ENABLING CONTEXT-AWARE APPLICATIONS IN ASSISTIVE ENVIRONMENTS

by

ROMAN ARORA

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

MASTER OF SCIENCE IN COMPUTER SCIENCE ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2009

ACKNOWLEDGEMENTS

This research project could not have been possible without the guidance and support of numerous people. I would like to thank my supervisor Dr. Fillia Makedon for the invaluable support as I pursued my research. I also wish to express my deepest gratitude to the members