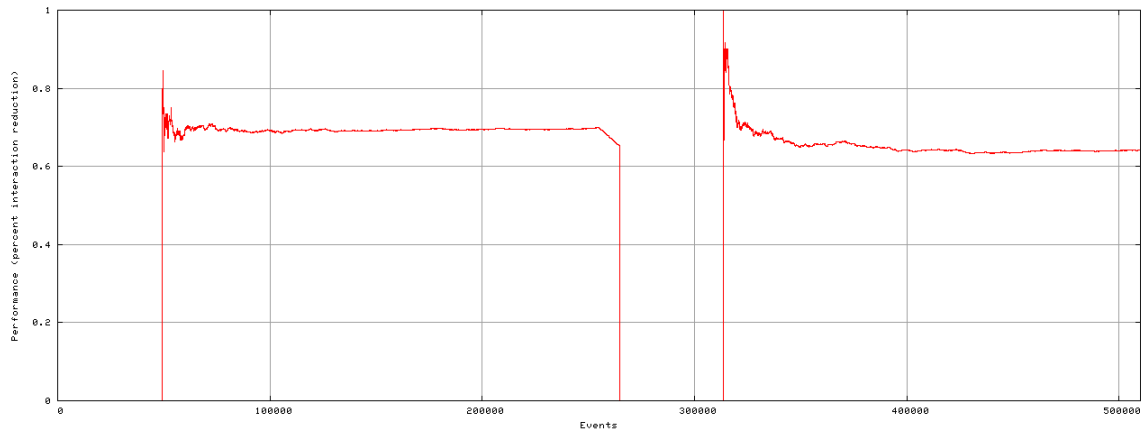


Figure 5.14. MavLab virtual inhabitant automation performance.

ALZ processed the data and converged to 99.99% accuracy during training on test data from the data set during both automation phases. ALZ performance on streaming data maintained between 18–59% accuracy converging to 59.0% in the first automation phase and between 22–59.5% accuracy converging to 59.5% in the second automation phase as illustrated in Figure 5.15. ALZ showed consistent performance as with previous studies.

ED processed the initial observation data set and found 10 interesting episodes, which filtered to 8 that correspond to automatable actions as with the previous experiment. This was abstracted through ED to two abstract nodes. A four-tier HPOMDP was constructed in ProPHeT and is shown in Figure 5.12. After the reboot, ED processed the second observation data set and discovered 11 interesting episodes, which filtered down to 7. ED discovered three abstractions in those episodes for a ProPHeT-generated four-tier HPOMDP as shown in Figure 5.16.

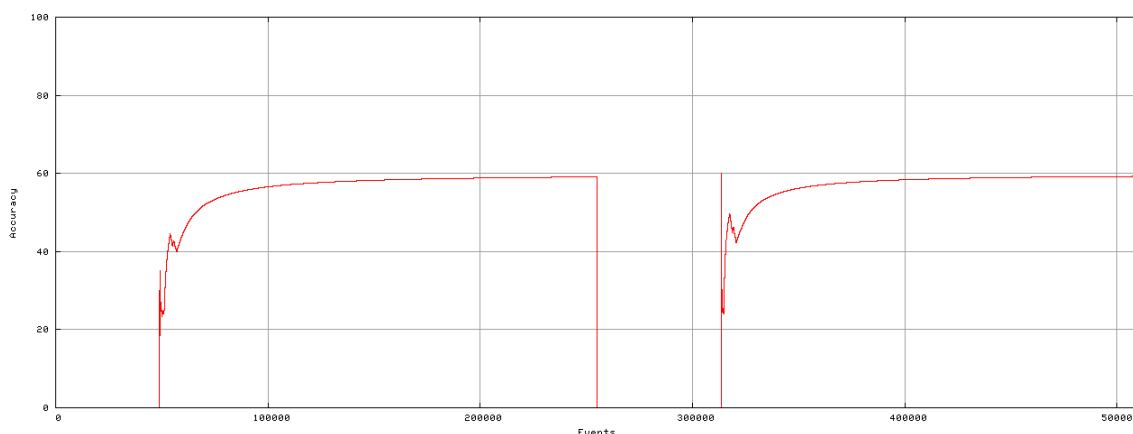


Figure 5.15. ALZ accuracy during long-term experimentation.

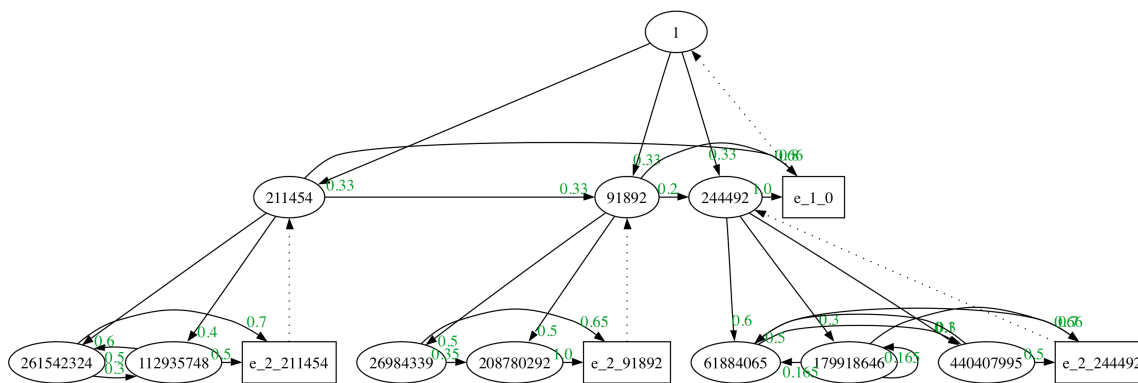


Figure 5.16. MavLab virtual inhabitant HHMM after reboot (production nodes omitted).

This system was initially able to reduce interactions by 69.9%—less than in the first case due to the safety rule, behavioral pattern rule, and programmed feedback preventing some automation as well as variance in the stochastic simulation process. After reboot, the system was able to reduce interactions by 64.2% exhibiting some trouble with two patterns that contained automations early in their pattern sequence—a consistent problem in episode discovery and the nature of many interactive patterns. Clean data and inhabitant consistency aid in the high automation rate. Figure 5.17 shows the effect of automations in reducing inhabitant interaction—the period of automation failure before the reboot can be clearly seen as a marked increase in the number of inhabitant interactions. There are no interactions between phase three

trials since ProPHeT was quiescent during the phase one observation period in order to gather data to construct a new model.

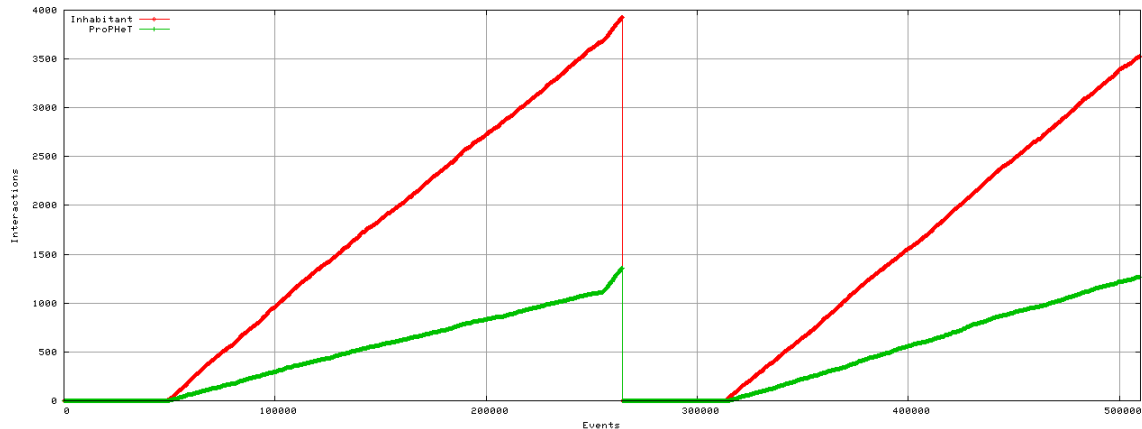


Figure 5.17. MavLab virtual inhabitant interaction reduction during long-term experimentation.

The ARBITER safety rule which was established to prevent turning on one of the lights in one of the discovered episodes was quickly learned in six firings by model adaptation through temporal-difference learning. The behavioral pattern rule which initiated the turning on of a room light after a light located in the same room was turned on was also learned in six firings. We also left this rule on during the second observation phase in order to force a rule-initiated pattern every time the virtual inhabitant initiated the action (the pattern involving the light was the only duplicate pattern in the two inhabitant models). In the second learned inhabitant model, the rule-initiated light was indeed discovered every time with its associated trigger light and was automatically incorporated in the model preventing the firing of ARBITER rules during the second automation phase.

It is important to note that a race condition was discovered between the new automation pattern and the ARBITER rule which initially masked the fact that the system had learned the pattern. ARBITER was modified to allow two seconds to pass between a trigger condition being

met and an action check of the incoming action request stream before firing the action directly from ARBITER. This allows ProPHeT a chance to automate and does not pose a major change in ARBITER operations.

The virtual inhabitant in this study was programmed to provide feedback for a pattern where a light was turned on that the inhabitant did not desire to be turned on—despite the fact that they had turned it on consistently in the observation phase. A variable amount of time between the countermanded action was programmed into the virtual inhabitant, but always within the 60 second window of negative reinforcement opportunity. This is an area where we have had trouble with real inhabitants correcting the system. Upon automation phase initiation, within the first eight automations involving that light, the virtual inhabitant had trained the system to stop those automations through inhabitant-initiated negative feedback.

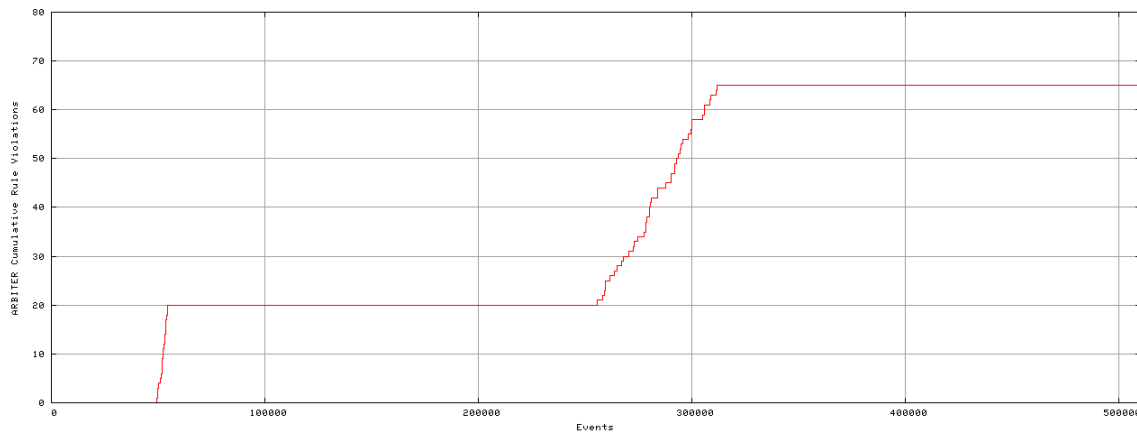


Figure 5.18. MavLab virtual inhabitant rule violations during long-term experimentation.

This study illustrated a typical inhabitant cycle over a period of a year where a system reboot was incurred. Minor adaptation of the learned inhabitant models was conducted. The system performed within operating parameters initiating the reboot when the set conditions were met, adapting to concept changes, and maintaining a high level of system operation through normal operation. Maintaining the ARBITER component in operation during the ob-

ervation phase after the reboot injected a sufficient pattern in the data stream to be discovered in the data-mining activity and become incorporated in the new set of activity episodes in the learned inhabitant model. Maintaining ARBITER active during the second observation phase maintained a level of security, safety, and user desires in the system even when the primary mechanism for learning and decision-making was inactive. This provides an interesting observation, important and consistent rules could be learned and added to the ARBITER rules making it a much quicker and stable mechanism for automation while leaving the more transient automation and life-long learning to ProPHeT—something left to explore in future work.

5.5 Component Dependence

Our approach to developing a decision-making system for the intelligent environment involves an integrated system with many components. It is often difficult to understand the contribution of each component in such systems. In this section we will discuss our observations, present known facts, and perform some limited experimentation to improve the understanding of the component relationships and their dependence on each other.

5.5.1 Core Component Relationship Requirements

There are five main components in our system: ProPHeT, ED, ALZ, Epi-M, and ARBITER. During the initial learning phase ED relies on ProPHeT to provide the data set upon which to perform data-mining and the setting of the widow capacity and time span. ProPHeT takes episode output information from ED and spawns additional instances of ED to sequentially mine the output of each ED instance in order to gain episode data. ED relies on ProPHeT for control (since it is really just a tool). ProPHeT relies on the episodes discovered by ED in order to produce an HPOMDP inhabitant model. ED finds all sorts of periodic and frequent episodes in the data. The more regular patterns that actually exist in the data, the more episodes found. Conversely, erratic data may produce few to no episodes. ProPHeT must filter

ED episodes for those in which it has an interest—those with automatable actions. ProPHeT cannot create an inhabitant model without ED.

ED discovered episodes may begin with an automatable action which is difficult for our system to automate. This is because the belief that the system is in that state may not occur until farther past the automatable action, and certainly too late to automate it. ED could use automatable action information to find some regularity in patterns before those events to provide better episodes that do not begin with automatable actions. In simulation evaluation with very regular inhabitants, ED-based models can automate in the 70th percentile, but this number drops significantly to the 40th percentile for real inhabitants which exhibit less regularity. ED may be missing many episodes in more complex environments. In general, ED takes a very long time to process data. It needs a speed improvement. ED may be too intensive as a data-miner and other techniques may be able to provide the same information.

In the operation phase, ProPHeT relies on Epi-M and ALZ to provide information used to determine belief state. Epi-M and ALZ only rely on ProPHeT to provide training data and current observation data to provide their respective information to ProPHeT. Belief state determination is heavily influenced by Epi-M and ALZ and is discussed in more detail in Section 5.5.2.

All ProPHeT actions go through ARBITER to ensure no safety and security or user preference rules are violated. Without ARBITER, ProPHeT may perform potentially unsafe actions. Removing ARBITER is possible, but would leave the environment potentially unprotected from detrimental decisions by ProPHeT. Environments that do not have rule barriers could employ a system without ARBITER; however, ARBITER also monitors user feedback. If there were no system desires to adapt the learned inhabitant model or be bound by rules, ARBITER could be removed as a component. ARBITER could also be improved over the current version by being able to handle temporal-based rules, improvements in multi-modal inhabitant feedback for actions, and methods to better incorporate user-tuning of rules.

In the adaptation phase, the feedback to ProPHeT from ARBITER is crucial for model modification through temporal difference learning. ProPHeT must have a feedback provider in-order to learn. The ED-ProPHeT relationship also returns in this phase as ProPHeT instantiates instances of ED to explore data compression values in-order to detect concept shift and drift. ED could be replaced with a component that may be able to better identify concept shift.

5.5.2 Belief State Component Analysis

Belief state is determined using observation history ε , the current observation o , ALZ prediction χ , and Epi-M membership probabilities Ξ . Episode membership probabilities are very important in determining which abstract episode the system believes it is currently observing and is tied to our HPOMDP representation. The system will not work without episode membership probabilities. The current observation anchors our current belief and coupled with history are the only absolutes we know at any given moment. History is used to support how we have arrived at our current observation, and it is also used to differentiate permutation branches in the Markov chain. For permutation branch direction determination history is invaluable; however, for episodes without multiple pathways through the Markov chain, it is not necessary. ALZ prediction is used to support current belief by projecting the next most-likely event to occur, but its usefulness could be evaluated.

In the course of belief state determination there are three areas of influence. The first influence comes from Epi-M and the narrowing down of belief to a certain set of abstract nodes with production nodes. At the beginning progression of a Markov chain comprised of production nodes, a sufficient amount of episode membership has to exist before the belief can be sufficiently strengthened by history, the current observation, and the next state prediction. Once a sufficient level of episode membership in a Markov chain is achieved, the next dominant terms are the current observation and the next state prediction because the history length is still small. As the observation events increase history increases, and it becomes the dominant factor

in belief state near the latter part of the Markov chains. Belief state is first guided by episode membership to the production Markov chain at the beginning, the current observation and next state prediction as the observations continue to support the chain near the middle, and then is largely influenced by history near the end.

In order to evaluate the contribution of ALZ next state prediction on belief state we provide the following virtual inhabitant study. We established a short virtual inhabitant study using ResiSim and the MavLab model involving a virtual inhabitant with five patterns. Two patterns that would yield a single pattern with a permutation, two similar patterns, and another regular pattern. All patterns were hand-generated with a length of 10 events and a single automatable event near the sixth position in the chain. An attempt to make the permutation pattern branch to separate automatable actions failed, because ED would identify them as separate patterns. An interesting observation during this experiment was that ED would only generate permutations from cycles or reductions in the event chain and prefer to uniquely identify separate paths involving a single unique event. This is a desirable feature, because complex permutations could lead to automation decisions that conflict given the observed belief—ED prevented our engineered attempt by discovering them as two patterns. We used those two patterns as the similar patterns, and then generated one with a simple cycle over two events. This set of four discovered patterns is representative of the typical patterns we observe in our environments with one exception—we intentionally designed the episodes to focus on ALZ prediction as the key factor in automation decisions by placing the automation decisions at a point where Epi-M provided a path to the correct chain with sufficient probability, but not enough history existed to dominate the belief state determination.

The four patterns were set to randomly fire five times a day. This generated 200 events per day with 20 automatable events. Noise was injected between the patterns to raise the daily event level to 300 and help separate the patterns. We generated 20 days of training data involving 6000 events to perform Episode Discovery. We intentionally kept the patterns

consistent and few in numbers to facilitate their discovery. ED discovered all four episodes and the knowledge was used to build a simple 3-tier hierarchy—no additional abstractions were performed. A test set of five days of data (1500 events) was generated that contained exactly 100 automatable events.

Our experimental focus was to determine the impact of prediction on automation by focusing on the area of key contribution for prediction in belief state determination. Epi-M was set to provide the single top membership from the four episode choices and was trained on the training data and ED output. History and current observation would provide the same level of support as in our earlier experiments. Since we are testing prediction accuracy, we removed ALZ and used a component to provide predictions based upon a look-ahead into the event data stream of the test data with a user-defined accuracy.

We ran an experiment over the test data to determine automation reduction under the heavy influence of prediction in which we varied prediction accuracy from 10–100% in 10% increments. Figure 5.19 shows the influence on prediction accuracy in automation reduction. Figure 5.20 shows correct automations over prediction accuracy. The variability in Figures 5.19 and 5.20 is due to the predictive accuracy being established for all observations and performance only being impacted by those that were correct near automatable actions. The better the accuracy of prediction, the greater the reduction in inhabitant interactions when prediction is the primary factor. If prediction were always 100% accurate it could automate the environment itself. This test provided a uniform distribution of predictive accuracy across all observation including the automatable events; however, in observing actual ALZ performance it tends to gain accuracy from predicting localized repetitions (e.g., pacing patterns and noise patterns with great regularity) and perform much worse over areas with automations. In general, any improvements in prediction would improve our system, particularly if the improvements are in predicting automations based upon observations.

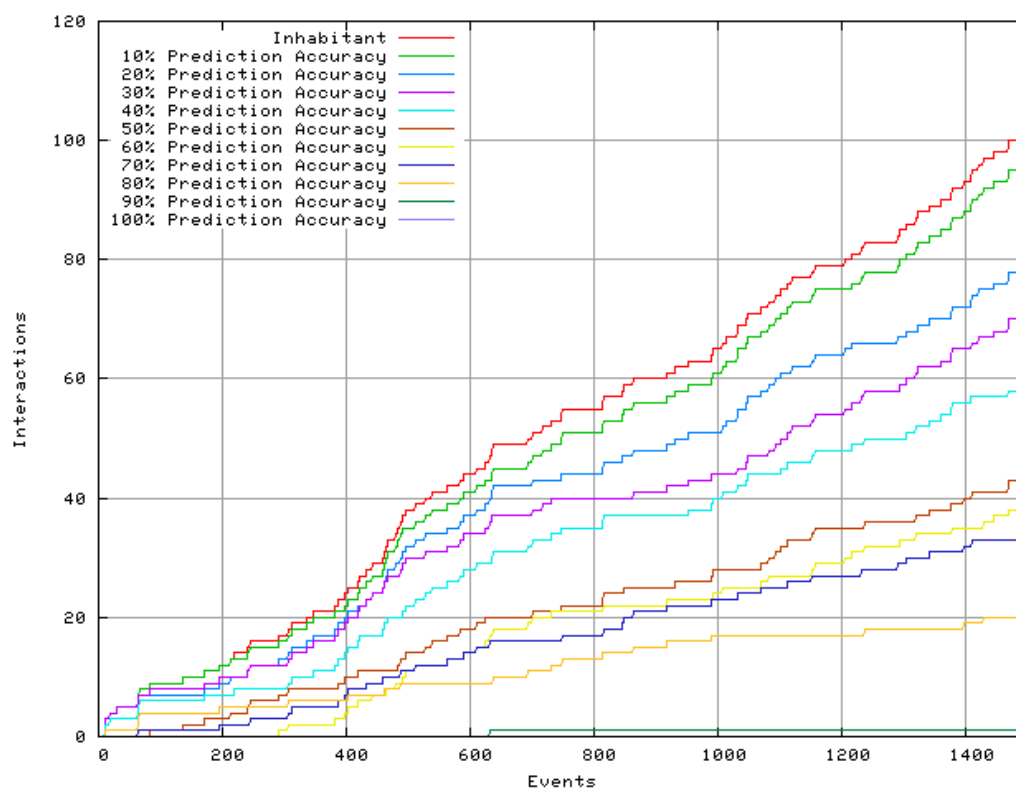


Figure 5.19. Interaction reduction from varying prediction accuracy.

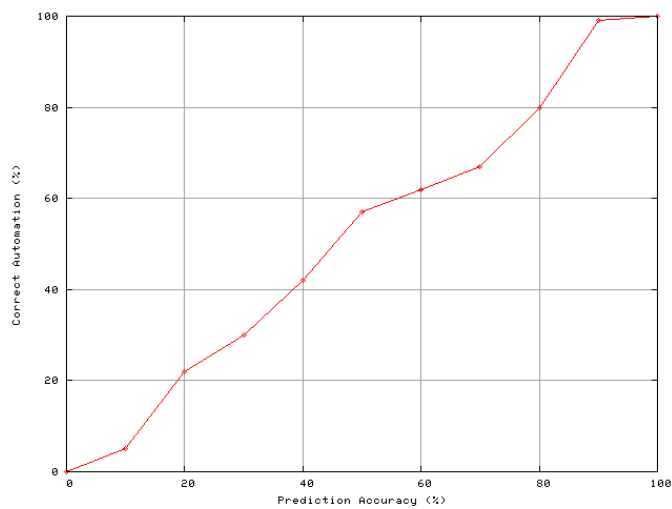


Figure 5.20. Correct automations versus prediction accuracy.

In general, our approach could be improved by enhancements in Episode Discovery (i.e., data mining) which would improve Epi-M selection and ProPHeT inhabitant modeling, as well as ALZ (i.e., prediction) which would improve belief state and automation action selections. Additional evaluation and testing will also yield additional improvements as our approach matures and is generalized to other domains.

5.6 Observations

There are a number of additional interesting observations we have made in the course of design, implementation, and experimentation of the work. The more unique challenges have come from working with our MavPad inhabitants. We did not automate our first inhabitant because we were still concentrating on sensor reading acquisition and determining if there was a perceivable pattern to inhabitant activities—fortunately, there *are* discoverable patterns. It was observing the first inhabitant that made it clear that we needed a good period of observation and that it would take several weeks if not months depending on the consistency of the inhabitant's lifestyle. Our second inhabitant participated for a summer and was automated for two weeks using ARBITER in the MavPad and a very basic version of ProPHeT (without adaptation) in simulation. This inhabitant lived a very regimented lifestyle—even taking showers at approximately the same time every day (surprisingly between 1:00–1:30 A.M.). Our third inhabitant lives a very chaotic lifestyle and has been a challenge for our systems.

Sensor network failure, unreliability, and general chaotic behavior at times forced a lot of effort toward improving the systems by adding additional fault tolerance mechanisms, watchdog timers, performance monitors, and many additional software objects that focus on maintaining high-availability. Sensor instability plagued the first couple of months (and periodically throughout our experimentation), but those issues have been corrected and now the sensors are very stable. One of the biggest problems on the project so far has been the stability of the

power grid and power loss issues in the stormy seasons. Operating system application stability for programs operating 24-7 has also been problematic. Recently, human interactions with the machines (i.e., users turning off machines or forcing reboots) in the MavLab as well as component failures have posed challenges.

A consistent problem we have with Episode Discovery is that many episodes actually begin with the automatable event; thus, automation is difficult if not impossible given our approach in those instances. Interestingly, ED finds many patterns that are filtered due to a lack of automatable actions. There is often a large number of pacing patterns—back and forth walking in the MavPad. The inhabitants, all students, have all admitted to this behavior when they are “thinking.”

Issues with inhabitants not willing to provide timely feedback and allowing the system to train them indicate a need for improved feedback mechanisms and more interaction between the decision-making system and the inhabitant. A general mechanism across all systems that prevent the system from fighting an inhabitant for control also needs to be designed and implemented.

The many modes of human behavior, emotion, and desire that change not only through life, but in many instances from day to day indicate a need for a combinational approach that employs several techniques that work in concert where the best-fit decisions for a given mode are chosen. A one-approach fits all solution will not work in an intelligent environment with a human inhabitant.

There is additional work to be done in the analysis of our approach involving further systemic exploration. More work on the effects of noise, its impact on system performance, and methods for improvement under noisy conditions is needed. Exploration of the effects of drift and shift involving the intensity, rate, and points of occurrence remain open. ProPHeT sensitivity to the components from which it receives information and the impact on system performance from degradation and improvements in those components would assist in better

understanding of the system and aid in pinpointing the areas of most needed improvement. These issues are left to future work.

Another area we also need to examine involves the issue of comparison of the ProPHeT hierarchy with other means of hierarchy generation. The key ability in ED-ProPHeT is to generate and utilize a hierarchy that captures specific activity at the lowest level allowing for high-resolution automation while abstracting those activities at the higher level in order to search in a low-resolution space. Selecting a hierarchy at random would never work in our system since the automation chains would be mangled. A manually-created hierarchy may be possible to evaluate, but it would be very difficult to generate in our stochastic environments. The work that needs to be investigated first involves developing a good measure of comparison of hierarchical models beyond more features than simple performance and second involves comparing other techniques (i.e., data-mining techniques) to generate hierarchies for evaluation. This is a significant amount of future work.

There is still much work to be done and many more hypotheses to test. With every experiment and case study we learn more and more of what works and, more importantly, what does not in the intelligent environment domain.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

Science develops best when its concepts and conclusions are integrated into the broader human culture and its concerns for ultimate meaning and value.

—Pope John Paul II, *Letter to the Reverend George V. Coyne, S.J. Director of the Vatican Observatory (1 June 1988)*

As established in our first chapter, the work in this dissertation focuses around the following central hypothesis.

Inhabitant interactions in home and workplace environments can be accurately automated through sensor observation and intelligent control using a data-driven approach to automatically generate hierarchical inhabitant interaction models in the form of HPOMDPs and these models may be modified using reinforcement learning techniques to continually adapt to changes in the inhabitant's patterns until a new model should be generated.

We started this dissertation with a statement of our mission and goals and went on to provide an introduction to past and current intelligent environment work that focused on a wide range of projects and initiatives from frameworks to applications to gadgets to healthcare to learning and adaption as well as including the bigger setting of a pervasive computing future. We then conducted a presentation of our learning architecture that started with an analysis of the intelligent environment problem, provided a solution that involves data-driven generation of advanced user (inhabitant) models, provided an abstract and concrete framework for the system, elaborated on a multi-component architecture that introduced our ProPHeT (decision-maker), Episode Discovery (data-miner), Active LeZi (predictor), Episode Membership (mem-

bership classifier), and ARBITER (rules engine) components, and ended with an introduction to our experimentation environments.

We further described in detail the methodology of our work from data characterization and flow through system control, training and learning in the components, model generation of hierarchical hidden Markov models (HHMMs), model extension into hierarchical partially-observable Markov decision processes (HPOMDPs), a walkthrough of events through system automation, system adaptation through temporal-difference (TD) learning, observations and handling of concept transformations involving concept shift and drift, and ending with an analysis of system complexity and runtime performance. To demonstrate these ideas, we provided case studies and analysis discussions involving both real and virtual inhabitants in real and virtual environments through both isolated test cases that examined Episode Discovery learning rates, automated HHMM construction, virtual inhabitant Markov model learning, adaptation learning, rule learning, and refactoring due to concept shift or drift, as well as case studies involving three inhabitants in the MavPad, mixed real/virtual experimentation in the MavLab, and a long-term simulation study.

The work presented, analyzed, and discussed in this dissertation supports satisfaction of our hypothesis by our MavCore components and supporting infrastructure. We have shown system-initiated light, fan, and mini-blind automation in both home (i.e., MavPad) and workplace (i.e., MavLab) environments that corresponded with inhabitant desires by reducing the number of manual inhabitant interactions over time. Automation decisions were made by mining observation data and generating hierarchical inhabitant models. Furthermore, we have shown adaption over time of these models to adjust to concept drift and identify concept shift indicating that a new model should be generated.

6.1 Conclusions

Overall, our approach, design, and experimentation provide a level of environmental automation for both virtual and real inhabitants from a data-driven automatically learned model that can adapt to user pattern changes over time. The key strength of our work is that it does not require a human to create the model or for knowledge to be created in the system for the model to be generated. A minimal amount of knowledge is required to automate and adapt—namely the automatable actions. Our model is also not state restricted since we do not consider all possible states but just the states actually observed. However, the challenges we are faced with are difficult. Our data-driven model is only as good as the data that is used to generate it. The less consistent the inhabitant, the less ability there is to automate their life. Our techniques are also not very noise tolerant having difficulty discovering and identifying episodes in noisy sensor environments. The intelligent environment domain presents some very difficult problems. We have provided an approach with some success at automation and insight into the unique challenges ahead.

6.1.1 Strengths

In general, our primary contribution is the automated generation of the inhabitant model which attests for the general sound feasibility our approach. We have shown that by using a data-mining technique (Episode Discovery) that we can identify patterns of regular periodicity and/or frequency and that these correspond to consistent patterns of activity in the intelligent environment domain. In fact, these patterns can be human identified as the normal activities of life such as leaving a room, watching television, or even playing video games. Performing data-mining on these discovered pattern sequences has also proven valuable in finding patterns of these activities. Our data-driven, automated construction of inhabitant models from observation data is a unique approach. Encapsulating this discovered information in a hierarchical structure suitable for modification through reinforcement learning techniques has proven

valuable for 1) translating observations into a belief state within the structure for event automation using lookahead and 2) modifying the structure through temporal-difference reinforcement learning (machine learning) techniques to adapt to minor concept changes (i.e., drift). Our ability to track performance and concept changes to initiate automatic system reset is an important contribution to developing life-long learning systems.

The architecture we provide solves many long-standing issues. Our use of zero configuration technology reduces the construction time for new environments and presents unparalleled flexibility in design and application. Modular construction allows for rapid replacement and testing of new or better components creating a pathway for streamlined improvements. We have also solved the issue of how to bridge reality with virtual reality. Utilizing logical proxies and zero configuration technology, our systems can interact in both real, virtual, or mixed environments. Overall, our system design, architecture, and implementation provides the flexibility needed for continued advancement of intelligent environment research.

6.1.2 Challenges

Our largest strength also presents our largest challenge. Episode Discovery (sequence data-mining) is a very time intensive process that poses significant challenges in large domains with large data sets. In some cases computation may be intractable causing our approach to be invalid for these large environments. Our approach also depends on a level of inhabitant consistency from observation to automation and beyond. Environments hosting inhabitants with inconsistent lifestyles will require a different type of control system. Inconsistent inhabitants present many challenges in the observation data used for learning since fewer patterns can be mined and when automated may no longer be consistent with the current inhabitant behavior. Our techniques are more suited to consistent inhabitants which usually consist of those later in life.

Noise in the data stream also poses a challenge as it masks identification of the current episode of activity and interferes with belief state tracking and thus proper automation of an environment. Improvements in environmental design, sensors, and improved noise cancellation and filtering techniques pose significant research challenges.

The lack of interaction and pure following of probabilistic and learned models challenge the automation capabilities of our system. Augmenting with careful interaction with the inhabitant could improve system performance by eliminating some of the guess work in automation. This could also assist in better model refinement since some patterns are not possible to automate based on observation alone. This especially applies to episodes which begin with an automatable event, which the system cannot automate because it has no prior context until the event occurs. If inhabitants can provide interaction signals (e.g., spoken commands, gestures, and so forth) as automation cues, automation decision accuracy could be improved.

Our work is just one possible approach in a new area of intelligent environment research. We face many challenges now and ahead which serve as motivation for continued research in this area.

6.1.3 Final Analysis

This work makes use of combined techniques in order to form a more powerful system of techniques. None of the individual components of the system in the work of this dissertation are powerful enough to solve the problem by themselves, but in combination using the strengths of each approach in a unifying framework under intelligent control they are able in concert to provide a particular solution to the problem at hand. We consider this a Gestalt system where the whole is greater than the sum of its parts.

Observing the related work mentioned in this dissertation, the reader will observe many system approaches to solving problems and even combinational approaches using several interwoven techniques. The integration of techniques to form more powerful systems is an emerging

area in which we are making a contribution with this dissertation. We hope that more work of a combinational nature will begin to surface in the artificial intelligence community as well as studies into the science of integration.

We have presented a complete system including architecture, algorithms, and supporting examples, test cases, and theory for the automation of single inhabitant home and workplace intelligent environments. In our work, we provide a unique method for the automatic data-driven generation of hierarchical inhabitant models and mechanisms that provide advantageous use of the knowledge structure for environmental automation and life-long learning through fine (adaptation) and coarse grain (reboot) learning. We have proven the effectiveness of this approach in both real and simulated environments with real and simulated inhabitants. This approach has been effective in home and workplace environment and should perform well in any event-driven environment where an actor interacts with interactive feature points of the domain. However, research in this area is still in the beginning stages, and we have yet to realize the full potential of intelligent environments.

6.2 Future Work

There is much work to be done and much knowledge to be gained in the intelligent environment domain. Our work has only scratched the surface of the possibilities, approaches, and potential rewards of such work. At the very core of what we are working on is the chance for a better world for humans to live, work, and play, both today and in the future. The promise of a more interactive world that actively seeks to improve the human condition and our personal experiences is a very powerful incentive to work in this area.

Throughout this dissertation we have discussed not only our approach, its basis, and strengths but also its weaknesses and areas of other potential research. Our modular system construction allows for easy replacement of the data-mining, prediction, and episode member-

ship components in which improvements would benefit the system as a whole. In addition to improvements in our existing components there are a number of fascinating directions that our work could explore.

6.2.1 Resource Consumption Reduction

Beyond considering the gratification needs of the inhabitants of an intelligent environment, we could introduce goals to satisfy specific interests such as reducing resource consumption. This may be a goal of the inhabitant, the inhabitant's parents (if they are a dependent), or an outside interest such as a company or government.

Energy efficiency in an increasingly energy-demanding world adds value to the intelligent environment by reducing its cost over time. Based upon the work presented in this dissertation, it is easy to see how electricity could be saved by only having lights on when necessary, fans on as needed, and mini-blinds open to provide light instead of lamps. Adding water heater control to heat water only when necessary could conserve electricity or natural gas. Improvements in HVAC system control along with air distribution control (e.g., through damper control) so that environments are only cooled and heated in occupied locations, as needed, and maintained within desired temperature conditions could improve HVAC overall system efficiency and electric/gas resource consumption. Water consumption reduction could be achieved by recirculation piping and control systems as well as controllable faucets with activity sensors. Any gains in energy efficiency will also need to be evaluated against the cost of energy to run the sensors, actuators, and the computational reasoning systems.

Improvements in resource consumption reduction partly depend on advancements in control and sensing systems, but there is also a need for innovative solutions that improve the efficiency in which resources are consumed and minimize the rates of the consumption. As we face a future of the depletion of some of our fossil fuels, better conservation of replacement sources will ensure that the lights in our homes will continue to work in the future.

6.2.2 Multiple Inhabitants

One drawback to our approach is that it is based on a single inhabitant. Unfortunately, this paradigm is not how most of us live our lives. Very few people live completely alone, and if they do, many have pets. Even bringing friends into our intelligent environments poses a challenge because multiple inhabitant patterns will not match any of those previously learned. It may be possible to learn a pattern initiated by a group of inhabitants, but this is challenging since it is unlikely that several individuals will interact with the environment in a consistent manner over a given periodicity. A good solution to maintaining environmental automation in the presence of many inhabitants is to isolate the activities of each and associate their patterns with their learned models (if available).

The key to isolating individual inhabitants is to localize them and associate the local observations. Inhabitant localization is a very challenging problem. Wireless signal tracking, vision processing, or electronic tagging methods can be applied similar to other projects as mentioned in Chapter 2, but none of the current technology is perfect and there is still much work to be done in this area. It would be interesting to start incorporating and testing such technologies with our approach in order to discover solutions and challenges.

6.2.3 Health Care

At this point in our research we have primarily focused on environmental sensors, but we could also include sensors that directly monitor the health and welfare of the inhabitant. Health care is a major issue in our lives. One of the motivations for our intelligent environment work is to help people live in their homes longer. A key aspect to that would be the identification and treatment recommendation for health issues. Most health care is reactive instead of preventative, we go to the medical doctor when something is not working properly—often much later than we should. Yet, early detection is very important for improving the survival

rate of such afflictions such as cancer. The development of sensors that can monitor aspects of our health, store that information, and reason about changes could help prolong our lives.

Imagine a future where all cancers are detected early enough that cancer deaths become uncommon or are very well planned in advance. A future where your physician receives detailed information on your health status (e.g., diet, heart rate, blood pressure, and so forth) for as far back in the past as necessary to make a proper diagnosis. Outbreaks of diseases can be tracked and proper immunizations and warnings deployed to minimize the spread. Mental decline can be better understood, and automation can be increased to provide improved assistance and safety. Technologies that can assist in understanding human health in non-invasive manners are vital to realizing this future. However, this is still a very open area of invention and research.

6.2.4 Synergistic Interaction

In our current work, our interaction with the user is through the control of objects in the environment mostly based on what we believe to be the correct course of action due to historical observation. However, we ignore a very important fact about the inhabitant—that they already do know their desired course of action. Working more closely with the inhabitants through natural language processing, intelligent interfaces, gesture recognition systems, and so forth could enhance the experience for the inhabitant and minimize control error.

Our role as an automator may be incorrect. Most humans tend to want to interact more closely with the systems, having them act as a direct enhancement of their clearly stated wishes rather than relying on them to control their environment based on the best guess at the time. The path to a more synergistic interaction seems more natural and should be examined in more detail.

6.2.5 Other Domains

Workplace and home environments are just two examples of environments that can be automated, but just as there are already automated factories there are many more environments to explore. Environments such as smart hospitals, restaurants, and the great outdoors provide fertile ground for further intelligent environment research. Beyond other environments there are also other domains in which our techniques may apply. The computer operating desktop seems like a domain with similar qualities—a click stream of events around interactive feature points of the environment. Workflow automation may also be possible in domains such as software engineering where an engineering process guides sequences of events in the development process. Domains characterized by a level of consistency in an event-driven space are good candidates for application of our techniques and further study.

6.2.6 Intelligent Neighborhoods and Beyond

Extending beyond the home to the neighborhood, city, state, country, and world holds the possibility for exploring intelligence on different granularities and decision boundaries. Intelligent homes can be grouped into intelligent neighborhoods where shared resource concerns can be addressed as well as community issues such as litter control, water, and sewage. Intelligent neighborhoods cluster into intelligent cities that can reason and manage city services, traffic control, and emergency management. Intelligent cities become valuable assets to intelligent states which control inter-city transportation, state laws, health and welfare, as well as large-area emergency management. Extending to the intelligent country and eventually the intelligent world, a future of better communication, better governance, coordinated resource control, and improved emergency response awaits.

6.2.7 Enhancement Technologies

Areas of technology improvements that would enhance this work include such items as sensors, appliance control, object and personnel localization, wireless, broadband adoption, and so forth. Advancements in home, control, and automation technologies provide ample areas for further investigation. The aging X-10 system is in desperate need of a replacement technology. Our Argus sensor networks could benefit from improved speed, wireless capabilities, and improved flexibility. Home appliances are in need of control and information interfaces. There is a wealth of opportunity in converting and improving everyday objects into objects that could be controlled or provide information to an intelligent controller. These enhancement technologies are needed in order to advance work in the intelligent environment domain.

6.2.8 Unprocessed Data

In our established environments not all sensor data is used. In fact, we filter out the majority of data because our systems cannot currently utilize the information. However, there is a lot of hidden value in the data that remains unused. For example, we can identify individual sleep durations and restlessness, patterns of shower usage, HVAC configurations and usage, room usefulness through time spent in each room, and many other potentially valuable nuggets of information. We have become very adept at capturing data, but still have a lot of work in utilizing all of it.

6.2.9 Security Issues

Security of the home and the data contained within the system is an important aspect to all inhabitants. Work in how to utilize the automated systems to mimic the inhabitant(s) when away from home and to detect possible intruders could assist in improving the security of homes and workspaces. Security around the captured observation data and the learned in-

habitant models needs to be safeguarded from those that could exploit the information, but also shared with those who could use it in support of the inhabitant's needs/desires. Thieves could exploit regularities in the inhabitant's activities for opportunistic gain; however, a personal physician or caregiver could use the same information in order to evaluate the well-being of the inhabitant. Work in this area should focus on ways to use inhabitant information to improve home security while protecting inhabitant data from nefarious entities and making it accessible to trusted ones.

This dissertation as a whole represents many years of hard work and dedication. The work presented in this dissertation represents part of the beginnings in advanced intelligent environment research. There is much more work to be done with great possibilities for an immediate and long-lasting impact on the quality of life for everyone.

APPENDIX A
Understanding X-10

A.1 Introduction

This appendix provides a basic introduction to power line control (PLC) with an emphasis on the X-10 system. X-10 is the PLC system utilized in the work of this dissertation. It is intended to provide the interested reader with an understanding of these types of technologies and to help provide a better understanding of this dissertation.

A.2 Experiment 10

In the late 1970s, Pico Electronics Ltd. was formed in Glenrothes, Scotland, to develop advanced integrated circuits for the growing electronic calculator market. All experiments at Pico were designated with an "X." Experiments one through eight (i.e., X-1 through X-8) concerned the development of increasingly more complicated integrated circuits for calculators. Experiment nine involved the development of a circuit to operate a programmable record changer for the BSR (British Sound Reproduction) company. BSR also contracted Pico to develop a wireless remote method to control their equipment. This was experiment ten or X-10. In the late 1970s X-10 had expanded beyond BSR's use and was marketed in the United States by Radio Shack and Sears Roebuck and Company [99]. Today, Pico Electronics Ltd. is better known as X-10 Ltd. who still retains Pico group in Scotland as a subsidiary for research and development augmented with personnel in Hong Kong and China [224].

A.3 X-10 Theory

The X-10 system works by a transmitter sending a digital message over the power line wires to a receiver that receives the message and performs the requested operation. The key piece of technology that enables this to work is a zero crossing detector that senses when the voltage sine wave crosses zero as shown in Figure A.1. Receivers listen for 0.6ms after each zero crossing (120 Hz) for a 120 kHz pulse. The presence of a pulse is detected as a

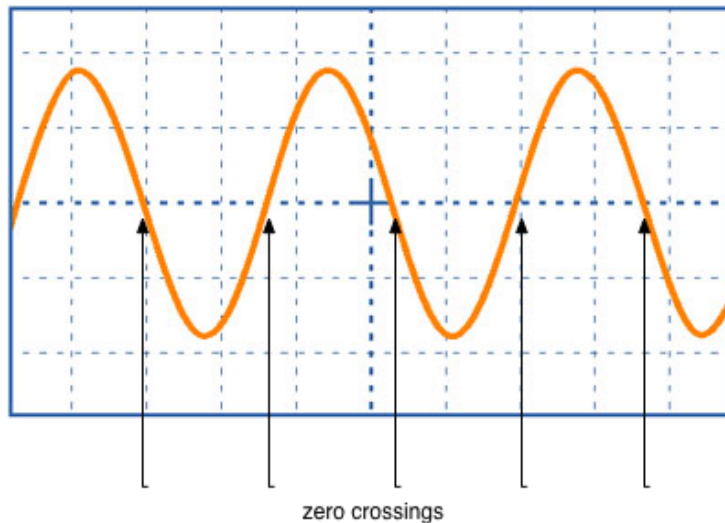


Figure A.1. Zero crossings in the 60Hz power sine wave.

binary “1” and the absence a binary “0.” Transmitters send a 1 ms 120 kHz pulse starting at the zero crossing for each binary “1” they are transmitting in a frame. Figure A.2 illustrates a transmitted bit pattern. The 0.4 ms difference between the transmit and receive durations accommodates for tolerance differences in components [99].

Data frames have a minimum of six leading zero bits between them. This resets the shift registers. The data frame starts with three binary ones followed by a zero (i.e., 1110). This start code is followed by a four bit device letter code from table A.1. The letter codes are not sequential in order to reduce similarity errors since most deployments utilize consecutive numbers in zones physically close together. The letter code is followed by a four bit number code to complete the X-10 address (e.g., A1, B10, and so forth). The number encoded is also not binary sequential and is shown in table A.2. After the number code is a function bit that when it is “0” indicates that the preceding nibble is a number code and when it is “1” indicates that there is no number code. This is to handle commands such as *all zone A lights off*. This frame which consists of a start code, a letter code, and a number code is repeated twice. A three cycle pause occurs (six zero bits) and then another frame is sent. This new frame consists

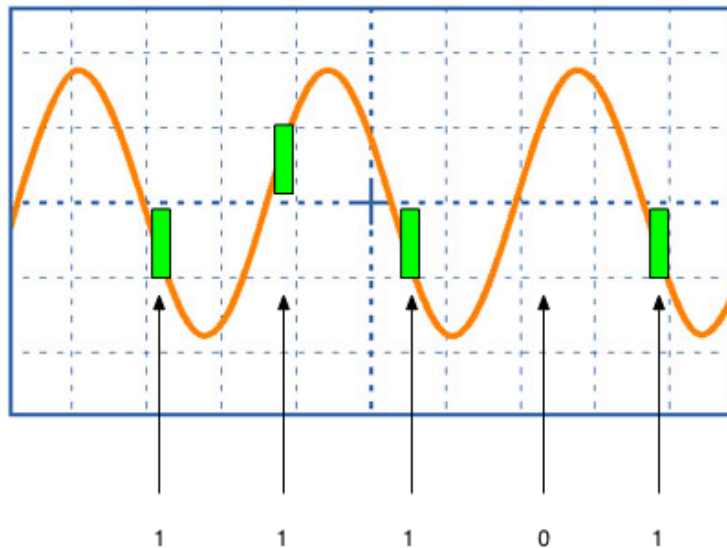


Figure A.2. Binary encoding of 120 kHz pulses in 60Hz power sine wave.

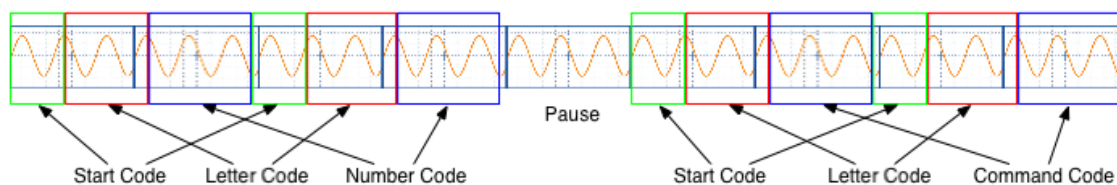


Figure A.3. Complete control message sequence.

of a start code, a letter code, and a command code. The command code specifies one of fifteen commands to be issued (e.g., on, off, bright, dim, and so forth) as shown in table A.3. This command frame is also sent twice. A complete control message sequence is shown in figure A.3 and takes place over 47 cycles of AC power for a duration of 0.7833 seconds [99].

X-10 devices can either be stateful or stateless. Devices that maintain and can report their state cost an order of magnitude more than the stateless versions, but they provide better control and state transparency. All of the devices used in this dissertation work were stateless. Device state was maintained by architectural components.

Table A.1. X-10 Letter Codes

Letter	Binary Code
A	0110
B	1110
C	0010
D	1010
E	0001
F	1001
G	0101
H	1101
I	0111
J	1111
K	0011
L	1011
M	0000
N	1000
O	0100
P	1100

Table A.2. X-10 Number Codes

Number	Binary Code
1	0110
2	1110
3	0010
4	1010
5	0001
6	1001
7	0101
8	1101
9	0111
10	1111
11	0011
12	1011
13	0000
14	1000
15	0100
16	1100

Table A.3. X-10 Command Codes

Command	Binary Code
ALL UNITS OFF	0000
ALL LIGHTS ON	0001
ON	0010
OFF	0011
DIM	0100
BRIGHT	0101
ALL LIGHTS OFF	0110
EXTENDED CODE 1	0111
HAIL REQUEST	1000
HAIL ACK	1001
EXTENDED CODE 3	1010
UNUSED	1011
EXTENDED CODE 2	1100
STATUS ON	1101
STATUS OFF	1110
STATUS REQ	1111

A.4 X-10 Equipment

There are many types of available X-10 modules and systems for home automation control. In our environments we have specifically used RF transceivers, computer interface modules, light modules, appliance modules, motion detectors, and an HVAC thermostat. The RF



Figure A.4. X-10 lamp modules and connected lamps.

receivers are used to capture open air wireless radio frequency remote control signals, translate the button commands into X-10, and transmit the command over the power line to an awaiting receiver. The computer interface uses RS-232 to communicate with a personal computer for both receiving and sending of X-10 protocol signals over the power line. The light modules shown in figure A.4 allow for on/off control of incandescent lamps as well as 32 different dim levels. Appliance modules are used for higher wattage appliances and fluorescent lights to turn them on or off—these modules may not be dimmed. Motion detectors use passive infrared sensors that detect infrared changes with the assistance of a Fresnel lens. The motion detectors work well, but have a two second or more delay in response and update due to transmitting detection over RF. The HVAC system uses X-10 to set the mode of the air conditioning unit, blower state, and current target heating or cooling temperature. These devices form the core of the effectors for the intelligent environments presented in this dissertation.

A.5 X-10 Problems

X-10 works well—most of the time. In the course of our experimental work we have established two very large X-10-based deployments. We have encountered a number of difficulties along the way due to some interesting problems.

After our first deployment of over fifty X-10 light modules we noticed that lights would often go on and off for no apparent reason. The problem seemed to get worse at night making it seem like we were hosting a prankster poltergeist. The problem seemed to come from the setup. We had intermixed lighting and computer equipment on the same power line circuits. The problem was that the low-voltage transformers used for the computer speakers and the computers themselves trapped the X-10 signals and often would generate noise that mimicked the X-10 protocol thus causing our apparent poltergeist. We exercised this demon from the system by separating the X-10 devices from the computer equipment and also introduced noise filters at the interface of these separate X-10 circuits and the main power supply circuit. After these modifications were complete we no longer witnessed random light control.

Dealing with fluorescent lights is also a difficult issue with X-10. First, the light modules cannot be used because they do not allow for sufficient wattage to be drawn to support starting the lights. Second, light modules regulate voltage to allow for dimming—have you ever dimmed a fluorescent light?—which caused problems with the lights and precluded the operation. Appliance modules can be used to successfully control fluorescent light, but the voltage conversion inside the ballast units of these types of lights also traps the X-10 signal and generates noise. Thus, we faced another poltergeist inherent to a device we actually sought to control. Eventually, we just replaced all of the fluorescent lights with rope lights, but we did observe that using an appliance module to power a noise filter with the fluorescent lamp on the other end worked successfully and exercised another demon, but the cost of adding filters to each light greatly exceeded just replacing them with an X-10 environment-friendly light.

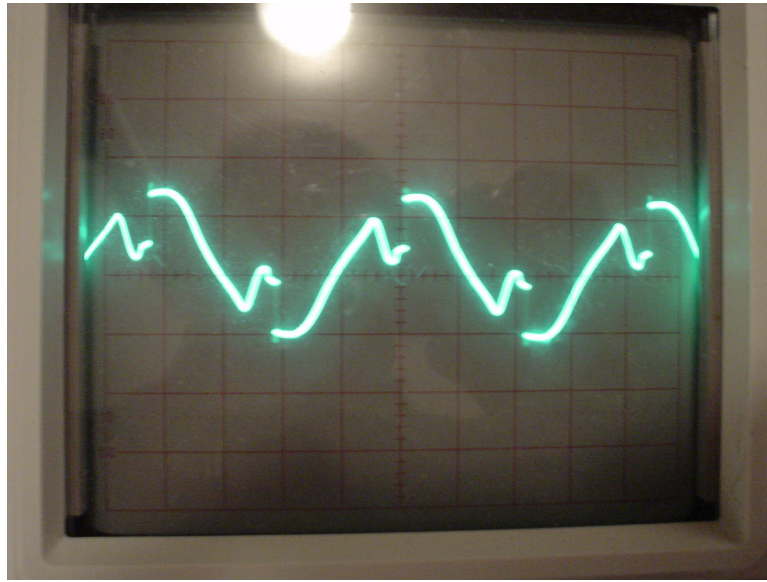


Figure A.5. Abnormal 60Hz 110V AC sine wave.

We have also had issues with some letter codes—they just stop working. We have had several areas we call zones that are under a single letter designation quit working. The modules still worked if we shifted them all to a different letter, but they would never work again on the originally assigned number. We still have no explanation for this behavior, and our only work around has been a permanent shift to a different letter designator for the affected zones.

In our second major deployment which is in an apartment, we encountered another interesting problem. The installed bathroom fan introduced a significant amount of noise as shown in Figure A.5 compared to a normal AC sine wave as shown in Figure A.6. It only introduced the noise, obviously, when it was turned on. The noise as can be seen in Figure A.5 shows voltage distortions that introduce additional zero crossings (explained earlier in this appendix) and therefore disrupt any X-10 control signals by injecting zeros into the frames. The fans can be turned on, but the noise prevents them from being turned off. We attempted to filter the line noise from the fan, but because it actually affects the power line voltage at the controller there is not an effective way to filter this anomaly. In the end, we developed a new computer-

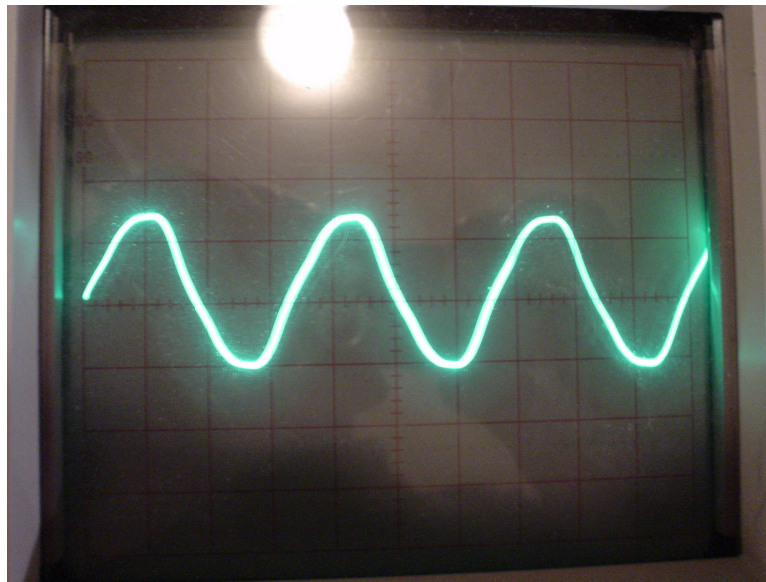


Figure A.6. Normal 60Hz 110V AC sine wave.

controlled circuit using our Argus system and discontinued X-10 control of this particular fan. It should be noted that a floor fan and a ceiling fan in the same environment can be controlled by X-10.

In general there is usually a delay of between one to four seconds in the normal operation of X-10. A number of our test inhabitants were impatient with the lights and felt they did not work. Working with the remotes also often requires pressing the command buttons more than once in order for the command to work. We have also noticed, especially with computer control, that many electrical systems are not wired correctly, have minor shorts, or use different phases from the 220V supply when it is split into 110V sections. These problems create unreachable areas and inconsistent power line signal reachability. We have often had to increase the number of RF transceivers and use alternate plugs for equipment to solve such problems.

Despite the problems we have encountered, we have always been able to find a way to adapt or work around the issues. X-10 does work and performs the job it was designed to do.

The ability to control household objects for the price at which X-10 modules are offered makes for a strong value.

A.6 X-10 Middleware

Connecting a personal computer to the power line through a computer interface module is as easy as plugging in the module into the power outlet and the line into the computer's RS-232 port. However, on a Linux system the next question is how to sense X-10 communication and how to initiate X-10 control. We started by using a simple software package called Heyu originally written by Daniel Suthers and later improved by Charles Sullivan which is based on the program called "x10" by Larry Campbell as modified by Paul Fox for Linux. It is freely available at heyu.tanj.com/heyu2 and contains a basic sensing daemon and command line control of X-10 devices. This software worked very well for our initial needs, but we grew to need something that seemed to be more integrated with our operating system. We have used the "Linux X-10 Universal Device Driver" (a.k.a., Project WISH) written by Scott Hiles for the last two years. WISH is publicly available at wish.sourceforge.net and takes a different approach from the typical command operation and daemon monitoring system. WISH adds X-10 devices to the /dev directory of the Linux OS and allows for X-10 devices to be treated as attached peripheral character devices. This allows for ease of monitoring by just polling the device handler and for control by simply setting the value of the device handler all in a very intuitive and standard Linux way. WISH is a very fast and well-integrated piece of middleware.

A.7 Power Line Control Alternatives

X-10 and its form of amplitude shift keying (ASK) modulation technique is one of the oldest PLC protocols. Other PLC protocols include the CEBus, LonWorks, and HomePlug. CEBus uses frequency sweeping and a carrier sense multiple access with collision resolution

and collision detection (CSMA/CRC) protocol for synchronization, collision handling, and sending data between peer-to-peer modules. This allows for rates of near 10 kb/s and a packet communication method. LonWorks uses a narrowband spread spectrum and a CSMA technique for peer-to-peer communications. Current LonWorks power line implementations are rated at 5 kb/s, but provide an Echelon Corporation patented noise cancellation technique that minimizes errors in noisy environments. HomePlug (1.0) uses an orthogonal frequency division modulation (OFDM) scheme to transmit information over a frequency band of 4.49 to 20.7 MHz with 128 subcarriers that yield a data rate from 1 to 14 Mb/s using a carrier sense multiple access / collision avoidance (CSMA/CA) protocol. These technologies represent a significant improvement over the 60 b/s X-10 protocol and are being used today to transmit Ethernet packets between modules over the power line [108]. Despite the speed and bandwidth advantages of these new technologies and protocols, there is a lack of affordable and available home and business automation products. For now, the availability and price of X-10 makes it a more reasonable system to work with since X-10 deployments are more likely found in homes, and the work of this dissertation can be used more readily to make an improvement in peoples' lives.

APPENDIX B

The Argus Sensor Networks

B.1 Introduction

This Appendix is included to provide the interested reader information concerning the design, construction, and operation of the Argus sensor network developed by researchers on the MavHome project at The University of Texas at Arlington. We created this sensor network in order to perceive motion, temperature, light, humidity, and door/window positions as well as to sense and control mini-blinds. The cost of sensor networks and general availability to the public persuaded us to pursue the option of building and designing our own at the lowest cost possible. This Appendix covers the design, implementation, and deployment details necessary to reproduce this affordable sensor network and its variants. Instead of patenting this work, we have made a conscious decision to release it and our algorithms under the terms outlined in Appendix F.

This Appendix includes many details and in some areas a step-by-step guide for setup. It is included to make this dissertation a more complete work of the efforts, engineering, and research conducted on the MavHome project. The prices as presented in this Appendix are those at the time of manufacture and may differ from the current prices (usually they get less expensive).

B.2 Hardware

The Argus Sensor Network is the main perception system for our work. Argus is comprised of a Master board that connects via a serial-based interface to a controlling personal computer (PC), Superslave boards that host sensor suites and report to the Master via an I^2C serial bus, and Dongle boards that host individual sensors. In this Appendix, we present the designs and costs for an extensible Master, a super slave capable of handling 64 sensors, and a Dongle board capable of hosting four individual sensors, together called the ArgusMS (Master-Slave) system. We also present three of these extensions to the Master called ArgusD (Digital),

ArgusM (Motor), and ArgusAD (Analog and Digital). The Argus Sensor Network system has been designed for deployment in the MavLab (see Appendix D) (including the MavKitchen), MavPad (see Appendix C), and any future MavHome environments.

All components were purchased at a local electronics wholesaler, board masks were made through a web order, and all designs were done in the freeware Eagle layout designer (www.cadsoft.de). The Eagle project packages and software used in our deployments are publicly available on the web at mavhome.uta.edu.

B.3 Software

We differentiate between three types of software; the software that is intended to directly communicate with an Argus device and provide functionality as a service is called *Hermes*¹, any software intended to communicate with the Hermes drivers is known as a *client*, and the software intended to be flashed onto the device microcontrollers is called *Argus*.

B.3.1 Hermes Software

Each Argus Master (including variants) communicates with a single interfaced host PC using a serial line (RS-232). Software running on the host PC may send commands to the Argus Master, retrieve data from the device, or send data to a remote PC. All software that communicates directly with an Argus device is called Hermes, and should be run on the PC to which Argus is physically connected. Hermes usually provides a mechanism to accept commands from a third party, and parse these commands for the Argus device. Essentially, Hermes is the driver program for an Argus device.

¹Hermes slayed the 100-eyed Argus beast in Greek mythology

B.3.2 Client Software

Client software is used to connect to Hermes and relay commands or receive information. It must run on the same machine as Hermes, and may often converse with remote applications. From the client's point of view, Hermes is serving as a middleman between client and Argus.

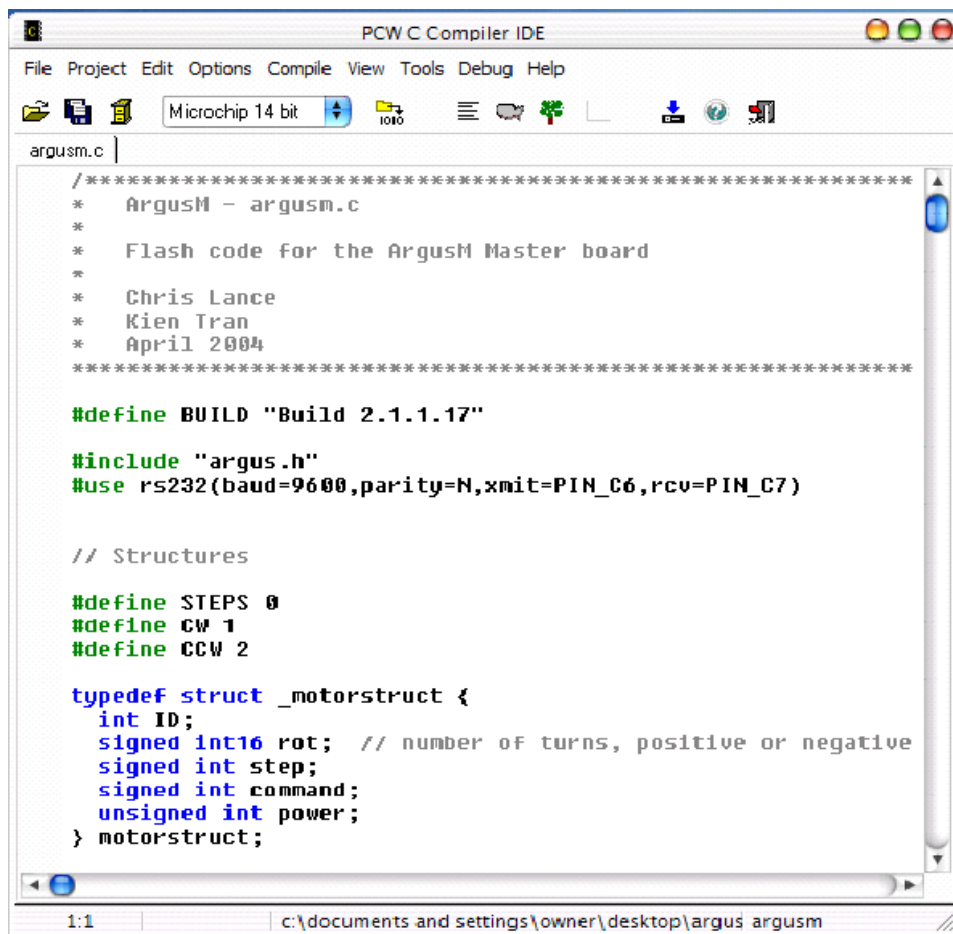
B.3.3 Argus Software

Each Argus device operates off of software stored on-chip. This firmware may need to be changed in case of software upgrade or microcontroller replacement. We program the microcontroller using a hardware programmer from CCS, called ICD-U. This ICD unit connects one end to the computer via USB, and the other end to the Argus device via an RJ-11 socket on the side. The Argus software (assumed to be already compiled as a .HEX file) is then loaded into the device using ICD-U programmer software, also supplied by CCS. Each Argus device class has its own .HEX file.

B.3.4 All Argus devices

These are the steps to setup Argus software (any Argus device):

1. Compile appropriate Argus source file using CCS's IDE compiler (PCW C Compiler or equivalent) as shown in Figure B.1
2. Connect ICD unit to both computer and Argus device
3. Power Argus Master device
4. Use the ICD programmer software/GUI to upload the appropriate .HEX file as shown in Figure B.2. If the ICD unit is not properly attached to both computer and Argus device, the programmer software will refuse to start.



```

PCW C Compiler IDE
File Project Edit Options Compile View Tools Debug Help
Microchip 14 bit
argusm.c
/*****
 * ArgusM - argusm.c
 *
 * Flash code for the ArgusM Master board
 *
 * Chris Lance
 * Kien Tran
 * April 2004
 *****/

#define BUILD "Build 2.1.1.17"

#include "argus.h"
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7)

// Structures

#define STEPS 0
#define CW 1
#define CCW 2

typedef struct _motorstruct {
    int ID;
    signed int16 rot; // number of turns, positive or negative
    signed int step;
    signed int command;
    unsigned int power;
} motorstruct;
1:1 | c:\documents and settings\owner\desktop\argus argusm

```

Figure B.1. PCW C Compiler.

B.4 Argus Sensor Networks

The low cost Argus networks are developed around a core Argus Master board. The Master in combination with Superslaves (separate component boards that relay sensor information to a Master board) and Dongles (separate component boards that host up to four sensors and connect to Superslaves) form the Argus Master-Slave (ArgusMS) network. The Master with a special daughter board for extending the system to allow only pure digital I/O form the Argus Digital (ArgusD) network. The Master with a special daughter board for extending the system to allow control of stepper motors form the Argus Motor (ArgusM) network. The Master with a special daughter board for extending the system to allow a pure digital I/O bank and an

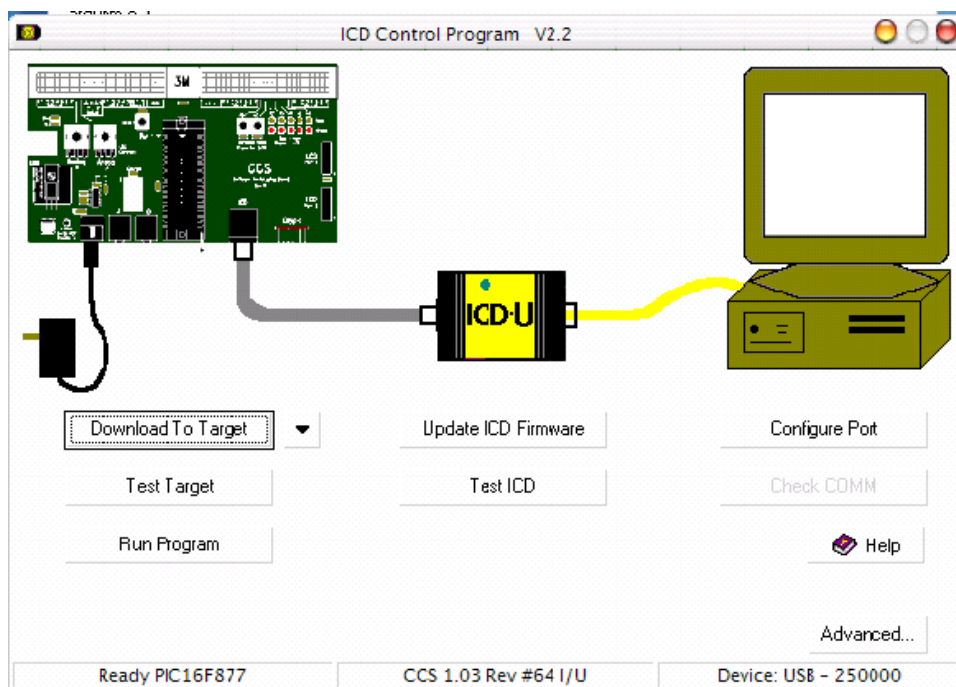


Figure B.2. ICD Programmer Software.

analog bank form the Argus Analog + Digital (ArgusAD) network. These networks have been deployed and working continuously for over a year in our test environments.

B.4.1 ArgusMS

ArgusMS is a branched spanning sensor network as shown in Figure B.3. The system consists of an Argus Master device, which connects to many Argus Superslave devices, each of which collects sensor data from its surroundings. The types of sensors that may be attached to an Argus Superslave are varied and configurable. Ultimately the Master device polls the Superslaves for their data and streams this data upward to a host PC on demand. Each Master can host up to 100 Superslaves and each Superslave can host 16 Dongles which, in turn, can host up to four sensors each for a total network capacity of 6400 sensors. In general, an ArgusMS network has the following features:

- Single computer reads entire network

- An Argus Master may connect to up to 100 Superslaves
- Supports up to 6400 analog sensors
- The serial communication bus uses CAT-5 Ethernet cable, which is plentiful and inexpensive
- Sensors may be added or detached from Superslaves while system is active
- Superslaves may be added or detached while system is active
- Cost effective, only one long serial bus needed to connect the Master to the Superslaves, instead of a star network
- To be used when a large amount of analog or digital data is to be collected remotely
- System runs at 2Hz

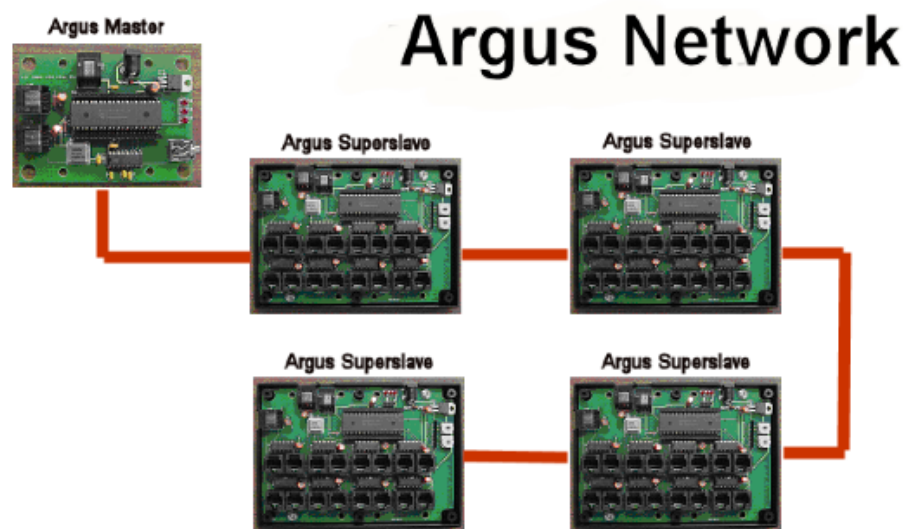


Figure B.3. Overview of Argus Master/Slave network.

B.4.1.1 Master

The Argus Master as shown in Figure B.4 is the centerpiece of the Argus network. All other devices directly or indirectly connect to it, and in some configurations other devices are not needed at all. All that is needed is a host PC with a serial line to interface to the device. In the ArgusMS configuration, the Argus Master is connected with one or more Argus Superslaves through an I^2C serial bus. This Master collects analog data from the Superslaves when prompted by its host computer. The Argus Master board serves as the I^2C controller for all attached Superslave boards. The Master interfaces with software on a PC to allow the user to read any sensor on any slave through a hardware selector switch numbering address. The four inch by three inch Master board circuit layout is shown in Figure B.5 and the circuit schematic is shown in Figure B.6. Table B.1 shows the components required to build a Master and the associated cost. A basic solder mask board can be ordered from www.pcbexpress.com for 3-day process (coated with legend) with the following specifications and cost:

- Quantity: 20
- Size: 4.000" x 2.900"
- CAD System: Eagle EE
- Finished Copper Weight: 1.0 ounce
- Thickness (FR4 Laminate): .062"
- TOTAL = \$404.00 (\$20.20 each)

Table B.1: Argus Master Parts List.

Part Name	Mouser Number	Price/Unit	Quantity	Price
0.1uF Capacitor	581-SR215C104M	\$0.26	6	\$1.56
47uF Capacitor	75-515D10V47	\$0.12	3	\$0.36
Zener Diode	78-1N4747A	\$0.07	1	\$0.07

Table B.1 – continued

Part Name	Mouser Number	Price/Unit	Quantity	Price
PIC16F877 DIP40	579-PIC16F87704P	\$8.40	1	\$5.39
7805 Voltage Regulator	511-L7805ABV	\$0.60	1	\$0.60
RS232 Drivers and Receivers	511-ST202CN	\$0.83	1	\$0.83
Power Jack	163-5003	\$0.64	1	\$0.64
RJ-45 8pin Jack SIDE ENTRY	571-5551641	\$0.56	2	\$1.12
RJ-12 6pin Jack SIDE ENTRY	571-5551651	\$0.53	1	\$0.53
20Mhz crystal oscillator	520-TCH2000	\$1.70	1	\$1.70
47K Resistor	71-RN55D-F-47.5K	\$0.16	2	\$0.32
BCD Rotary Dial	106-RI40012	\$2.25	2	\$4.50
24 Pin Header	538-26-48-1241	\$1.47	2	\$2.94
LED Lamp w5V resistor	606-4302F1-5V	\$0.49	3	\$1.47
4.38x3.25x2.5 Enclosure	635-133-B	\$5.38	1	\$5.38
DIP16 Socket	506-516AG11D-ES	\$0.90	1	\$0.90
DIP40 Socket	506-540AG11D-ES	\$1.90	1	\$1.90
RS232 Jack, Stereo Audio	161-3508	\$0.74	1	\$0.74
AC 12VDC 500MA 2.5X5.5M	412-112053	\$5.89	1	\$5.89
Total Cost				\$36.84
+ PCB				\$20.20
Grand Total Cost				\$57.04

The Argus Master board has two RJ-45 Ethernet jacks located on the side panels as shown in Figure B.7 for the I^2C serial bus for communication with the Superslaves. These jacks are internally connected together, and have lines for data, power and ground. While the

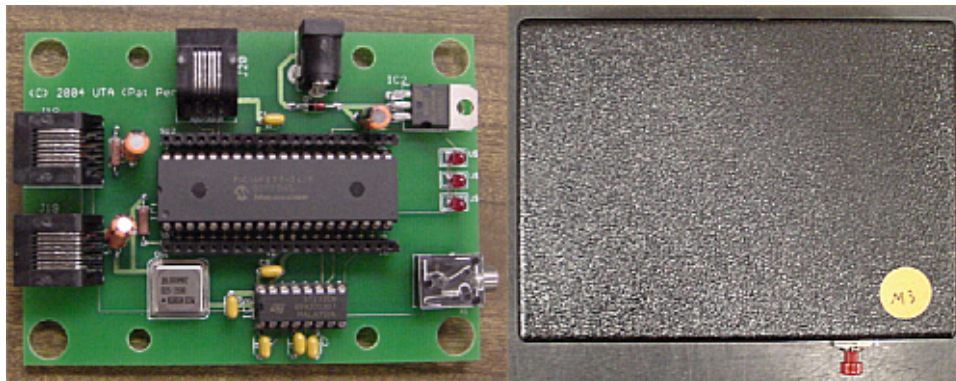


Figure B.4. Argus Master.

I^2C serial bus is connected all boards share a common power connection and so an Argus device without a power supply may still be powered through the I^2C bus. However, do **NOT** leave the system idle for long periods of time without a power supply for each and every individual Argus Master or Slave. Each device without its own power supply will draw power from neighboring devices, possibly causing an overload! The process for connecting and starting a Argus Master and Superslaves is as follows:

1. Disconnect the I^2C serial bus line from all Argus devices
2. Connect power to Argus Master
3. Connect power to all Argus Superslaves
4. Keep tabs on the status lights, to identify a faulty subsystem
5. Connect I^2C serial line from Master to a Superslave
6. Connect I^2C serial line from Superslave to Superslave, until all Superslaves and the Master are connected in one long chain. Do not complete a loop from the last Superslave back to the Master.
7. Connect RS-232 Serial cable into the Master's 1/8 inch stereo socket
8. Connect RS-232 Serial cable to the host PC's DB-9 serial socket
9. Check lights on Master and all Superslaves to verify systems are operational

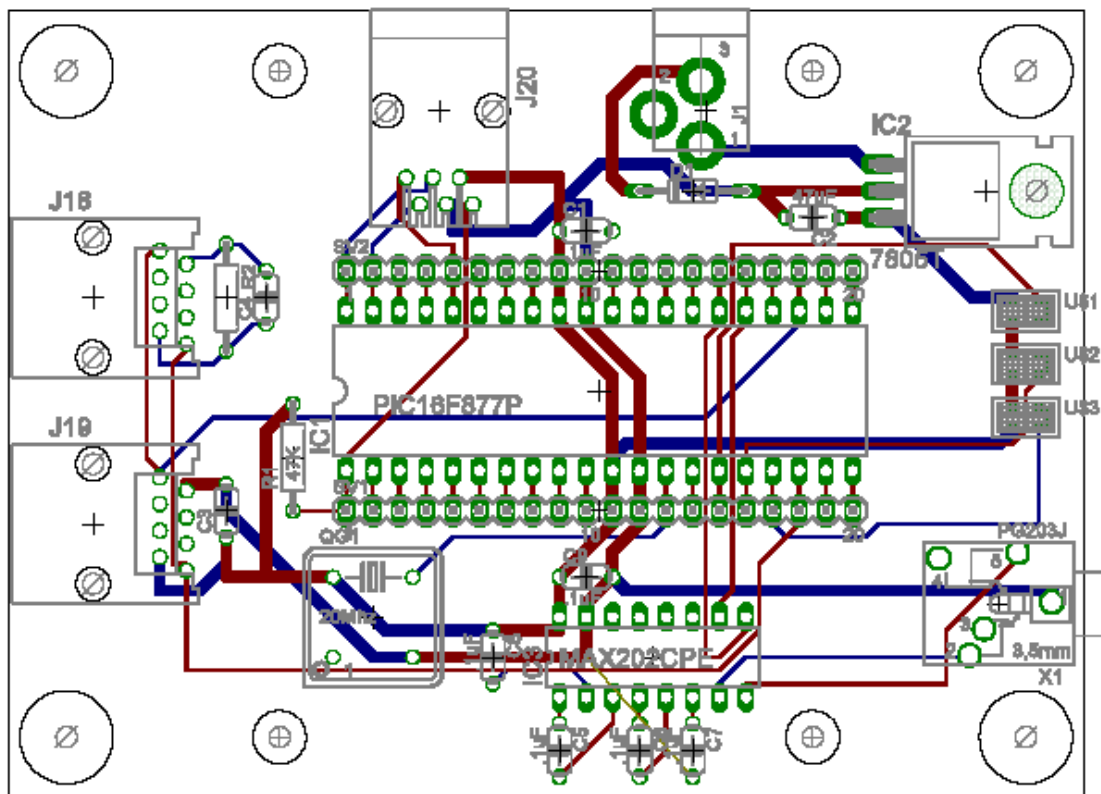


Figure B.5. Argus Master layout schematic (4 in. x 3 in.).

B.4.1.2 Superslave

The Argus Superslave shown on Figure B.8 is the backbone of the Argus network. The Superslave is the device that directly reads sensor output for all sensors, converts them to binary values, and stores this information for later retrieval. Upon request by an Argus Master, the Argus Superslave will transmit its data to the Argus Master for processing. A Dongle system groups sensors into clusters of four, and one Dongle connects four sensors to the Superslave—efficiently using one port for four sensors. Multiple Superslaves can co-exist on the same serial communication bus through a flexible addressing system, where the addresses are set manually by the user before power up. The Superslave featuring control of 64 individual sensors

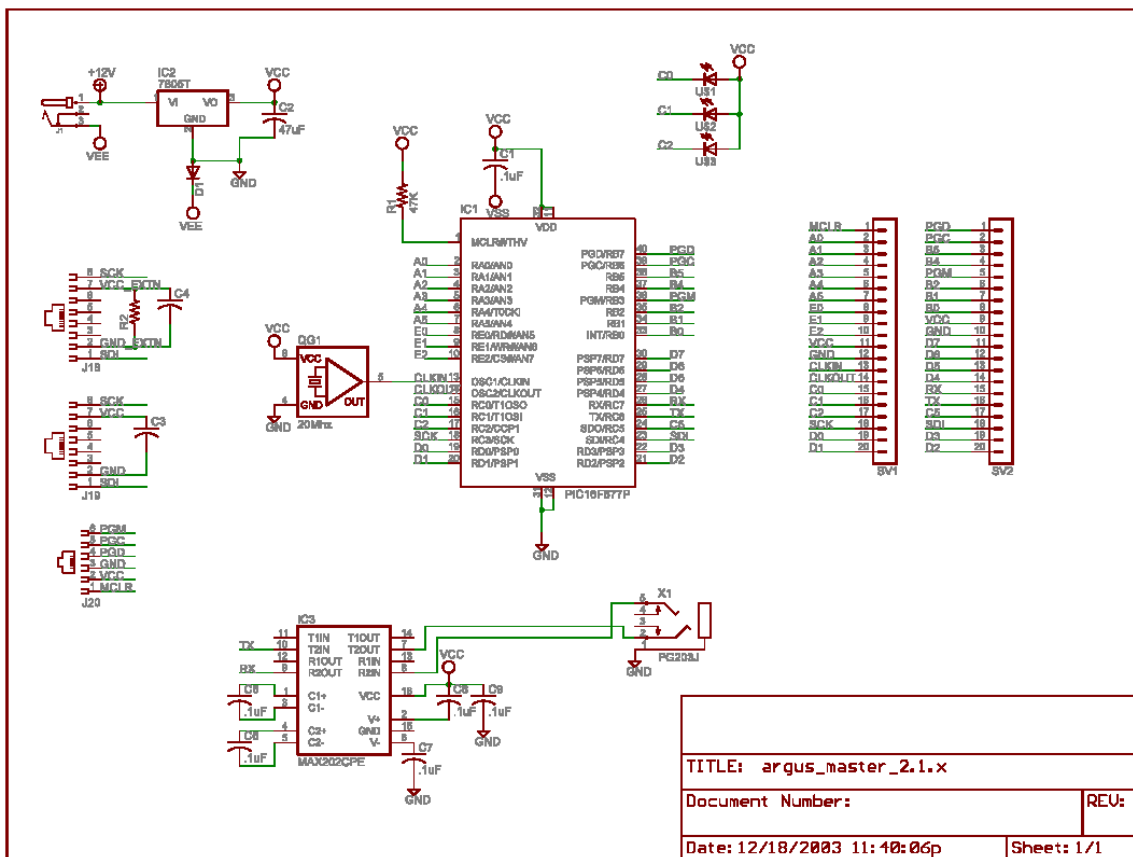


Figure B.6. Argus Master circuit schematic.

was designed to replace the eight individual sensor control standard slave we had previously developed. Argus Superslaves have the following features:

- On board analog-to-digital converter
- The serial communication bus connects via CAT-5 Ethernet, using an I^2C protocol
- Sensor Dongles also attach via CAT-5 Ethernet
- Supports up to 64 sensors per Superslave
- Allows convenient sensor placement and installation by placing the Superslave local to the sensors, yet remote from the Master and host PC
- Addressable from 00–99, providing 100 unique IDs
- Wide variety of sensor types supported—any resistive or voltage-based sensor

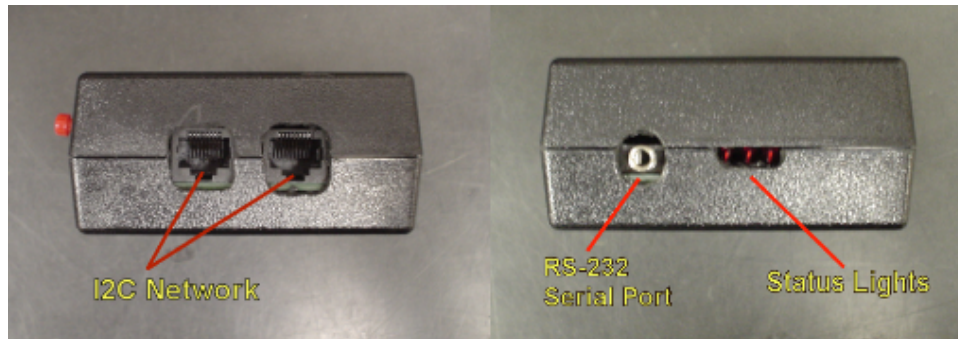


Figure B.7. Argus Master side view.

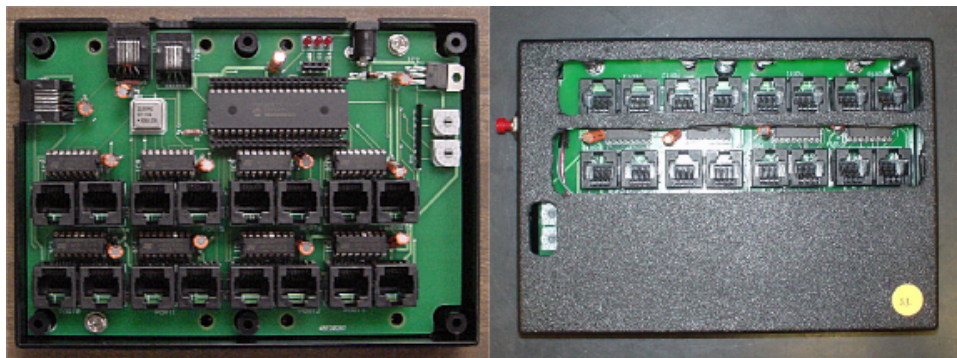


Figure B.8. Argus Superslave.

The 6.5 inch by 4.5 inch Superslave board circuit layout is shown in Figure B.9 and the circuit schematic is shown in Figures B.10, B.11, and B.12. Table B.2 shows the components required to build a Superslave and the associated cost. A basic solder mask board can be ordered from www.pcbexpress.com for 3-day process (coated with legend) with the following specifications and cost:

- Quantity: 10
- Size: 6.500" x 4.500"
- CAD System: Eagle EE
- Finished Copper Weight: 1.0 ounce
- Thickness (FR4 Laminate): .062"
- TOTAL = \$334.00 (\$33.40 each)

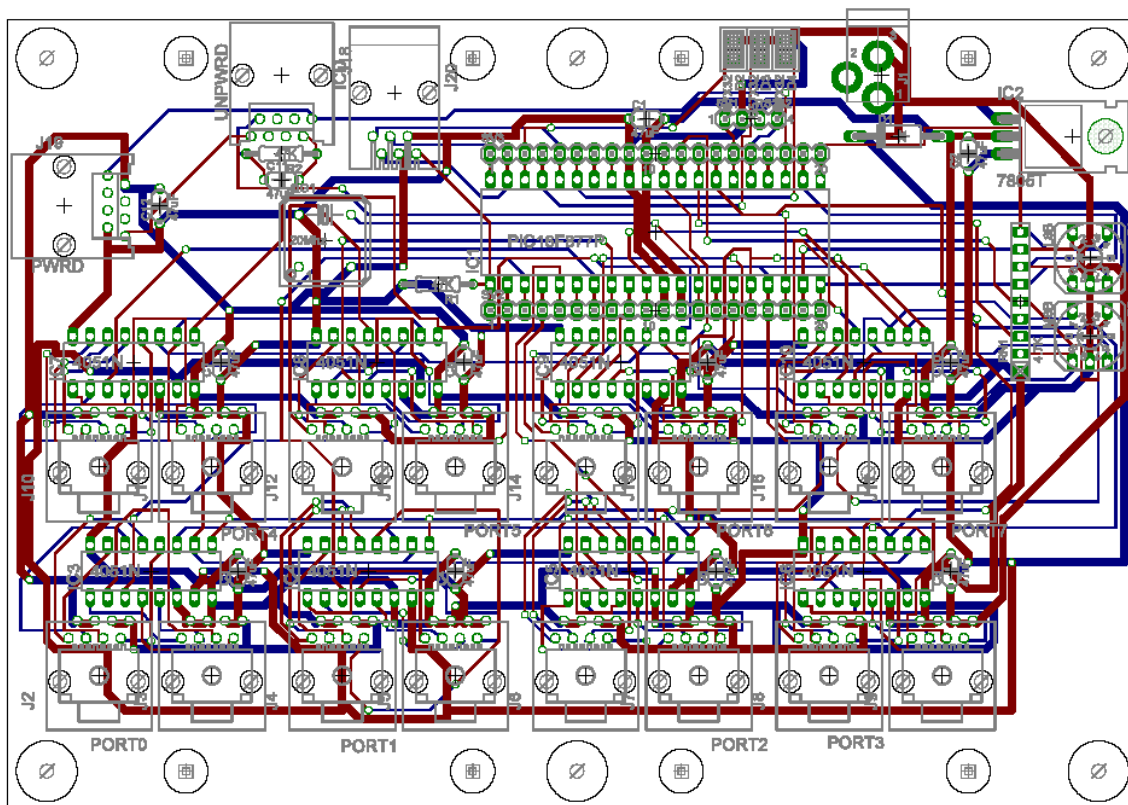


Figure B.9. Argus Superslave layout schematic (6.5 in. x 4.5 in.).

Table B.2: Argus Superslave Parts List.

Part Name	Mouser Number	Price/Unit	Quantity	Price
47uF Capacitor	75-515D10V47	\$0.12	12	\$1.44
Zener Diode	78-1N4747A	\$0.07	1	\$0.07
LED Lamp w/5V resistor	606-4302F1-5V	\$0.49	3	\$1.47
PIC16F877 DIP40	579-PIC16F87704P	\$8.40	1	\$5.39
7805 Voltage Regulator	511-L7805ABV	\$0.60	1	\$0.60
4051N DIP-16 8TO1 ANA MUX	511-M74HC4051M	\$0.84	8	\$6.72
Power Jack	163-5003	\$0.64	1	\$0.64

Table B.2 – continued

Part Name	Mouser Number	Price/Unit	Quantity	Price
RJ-45 8pin Jack TOP ENTRY	571-5202594	\$0.51	16	\$8.16
RJ-45 8pin Jack SIDE ENTRY	571-5551641	\$0.56	2	\$1.12
RJ-12 6pin Jack SIDE ENTRY	571-5551651	\$0.53	1	\$0.53
20Mhz crystal oscillator	520-TCH2000	\$1.70	1	\$1.70
47K Resistor	71-RN55D-F-47.5K	\$0.16	2	\$0.32
4.7K 8bit Resistor Network	71-CSC10A01-47K	\$0.29	1	\$0.29
BCD Rotary Dial	106-RI40012	\$2.25	2	\$4.50
24 Pin Header	538-26-48-1241	\$1.47	2	\$2.94
4.88x6.88x1.5 Enclosure	635-171-B	\$8.00	1	\$8.00
DIP16 Socket	506-516AG11D-ES	\$0.90	8	\$7.20
DIP40 Socket	506-540AG11D-ES	\$1.90	1	\$1.90
AC 12VDC 500MA 2.5X5.5M	412-112053	\$5.89	1	\$5.89
Total Cost				\$58.88
+ PCB				\$33.40
Grand Total Cost				\$92.28

Each Superslave board has two RJ-45 Ethernet jacks located on the side panels for I^2C communications as shown on Figure B.14. These jacks are also internally connected together. While connected to each other, all boards share a common power connection. Again, do **NOT** leave the system running for long periods of time without a power supply for each and every individual Argus Master or Superslave. There are 16 RJ-45 jacks located on the top panel for Argus sensor Dongles as shown in Figure B.13. Sensors do not directly connect to the Superslave, but are instead loaded onto Argus Dongles which then send a raw or amplified sensor

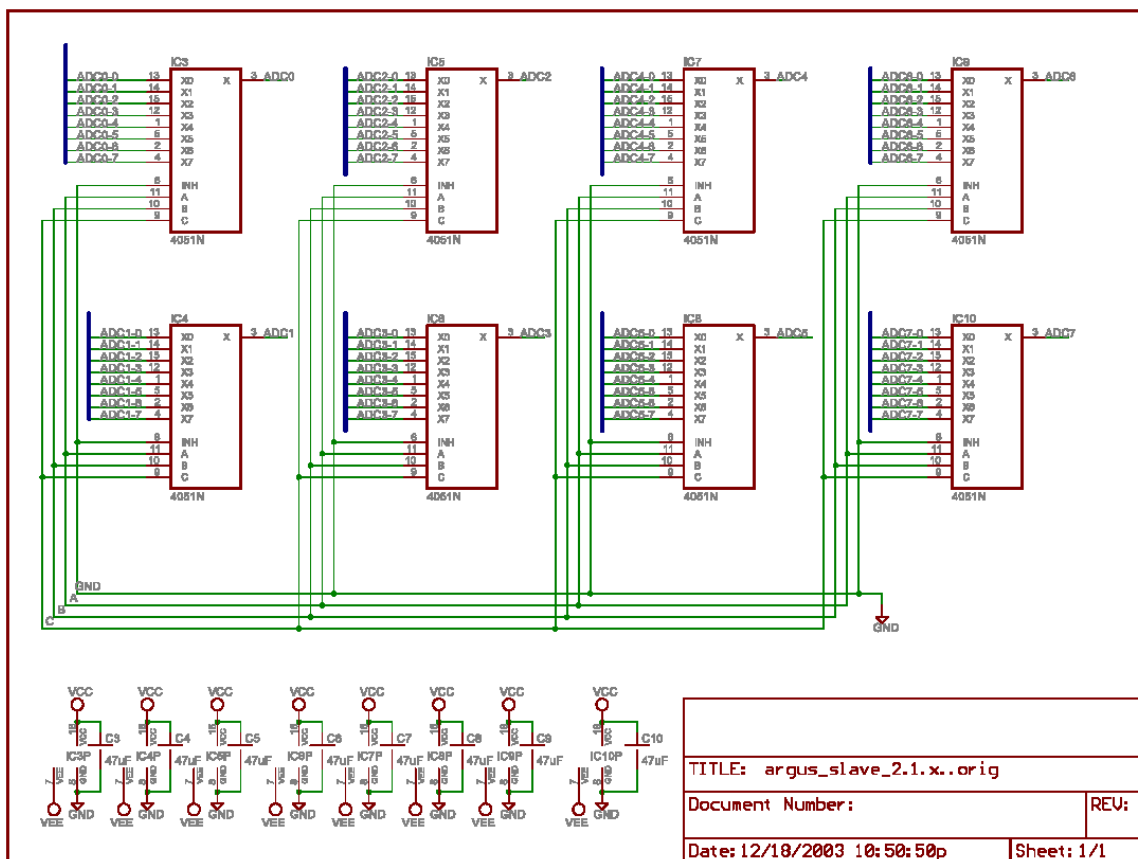


Figure B.11. Argus Superslave Circuit Diagram.

6. Check lights to verify that Superslave subsystem is up
7. Connect the Superslave to the I^2C serial bus
8. Check lights again to verify systems operational

B.4.1.3 Dongle

The Argus sensor Dongle shown in Figure B.15 accepts up to four sensors for data collection. The Dongle amplifies readings from a resistive sensor (the amplification of which is user configurable) and transmits this amplified analog signal to the Argus Superslave for processing and conversion. Digital signals can also be transmitted in the form of a high-low steady signal, as well as unamplified analog signals (in the case of black-box sensor devices).

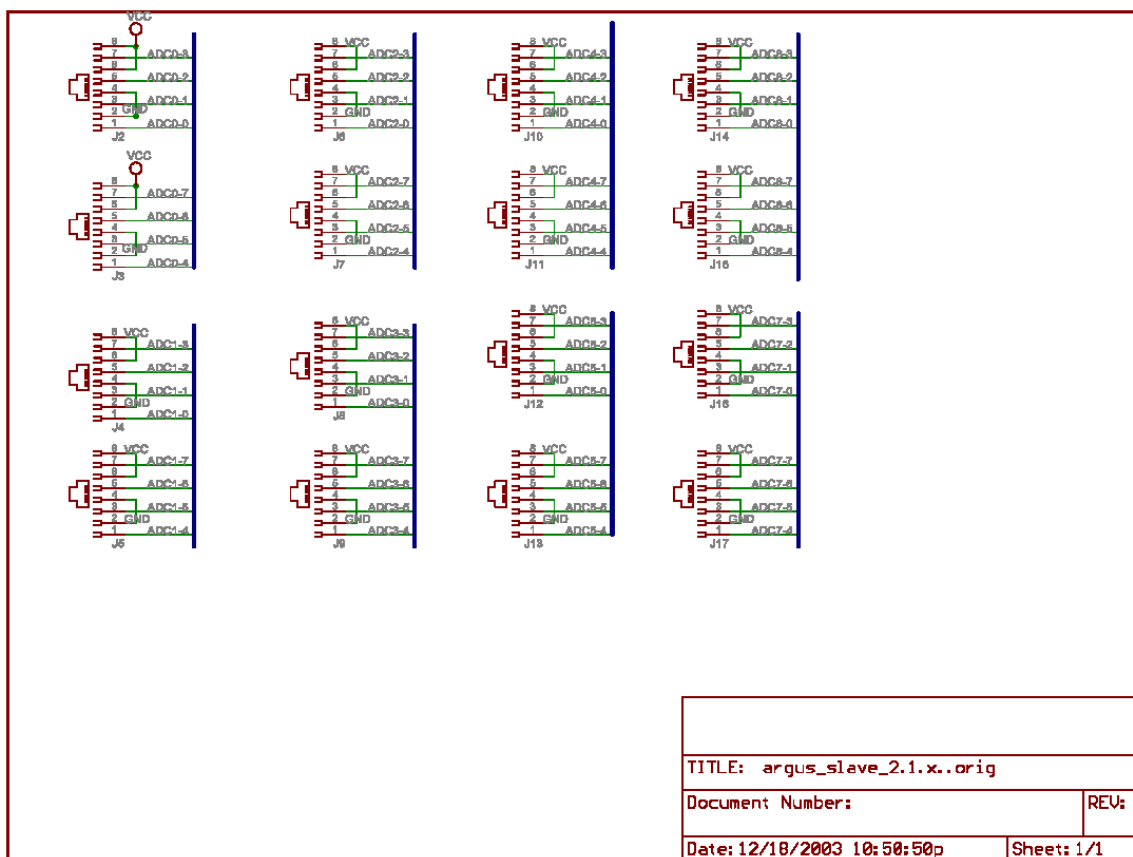


Figure B.12. Argus Superslave Circuit Diagram.

The Dongle is a small packaged board used to reduce the complexity of the Superslave and increase the sensor flexibility of the Argus sensor network. The Dongle incorporates circuitry and input slots to accommodate four individual analog or digital sensors and associated resistance networks. The two inch by 1.25 inch Dongle board circuit layout is shown in Figure B.16 and the circuit schematic is shown in Figures B.17. Table B.3 shows the components required to build a Dongle and the associated cost. A basic solder mask board can be ordered from www.barebonespcb.com where $UnitPrice = (\$25.00 + Quantity * X_{Dim} * Y_{Dim} * \$0.50) / Quantity$ and for 50 units, price is \$1.75 each (\$87.50 for the order). Argus Dongles have the following features:

- On board non-inverting amplifier circuit

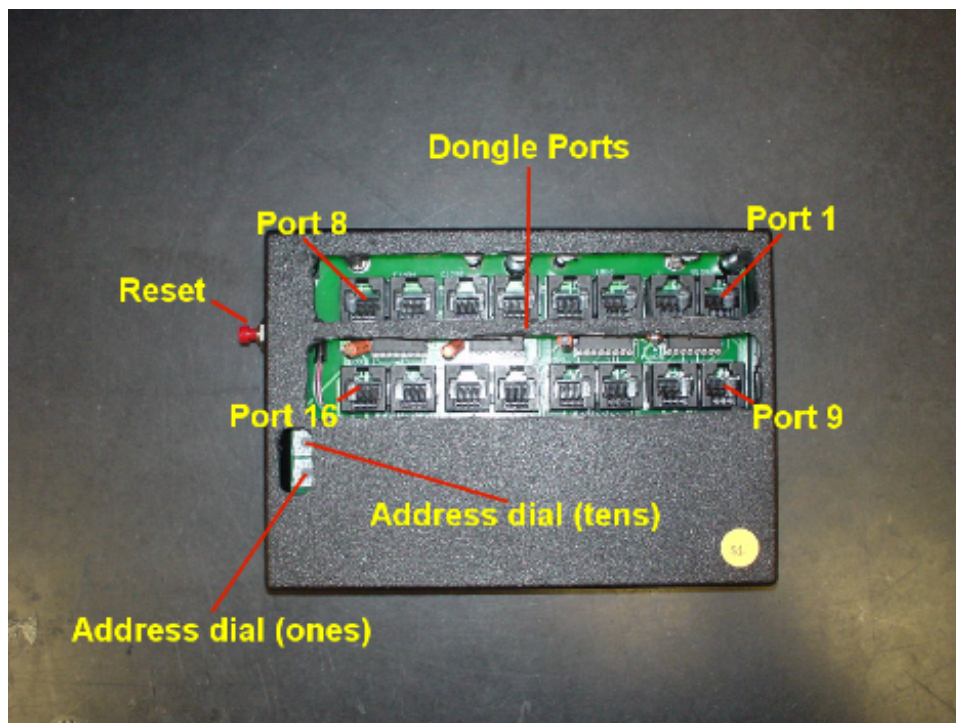


Figure B.13. Argus Superslave top view.

- Four sensor banks
- Amplifies signals from resistive sensors
- Accepts signals from digital input (i.e., switches)
- Modifiable for accepting unamplified analog input
- Wall mountable

Table B.3: Argus Dongle Parts List.

Part Name	Mouser Number	Price/Unit	Quantity	Price
0.1uF Capacitor	581-SR215C104M	\$0.26	1	\$0.26
Op Amp 4-gate bipolar DIP-14	511-TSH24IN	\$1.28	1	\$1.28
RJ-45 8pin Jack SIDE ENTRY	571-5551641	\$0.56	1	\$0.56
16 pin single row pin socket	517-974-01-16	\$1.54	1	\$1.54

Table B.3 – continued

Part Name	Mouser Number	Price/Unit	Quantity	Price
36 pin double row pin socket	517-975-01-18	\$1.55	1	\$1.55
IC socket DIP-14	506-514AG11D-ES	\$0.88	1	\$0.88
Bud Ind. Plastic Box	563-CU741MB	\$2.80	1	\$2.80
Total Cost				\$8.87
+ PCB				\$1.75
Grand Total Cost				\$10.62

Each sensor Dongle has an RJ-45 Ethernet jack for connection to an Argus Superslave. Each sensor Dongle also contains four banks of 12-pin sockets for sensor configuration as shown in Table B.4, although with some configurations certain pins are optional. The pin functions are as follows:

1. Out: Dongle output. The splices into the sensor feed going back to the Superslave. The pin is used only for un-amplified sensors, when we want to bypass the amplifier altogether.
2. GND: Superslave ground. Used as a reference ground.
3. Input: Sensor input. Feeds into the voltage divider part of an amplifier circuit. One leg of a resistive sensor is to be placed here, although our usual configuration bypasses this.
4. Vcc: Superslave power. Used to power sensors or sensor devices.
5. R3, R4: Amplifier gain. Resistors are plugged in these to set the gain of the Op-Amp. The gain is $1 + R3 / R4$.
6. R2, R1: Pre-amplifier voltage divider. The voltage that is fed into the amplifier is determined by R2, R1, and the resistive sensor (Input).

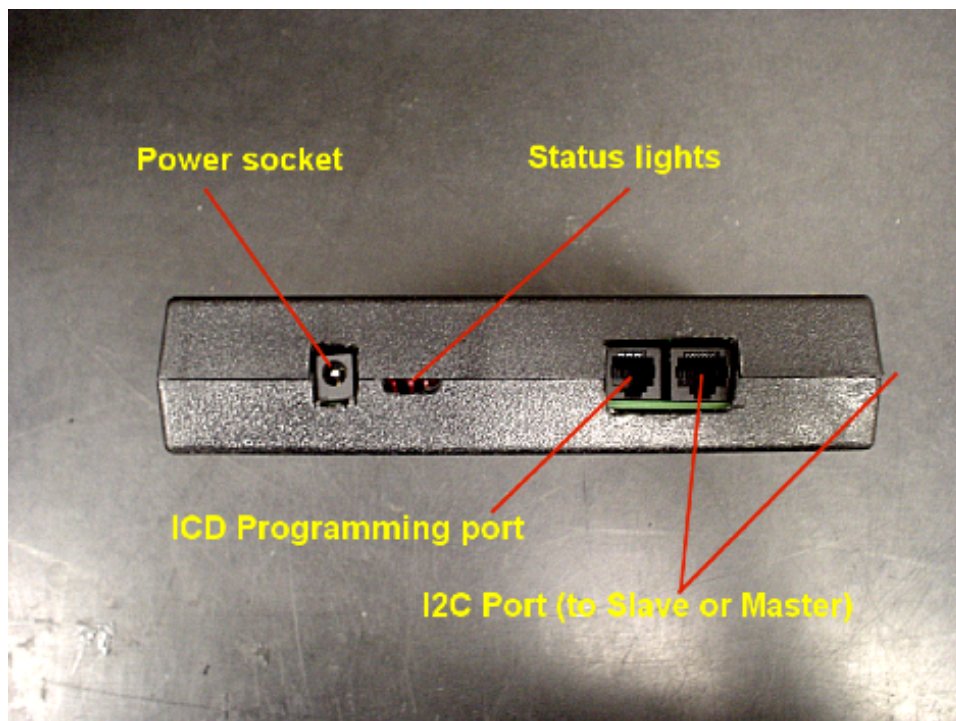


Figure B.14. Argus Superslave side view.

There are four such banks on an Argus Dongle, ordered from left to right, with each bank having the configuration shown in Table B.4 and Figure B.18.

Table B.4. Single Argus Dongle bank

Out	GND	Input	Vcc
R4	R3	R2	R1
R4	R3	R2	R1

Each Argus Dongle bank utilizes a non-inverting amplifier circuit as shown in Figure B.19 (which may be bypassed if the sensor does not require analog amplification). In this circuit, $R1$ shall represent the resistive sensor (actually $R1$ and the sensor are in series, but

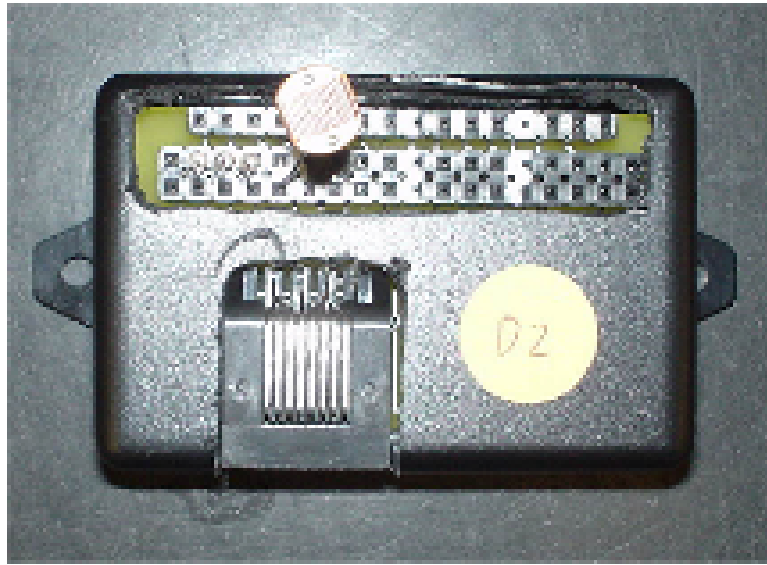


Figure B.15. Argus sensor Dongle with light sensor.

$R1$ is optional and usually bypassed, and we refer to the sensor as $R1$). $R2$ forms a voltage divider along with $R1$. $R3$ and $R4$ change the gains on the operational amplifier. The output of a Dongle sensor bank (amplified) is as follows:

- $Amplifier_input_voltage(A) = (R2/(R1 + R2)) * 5V$
- $Amplifier_gain(G) = 1 + R3/R4$
- $Amplifier_output(O) = A * G$
- $Argus_conversion(C) = 255/5V$
- $Argus_output(X) = O * C, or (A * G * C)$

A quick way we use to calculate the raw reading that Argus returns is by using the equation, $X = (R2 * 510)/(R1 + R2)$, if given $R1$ (the sensor) and $R2$, and assuming gains is set to 2 ($R3$ is equal to $R4$) Since $R1$ is in the denominator, the Argus range is nonlinear compared with the sensor's range. That is, the larger the difference between $R1$ and $R2$, the less Argus can differentiate the readings. This can be summed up as:

- When $R1 < R2$, we get the maximum reading from Argus.
- When $R1 = R2$, we get the maximum reading from Argus.

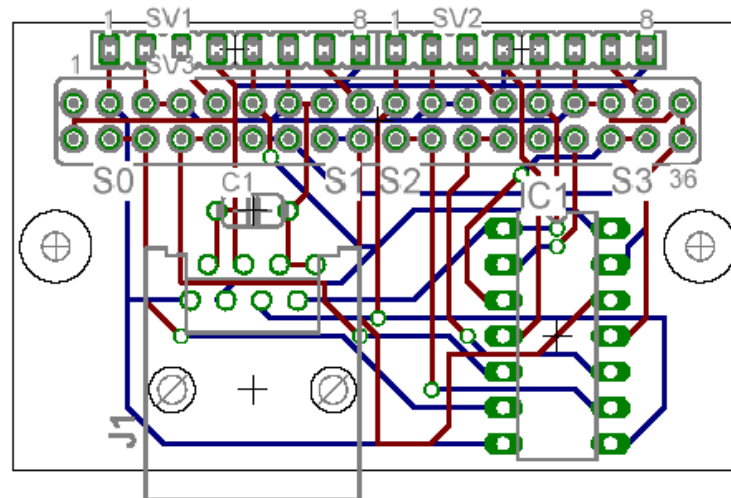


Figure B.16. Argus Dongle layout schematic (2 in. x 1.25 in.).

- When $R1 > R2$, we get a decreasing reading from Argus, where the precision also decreases.

When choosing a resistance for $R2$, we want a value that is near the lower extreme of the sensor's desired range. For example, if we have a light sensor which reads from 500 ohms to 5000 ohms in our desired range, we would choose an $R2$ of 500. This does hurt our precision badly at readings approaching 5000, but luckily the light sensor's resistive values also increase exponentially as it gets darker! However, this is not the case for all sensors. We may have to sacrifice all readings at the low extreme in order to get more accurate readings at the high extreme (i.e., choosing an $R2$ of 1000 instead cuts off all readings under 1000), but this gives us a little better precision as it approaches 5000. A poor tradeoff, but we do have another option which is to swap $R1$ and $R2$. The sensor goes where $R2$ is slotted, and $R2$ goes where the sensor is slotted. In which case our equation becomes $X = (R1 * 510)/(R1 + R2)$. With a corresponding swap in effect:

- When $R1 < R2$, we get a decreasing reading from Argus, where the precision also drops off.

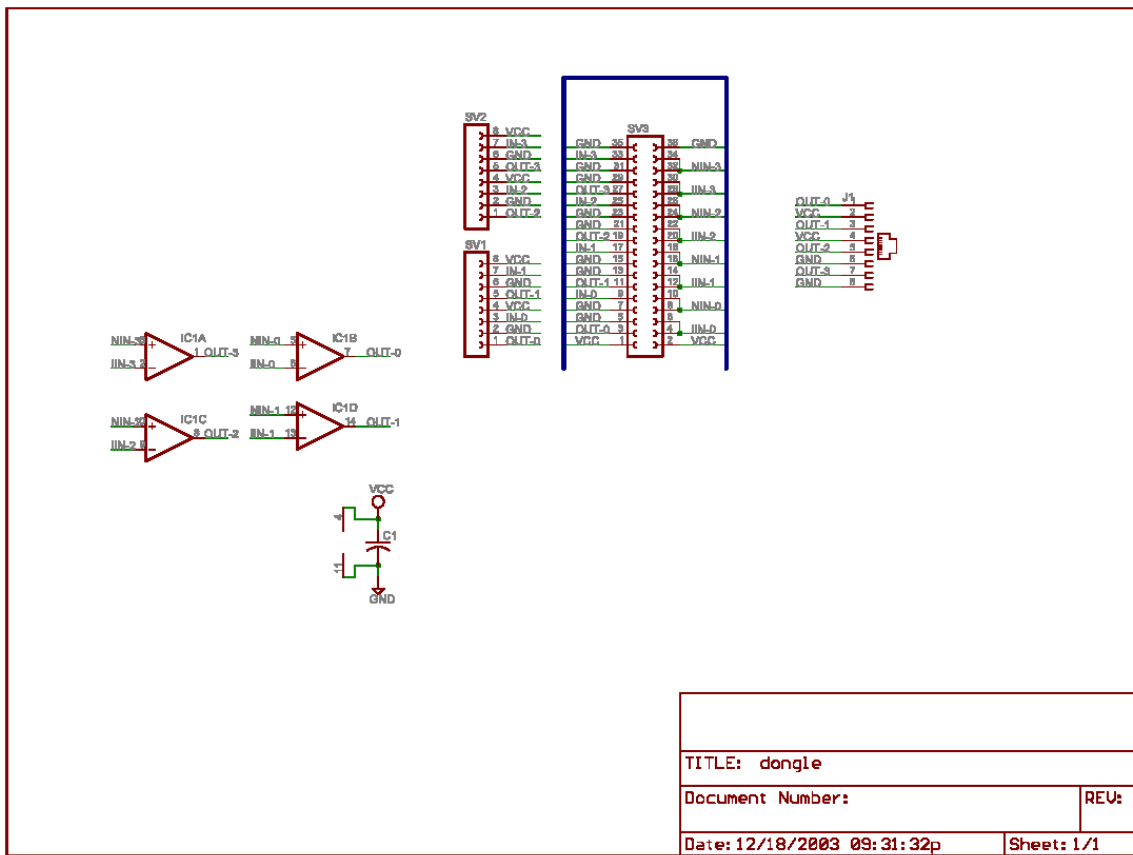


Figure B.17. Argus Dongle circuit diagram.

- When $R1 = R2$, we get the maximum reading from Argus.
- When $R1 > R2$, we get the maximum reading from Argus.

That is, we now choose the maximum extreme, and decreasing resistances increasingly lose precision. Readings are also inverted, so in the case of our light sensor, where high readings used to correspond to more light, they now correspond to less light.

If one wishes to decrease the Argus range in trade for more precision, one can increase the gains $(1 + R3/R4)$. For example, an $R3$ of 200 and an $R4$ of 200 gives you a gain of $(1 + 200/200) = 2$; while an $R3$ of 400 and an $R4$ of 200 gives you a gain of $(1 + 400/200) = 3$. Assumably you could invert the gain, and get more range for less precision. Gain modification has had little testing.

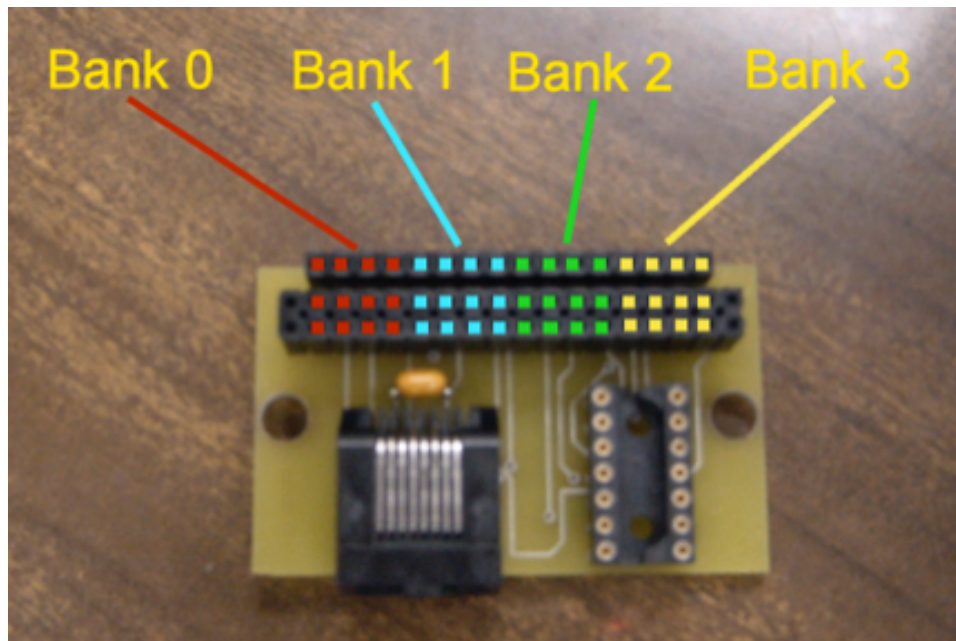


Figure B.18. Argus Dongle sensor banks.

Figure B.20 is an example of a simple analog sensor circuit that requires an op-amp gain circuit. Note that the *R1* slot is optional. This is the typical configuration of temperature and light sensors.

Figure B.21 is an example of a digital input. The digital input is directly attached to output socket header. In certain devices (such as the CO₂ sensors) it is important that the positive line is not switched with the signal out. Positive is usually red or orange and signal out is blue or black; otherwise, damage could be caused to the optoisolator. This is a typical configuration for reed switches on doors, windows, and cabinets as well as pressure switches placed in furniture seating surfaces.

Figure B.22 is an example of an analog sensor that supplies its own voltage output signal. It does **NOT** require the op-amp, so we bypass it. This is a typical configuration for humidity sensors. This configuration may require modification to the op-amp circuit as shown in Figure B.23. The following output pin must be lifted (i.e., bent) from the socket or the output pin on

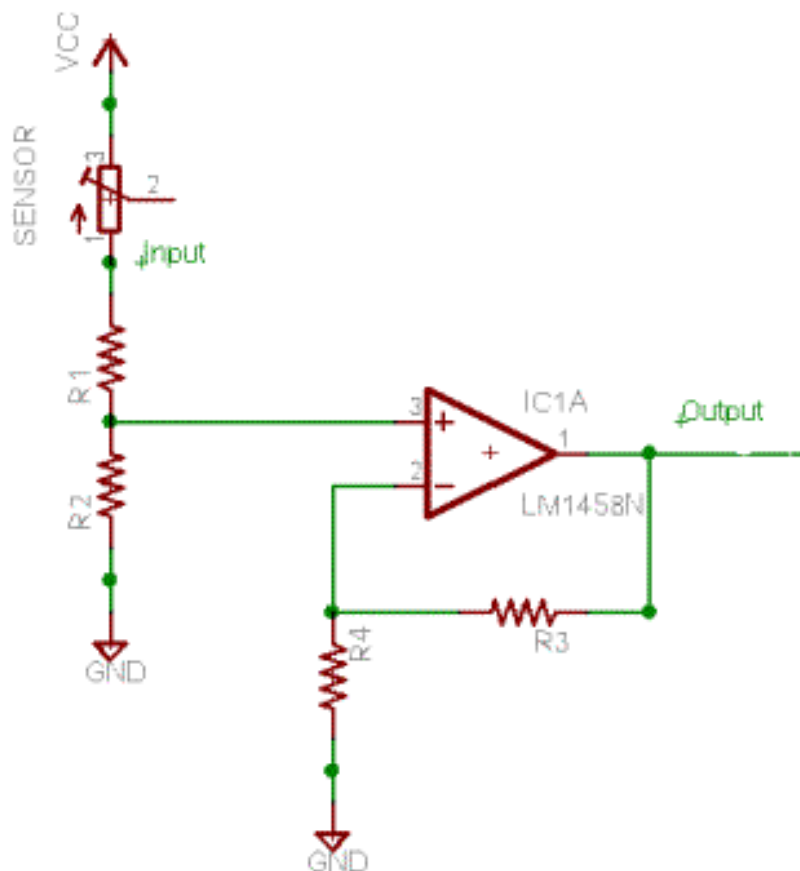


Figure B.19. Dongle amplification circuit.

the operational amplifier acts as a voltage divider and affects the reading. If one does not care about losing minor output accuracy, this alteration need not be performed.

Table B.5 shows the recommended resistors for the particular sensors we use, although the resistive values of sensors may vary wildly among manufacturers, or even among different models in the same line. For these sensors we have decided to choose a value somewhat higher than the lower extreme, but that still gave us good range among the possible readings in the environment—the MavPad is unlikely to give a thermistor reading of freezing or boiling temperatures. In all cases we experimented with the sensor first to get an idea of the possible ranges in the environment in which we were interested.

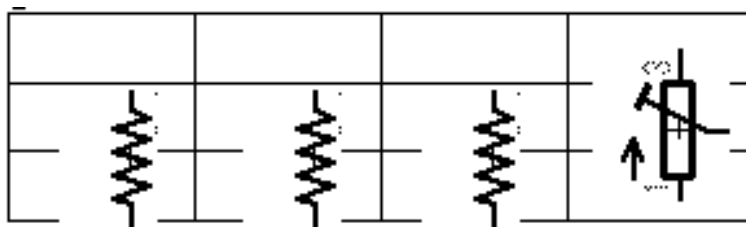


Figure B.20. Dongle loadout for a typical analog sensor.

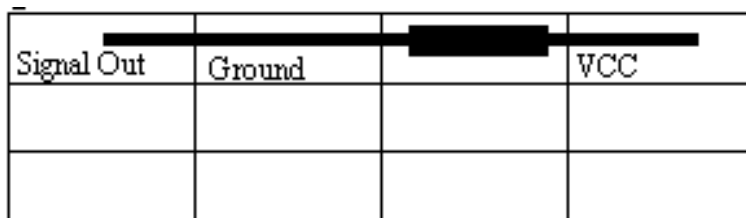


Figure B.21. Dongle loadout for a typical digital sensor/switch.

Table B.5. Example resistor values for Dongle sensor loadout

Sensor Type	Typical Values	R4 Gain Denominator	R3 Gain Numerator	R2 Lower Extreme	R1 Optional
Light sensor	500 to 3K ohms	1K ohms	1K ohms	200 ohms	None
Thermistor	900 to 1.1K ohms	1K ohms	1K ohms	360 ohms	None
Pressure	20K to 20M ohms	1K ohms	1K ohms	33K ohms	None

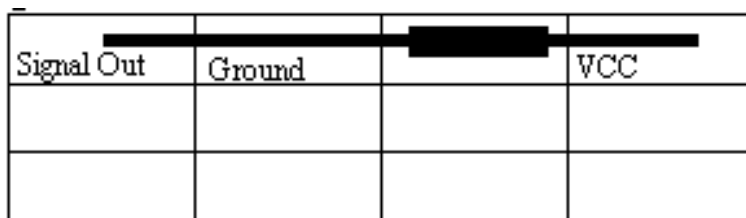


Figure B.22. Dongle loadout for a “blackbox” analog sensor device.

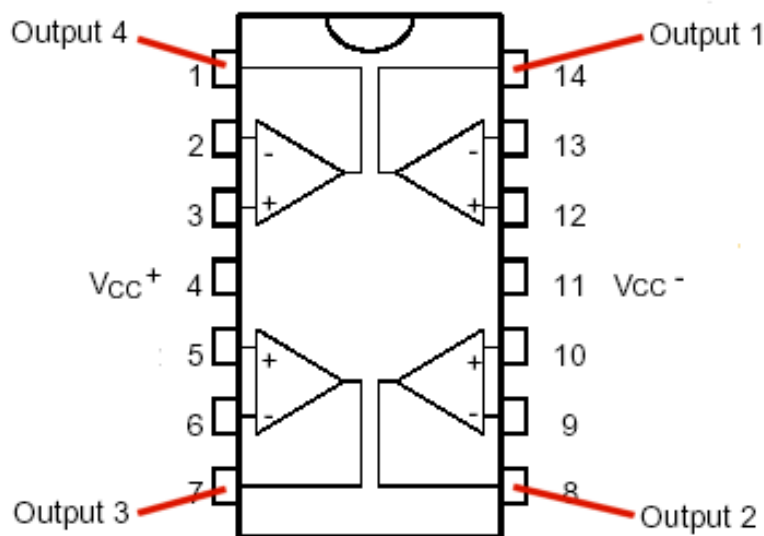


Figure B.23. Output pins on operational amplifier.

Interested implementers should visit the MavHome project website at mavhome.uta.edu to download all of the appropriate eagle files, latest information, documentation, and Argus-Hermes software for ArgusMS. We also provide example application clients to interface with the Hermes drivers.

B.4.2 ArgusD

The Argus Master may be modified to collect only digital data at a very high rate of speed. With an add-on daughterboard, the Master has extra ports to both connect to and power digital sensors (or switches) and is called ArgusD for Argus Digital as shown in Figure B.24. The daughterboard schematic is shown in Figure B.25 and the layout schematic in Figure B.26. ArgusD has the following features:

- No Superslave required
- Supports up to twenty digital sensors or switches
- Fast rate of data collection—8 Hz
- Supports long lengths of cable to the sensors (100+ ft.)

- Requires only a software change
- Feeds data directly to a host computer
- To be used when only digital data is needed

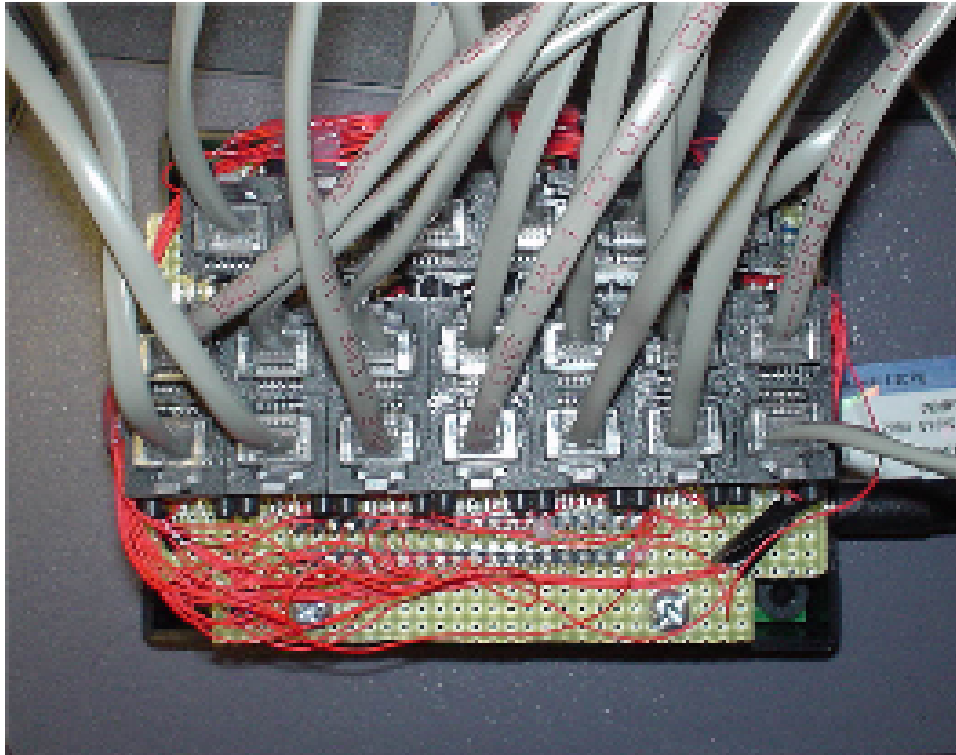


Figure B.24. Argus Digital with motion sensor loadout.

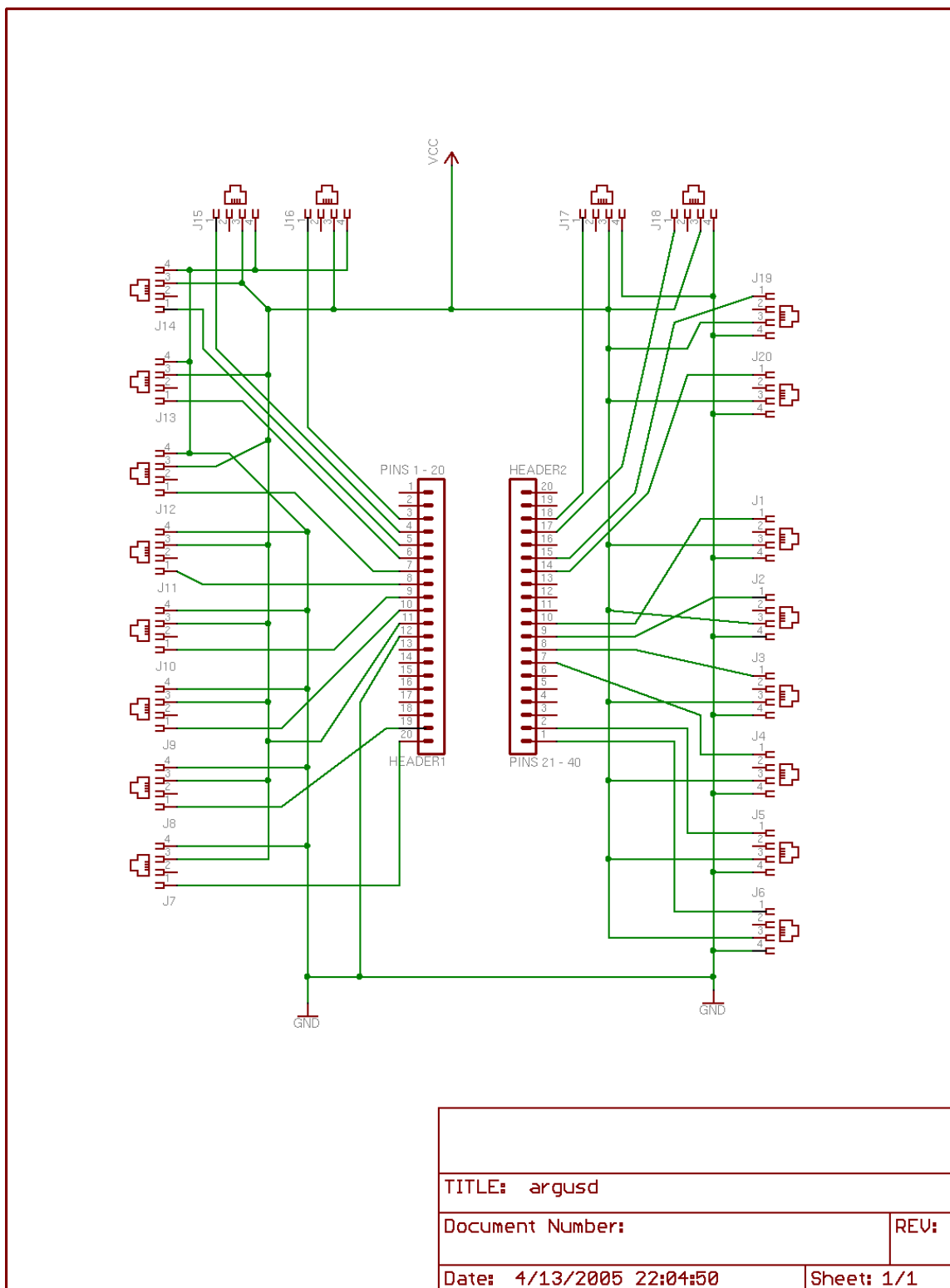


Figure B.25. Argus Digital circuit schematic.

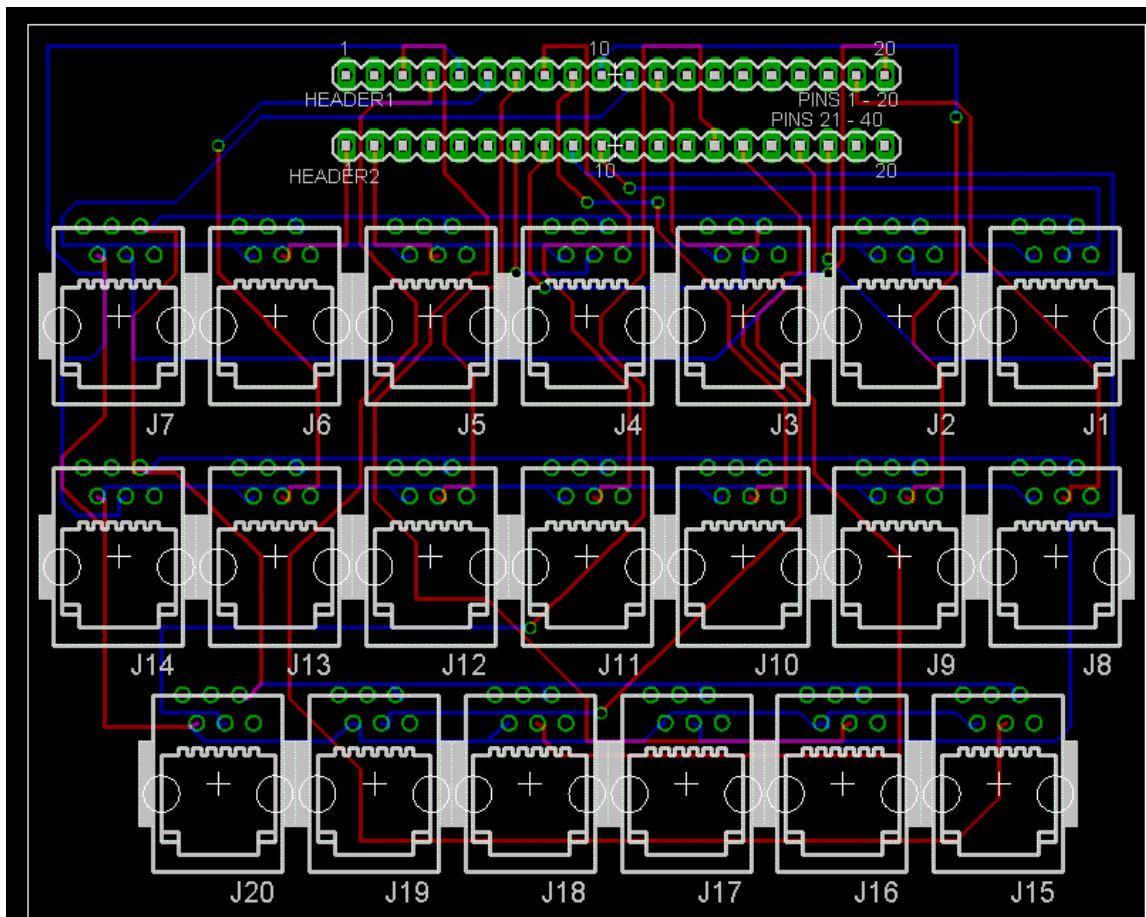


Figure B.26. Argus Digital layout schematic.

Each high speed analog board has 20 RJ-12 jacks located on the top panel for sensor input. To setup ArgusD for operation follow these steps:

1. Connect power to the Argus Digital
2. Keep tabs on the status lights
3. Connect digital sensors to Digital ports (RJ-11 phone sockets)
4. Connect RS-232 serial cable to the Master's 1/8 inch stereo socket
5. Connect RS-232 serial cable to the host PC's DB-9 serial socket

Interested implementers should visit the MavHome project website at mavhome.uta.edu to download all of the appropriate eagle files, latest information, documentation, and Argus-Hermes software for ArgusD. We also provide example application clients to interface with the Hermes drivers.

B.4.3 ArgusM

The Argus Master may also be modified to control stepper motors. When installed with an add-on daughter board, the Master may power and control up to four stepper motors in a setup we call Argus Motors (ArgusM) as shown in Figure B.27. The schematic for ArgusM is shown in Figure B.28 and the layout schematic in Figure B.29. The Argus Motor controller interfaces directly with a host PC. ArgusM has the following features:

- No Superslave required
- Uses daughter board configured for stepper motors
- Supports up to four motors
- Onboard ULN2003 stepper motor driver chip isolates and protects the microcontroller from the large currents needed by the motors
- Requires only software changes
- Connects directly to a host PC

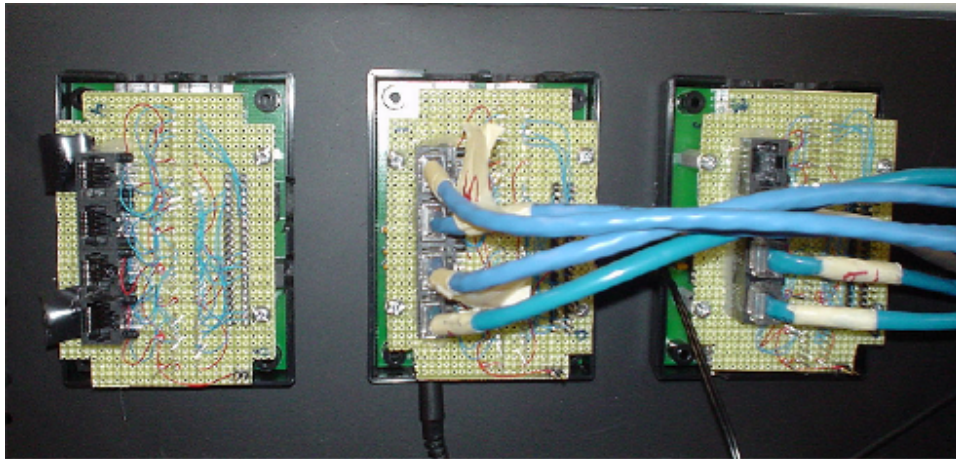


Figure B.27. Argus Motors with two sets connected to mini-blinds.

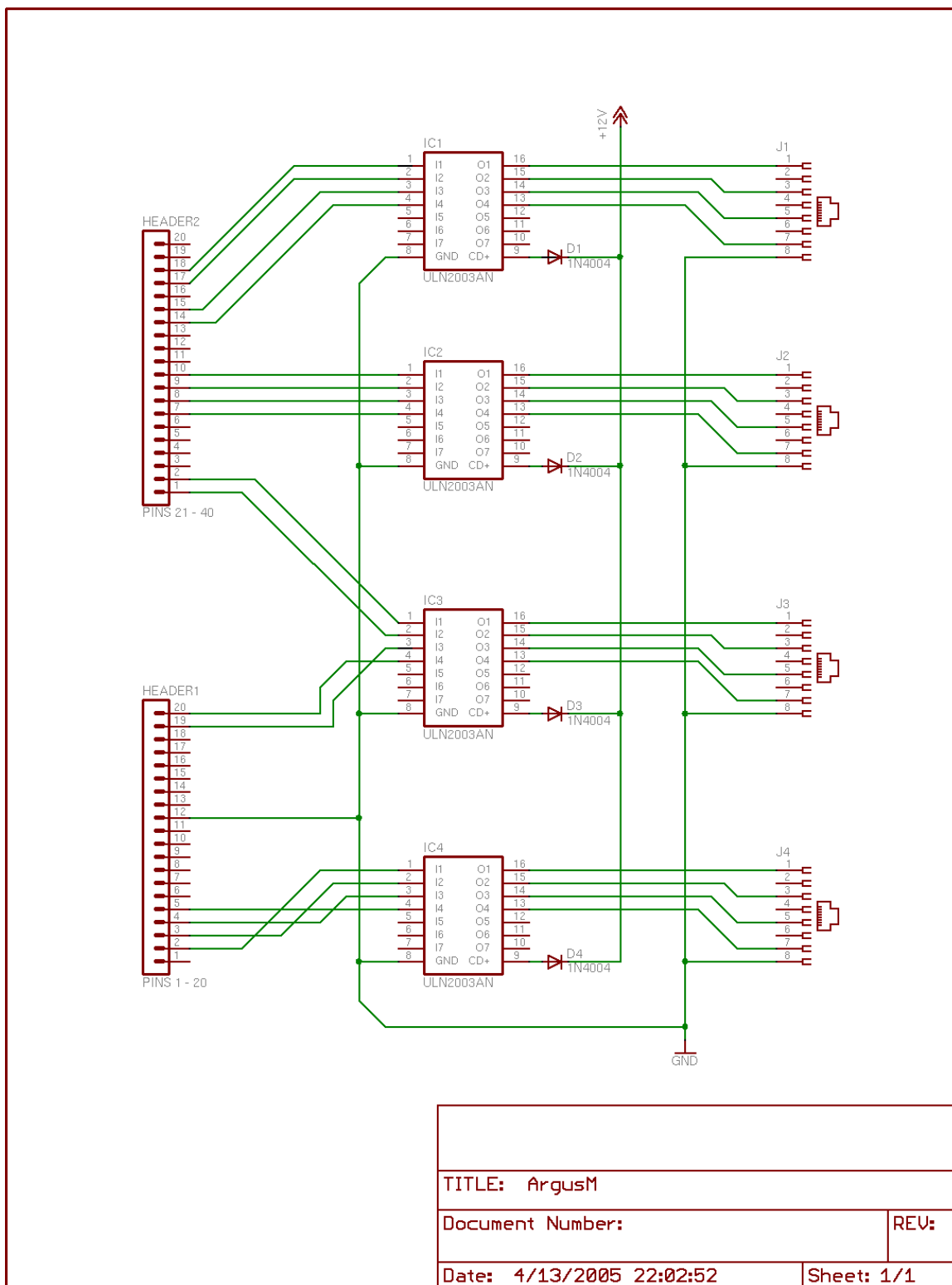


Figure B.28. Argus Motor circuit schematic.

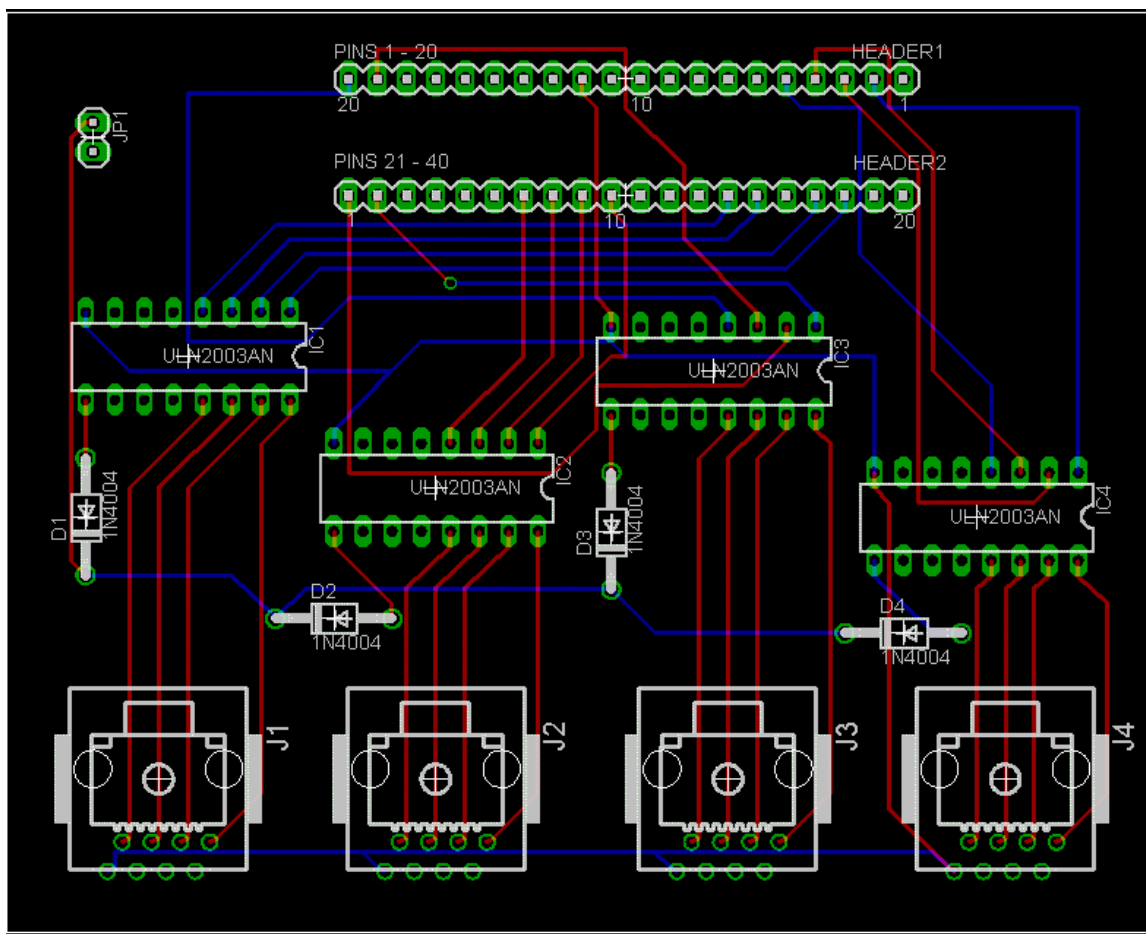


Figure B.29. Argus Motor layout schematic.

ArgusM controls up to four separate stepper motors. These motors are four-phase unipolar stepper motors. There are four coils in such motors, one line per coil, plus a common line. The minimum lines coming out of these motors is five. Some unipolar stepper motors have six wires, but these simply double the common line. ArgusM sends a signal to one line at a time—an active-low signal to activate one of the coils. The common is at high potential, so a low signal from ArgusM activates and a high signal deactivates. While any coil is active, the motor will prevent any movement. ArgusM deactivates all coils when done, allowing someone to change the position manually. We typically step through one coil at a time in a clockwise or

counterclockwise fashion, for a four-step revolution. We could possibly make it an eight-step revolution for more power, by using two coils at the same time to give us in-between points, but the system is currently programmed for four steps.

Since stepper motors use different color coding of their wires, the best way to figure out the mapping is by testing their resistances with a multimeter and power supply. This procedure is as follows:

1. For a five wire stepper motor, we are trying to find four coils and one common. For a six wire, we are trying to find four coils and two commons.
2. Test the resistance between all wires. The common wire should have half the resistance to a coil, as the coils have to each other.
3. Supply voltage to the common(s). 5 volts or 12 volts are acceptable
4. Ground one wire and assume this is Coil 4.
5. Ground the other wires one by one and release when done.
6. If the motor turns clockwise, label this wire Coil 3.
7. If the motor does not turn, label this wire Coil 2.
8. If the motor turns counterclockwise, label this wire Coil 1.

Now that we have the coils identified, we may map them to a CAT-5 Ethernet cable we use to carry the control signals. CAT-5 is chosen because of its ubiquity and price—plus it reduces the number of tools necessary in the lab and personnel capable of making CAT-5 wire are easy to find when needed (most likely CS, CSE, and EE departments are full of people with this skill). Table B.6 shows the mapping from CAT-5 to the stepper motor wires.

Table B.6. CAT-5 Ethernet cable wire mapping to stepper motor coils

Ethernet	Coil #
Green	Coil 1
Orange	Coil 2
Blue	Coil 3
Brown	Coil 4
Orange/White	Common

Each motor controller has four RJ-45 Ethernet jacks for stepper motor power and control, located on the top panel. To setup ArgusM follow these steps:

1. Connect power to the Argus Motor
2. Keep tabs on the status lights
3. Connect Argus stepper motors to RJ-45 Ethernet jack
4. Connect RS-232 serial cable to Argus Motor's 1/8 inch stereo socket
5. Connect RS-232 serial cable to host PC's DB-9 serial port
6. Check lights again to verify system is operational

Interested implementers should visit the MavHome project website at mavhome.uta.edu to download all of the appropriate eagle files, latest information, documentation, and Argus-Hermes software for ArgusM. We also provide example application clients to interface with the Hermes drivers.

B.4.4 ArgusAD

The Argus Master may also be configured to collect both analog and digital data at a high rate of speed, a compromise between Argus Digital and Argus Master-Slave. Using a

daughterboard, the Argus Analog + Digital, or ArgusAD, as shown in Figure B.30 may act as a single analog and digital data collection device that streams the data directly to a host PC. The circuit schematic for ArgusAD is shown in Figure B.31 and the layout schematic in Figure B.32. ArgusAD has the following features:

- No Superslave required
- Supports up to eight analog sensors (via Argus Dongles)
- Supports up to ten digital sensors/switches
- Fast rate of data collection—8 Hz
- Requires only a software change
- Feeds data directly to a host computer

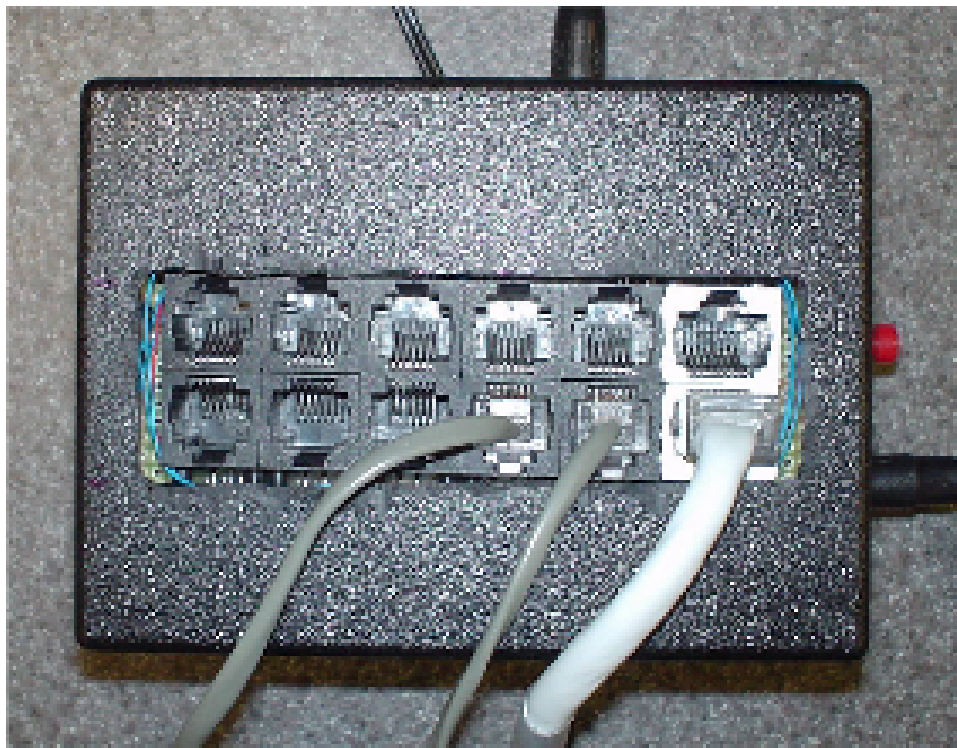


Figure B.30. Argus Analog + Digital.

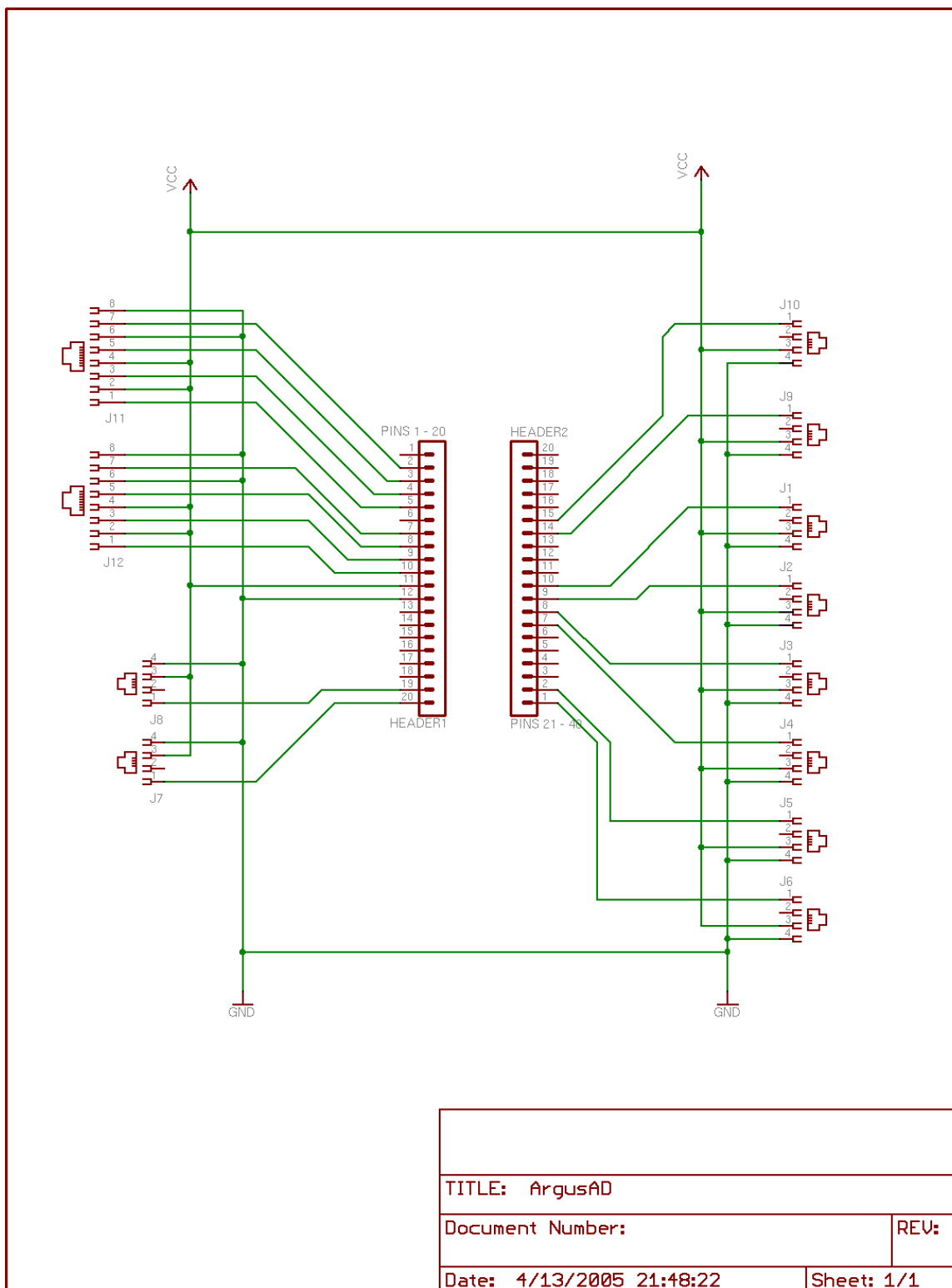


Figure B.31. Argus Analog + Digital circuit schematic.

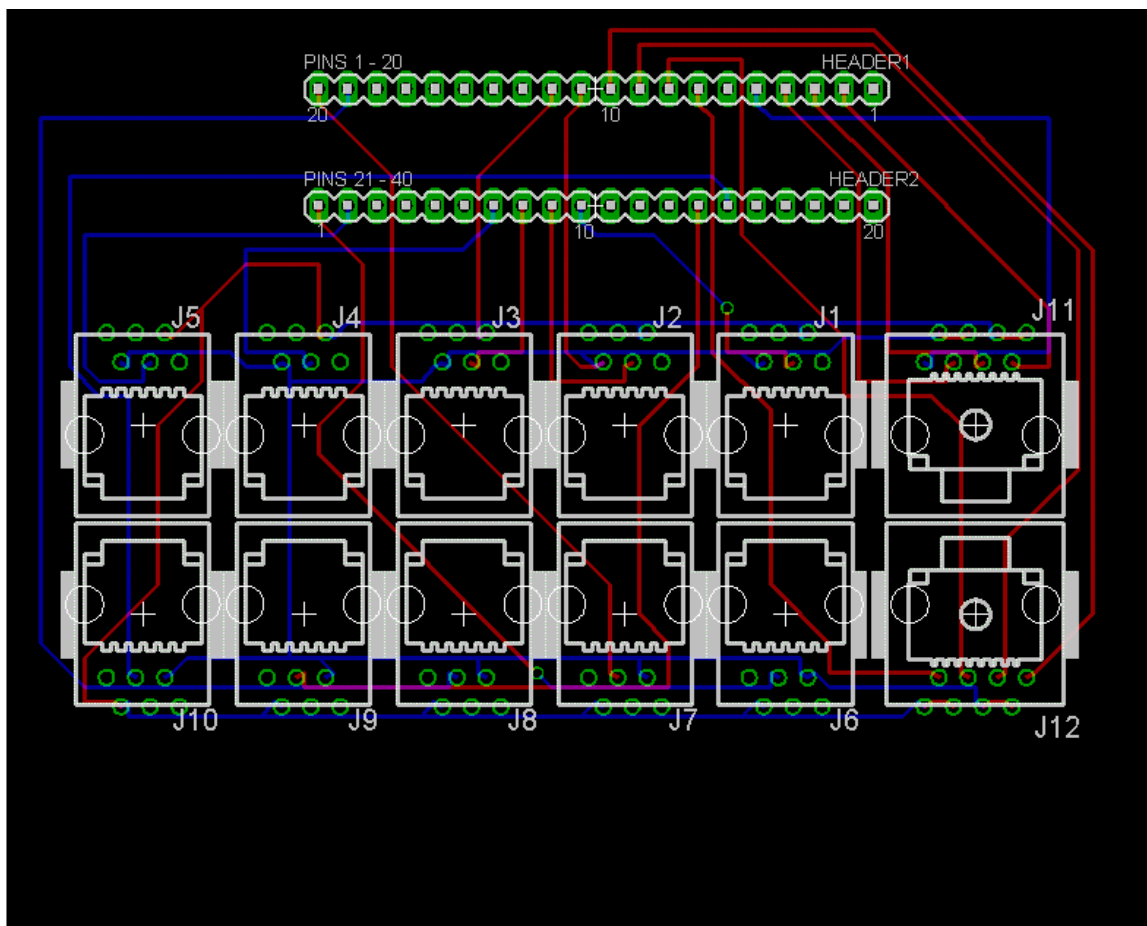


Figure B.32. Argus Analog + Digital layout schematic.

Each Argus Analog-Digital board has two RJ-45 Ethernet jacks for analog sensors and ten RJ-11 phone jacks for digital sensors, all located on the top panel as shown in Figure B.33.

To setup ArgusAD follow these steps:

1. Connect power to the Argus Analog-Digital
2. Keep tabs on the status lights
3. Connect Argus Sensor Dongles to Analog Ports (RJ-45 Ethernet sockets)
4. Connect digital sensors to Digital ports (RJ-11 phone sockets)
5. Connect RS-232 serial cable to the Master's 1/8 inch stereo socket

6. Connect RS-232 serial cable to the host PC's DB-9 serial socket

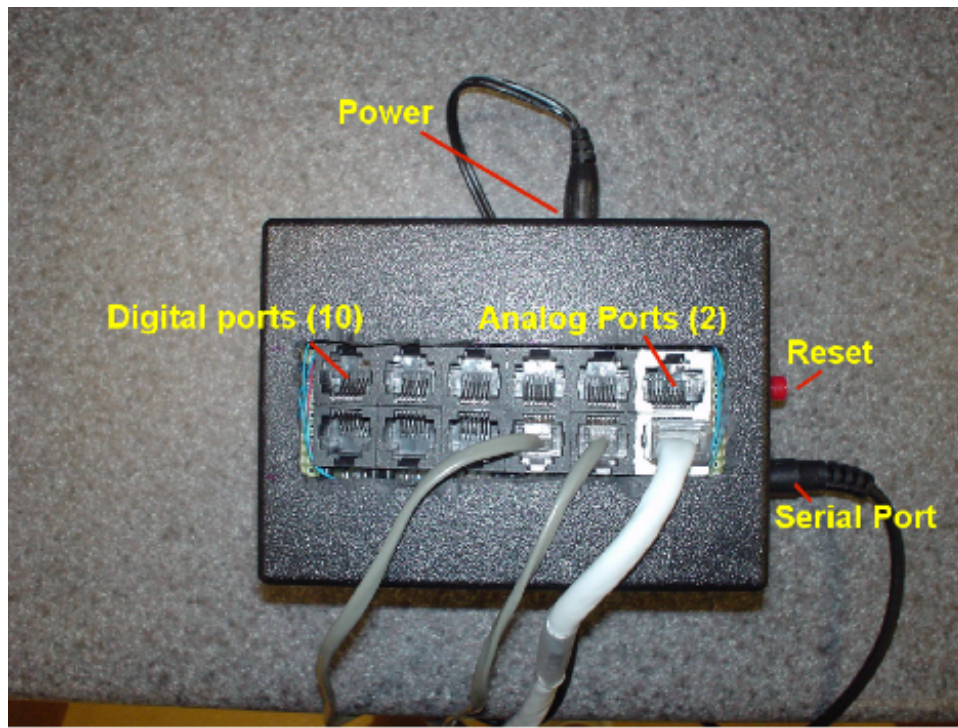


Figure B.33. ArgusAD top view.

Interested implementers should visit the MavHome project website at mavhome.uta.edu to download all of the appropriate eagle files, latest information, documentation, and Argus-Hermes software for ArgusAD. We also provide example application clients to interface with the Hermes drivers.

B.5 Acknowledgements

Work on the sensor networks for the intelligent environment has been ongoing at The University of Texas at Arlington since 2001. It started with a senior design project called MUSCLE lead by Eric Kolkmeier under the direction of Dr. Farhad Kamangar. In 2003 Michael Youngblood and Kien Tran redesigned the system based on the original MUSCLE designs. We

even consulted with Kolkmeier for a time. Tran finished the initial design work in 2003. Tran and Chris Lance under the direction of Youngblood completed the designs and software and produced the first full prototypes in early 2004. From mid-2004 through the present Youngblood and Lance have continued to improve the system and software.

APPENDIX C

The MavPad Environment

C.1 MavPad Description

The MavPad is an on-campus student apartment located at University Village on The University of Texas at Arlington's campus. The apartment consists of a living/dining room combination, a kitchen, a bathroom, a large bedroom, and a walk-in closet. The living room is shown in Figure C.1 and the dining portion is shown in Figure C.2. The kitchen is small and shown in Figure C.3. The bedroom supports two people and can be seen in Figure C.4. The bathroom contains a sink, toilet, and a shower. The bedroom closet is large enough to hold clothes as well as computer equipment including the network routers and our master server.



Figure C.1. MavPad living room.



Figure C.2. MavPad dining room.

C.2 Computing Power

The Mavpad hosts four Pentium 4 class SuSe 9.1-based Linux machines. Two machines act as sensor collectors for the ArgusD networks as well as for X-10. These machines are located in the living room and the bedroom. The machine in the kitchen supports X-10, the ArgusM network, and the ArgusMS network. The machine in the closet is the master server which hosts the experimental software systems for decision-making as presented in this dissertation. The system uses a 100bTX network and is connected to the Internet through a 4Mb cable modem donated by Comcast. There currently is not any wireless connectivity in this environment.



Figure C.3. MavPad kitchen.

C.3 Sensing and Actuation

The MavPad has a deployed ArgusD network with two units supporting up to forty sensors, a single ArgusM network supporting the two mini-blinds in the apartment, an ArgusMS network with three Superslaves and fourteen Dongles total, and an X-10 system covering three zones. This environment is also equipped with an X-10 controlled HVAC system and a water leakage detection system. Figures C.5 and C.6 show some of the ArgusMS Dongles deployed. Figure C.7 shows the ArgusD passive infra-red (PIR) motion sensor array mounted on the ceiling. The X-10 deployment is shown in Figure C.8 along with the ArgusM actuators for the mini-blinds. The ArgusMS and ArgusD sensors are shown in Figure C.9. A complete list of deployed sensors and actuators for the MavPad is presented in Table C.1 and the Argus network deployment costs are shown in Table C.2.



Figure C.4. MavPad bedroom.



Figure C.5. ArgusMS Dongle on the wall.



Figure C.6. ArgusMS Dongle sensors by the window near an X-10 controlled floor fan.



Figure C.7. ArgusD ceiling mounted PIR motion sensors.

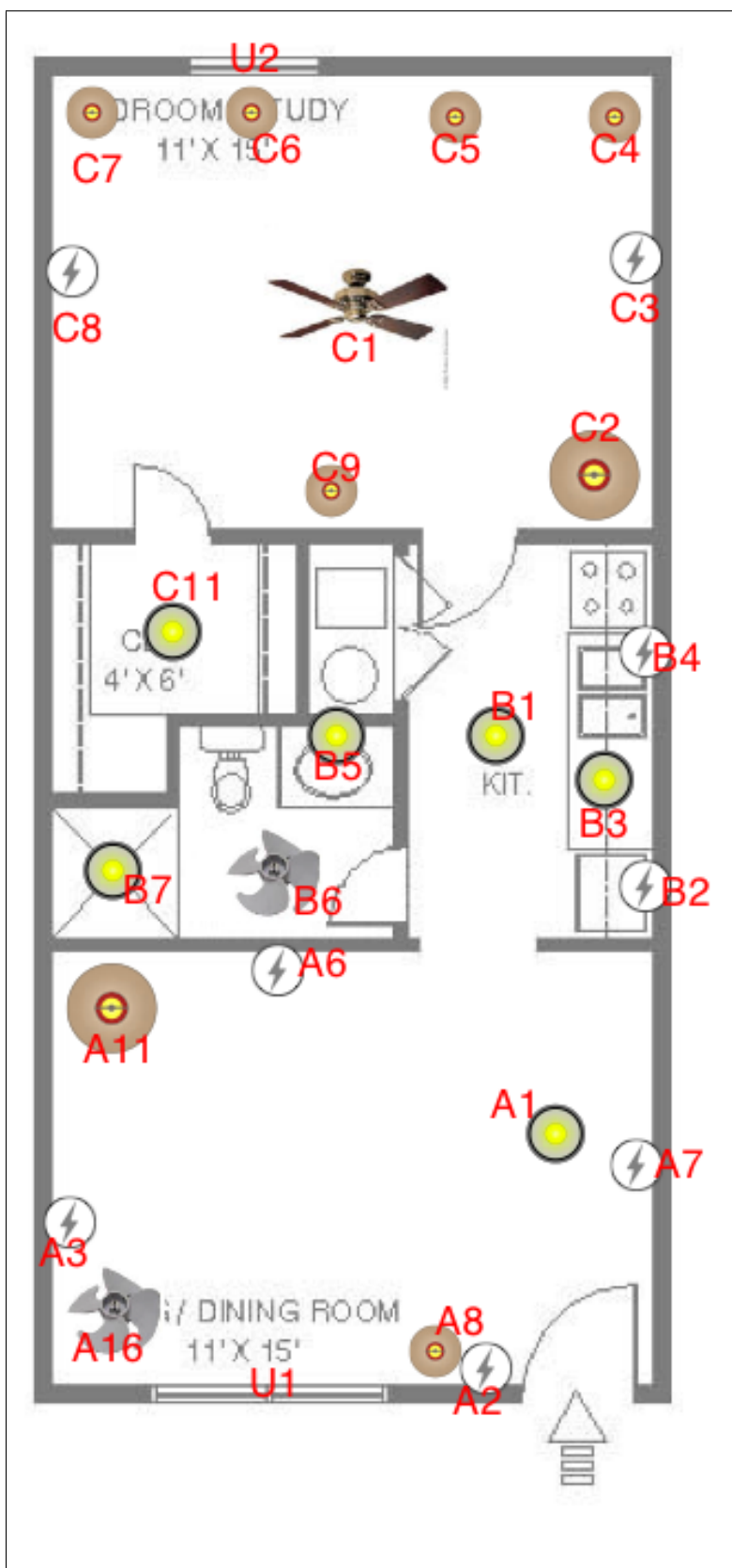


Figure C.8. MavPad X-10 and ArgusM Actuators.

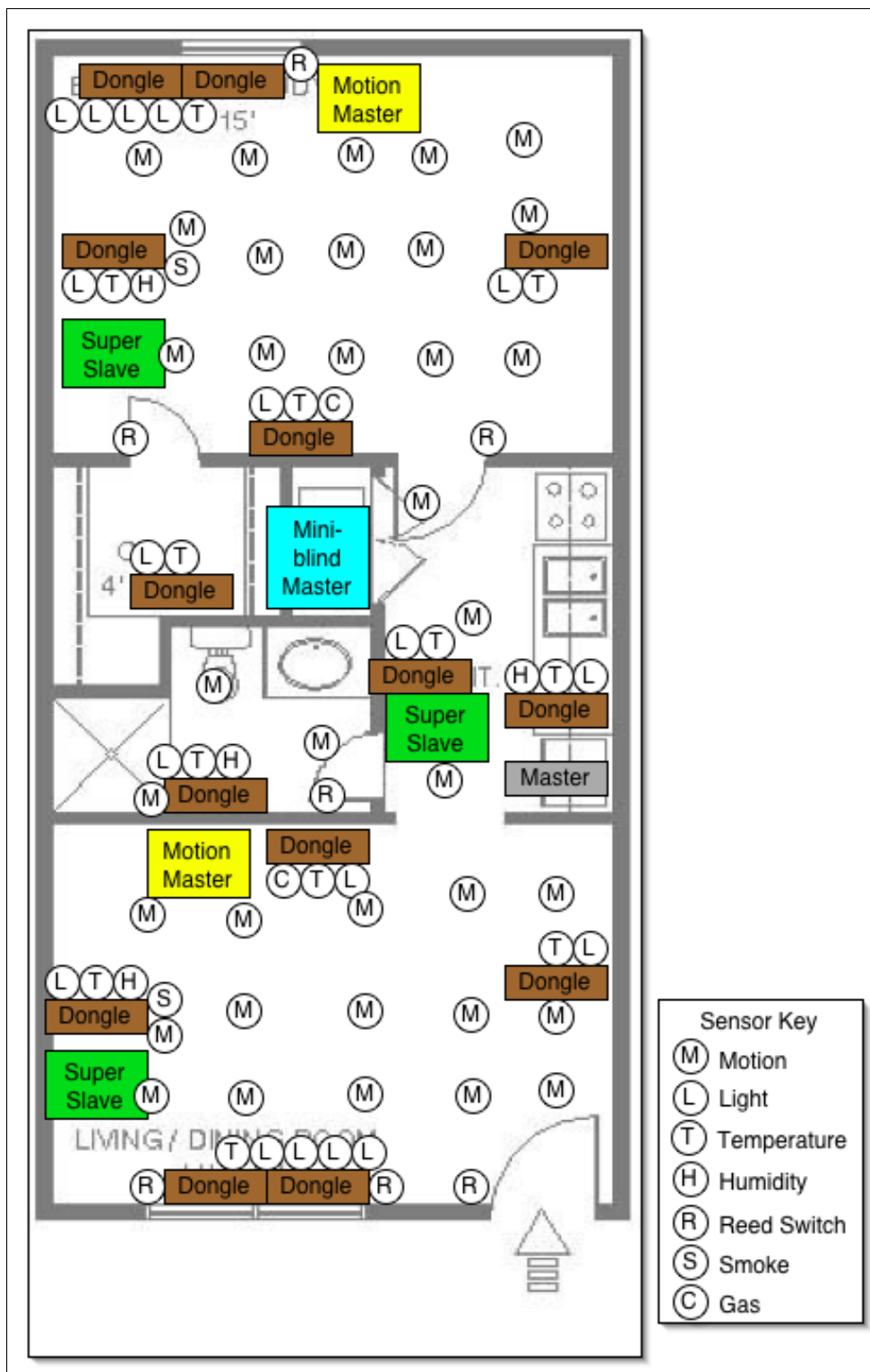


Figure C.9. MavPad ArgusMS and ArgusD sensors.

Table C.1: MavPad Sensors and Actuators.

Zone	ID	Type	Source	Room	Location
A	1	Ceiling light	X10	Living Room	Overhead light near north wall
A	2	Electrical outlet	X10	Living Room	East wall near door, top outlet
A	3	Electrical outlet	X10	Living Room	Southeast corner of room, top outlet
A	6	Electrical outlet	X10	Living Room	West wall behind entertainment center, top outlet
A	7	Electrical outlet	X10	Living Room	North wall near table, top outlet
A	8	Table lamp	X10	Living Room	Near door on side table
A	9	Electrical outlet	X10	Kitchen	North wall over counter, top outlet
A	11	Floor Lamp	X10	Living Room	Southwest corner
A	16	Floor Fan	X10	Living Room	Southeast corner of room, top outlet
B	1	Ceiling light	X10	Kitchen	Overhead in kitchen
B	2	Electrical outlet	X10	Kitchen	Behind refrigerator
B	3	Counter lights	X10	Kitchen	Over counter, under cabinets
B	4	Electrical outlet	X10	Kitchen	Over counter, on left near oven, top outlet
B	5	Bathroom light	X10	Bathroom	Over mirror
B	6	Bathroom fan	X10	Bathroom	On ceiling. Will not turn off with remote.
B	7	Shower light	X10	Bathroom	Over shower

Table C.1 – continued

Zone	ID	Type	Source	Room	Location
B	13	Argus Motor	X10	Kitchen	On top of fridge
C	1	Overhead fan	X10	Bedroom	On ceiling
C	2	Room light	X10	Bedroom	Northeast corner
C	3	Luxo lamp	X10	Bedroom	Attached to north bed
C	4	Luxo lamp	X10	Bedroom	On north computer desk
C	5	Luxo lamp	X10	Bedroom	On south computer desk
C	6	Luxo lamp	X10	Bedroom	Attached to south bed
C	7	Electrical outlet	X10	Bedroom	South wall behind bed, top outlet
C	8	Table lamp	X10	Bedroom	East wall, on dresser
C	10	Closet light	X10	Bedroom	In closet, on ceiling
S	1	Light	ArgusMS	Living Room	In cluster P9, west wall. Facing into room
S	2	Heat	ArgusMS	Living Room	In cluster P9, west wall. Facing into room
S	3	CO	ArgusMS	Living Room	Next to cluster P9, west wall.
S	5	Light	ArgusMS	Living Room	In cluster P10, south wall. Facing into room
S	6	Heat	ArgusMS	Living Room	In cluster P10, south wall. Facing into room
S	7	Humidity	ArgusMS	Living Room	In cluster P10, south wall. Facing into room

Table C.1 – continued

Zone ID	Type	Source	Room	Location
S 8	Smoke	ArgusMS	Living Room	Next to cluster P10, south wall.
S 9	Heat	ArgusMS	Living Room	In cluster P11, east wall. Facing into room
S 10	Reed switch	ArgusMS	Living Room	Reed switch attached to left window
S 11	Reed switch	ArgusMS	Living Room	Reed switch attached to right window
S 13	Light	ArgusMS	Living Room	On cardboard between window and blinds, facing out
S 14	Light	ArgusMS	Living Room	On cardboard between window and blinds, facing in
S 15	Light	ArgusMS	Living Room	Taped near window, facing miniblinds
S 16	Light	ArgusMS	Living Room	In cluster P12, east wall. Facing into room.
S 17	Light	ArgusMS	Living Room	In cluster P13, north wall. Facing into room.
S 18	Heat	ArgusMS	Living Room	In cluster P13, north wall. Facing into room.
S 19	Reed switch	ArgusMS	Living Room	Reed switch over front door.
S 65	Light	ArgusMS	Bedroom	On cardboard between window and blinds, facing out
S 66	Light	ArgusMS	Bedroom	On cardboard between window and blinds, facing in
S 67	Light	ArgusMS	Bedroom	Taped near window, facing miniblinds
S 68	Light	ArgusMS	Bedroom	In cluster P1, west wall. Facing into room
S 69	Heat	ArgusMS	Bedroom	In cluster P2, west wall. Facing into room

Table C.1 – continued

Zone	ID	Type	Source	Room	Location
S	70	Reed switch	ArgusMS	Bedroom	Reed switch attached to window
S	73	Light	ArgusMS	Bedroom	In cluster P3, south wall. Facing into room
S	74	Heat	ArgusMS	Bedroom	In cluster P3, south wall. Facing into room
S	75	Smoke	ArgusMS	Bedroom	Near cluster P3, south wall.
S	77	Light	ArgusMS	Bedroom	In cluster P4, north wall. Facing into room
S	78	Heat	ArgusMS	Bedroom	In cluster P4, north wall. Facing into room
S	79	Humidity	ArgusMS	Bedroom	In cluster P4, north wall. Facing into room
S	81	Light	ArgusMS	Bedroom	In cluster P5, east wall. Facing into room
S	82	Heat	ArgusMS	Bedroom	In cluster P5, east wall. Facing into room
S	83	Reed switch	ArgusMS	Bedroom	Reed switch over front door.
S	84	CO	ArgusMS	Bedroom	Near cluster P5, east wall.
S	85	Light	ArgusMS	Bedroom	In cluster P6, middle of closet. Facing into room.
S	86	Heat	ArgusMS	Bedroom	In cluster P6, middle of closet. Facing into room.
S	87	Leak detector	ArgusMS	Bathroom	Bioguard 1 - under toilet
S	88	Leak detector	ArgusMS	Bedroom	Bioguard 2 - under sink
S	89	Leak detector	ArgusMS	Utility Closet	Bioguard 3 - in water heater drip pan

Table C.1 – continued

Zone	ID	Type	Source	Room	Location
S	90	Leak detector	ArgusMS	Kitchen	Bioguard 4 - under sink
S	129	Light	ArgusMS	Kitchen	In cluster P8, south wall. Facing into room
S	130	Heat	ArgusMS	Kitchen	In cluster P8, south wall. Facing into room
S	131	Reed switch	ArgusMS	Kitchen	Reed switch over left utility door
S	132	Reed switch	ArgusMS	Kitchen	Reed switch over right utility door
S	133	Light	ArgusMS	Kitchen	In cluster P7, over counter. Facing into room
S	134	Heat	ArgusMS	Kitchen	In cluster P7, over counter. Facing into room
S	137	Light	ArgusMS	Bathroom	In cluster P14, east wall. Facing into room
S	138	Heat	ArgusMS	Bathroom	In cluster P14, east wall. Facing into room
S	139	Humidity	ArgusMS	Bathroom	In cluster P14, east wall. Facing into room
S	140	Reed switch	ArgusMS	Bathroom	Reed switch over door
S	141	Vent position	ArgusMS	Living Room	Inside living room vent
S	142	Vent position	ArgusMS	Kitchen	Inside kitchen vent
S	143	Vent position	ArgusMS	Bedroom	Inside bedroom vent
S	144	Vent position	ArgusMS	Bathroom	Inside bathroom
V	1	Motion	ArgusD	Bedroom	Northwest corner of ceiling

Table C.1 – continued

Zone	ID	Type	Source	Room	Location
V	2	Motion	ArgusD	Bedroom	1 sensor south of V1
V	3	Motion	ArgusD	Bedroom	2 sensors south of V1
V	4	Motion	ArgusD	Bedroom	3 sensors south of V1
V	5	Motion	ArgusD	Bedroom	Southwest corner of ceiling
V	6	Motion	ArgusD	Bedroom	North center of ceiling
V	7	Motion	ArgusD	Bedroom	1 sensor south of V6
V	8	Motion	ArgusD	Bedroom	2 sensors south of V6
V	9	Motion	ArgusD	Bedroom	3 sensors south of V6
V	10	Motion	ArgusD	Bedroom	South center of ceiling
V	11	Motion	ArgusD	Bedroom	Northeast corner of ceiling
V	12	Motion	ArgusD	Bedroom	1 sensor south of V11
V	13	Motion	ArgusD	Bedroom	2 sensors south of V11
V	14	Motion	ArgusD	Bedroom	3 sensors south of V11
V	15	Motion	ArgusD	Bedroom	Southeast corner of ceiling
V	16	Motion	ArgusD	Bedroom	On ceiling in closet
V	17	Motion	ArgusD	Kitchen	West on ceiling

Table C.1 – continued

Zone	ID	Type	Source	Room	Location
V	19	Motion	ArgusD	Kitchen	Center on ceiling
V	20	Motion	ArgusD	Kitchen	East on ceiling
V	21	Motion	ArgusD	Bathroom	On ceiling over toilet
V	22	Motion	ArgusD	Bathroom	On ceiling over shower
V	23	Motion	ArgusD	Bathroom	On ceiling over bathroom door
V	24	Motion	ArgusD	Living Room	Northwest corner of ceiling
V	25	Motion	ArgusD	Living Room	1 sensor south of V24
V	26	Motion	ArgusD	Living Room	2 sensors south of V24
V	27	Motion	ArgusD	Living Room	3 sensors south of V24
V	28	Motion	ArgusD	Living Room	Southwest corner of ceiling
V	29	Motion	ArgusD	Living Room	North center of ceiling
V	30	Motion	ArgusD	Living Room	1 sensor south of V29
V	31	Motion	ArgusD	Living Room	2 sensors south of V29
V	32	Motion	ArgusD	Living Room	3 sensors south of V29
V	33	Motion	ArgusD	Living Room	South center of ceiling
V	34	Motion	ArgusD	Living Room	Northeast corner of ceiling

Table C.1 – continued

Zone	ID	Type	Source	Room	Location
V	35	Motion	ArgusD	Living Room	1 sensor south of V34
V	36	Motion	ArgusD	Living Room	2 sensors south of V34
V	37	Motion	ArgusD	Living Room	3 sensors south of V34
V	38	Motion	ArgusD	Living Room	Southeast corner of ceiling
V	39	Couch Seat Switch	ArgusD	Living Room	Couch - east wall
V	40	Chair Seat Switch	ArgusD	Living Room	Chair - southeast corner

Table C.2. MavPad Argus Sensor Components

Component	Quantity	Unit Price	Total Price
Master	1	\$57.04	\$57.04
Mini-blind Master	1	\$69.76	\$69.76
Motion Master	2	\$78.20	\$78.20
Super Slave	3	\$92.28	\$276.84
Dongles	14	\$8.87	\$124.18
Light Sensor	18	\$1.56	\$28.08
Temperature Sensor	12	\$1.26	\$15.12
Reed Switch	6	\$2.47	\$14.82
Humidity Sensor	4	\$20.00	\$80.00
Motion Sensor	36	\$12.00	\$432.00
Grand Total			\$1,176.04

APPENDIX D

The MavLab Environment

D.1 MavLab Description

The MavLab shown in Figure D.1 is commonly referred to as the Artificial Intelligence Lab, or AI Lab, located in 250 Nedderman Hall of The University of Texas at Arlington campus. The lab is divided into ten areas which include six cubicle workspace areas (designated A–F as shown on Figure D.7) with four desks per area as shown in Figure D.2, one large office (designated as G) as shown in Figure D.3, a living room area (designated as H) often called the MavDen and shown in Figure D.4, a kitchen area (designated as I) often called the MavKitchen and shown in Figure D.5, and a conference room (designated as J) as shown in Figure D.6.



Figure D.1. MavLab in 250 Nedderman Hall at UTA.



Figure D.2. MavLab cubicle workspaces.

D.2 Computing Power

As a work environment, the MavLab hosts over thirty computers. One computer in each space, for a total of ten computers, is designated to handle the X-10 for each area. Some of these computers are also tasked to interface to the ArgusMS network, four ArgusM networks, single ArgusAD network, and two ArgusD networks. Two additional computers are used for database services and to host the decision-making software. The Mavlab is connected through a gigabit network and includes 802.11b wireless coverage for mobile devices such as our Navigator PDA controller (a control interface for the MavLab environment implemented on a Dell Axim PDA which allows for control of the light and mini-blinds as well as providing information from all room sensors to the user).

D.3 Sensing and Actuation

The MavLab has a deployed ArgusD network with two units supporting up to forty sensors, four ArgusM networks supporting fourteen mini-blinds, an ArgusMS network with two Superslaves and fifteen Dongles total, and an X-10 system covering ten zones. The X-10 deployment is shown in Figure D.7 along with the ArgusM actuators for the mini-blinds. The ArgusMS and ArgusD sensors are shown in Figure D.8. A complete list of deployed sensors and actuators for the MavLab is presented in Table D.1 and the Argus network deployment costs are shown in Table D.2.



Figure D.3. MavLab large office.



Figure D.4. MavLab living room (a.k.a., MavDen).



Figure D.5. MavLab kitchen (a.k.a., MavKitchen).



Figure D.6. MavLab conference room area.

Table D.1: MavLab Sensors and Actuators.

Zone	ID	Type	Source	Room	Location
A	11	Light	X-10	Room A	Northeast corner
A	12	Light	X-10	Room A	Northwest corner
A	13	Light	X-10	Room A	Southeast corner
A	14	Light	X-10	Room A	Southwest corner
A	15	Light	X-10	Room A	Center East
A	16	Light	X-10	Room A	Center West
B	9	Light	X-10	Hallway	North rope light between rooms A & B
B	11	Light	X-10	Room B	Northeast corner
B	12	Light	X-10	Room B	Northwest corner
B	13	Light	X-10	Room B	Southeast corner
B	14	Light	X-10	Room B	Southwest corner
B	15	Light	X-10	Room B	Center East
B	16	Light	X-10	Room B	Center West
C	8	Light	X-10	Hallway	North rope light between rooms C & G
C	9	Light	X-10	Hallway	North rope light between rooms B & C
C	11	Light	X-10	Room C	Northeast corner

Table D.1 – continued

Zone	ID	Type	Source	Room	Location
C	12	Light	X-10	Room C	Northwest corner
C	13	Light	X-10	Room C	Southeast corner
C	14	Light	X-10	Room C	Southwest corner
C	15	Light	X-10	Room C	Center East
C	16	Light	X-10	Room C	Center West
D	9	Light	X-10	Hallway	South rope light between rooms D & E
D	11	Light	X-10	Room D	Northeast corner
D	12	Light	X-10	Room D	Northwest corner
D	13	Light	X-10	Room D	Southeast corner
D	14	Light	X-10	Room D	Southwest corner
D	15	Light	X-10	Room D	Center East
D	16	Light	X-10	Room D	Center West
E	9	Light	X-10	Room E	South rope light between rooms E & F
E	11	Light	X-10	Room E	Northeast corner
E	12	Light	X-10	Room E	Northwest corner
E	13	Light	X-10	Room E	Southeast corner

Table D.1 – continued

Zone	ID	Type	Source	Room	Location
E	14	Light	X-10	Room E	Southwest corner
E	15	Light	X-10	Room E	Center East
E	16	Light	X-10	Room E	Center West
F	11	Light	X-10	Room F	Northeast corner
F	12	Light	X-10	Room F	Northwest corner
F	13	Light	X-10	Room F	Southeast corner
F	14	Light	X-10	Room F	Southwest corner
F	16	Light	X-10	Room F	Center West
G	11	Light	X-10	Room G	Northeast corner on desk tower
G	12	Light	X-10	Room G	Center on desk
G	13	Light	X-10	Room G	South center on printer stand
G	14	Light	X-10	Room G	Northwest corner floor lamp
H	6	Appliance	X-10	Room H	Receiver in cabinet
H	7	Appliance	X-10	Room H	DVD player in cabinet
H	9	Light	X-10	Room H	North of couch floor lamp
H	11	Light	X-10	Room H	Rope light by book shelf

Table D.1 – continued

Zone	ID	Type	Source	Room	Location
H	12	Light	X-10	Room H	Western pink table lamp
H	13	Light	X-10	Room H	Southeast pink table lamp
H	14	Light	X-10	Room H	Television on stand
I	11	Light	X-10	Room I	Hanging from ceiling in kitchen
I	14	Appliance	X-10	Room I	Radio on counter
I	15	Appliance	X-10	Room I	Microwave on counter
I	16	Appliance	X-10	Room I	Coffee maker on counter
J	10	Appliance	X-10	Room J	Projector screen circuit
J	11	Appliance	X-10	Room J	Projector screen up
J	12	Appliance	X-10	Room J	Projector screen down
J	15	Light	X-10	Room J	On conference table
J	16	Light	X-10	Room J	On conference table
S	1	Light	ArgusMS	Room F	Room F. Dongle L9 Slot 1
S	2	Heat	ArgusMS	Room F	Room F. Dongle L9 Slot 2
S	4	Light	ArgusMS	Room F	Room F. Dongle L9 Slot 4
S	5	Light	ArgusMS	Room D	Room D. Dongle L5 Slot 1

Table D.1 – continued

Zone ID	Type	Source	Room	Location
S 6	Heat	ArgusMS	Room D	Room D. Dongle L5 Slot 2
S 7	Light	ArgusMS	Room D	Room D. Dongle L5 Slot 3
S 9	Light	ArgusMS	Room E	Room E. Dongle L7 Slot 1
S 10	Heat	ArgusMS	Room E	Room E. Dongle L7 Slot 2
S 11	Light	ArgusMS	Room E	Room E. Dongle L7 Slot 3
S 13	Light	ArgusMS	Conference	Conference room. Dongle L15 Slot 1
S 14	Heat	ArgusMS	Conference	Conference room. Dongle L15 Slot 2
S 15	Light	ArgusMS	Conference	Conference room. Dongle L15 Slot 3
S 17	Light	ArgusMS	Kitchen	Over sink. Dongle L14. Slot 1
S 18	Heat	ArgusMS	Kitchen	Over sink. Dongle L14. Slot 2
S 19	Light	ArgusMS	Kitchen	Over sink. Dongle L14. Slot 3
S 20	Humidity	ArgusMS	Kitchen	Over sink. Dongle L14. Slot 4
S 21	Light	ArgusMS	Living Room	Over large couch. Dongle L12 Slot 1
S 22	Heat	ArgusMS	Living Room	Over large couch. Dongle L12 Slot 2
S 23	Humidity	ArgusMS	Living Room	Over large couch. Dongle L12 Slot 3
S 24	Light	ArgusMS	Living Room	Over large couch. Dongle L12 Slot 4

Table D.1 – continued

Zone	ID	Type	Source	Room	Location
S	25	Light	ArgusMS	Kitchen	Kitchen area. Dongle L13 Slot 1
S	26	Heat	ArgusMS	Kitchen	Kitchen area. Dongle L13 Slot 2
S	27	Light	ArgusMS	Kitchen	Kitchen area. Dongle L13 Slot 3
S	65	Light	ArgusMS	Room B	On window. Dongle L2 Slot 1. Facing outside
S	66	Light	ArgusMS	Room B	On window. Dongle L2 Slot 2. Facing outside blinds
S	67	Light	ArgusMS	Room B	On window. Dongle L2 Slot 3. Facing inside blinds
S	68	Light	ArgusMS	Room B	On window. Dongle L2 Slot 4. Facing inside
S	69	Light	ArgusMS	Room A	On sill. Dongle L6 Slot 1
S	70	Heat	ArgusMS	Room A	On sill. Dongle L6 Slot 2
S	71	Humidity	ArgusMS	Room A	On sill. Dongle L6 Slot 3
S	72	Light	ArgusMS	Room A	On sill. Dongle L6 Slot 4
S	73	Light	ArgusMS	Room B	On sill. Dongle L8 Slot 1
S	74	Heat	ArgusMS	Room B	On sill. Dongle L8 Slot 2
S	75	Light	ArgusMS	Room B	On sill. Dongle L8 Slot 3
S	77	Light	ArgusMS	Room A	On window. Dongle L1 Slot 1. Facing outside
S	78	Light	ArgusMS	Room A	On window. Dongle L1 Slot 2. Facing outside blinds

Table D.1 – continued

Zone	ID	Type	Source	Room	Location
S	79	Light	ArgusMS	Room A	On window. Dongle L1 Slot 3. Facing inside blinds
S	80	Light	ArgusMS	Room A	On window. Dongle L1 Slot 4. Facing inside
S	81	Light	ArgusMS	Room C	On sill. Dongle L10 Slot 1
S	82	Heat	ArgusMS	Room C	On sill. Dongle L10 Slot 2
S	83	Light	ArgusMS	Room C	On sill. Dongle L10 Slot 3
S	85	Light	ArgusMS	Room C	On window. Dongle L3 Slot 1. Facing outside
S	86	Light	ArgusMS	Room C	On window. Dongle L3 Slot 2. Facing outside blinds
S	87	Light	ArgusMS	Room C	On window. Dongle L3 Slot 3. Facing inside blinds
S	88	Light	ArgusMS	Room C	On window. Dongle L3 Slot 4. Facing inside
S	89	Light	ArgusMS	Room G	On window. Dongle L4 Slot 1. Facing outside
S	90	Light	ArgusMS	Room G	On window. Dongle L4 Slot 2. Facing outside blinds
S	91	Light	ArgusMS	Room G	On window. Dongle L4 Slot 3. Facing inside blinds
S	92	Light	ArgusMS	Room G	On window. Dongle L4 Slot 4. Facing inside
S	93	Light	ArgusMS	Room G	On east wall. Dongle L11 Slot 1
S	94	Heat	ArgusMS	Room G	On east wall. Dongle L11 Slot 2
S	95	Light	ArgusMS	Room G	On east wall. Dongle L11 Slot 3

Table D.1 – continued

Zone	ID	Type	Source	Room	Location
S	96	Door Switch	ArgusMS	Room G	On east wall. Dongle L11 Slot 4
V	1	Motion	ArgusD	Living Room	Over door
V	2	Motion	ArgusD	Living Room	Over large couch
V	3	Motion	ArgusD	Living Room	Over armchair
V	4	Motion	ArgusD	Living Room	Over kitchen area entrance
V	5	Motion	ArgusD	Living Room	Center on ceiling, in front of TV
V	6	Motion	ArgusD	Living Room	Over bookcase
V	7	Motion	ArgusD	Living Room	Over small couch
V	8	Motion	ArgusD	Living Room	Over office entrance
V	9	Pressure	ArgusD	Living Room	In small couch, southern end
V	10	Pressure	ArgusD	Living Room	In small couch, middle
V	11	Pressure	ArgusD	Living Room	In small couch, northern end
V	12	Pressure	ArgusD	Living Room	In large couch, southern end
V	13	Pressure	ArgusD	Living Room	In large couch, 2nd from northern end
V	14	Pressure	ArgusD	Living Room	In large couch, northern end
V	15	Pressure	ArgusD	Living Room	In large couch, 3rd from northern end

Table D.1 – continued

Zone	ID	Type	Source	Room	Location
V	16	Pressure	ArgusD	Living Room	In armchair
V	17	Switch	ArgusD	Living Room	Reed switch on door
V	18	Motion	ArgusD	Kitchen	Kitchen hallway, over microwave
V	19	Motion	ArgusD	Conference	East end of conference table
V	20	Motion	ArgusD	Kitchen	Kitchen hallway, over sink
V	21	Motion	ArgusD	Office	Michael's office. Center on ceiling
V	22	Motion	ArgusD	Hallway	Hallway ceiling at FC junction
V	23	Motion	ArgusD	Room F	Center on ceiling
V	24	Motion	ArgusD	Room C	Center on ceiling
V	25	Motion	ArgusD	Hallway	Hallway ceiling west of FC junction
V	26	Motion	ArgusD	Room E	Center on ceiling
V	27	Motion	ArgusD	Room B	Center on ceiling
V	28	Motion	ArgusD	Hallway	Hallway ceiling west of EB junction
V	29	Motion	ArgusD	Room D	Center on ceiling
V	30	Motion	ArgusD	Room A	Center on ceiling
V	31	Motion	ArgusD	Hallway	Hallway ceiling at DA junction (near rack)

Table D.1 – continued

Zone ID	Type	Source	Room	Location
V 36	Motion	ArgusD	Kitchen	Over kitchen entrance
V 37	Motion	ArgusD	Kitchen	Over refrigerator
V 38	Motion	ArgusD	Kitchen	Over shelves

Table D.2. MavLab Argus Sensor Components

Component	Quantity	Unit Price	Total Price
Master	1	\$57.04	\$57.04
Mini-blind Master	4	\$69.76	\$279.04
Motion Master	2	\$78.20	\$156.40
AD Kitchen Master	1	\$82.64	\$82.64
Super Slave	2	\$92.28	\$184.56
Dongles	15	\$8.87	\$133.05
Light Sensor	37	\$1.56	\$57.72
Temperature Sensor	11	\$1.26	\$13.86
Reed Switch	2	\$ 2.47	\$4.94
Humidity Sensor	3	\$20.00	\$60.00
Motion Sensor	25	\$12.00	\$300.00
Grand Total			\$1,329.25

APPENDIX E

ResiSim: A Zero Configuration Distributed Simulation

E.1 Preface

This appendix is an introduction to the simulator we use for testing, evaluation, and experimentation for the work in this dissertation. We call this simulator ResiSim as it was originally intended to provide a residential home simulation. Several facets of this simulator and its design are novel in the world of modeling and simulation. We believe this new paradigm will prove useful in intelligent environment work and help bridge the gap between reality and virtual reality.

E.2 Introduction

Simulation is an important tool for performing evaluation and research in many areas. However, the current simulation paradigms do not lend themselves well to all environments. In our primary research, we are involved in exploring artificial intelligence techniques in intelligent environments—smart homes and workplaces. We seek to simulate these environments as a discrete event simulation with dozens of objects. Each object should contain public and private information, operations, and interfaces. They need to be self contained, accurate simulations of the items they represent and even have the ability to be replaced by actual objects—bridging the gap between reality and virtual reality. As in most real-world environments, new objects can be introduced at any time and existing objects removed. An ideal simulator for this type of work would allow for easy object addition and removal, preferably with little or no changes in configuration.

Imagine a scenario occurring in our lab (i.e., the UTA AI Lab shown in Figure E.1) where we add a new desk lamp and salvage some of the computers. Some manufacturer has already produced the new lamp, is intimately familiar with its form and function, and probably already has existing 2D and 3D models. It would be very beneficial to have this information instead of having to create it ourselves. Couple this with the increasing growth in wireless control and



Figure E.1. UTA AI Lab (MavLab) composed of zeroconf objects.

movement towards making smarter objects (e.g., even static furniture with an RFID tag can store and provide information about itself) along with wireless localization technologies [68] and the future contains objects that 1) can provide information about themselves (e.g., a link to their information on the Internet), 2) can have an externally (possibly wireless) controlled interface, 3) can provide state information, and 4) can be localized in their environment. If environmental systems can find these objects, then logical proxies can be created in cyberspace to represent these real-world items and even simulate them if they are not physically present.

Current distributed simulation systems are usually built as specific representations of the environments they simulate. Changes in the environmental objects are usually planned and well-defined as in HLA [85]. Simulation systems usually do not handle entry of new entities of which they have no configuration information and require that entities maintain their own real-world information with additional processing and tracking requirements [84]. Commercial simulation systems usually allow the users to develop environments and environmental conditions with great flexibility, but require configuration and reloading in order to accommodate change.

We propose a new type of simulation system that accommodates a high level of dynamic flexibility, requires no configuration, is object-oriented, easy to develop, and scales while maintaining reasonable performance. It is based on the IETF Zero Configuration Working Group's Zeroconf specification [87] and the notion of small components that encapsulate a set of focused operations and information pertaining to a single object or functionality. This paper will present the main ideas of this simulation system, the architecture, a discussion of our implementation, and performance evaluation.

E.3 Distributed Simulation

Distributed simulation originated in the late 1970's in the high performance computing community. Their work sought to develop synchronization algorithms that would produce quicker results across many machines compared to a single machine. In the early 1980's the defense community started to use distributed simulation to create virtual real-worlds that integrated flight, ground vehicle, and other simulators with computer generated forces into cohesive battlefield training exercises. They started with the homogenous SIMNET, advanced to the heterogeneous DIS (Distributed Interactive Simulation)[84], and have now arrived at the HLA (High-Level Architecture) [85] standard. During the same time frame, the computer gaming community has developed from Multi-User Dungeon (MUD) games to the massively multi-player games of today (e.g., Everquest [190]) which distribute the simulation to thousands of participants. [58] However, all of these examples contain *a priori* well-defined interfaces, configuration requirements, and heavy-weight entities (i.e., each entity is an entire simulation in itself containing information about the entire local real-world).

E.4 Zero Configuration Networking

In 1999, the IETF Zeroconf Working Group was chartered in order to develop standards for zero configuration networking not only for computers, but for all types of networked products. Zeroconf is currently comprised of three main technologies: IPv4 link-local addressing, multicast DNS [134], and DNS service discovery [46]. Zeroconf objects must be assigned an IP number without being pre-configured with it and without a DHCP server. Link-local addressing is a defined method for networked entities to self-assign addresses and check for conflicts. Multicast DNS allows for the translation between names and IP addresses without a centralized DNS. This is accomplished by querying name resolution and browsing over multicast IP, where all machines listen and, if applicable, respond to the query based upon their own view of the network and name resolution memory. In order to find services without a directory server, DNS service discovery provides descriptive information stored in SRV and TXT records that are available for query through multicast DNS [87].

When a zeroconf object is started, it first seeks an IP number from a DHCP server, if available, or uses link-local addressing. Link-local addressing works by a random selection of an IP number from the IANA [83] specified range of link-local addresses (currently 169.254.x.x). It then broadcasts out a message to see if any other machine has assigned itself this number as well. If so, then it randomly chooses another number and rechecks for conflicts. This process continues until the device finds a nonconflicting address. Once the device becomes a member of the IP network, it then needs to advertise itself and find other services. The first step in sharing is to create a unique name for the object. Zeroconf naming uses the convention *uniquename.typeofservice.networkprotocol.local*, where the *unique name* is a local network unique name for the object instance (e.g., KitchenMicrowave), *protocol* specifies the protocol name of the type of service through which the object communicates (e.g.,

*xyzmicrowave*¹), *networkprotocol* is the actual network protocol used in communication with the object (currently only *tcp* and *udp* are supported), and *local* is the network domain, currently set to the link-local domain. This unique name is then registered with all other devices through a Multicast DNS-Service Discovery daemon which broadcasts the information out to the local network. Other objects on the network note the name and associated IP for future reference. A text (TXT) record can also be transmitted and associated with the name-address pairing to provide additional information (e.g., model URL, contact port, IDL name for CORBA IDL lookup, and so forth). The object is now discoverable and all of the information needed to contact it is available. Discovery is initiated by a network query for a type of object such as *xyzmicrowave.tcp.local*, in which a response of *KitchenMicrowave.xyzmicrowave.tcp.local* should be received. The IP and TXT record can be retrieved with the name as well. With this information an entity now has the ability to contact the *KitchenMicrowave* on the *local* network using the *TCP* network protocol and following the *xyzmicrowave* protocol to perform actions consistent with the defined interface [87].

The competing standard to zeroconf is Universal Plug-n-Play (UPnP) [218] which is similar to zeroconf with a few notable exceptions. UPnP is a consortium of 728 members, mostly corporations, who define the “standard” and the protocols of the devices using UPnP. There are specifications for printers, scanners, and many other devices, but all have to be approved by the consortium before being available. UPnP uses link-local addressing and multicast DNS, but also adds the usage of SOAP. In general, it is a similar mechanism, but has more rules and restrictions and is governed by a closed business consortium.

¹Usually well-known IANA registered protocols are used, but any protocol understood by using objects can be specified. The idea is to promote the usage of existing protocols and standards and not invent new ones.

E.5 ZCoDS

Simulation at its core consists of an environment and objects that exist in that environment. Mathematical models of behavior govern the interactions and results of the constituent parts. In distributed simulation the objects do not exist on the same machine, but rather on the same network. Computation, control, state, and interactions are spread across the participating machines on the network. In some simulation environments any process that starts participating is included in the simulation and in others they must be specifically configured to participate. The former extreme requires absolute control over the machines and their processes in order to prevent chaos, and the latter requires the labor of complete configuration. Both require a level of control to ensure proper simulation. The well-defined and configured simulation is the obviously preferred method, especially in research, but the burden of configuration can be quite expensive in time and labor. By integrating zeroconf technology into the distributed simulation we can gain the power and stability of the well-defined simulation while avoiding a high penalty for configuration.

A ZCoDS (**Z**ero **C**onfiguration **D**istributed **S**imulation) environment is comprised of one or more objects. Objects can be of any size and functionality, but are categorized into one of five types: static, dynamic, simulation, observation, and interface. Static objects simply exist and do not move. Dynamic objects can move in the environment or change state. Simulation objects provide mathematical support to coordinate interactions between other objects. Observation objects provide a view into the simulation. Interface objects allow either users or other external entities to interact with the simulation. Every object contains attributes and a state, it may also provide functionality in the form of operators. To run a simulation all that is necessary is to start all of the objects. The static objects make up the environment while the dynamic objects are the key actors. The simulation objects determine the effects of the dynamic objects on each other causing state changes as necessary. The simulation can be viewed through an observation object that may appropriately display the state of the simulation. User interaction or

external information may be channeled through an interface object. Updating is usually event driven, but other types of simulations could be established depending on the core programming of the objects and their defined attributes and operations.



Figure E.2. Simulation object types in ResiSim.

E.6 Case Study: ResiSim

We have developed a simulator using the principles described in the last section. Figure E.2 shows a view provided by an observation object in a ZCoDS environment established by the ResiSim simulator. ResiSim consists of three main parts: logical proxy objects, core simulation objects, and user interface objects. It was designed to provide an accurate simulation of a residential environment, but has been used for workplace environments as well.

The logical proxies are zeroconf enabled object processes that are distributed among the available computing assets on the network. They represent an actual object in the real-world (e.g., a lamp or a chair). Logical proxies in our work are usually static objects, but could be

dynamic if they move around or change state in the environment. Our logical proxies usually take one of two forms—real-world or simulated. Real-world logical proxies are used to bridge the gap between reality and virtual reality by projecting a logical interface to a real-world object into the simulation environment so that the simulator can actually control the real-world object. For example, we can allow some lights in the real-world to be turned on and off through simulation actions (e.g., a virtual inhabitant can turn on real-world lights). Simulated logical proxies are pure virtual reality objects that simulate the behavior of the real-world objects they represent.

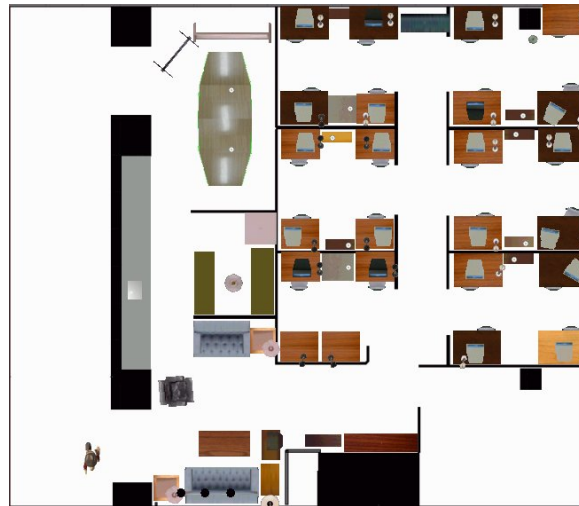


Figure E.3. ResiSim::Server 2D UTA AI Lab observation object view.

The core simulation objects include zeroconf enabled virtual inhabitant dynamic objects, simulation objects, interface objects, and an observation object. Figure E.3 shows the 2D observation object view of the UTA AI Lab simulation, and Figure E.4 shows an interface panel which allows selection of the discovered objects included in the simulation. In our implementation we aggregate the core simulation objects with an interface and observation object into the ResiSim Server component. We call this a server because from the simulation core

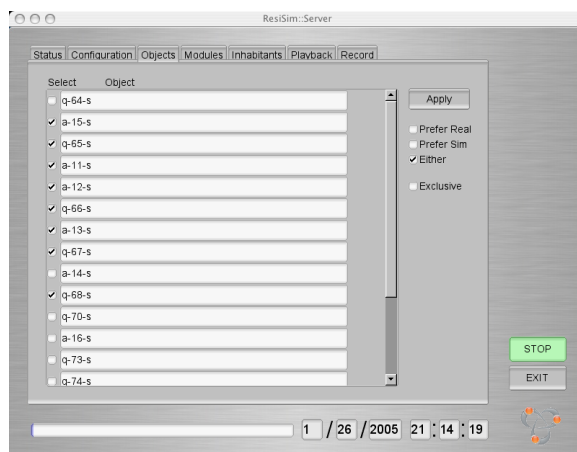


Figure E.4. ResiSim::Server interface object panel.

we can serve to clients outside our link-local addresses through IP connections. If the clients are within the link-local network, then they participate as zeroconf objects; otherwise, they must be configured to locate the server on the Internet. The server provides a collecting point for all of the necessary components for a complete simulation; however, all of the constituent components are independent zeroconf objects. This allows for easy dynamic changes in the simulation where any object can be added or removed to the simulation. Removed objects notify through zeroconf their removal allowing interacting objects to accommodate the change. Added objects notify their availability and service through zeroconf allowing other objects to seek interactions with them. These components comprise a complete ZCoDS environment.

An example of a useful user interface object is the ResiSim Client which is a zeroconf-enabled (but not required) process that combines an observation object and an interface object in order to provide an interactive 3D environment to the user as seen in Figure E.1. This component was created to accommodate connections outside of the link-local address by communicating with the ResiSim Server through the Internet, which requires minimal configuration in order to locate the server. When run on the link-local network, the client can find the server

through zeroconf without any configuration. The server is also better informed of its connection to clients since it registers any client removals.

Using zeroconf technology to create a simulation environment has many strengths and few weaknesses. The obvious strengths are the lack of configuration needed to create a distributed simulation since objects can discover each other and utilize information and operations of each other as designed and the dynamic capability of adding and removing simulation objects during runtime with proper notifications and discovery occurring as necessary. The inclusion of standard, well-defined, and Internet approved protocols, and not necessarily waiting for ones designed by committee, promotes the use of IANA and development and sharing of protocols that helps advance the computing community. An added strength is the cross-platform availability of the technology since Apple Computer, Inc. has made their cross-platform zeroconf implementation, called Bonjour, freely available over the Internet [12]. The weaknesses of using zeroconf are those shared by many distributed simulations. Network utilization is increased by all of the communication required for address allocation, multicast DNS communication, service discovery, and network query. Distributed objects must also communicate through the network, increasing bandwidth utilization. Single component failures can damage the simulation, though in zeroconf it is easy to recover since a replacement object can be dynamically reinserted. Network speed and bandwidth are improving daily and zeroconf makes it easy to establish backup objects that can instantly replace failed components.

In ResiSim the location of objects is maintained by the logical proxy that represents the object. Movement of objects is reflected in state changes which are broadcast to the ResiSim server. These changes may be initiated by software for simulated objects, or by localization methods in the real-world reflected by state changes in position in the logical proxies.

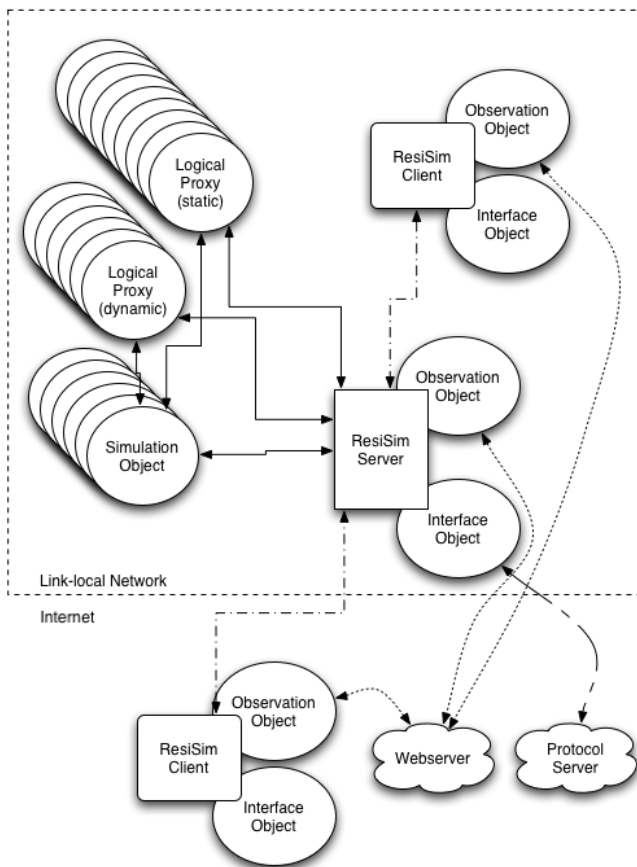


Figure E.5. ResiSim architecture.

E.6.1 Implementation

As introduced in the last section, we have implemented a ZCoDS type simulator we call ResiSim (Residential Simulation). We use this simulator to conduct artificial intelligence research in intelligent environments using two environments: the MavLab which is the UTA AI Lab and the MavPad which is an on-campus apartment. “Mav” is an prefix acronym for our MavEnvironments which stands for **M**anaging an **a**daptive **v**ersatile **E**nvironment. The basic architecture of our simulation is illustrated in Figure E.5. We utilize both real-world and simulated logical proxy objects in homogeneous and heterogeneous modes sometimes preferring only simulated, only real-world, or whatever is available possibly with a preference



Figure E.6. MavPad apartment composed of zeroconf objects.

either towards real-world or simulated. The power of zeroconf means that both objects can exist in the link-local network, can be discovered, and the preference chosen for inclusion in the desired simulation experiment. Another advantage is that our real-world and simulated logical proxies both have the same attributes and operations and due to careful modeling they both behave in the same manner—thus developing systems to work with objects in virtual reality directly translates into working with objects in reality with no changes. Real-world objects are zeroconf enabled as well, so they are found and controlled by our systems automatically. The key difference between running in reality and virtual reality is the presence of the core simulation objects aggregated by the ResiSim Server.

ResiSim Server starts the simulation environment through a number of steps. First all logical proxies are started then located and cataloged by name and type (static and dynamic objects), then all simulation objects are started then located and cataloged by name and type, the simulation objects will find the static and dynamic objects with which they interact, then an interface object is started in which the user can manipulate the simulation, and finally an observation object is displayed showing a 2D representation of the simulation environment. Virtual inhabitant dynamic objects can be started at any time as well as any other interface and

observation objects (e.g., ResiSim clients). Startup takes a slightly longer period of time than a typical predefined simulation because of the discovery process and dynamic configuration. Regular runtime performance is consistent with pre-configured distributed simulations. Once the simulation objects are started by a interface object command, the simulation begins in an event driven manner. Zeroconf objects are free to come and go as necessary to support the desired simulation.

When observation objects start, they find the environment static object and all of the static and dynamic objects in the environment. Depending on the display type, either 2D or 3D graphical models are obtained for each object by referring to a URL embedded in the TXT record of each zeroconf object. The observation object checks to see if the object model is already cached and if not follows the URL to download the model from the Internet (we use *wget* and serve models from a web server). The observation object can also verify if the local cached version is the latest and download an update if necessary. Decoupling the object from its model in this manner allows for easy updating of the object model which may be maintained by an outside organization.

An idea that we would like to promote is that any company producing a product should generate both 2D and 3D models of the product, a complete attribute and interface specification, and an example logical proxy. Organizations should make these available for free download over the Internet. This promotes reuse and widely accepted models common to simulations that use these types of objects.

E.6.2 Environments

We have complete 2D and 3D models for both of our research environments. The MavLab shown in Figure E.1 contains a total of 139 models which include static objects such as the room environment and furniture, and dynamic objects such as motion sensors, temperature sensors, lights, and computers. The 2D pixel size of the MavLab environment is 651x568.

The MavPad shown in Figure E.6 contains a total of 111 models which includes similar objects at those in the MavLab, but more typical of a residence than an office. The 2D pixel size of the MavPad environment is 840x394. All models were created using Blender [173] and built to scale. The base unit for our simulations is 1 decimeter. Scaling in the MavLab is 4.2 pixels/dm and in the MavPad 8.45 pixels/dm.

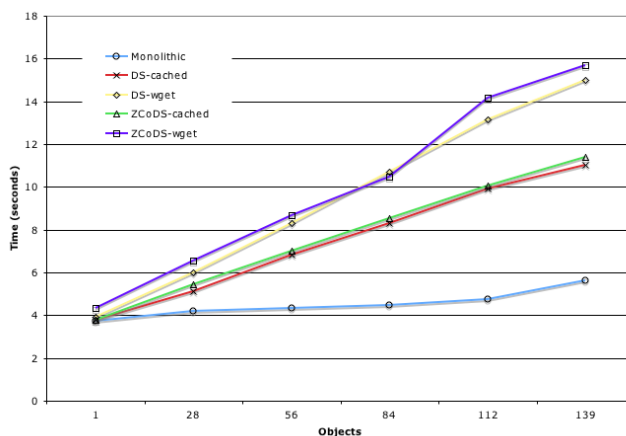


Figure E.7. MavLab visualization performance.

E.6.3 Performance

We evaluated performance of ResiSim using both the MavLab and MavPad simulations and compared them against a monolithic simulation and a distributed simulation. The tests were conducted to measure the time to display the entire simulation in a 3D observation object as seen in Figures E.1 and E.6 from initial start of the simulation logical proxies until the first complete view of all objects. This represents a complete path to visualization and the beginning of interaction. The monolithic simulation was a modification of a ResiSim Client that loaded all of the objects locally on the same machine similar to a video game. The distributed simulation (DS) used a modified ResiSim Client that was pre-configured with the logical proxy

information instead of having to discover them. ZCoDS uses zeroconf to find all of the objects in the simulation with no pre-configuration. The ZCoDS and DS simulations were run with models cached and without models cached facilitating the need to retrieve the models from a local webserver using the *wget* command. Distributed tests were run on four Intel-based Pentium 4 machines running at 2 GHz, two with 512MB of memory and two with 1 GB of memory, and accelerated NVidia-based graphics cards on all. All are connected through a gigabit ethernet network. The webserver is also located on the local network.

MavLab visualization performance is presented in Figure E.7. Performance between the DS and ZCoDS simulations scales similarly with ZCoDS taking from -0.3 to 6.8% longer with an average of 2.9% more time required than the pre-configured DS for the cached versions and from -2.3 to 10.8% longer with an average of 5.7% more time required for the simulations that retrieved the models through the web. Compared to a monolithic simulation, the ZCoDS was consistent in performance for distributed simulations taking from 1.4 to 111.8% longer with an average of 66.3% for the cached version and from 15.8 to 198.6% longer with an average of 113.8% for the *wget* version. It is difficult to outperform the monolithic simulations that do not incur network latency and overhead until the processing requirements exceed the capacity of the single machine hosting the simulation. Overall, the difference between ZCoDS for the tests we conducted only differed by milliseconds—a negligible difference for the benefits gained by using zeroconf technology.

MavPad visualization performance is presented in Figure E.8. It tracks very similar to the performance in the MavLab and shows a consistency in the simulations under different conditions. Performance between the DS and ZCoDS simulations scales similarly with ZCoDS taking from -0.4 to 3.2% longer with an average of 1.2% more time required than the pre-configured DS for the cached versions and from -3.8 to 3.7% longer with an average of 0.6% more time required for the simulations that retrieved the models through the web. Compared to a monolithic simulation, the ZCoDS was consistent in performance for distributed simulations

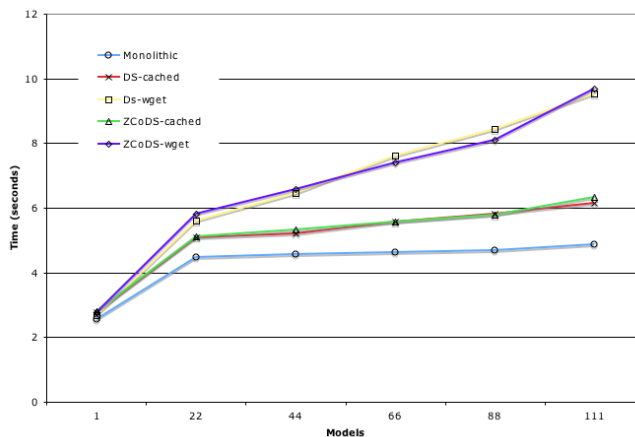


Figure E.8. MavPad visualization performance.

taking from 8.7 to 30.7% longer with an average of 19.2% for the cached version and from 7.9 to 99.3% longer with an average of 52.4% for the *wget* version. The quick ramp up in time required is due to the loading of the furniture objects first in the simulation which tend to be larger and more complex models than the others. These results indicate much closer performance between ZCoDS and DS and not that distant of a performance from monolithic.

E.7 Summary

Using zero configuration technology in distributed simulation adds flexibility through the ability to develop simulations without time and labor intensive configuration, to support dynamic object addition and deletion, and to distribute computation among a network of local computers. Zero Configuration Distributed Simulation (ZCoDS) promotes model reuse, the object-oriented paradigm, standardization, and community development to improve simulation systems. In empirical studies of performance, the use of zeroconf is similar in runtime to pre-configured distributed simulation and performs within acceptable time in comparison with monolithic single-computer simulation.

E.8 Acknowledgements

This work was done with the assistance of Chris Lance who is responsible for developing the 3D visual simulator as well as performing all of the general tedious work once the designs and basic implementations were complete. His work has been greatly appreciated. The ideas, approach, 2D visualization, and server side were all designed and developed by Michael Youngblood.

APPENDIX F
Licenses and Terms

F.1 Intellectual Property Rights

The Argus Sensor Network designs (including all ideas, algorithms, artwork, and schematics), Hermes driver software, the ProPHeT algorithms, the ARBITER, and the entire architecture and designs contained in this dissertation are released under the GNU General Public License, Version 2 as outlined in the next section. We retain the copyright, but release the use to the scientific community in order to further discovery in this field. This license applies to all algorithms, software, and hardware work contained in this dissertation—please substitute “program,” “software,” or “source code” with the appropriate copyrighted object.

We have chosen not to patent these technologies in order to further scientific and engineering discovery in this area of interest. Please do not violate these rights as outlined in the next section.

F.2 The GNU General Public License

Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.

59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

F.2.1 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation’s software and to any other program whose authors commit to using it. (Some other Free Software Foundation

software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION
AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The “Program”, below, refers to any such program or work, and a “work based on the Program” means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term “modification”.) Each licensee is addressed as “you”.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program’s source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
 - (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
 - (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
 - (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
 - (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
 - (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
 - (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.
6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if

a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and “any later version”, you have the

option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INAC-

CURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

REFERENCES

- [1] G. Abowd, J. Engelsma, L. Guadagno, and O. Okon. Architectural Analysis of Object Request Brokers. *Object Magazine*, March, 1996.
- [2] Gregory D. Abowd, Agathe Battestini, and Thomas O'Connell. The Location Service: A Framework for Handling Multiple Location Sensing Technologies, 2002. Website: www.cc.gatech.edu/fce/ahri/publications/location_service.pdf.
- [3] Accenture. Room of the Future, 2005. Website: www.accenture.com/xd/xd.asp?it=enweb&xd=services/technology/research/i%hs/room_future.xml.
- [4] R. Agrawal and R. Srikant. Mining Sequential Patterns. In *Proceedings of the 11th International Conference on Data Engineering*, pages 3–14, 1995.
- [5] AHRI. [AHRI] - Aware Home Research Initiative, Oct 2003. Website: www.cc.gatech.edu/fce/ahri.
- [6] AIRE Group. MIT Project AIRE, March 2005. Website: aire.csail.mit.edu/projects.shtml.
- [7] J. S. Albus. A Theory of Cerebellar Function. *Mathematical Biosciences*, 10:25–61, 1971.
- [8] Open Mobile Alliance. Open Mobile Alliance, Oct 2003. Website: www.openmobilealliance.org.
- [9] Ambient Intelligence research and Development Consortium. Ambient Intelligence for the Networked Home Environment, 2005. Website: www.amigo-project.org.
- [10] AMBIENTE. AMBIENTE activity: i-LAND, Oct 2003. Website: www.darmstadt.gmd.de/ambiente/i-land.html.

- [11] D. Andre. Learning hierarchical behaviors. *NIPS Hierarchical Reinforcement Learning Workshop*, 1998.
- [12] Apple Computer, Inc., 2005. Website: developer.apple.com/darwin/projects/bonjour.
- [13] Sachiyo Arai and Katia Sycara. Credit assignment method for learning effective stochastic policies in uncertain domain. In Lee Spector, Erik D. Goodman, Annie Wu, W. B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 815–822, San Francisco, California, USA, 7-11 2001. Morgan Kaufmann.
- [14] Andrew G. Barto and Sridhar Mahadevan. Recent Advances in Hierarchical Reinforcement Learning. *Discrete Event Systems*, 2003.
- [15] Edward Batschelet. *Circular Statistics in Biology*. Academic Press, 1981.
- [16] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, New Jersey, 1957.
- [17] Alastair Beresford. CUED: Laboratory for Communication Engineering, Oct 2003. Website: www-lce.eng.cam.ac.uk/research/\?view=2\&id=7.
- [18] D. Bernstein and S. Zilberstein. Reinforcement learning for weakly-coupled MDPs and an application to planetary rover control. In *Mt*, 2001. European Conference on Planning, 2001.
- [19] Philip A. Bernstein. Middleware: a model for distributed system services. *Communications of the ACM*, 39(2):86–98, 1996.
- [20] Amit Bhattacharya and Sajal K. Das. LeZi-Update: An Information-theoretic framework for personal mobility tracking in PCS networks. *ACM/Kluwer Wireless Networks Journal*, 8:121–135, 2002.
- [21] John Bloomer. *Power Programming with RPC*. O’Reilly, Sebastopol, CA, 1992. Website: www.oreilly.com/catalog/rpc.

- [22] Fred Brooks. *The Mythical Man-Month*. Addison-Wesley, 1975.
- [23] Barry Brumitt, Brian Meyers, John Krumm, Amanda Kern, and Steven A. Shafer. Easyliving: Technologies for intelligent environments. In *HUC*, pages 12–29, 2000.
- [24] BT. Telecare Overview. *BT Exact*, 2005. Website: www.btexact.com/research/researchprojects/currentresearch?doc=42834.
- [25] AT&T Laboratories Cambridge. Sentient Computing Project Home Page, 2001. Website: www.uk.research.att.com/spirit.
- [26] Anthony R. Cassandra, Leslie Pack Kaelbling, and James A. Kurien. Acting under uncertainty: Discrete bayesian models for mobile robot navigation. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1996.
- [27] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, November 2000.
- [28] Stuart Cheshire. Bonjour. Technical report, Apple Computer, Inc., 2003.
- [29] Stuart Cheshire. Zero Configuration Networking (Zeroconf), Jan 2004. Website: www.zeroconf.org.
- [30] Jen-Tzung Chien. Chien lab. Website: chien.csie.ncku.edu.tw.
- [31] Jen-Tzung Chien and Sadaoki Furui. Predictive Hidden Markov Model Selection for Decision Tree State Tying. In *Proceedings of Eurospeech*, pages 2701–2704, 2003.
- [32] Grzegorz Cielniak, Maren Bennewitz, and Wolfram Burgard. Where is...? learning and utilizing motion patterns of persons with mobile robots. In *IJCAI*, 2003.
- [33] CISCO. The Internet Home, 2005. Website: www.cisco.com/warp/public/3/uk/ihome.
- [34] Michael Coen, Brenton Phillips, Nimrod Warshawsky, Luke Weisman, Stephen Peters, and Peter Finin. Meeting the computational needs of intelligent environments: The metagluue system. In *Proceedings of MANSE'99*, Dublin, Ireland, 1999.

- [35] Michael H. Coen. Design principles for intelligent environments. In *AAAI/IAAI*, pages 547–554, 1998. Website: citeseer.nj.nec.com/coen98design.html.
- [36] World Wide Web Consortium. World Wide Web Consortium, Oct 2003. Website: www.w3c.org.
- [37] D. J. Cook, S. Das, Karthik Gopalratnam, and Abhishek Roy. Health Monitoring in an Agent-Based Smart Home. In *Proceedings of the International Conference on Aging, Disability and Independence Advancing Technology and Services to Promote Quality of Life*, 2003.
- [38] D. J. Cook and M. Youngblood. Smart Homes. *Encyclopedia of Human-Computer Interaction*, 2004.
- [39] D. J. Cook, M. Youngblood, E. Heierman, K. Gopalratnam, S. Rao, A. Litvin, and F. Khawaja. MavHome: An Agent-Based Smart Home. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications*, pages 521–524, 2003.
- [40] Joelle Coutaz, James L. Crowley, Simon Dobson, and David Garlan. Context is Key. *Communications of the ACM*, 48:3:49–53, 2005.
- [41] CSIRO. NanoHouse, 2005. Website: www.csiro.au/index.asp?type=blank\&id=Nanotechnology_NanoHouse.
- [42] S. K. Das, D. J. Cook, A. Bhattacharya, E. O. Heierman, III, and T.-Y. Lin. The Role of Prediction Algorithms in the MavHome Smart Home Architecture. *IEEE Wireless Communications Special Issue on Smart Homes*, 9(6):77–84, 2002.
- [43] A. Dey, G. Abowd, and D. Salber. A Context-based Infrastructure for Smart Environments. In *MANSE*, 1999.
- [44] T. Dietterich. State abstraction in maxq hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems 12*, pages 994–1000, 2000.

- [45] Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.
- [46] DNS-SD.org. DNS Service Discovery (dns-sd), 2005. Website: www.dns-sd.org.
- [47] Jean-Yves Donnat and Jean-Arcady Meyer. Learning Reactive and Planning Rules in a Motivationally Autonomous Animat. *IEEE Transactions on Systems, Man and Cybernetics - Part B: Cybernetics*, 26(3):381–395, 1996.
- [48] Brier Dudley. Microsoft Home smarter at age 10. *The Seattle Times*, September 28, 2004. Website: seattletimes.nwsourc.com/html/business/technology/2002048005_microsoft%home28.html.
- [49] DWRC. Digital World Research Centre, 2005. Website: www.surrey.ac.uk/dwrc/Index.htm.
- [50] Herb Edelstein. Unraveling client/server architecture. *DBMS*, 34(7), 1994.
- [51] M. Edge, B. Taylor, G. Dewsbury, and M. Groves. The potential for 'smart home' systems in meeting the care needs of older persons and people with disabilities. *Seniors' Housing Update*, 10:1, 2000. Website: www.rgu.ac.uk/sss/research/page.cfm?page=2546.
- [52] Wolfgang Emmerich. Software engineering and middleware: a roadmap. In *Proceedings of the conference on The future of Software engineering*, pages 117–129. ACM Press, 2000.
- [53] Equator. Equator Website, 2005. Website: www.equator.ac.uk.
- [54] Shai Fine, Yoram Singer, and Naftali Tishby. The Hierarchical Hidden Markov Model: Analysis and Applications. *Machine Learning*, 32(1):41–62, 1998.
- [55] International Organization for Standardization. ISO - international organization for standardization, Oct 2003. Website: www.iso.ch/iso/en/ISOOnline.openpage.
- [56] D. Franklin. Cooperating with people: The Intelligent Classroom. *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 1998.

- [57] Fraunhofer IPSI. Ambient Agoras: Dynamic Information Clouds in a Hybrid World, 2005. Website: www.ambient-agoras.org.
- [58] Richard M. Fujimoto. Parallel simulation: distributed simulation systems. In *WSC '03: Proceedings of the 35th conference on Winter simulation*, pages 124–134. Winter Simulation Conference, 2003.
- [59] Sadaoki Furui. Furui laboratory homepage. Website: www.furui.cs.titech.ac.jp/english.
- [60] Mohammad Ghavamzadeh and Sridhar Mahadevan. Continuous-time hierarchial reinforcement learning. In *Proc. 18th International Conf. on Machine Learning*, pages 186–193. Morgan Kaufmann, San Francisco, CA, 2001.
- [61] K. Gopalratnam and D. J. Cook. Active LeZi: An Incremental Parsing Algorithm for Device Usage Prediction in the Smart Home. In *Proceedings of the Florida Artificial Intelligence Research Symposium*, pages 38–42, May 2003.
- [62] Karthik Gopalratnam and Diane J. Cook. Online sequential prediction via incremental parsing: The active lezi algorithm. *IEEE Intelligent Systems*, 2005.
- [63] Duncan Grisby. omniORB, Jan 2004. Website: omniorb.sourceforge.net.
- [64] CMU Intelligent Software Agents Group. Intelligent Software Agents, Oct 2003. Website: www-2.cs.cmu.edu/~softagents/intro.htm.
- [65] Microsoft Research Vision Group. Easy Living, Oct 2003. Website: research.microsoft.com/easyliving.
- [66] The Open Group. DCE Portal, Oct 2003. Website: www.opengroup.org/dce.
- [67] C. Guestrin and G. Gordon. Distributed planning in hierarchical factored MDPs. In *Uncertainty in Artificial Intelligence (UAI)*, 2002.
- [68] Andreas Haeberlen, Eliot Flannery, Andrew M. Ladd, Algis Rudys, Dan S. Wallach, and Lydia E. Kavraki. Practical robust localization over large-scale 802.11 wireless networks. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 70–84. ACM Press, 2004.

- [69] Hani Hagra, Victor Callaghan, Martin Colley, Graham Clarke, Anthony Pounds-Cornish, and Hakan Duman. Creating an Ambient-Intelligence Environment Using Embedded Agents. *IEEE Intelligent Systems*, pages 12–29, 2004.
- [70] Richard Harper. *Inside the Smart Home*. Springer-Verlag UK, 2003.
- [71] Milos Hauskrecht, Nicolas Meuleau, Craig Boutilier, Leslie Pack Kaelbling, and Thomas Dean. Hierarchical Solution of Markov Decision Processes using Macro-Actions. In *UAI*, 1998.
- [72] Milos Hauskrecht, Nicolas Meuleau, Leslie Pack Kaelbling, Thomas Dean, and Craig Boutilier. Hierarchical solution of Markov decision processes using macro-actions. In *Uncertainty in Artificial Intelligence*, pages 220–229, 1998.
- [73] E. Heierman and D. J. Cook. Improving Home Automation by Discovering Regularly Occurring Device Usage Patterns. In *Proceedings of the International Conference on Data Mining*, 2003.
- [74] Ed Heierman, G. Michael Youngblood, and Diane Cook. Mining temporal sequences to discover interesting patterns. In *3rd Workshop on Mining Temporal and Sequential Data (TDM'04)*, August 2004.
- [75] Edwin O. Heierman. *Using Information-theoretic Principles to Discover Interesting Episodes in a Time-ordered Input Sequence*. PhD thesis, The University of Texas at Arlington, 2004.
- [76] Sumi Helal, William Mann, Hicham El-Zabadani, Jeffrey King, Youssef Kaddoura, and Erwin Jansen. The Gator Tech Smart House: A Programmable Pervasive Space. *IEEE Computer*, pages 50–60, March 2005.
- [77] James Hendler. Fathoming Funding. *IEEE Intelligent Systems*, 20:2:2–3, 2005.
- [78] B. Hengst. Safe state abstraction and discounting in hierarchical reinforcement learning. *Technical Report UNSW CSE TR 0308, National ICT*

Australia, School of Computer Science and Engineering, University of New South Wales, Sydney NSW Australia, 2003.

- [79] Bernhard Hengst. Variable Resolution Hierarchical RL. *Technical Report UNSW-CSE-TR #0309, National ICT Australia, School of Computer Science and Engineering, University of New South Wales, 2003.*
- [80] Michi Henning and Steve Vinoski. *Advanced CORBA Programming with C++*. aw-pcs. Addison-Wesley, 1999.
- [81] Denis Howe. *Free Online Dictionary of Computing*. Website: www.foldoc.org/.
- [82] Manfred Huber. *A Hybrid Architecture For Adaptive Robot Control*. PhD thesis, University of Massachusetts at Amherst, 2000.
- [83] IANA. IANA Home Page, 2005. Website: www.iana.org.
- [84] IEEE. IEEE 1278 Standard for Distributed Interactive Simulation, 1995.
- [85] IEEE. IEEE 1516 Standard for High Level Architecture, 2000.
- [86] IETF. IETF RFC page, Oct 2003. Website: www.ietf.org/rfc.html.
- [87] IETF Zeroconf Working Group. Zero Configuration Networking, 2005. Website: www.zeroconf.org.
- [88] IIEG. Welcome to IIEG, 2005. Website: cswww.essex.ac.uk/intelligent-buildings.
- [89] INRIA. PRIMA Project, 2005. Website: [www.inria.fr/recherche/equipes_ur/prima.en.html](http://www.inria.fr/recherche/equipes/_ur/prima.en.html).
- [90] American National Standards Institute. American National Standards Institute, Oct 2003. Website: www.ansi.org.
- [91] Software Engineering Institute. Software engineering institute (SEI) home page, Sep 2003. Website: www.sei.cmu.edu/sei-home.html.
- [92] Intel Corporation. Digital Home Technologies for Aging in Place, 2005. Website: www.intel.com/research/exploratory/digital_home.htm.

- [93] Anders Jonsson and Andrew G. Barto. Automated state abstraction for options using the u-tree algorithm. In *NIPS*, pages 1054–1060, 2000.
- [94] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and Acting in Partially Observable Stochastic Domains. Technical Report CS-96-08, Brown University, Providence, RI, 1996.
- [95] Leslie Pack Kaelbling, Michael L. Littman, and Andrew P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.
- [96] Marcelo Kallmann, Etienne de Sevin, and Daniel Thalmann. Constructing virtual human life simulations. In *DEFORM/AVATARS*, pages 240–, 2000.
- [97] Kristian Kersting and Luc De Raedt. Logical Markov Decision Programs. In *Working Notes of the IJCAI-2003 Workshop on Learning Statistical Models from Relational Data (SRL-03)*, pages 63–70, 2003.
- [98] F. Khawaja, D. Gjoni, M. Huber, D. Cook, and M. Youngblood. Achieving Faster Convergence to the Optimal Policy by Using Knowledge of the Unimodal Reward Structure. In *Proceedings of IASTED Artificial Intelligence and Applications*, 2003.
- [99] Phil Kingery. Digital X-10, 2005. Website: www.hometoys.com/htinews/feb99/articles/kingery/kingery13.htm\#Digital%\%20X-10.
- [100] Sven Koenig and Reid G. Simmons. The effect of representation and knowledge on goal-directed exploration with reinforcement-learning algorithms. *Machine Learning*, 22(1-3):227–250, 1996.
- [101] Sven Koenig and Reid G. Simmons. Complexity analysis of real-time reinforcement learning. In *National Conference on Artificial Intelligence*, pages 99–107, 1997.
- [102] Takayuki Kohri, Kei Matsubayashi, and Mario Tokoro. An adaptive architecture for modular q-learning. In *IJCAI (2)*, pages 820–825, 1997.
- [103] MIT Media Lab. MIT Media Lab: Projects List Database, Oct 2003. Website: www.media.mit.edu/research/index.html.

- [104] T. Lane and L. P. Kaelbling. Approaches to Macro Decompositions of Large Markov Decision Process Planning Problems. In *Proceedings of the 2001 SPIE Conference on Mobile Robotics*, Newton, MA, 2001. SPIE.
- [105] Pier Luca Lanzi. Adaptive Agents with Reinforcement Learning and Internal Memory. In *Sixth International Conference on the Simulation of Adaptive Behavior (SAB2000)*, 2000.
- [106] Pier Luca Lanzi and Stewart W. Wilson. Toward optimal classifier system performance in non-markov environments. *Evolutionary Computation*, 8(4):393–418, 2000.
- [107] P. Laroche, F. Charpillet, and R. Schott. Decomposition of markov decision processes using directed graphs. In *Poster session of the European Conference on Planning (ECP'99)*, 1999.
- [108] Haniph A. Latchman and Anuj V. Mundi. *Smart Environments: Technology, Protocols, and Applications*, chapter Power Line Communication Technologies, pages 47–62. Wiley, 2004.
- [109] Victor Lesser, Michael Atighetchi, Brett Benyo, Bryan Horling, Anita Raja, Regis Vincent, Thomas Wagner, Ping Xuan, and Shelly Zhang. The Intelligent Home Testbed. *Proceedings of the Autonomy Control Software Workshop (Autonomous Agent Workshop)*, January 1999.
- [110] Lin Liao, Dieter Fox, and Henry Kautz. Learning and inferring transportation routines. In *AAAI*, 2004.
- [111] David S. Linthicum. Next Generation Middleware, Jul 1997. Website: www.dbmsmag.com/9709d14.html.
- [112] Reed Little. Architectures for Distributed Interactive Simulation, 2004. Website: www.sei.cmu.edu/publications/articles/arch-dist-int-sim.html.

- [113] Sebastian Lühr, Hung H. Bui, Svetha Venkatesh, and Geoff A. W. West. Recognition of human activity through hierarchical stochastic learning. In *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, March 2003.
- [114] Rajbala Makar, Sridhar Mahadevan, and Mohammad Ghavamzadeh. Hierarchical multi-agent reinforcement learning. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 246–253, Montreal, Canada, 2001. ACM Press.
- [115] H. Mannila, H. Toivonen, and A. Verkamo. Discovering Frequent Episodes in Sequences. In *Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining*, pages 210–215, 1995.
- [116] MARC. Smart In-Home Monitoring System, 2005. Website: marc.med.virginia.edu/projects/_smarthomemonitor.html.
- [117] A. A. Markov. Eugene onegin. In *Proceedings of the Academy of Sciences of St. Petersburg*. Russian Academy of Sciences, 1913.
- [118] N. Massios and Voorbraak F. Hierarchical decision theoretic planning for autonomous robotic surveillance. In *EUROBOT'99 3rd European Workshop on Advanced Robotics*, pages 219–226, 1999.
- [119] A. McCallum. Learning to use selective attention and short-term memory in sequential tasks. In *From Animals to Animats, Fourth International Conference on Simulation of Adaptive Behavior, (SAB'96)*, 1996.
- [120] Andrew Kachites McCallum. Efficient exploration in reinforcement learning with hidden state. In *AAAI Fall Symposium on Model-directed Autonomous Systems*, 1997.
- [121] R. Andrew McCallum. Hidden State and Reinforcement Learning with Instance-based State Identification. *IEEE Transactions on Systems, Man, and Cybernetics*, 26(3):464–473, 1996.

- [122] A. McGovern. acQuire-macros: An algorithm for automatically learning macro-actions. In *NIPS'98 Workshop on Abstraction and Hierarchy in Reinforcement Learning*, 1998.
- [123] A. McGovern and R. Sutton. Macro-actions in reinforcement learning: An empirical analysis. *Technical Report 98-70, University of Massachusetts, Department of Computer Science*, 1998.
- [124] Amy McGovern, Doina Precup, Balaraman Ravindran, Satinder Singh, and Richard S. Sutton. Hierarchical Optimal Control of MDPs. In *Yale Workshop on Adaptive and Learning Systems*, 1998.
- [125] Amy McGovern and Richard S. Sutton. Roles of macro-actions in accelerating reinforcement learning TITLE2:. Technical Report UM-CS-1998-070, University of Massachusetts, March 1998.
- [126] R. Mehta, D. J. Cook, and L. B. Holder. Identifying Inhabitants of an Intelligent Environment using a Graph-Based Data Mining System. In *Proceedings of the Florida Artificial Intelligence Research Symposium*, pages 314–318, May 2003.
- [127] Merriam-Webster. *Webster's Revised Unabridged Dictionary*. MICRA, 1998.
- [128] Nelson Minar, Matthew Gray, Oliver Roup, Raffi Krikorian, and Pattie Maes. Hive: Distributed agents for networking things. In *First International Symposium on Agent Systems and Applications (ASA'99)/Third International Symposium on Mobile Agents (MA'99)*, Palm Springs, CA, USA, 1999.
- [129] House_n Research Group House_n Research Group MIT Dept of Architecture House_n Research Group. MIT House_n, 2005. Website: architecture.mit.edu/house_n.
- [130] Tom Mitchell. *Machine Learning*. McGraw-Hill, Boston, MA, 1997.
- [131] M. Mozer. An Intelligent Environment must be Adaptive. *IEEE Intelligent Systems*, 14(2):11–13, March/April 1999.
- [132] Michael Mozer. The adaptive house. Website: www.cs.colorado.edu/~mozer/house.

- [133] Michael C. Mozer. *Smart Environments: Technology, Protocols, and Applications*, chapter Lessons from an Adaptive House, pages 273–294. J. Wiley & Sons, Hoboken, NJ, 2004.
- [134] MulticastDNS.org. Multicast DNS, 2005. Website: www.multicastdns.org.
- [135] K. Murphy and M. Paskin. Linear time inference in hierarchical HMMs. In *NIPS*, 2001.
- [136] Hani Naguib, George Coulouris, and Scott Mitchell. Middleware support for context-aware multimedia applications. In *DAIS*, pages 9–22, 2001.
- [137] Trent Reznor (Nine Inch Nails). *WITH TEETH*, chapter Every Day is Exactly the Same, page Track 6. Interscope Records, 2005.
- [138] NiCT. The Ubiquitous Home, 2005. Website: www2.nict.go.jp/jt/a135/eng.
- [139] NIST. Welcome to the NIST Smart Space Laboratory Web Site, Oct 2003. Website: www.nist.gov/smartspace.
- [140] OASIS. OASIS, Oct 2003. Website: www.oasis-open.org.
- [141] OASIS. UDDI.org, Oct 2003. Website: www.uddi.org.
- [142] Object Management Group. Object Management Group, 2005. Website: www.omg.org.
- [143] Institute of Electrical and Electronic Engineers. IEEE standard computer dictionary: A compilation of IEEE standard computer glossaries, 1990.
- [144] Bureau of Labor Statistics. American time-use survey summary. Technical Report USDL 04-1797, United States Department of Labor, 2004. Website: www.bls.gov/news.release/atus.nr0.htm.
- [145] R. Parr. *Hierarchical Control and learning for Markov decision processes*. PhD thesis, University of California at Berkeley, 1998.
- [146] Ronald Parr. A Unifying Framework for Temporal Abstraction in Stochastic Processes. In *Symposium on Abstraction Reformulation and Approximation*, 1998.

- [147] Ronald Parr. Flexible Decomposition Algorithms for Weakly Coupled Markov Decision Problems. In *In Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 422–430, 1998.
- [148] Ronald Parr and Stuart Russell. Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995.
- [149] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1997.
- [150] Donald J. Patterson, Lin Liao, Dieter Fox, and Henry Kautz. Inferring high-level behavior from low-level sensors. In *UbiComp 2003*, 2003.
- [151] Mark D. Pendrith and Malcolm R.K. Ryan. C-Trace: A new algorithm for reinforcement learning of robotic control. In *ROBOLEARN*, 1996.
- [152] Stephen Peters, Gary Look, Kevin Quigley, Howard Shrobe, and Krzysztof Gajos. Hyperglue: Designing High-Level Agent Communication For Distributed Applications. 2003.
- [153] Karen E. Peterson. Home Sweet Ambient Home, from Philips, 2002. Website: www.10meters.com/homelab1.html.
- [154] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.
- [155] J. Pineau, N. Roy, and S. Thrun. A hierarchical approach to POMDP planning and execution. In *Workshop on Hierarchy and Memory in Reinforcement Learning (ICML)*, 2001.
- [156] J. Pineau, N. Roy, and S. Thrun. A Hierarchical Approach to POMDP Planning and Execution, 2001. Workshop on Hierarchy and Memory in Reinforcement Learning (ICML).
- [157] Joelle Pineau. Hierarchical Methods for Planning under Uncertainty. 2001.

- [158] Joelle Pineau and Geoffrey Gordon. Policy-contingent state abstraction for hierarchical MDPs. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2003.
- [159] Joelle Pineau and Sebastian Thrun. Hierarchical POMDP Decomposition for A Conversational Robot. 2001.
- [160] Shankar R. Ponnekanti, Brad Johanson, Emre Kiciman, and Armando Fox. Portability, extensibility and robustness in iros. In *Proc. IEEE International Conference on Pervasive Computing and Communications*, Mar 2003.
- [161] PostgreSQL Global Development Group. PostgreSQL, 2004. Website: www.postgresql.org.
- [162] D. Precup, R. Sutton, and S. Singh. Eligibility Traces for Off-Policy Policy Evaluation. In *In Proceedings of the Seventeenth Conference on Machine Learning*, pages 759–766. Morgan Kaufmann, 2000.
- [163] Princeton University. Wordnet 2.0, 2003. Website: wordnet.princeton.edu.
- [164] L. R. Rabiner and B. H. Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, pages 4–15, January 1986.
- [165] B. R. Rao. Making the most of middleware. *Data Communications International*, 24(12):89–96, Sep 1995.
- [166] S. Rao and D. J. Cook. Identifying Tasks and Predicting Actions in Smart Homes using Unlabeled Data. In *Proceedings of the Machine Learning Workshop on The Continuum from Labeled to Unlabeled Data*, 2003.
- [167] S. Rao and D. J. Cook. Improving the Performance of Action Prediction through Identification of Abstract Tasks. In *Proceedings of the Florida Artificial Intelligence Research Symposium*, pages 43–47, May 2003.
- [168] S. Rao and D. J. Cook. Predicting Inhabitant Actions Using Action and Task Models with Application to Smart Homes. In *International Journal of Artificial Intelligence Tools*, 2004.

- [169] Balaraman Ravindran and Andrew G. Barto. Model Minimization in Hierarchical Reinforcement Learning. In *Fifth Symposium on Abstraction, Reformulation and Approximation*, 2002.
- [170] Mark Reid and Malcolm Ryan. Using ILP to improve planning in hierarchical reinforcement learning. *10th International Conference on Inductive Logic Programming*, 1866:174–190, 2000.
- [171] J. Rissanen. *Stochastic Complexity in Statistical inquiry*. World Scientific Publishing Company, 1989.
- [172] Manuel Román, Christopher K. Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. Gaia: A Middleware Infrastructure to Enable Active Spaces. *IEEE Pervasive Computing*, pages 74–83, 2002.
- [173] Ton Roosendaal. Blender 3D, 2005. Website: www.blender3d.com.
- [174] A. Roy, S. Bhaumik, A. Bhattacharya, K. Basu, D. J. Cook, and S. Das. Location Aware Resource Management in Smart Homes. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications*, pages 0–0, 2003.
- [175] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ, 1995.
- [176] Malcolm Ryan and Mark Reid. Learning to fly: An application of hierarchical reinforcement learning. In *Proc. 17th International Conf. on Machine Learning*, pages 807–814. Morgan Kaufmann, San Francisco, CA, 2000.
- [177] Malcolm R. K. Ryan and Mark D. Pendrith. RL-TOPs: an architecture for modularity and re-use in reinforcement learning. In *Proc. 15th International Conf. on Machine Learning*, pages 481–487. Morgan Kaufmann, San Francisco, CA, 1998.
- [178] Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *CHI*, pages 434–441, 1999.

- [179] Christian Schuckmann, Lutz Kirchner, Jan Schummer, and Jorg M. Haake. Designing object-oriented synchronous groupware with COAST. In *Computer Supported Cooperative Work*, pages 30–38, 1996.
- [180] George Schussel. Client/server past, present, and future, 1995. Website: news.dci.com/geos/dbsejava.htm.
- [181] Yuanchun Shi, Weikai Xie, Guangyou Xu, Runtong Shi, Enyi Chen, Yanhua Mao, and Fang Liu. The Smart Classroom: Merging Technologies for Semless Tele-education. *IEEE Pervasive Computing*, 2, 2003.
- [182] Siemens. Smart Homes, 2005. Website: www.siemens-industry.co.uk/main/business%20groups/et/smart%20homes.
- [183] Bryan Singer and Manuela M. Veloso. Learning state features from policies to bias exploration in reinforcement learning. In *AAAI/IAAI*, page 981, 1999.
- [184] S. P. Singh. Scaling reinforcement learning algorithms by learning variable temporal resolution models. In *MLC-92*, 1992.
- [185] Satinder Singh. Reinforcement Learning with a Hierarchy of Abstract Models. In *Proceedings of the National Conference on Artificial Intelligence*, 1992.
- [186] Satinder Pal Singh. Learning to solve markovian decision processes. Technical Report UM-CS-1993-077, University of Massachusetts, , 1993.
- [187] M. Skounakis, M. Craven, and S. Ray. Hierarchical hidden markov models for information extraction. In *18th International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.
- [188] James Snell, Doug Tidwell, and Pavel Kulchenko. *Programming Web Services with SOAP*. O'Reilly, Sebastapol, CA, 2002. Website: www.oreilly.com/catalog/progwebsoap.
- [189] Edward J. Sondik. The optimal control of partially observable Markov Decision Processes, 1971. PhD thesis, Stanford University, Palo Alto, CA.

- [190] Sony Entertainment. Everquest, 2005. Website: eqlive.station.sony.com.
- [191] R. Srikant and R. Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements. In *Proceedings of the 5th International Conference on Extending Database Technology*, pages 3–17, 1996.
- [192] Ambika Srinivasan and Sharma Chakravarthy. Discovery of Interesting Episodes in Sequence Data. In *In proceedings of PAKDD Workshops*. PAKDD, May 2004.
- [193] Stanford Interactivity Lab. Interactive Workspaces, Oct 2003. Website: iwork.stanford.edu.
- [194] Steve Steinke. Middleware meets the network. *LAN: The Network Solutions Magazine*, 10(13):56, 1995.
- [195] Peter Stone and Manuela M. Veloso. Layered learning. In *Machine Learning: ECML 2000, 11th European Conference on Machine Learning, Barcelona, Catalonia, Spain, May 31 - June 2, 2000, Proceedings*, volume 1810, pages 369–381. Springer, Berlin, 2000.
- [196] Thomas Strang and Claudia Linnhoff-Popien. A context modeling survey. In *Workshop on Advanced Context Modelling, reasoning, and Management UbiComp 2004*, 2004.
- [197] Norbert Streitz, Carsten Rucker, Thorsten Prante, Daniel van Alphen, Richrd Stenzel, and Carsten Magerkurth. Designing Smart Artifacts for Smart Environments. *IEEE Computer*, March:41–49, 2005.
- [198] R. Sun and C. Sessions. Self segmentation of sequences. In *IEEE International Conference on Neural Networks*, 1999.
- [199] R. Sun and C. Sessions. Self-segmentation of sequences: Automatic formation of hierarchies of sequential behaviors. Technical Report 609951-2781, NEC Research Institute, Princeton, 1999.
- [200] Sun Microsystems, Inc. The Source for Java Technology, Oct 2003. Website: java.sun.com.

- [201] Richard S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 39:9–44, 1988.
- [202] Richard S. Sutton. TD models: Modeling the world at a mixture of time scales. In *International Conference on Machine Learning*, pages 531–539, 1995.
- [203] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.
- [204] Richard S. Sutton, Doina Precup, and Satinder P. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.
- [205] K. Sycara, K. Decker, A. Pannu, M. Williamson, and D. Zeng. Distributed intelligent agents. *IEEE Expert*, 11(6):36–46, December 1996.
- [206] Prasad Tadepalli and Thomas G. Dietterich. Hierarchical explanation-based reinforcement learning. In *Proc. 14th International Conference on Machine Learning*, pages 358–366. Morgan Kaufmann, 1997.
- [207] Tampere University of Technology Institute of Electronics. The Smart Home Project, 2005. Website: www.ele.tut.fi/research/personalelectronics/projects/smart_home.htm.
- [208] Thuan L. Thai. *Learning DCOM*. O’Reilly, Sebastapol, CA, 1999. Website: www.oreilly.com/catalog/ldcom.
- [209] G. Theodorou and S. Mahadevan. Approximate planning with hierarchical partially observable markov decision processes for robot navigation. In *IEEE International Conference on Robotics and Automation*, 2002.
- [210] G. Theodorou, K. Rohanimanesh, and S. Mahadevan. Learning Hierarchical Partially Observable Markov Decision Processes for Robot Navigation, 2001. IEEE Conference on Robotics and Automation.

- [211] Georgios Theodorou and Sridhar Mahadevan. Learning the hierarchical structure of spatial environments using multiresolution statistical models. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2002.
- [212] Georgios Theodorou, Kevin Murphy, and Leslie Pack Kaelbling. Representing hierarchical POMDPs as DBNs for multi-scale robot localization. In *International Conference on Robotics and Automation*, 2004.
- [213] Sebastian Thrun and Anton Schwartz. Finding structure in reinforcement learning. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 385–392. The MIT Press, 1995.
- [214] Khai N. Truong and Gregory D. Abowd. INCA: Architectural support for building automated capture and access applications, 2002. Website: www.cc.gatech.edu/fce/ahri/publications/inca_final.pdf.
- [215] UCLA HyperMedia Studio. UCLA HyperMedia Studio Projects, 2005. Website: hypermedia.ucla.edu/projects/atc.php.
- [216] UIUC Software Research Group. Gaia homepage, 2005. Website: gaia.cs.uiuc.edu.
- [217] International Telecommunications Union. The ITU telecommunications standardization sector (itu-t), Oct 2003. Website: www.itu.int/ITU-T.
- [218] UPnP Forum. Welcome to the UPnP Forum!, 2005. Website: www.upnp.org.
- [219] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, 1989.
- [220] Webopedia. SQL, Oct 2003. Website: www.webopedia.com/TERM/S/SQL.html.
- [221] Mark Weiser. The Computer for the 21st Century. *Scientific American*, September 1999.
- [222] M. Wiering and J. Schmidhuber. HQ-learning. *Adaptive Behavior*, 6(2):219–246, 1997.
- [223] Ian H. Witten. An adaptive optimal controller for discrete-time Markov environments. *Information and Control*, 34:286–295, 1977.
- [224] X10 Corporation. X10.com, 2005. Website: www.x10.com.

- [225] Lexing Xie, Shih-Fu Chang, Ajay Divakaran, and Huifang Sun. Unsupervised discovery of multilevel statistical video structures using hierarchical hidden markov models. In *International Conference on Multimedia and Expo (ICME)*, 2003.
- [226] Candice A. York. IBM's advanced PvC technology laboratory, Oct 2003. Website: www-106.ibm.com/developerworks/wireless/library/wi-pvc.
- [227] G. Michael Youngblood. MavHome: Managing an Adaptive Versatile Home, Oct 2003. Website: mavhome.uta.edu.
- [228] G. Michael Youngblood, Edwin O. Heierman, Diane J. Cook, and Lawrence B. Holder. Automated Hierarchical POMDP Construction through Data-mining Techniques in the Intelligent Environment Domain. In *FLAIRS*, 2005. Website: www.aimachines.com/youngblood/publications.html.
- [229] G. Michael Youngblood, Lawrence B. Holder, and Diane J. Cook. Managing Adaptive Versatile Environments. In *Innovative Applications of Artificial Intelligence*, 2005. Website: www.aimachines.com/youngblood/publications.html.

BIOGRAPHICAL STATEMENT

The author received his Doctorate of Philosophy in Computer Science and Engineering from The University of Texas at Arlington in August 2005.

Gregory Michael Youngblood, a native of Bishop, Texas, born in Atlanta, Georgia, received a Masters in Computer Science and Engineering in December 2002 and an Honors Bachelor of Science in Computer Science and Engineering in May 1999 from UT Arlington. He received an Associate of Science in Applied Science and Technology in Nuclear Engineering Technologies from Thomas A. Edison State College, Trenton, New Jersey, in November 1995. He is also a 1990 graduate of the U.S. Naval Nuclear Power Program.

Dr. Youngblood is a member of Sigma Xi (UT Arlington 2005), Tau Beta Pi (TX Eta 98), Upsilon Pi Epsilon (TX Gamma 99), Omicron Delta Kappa (Sincerus Orbis Circle 98), and the Golden Key Honor Society (98). He was a Tau Beta Pi Fellow in 2002-2003. He has been included in Who's Who in Science and Engineering 1998-2000, and Who's Who Among Students in American Universities and Colleges, 1998-99, 2005.

From 1988 to 1996, Dr. Youngblood served in the United States Navy as an Engineering Laboratory Technician and Nuclear Engine Room Supervisor in the Reactor Laboratories Division aboard the USS BOISE (SSN-764) and as a Radiological Controls Shift Supervisor aboard the USS CANOPUS (AS-34, Decommissioned) and USS EMORY S. LAND (AS39). His military awards include the Navy Achievement Medal, Meritorious Unit Commendation, Good Conduct Medal (two awards), and the National Defense Medal.

Dr. Youngblood is married to the former Julie Rose of Apopka, Florida. They and their two sons, Gregory and Austin, reside in Arlington, Texas.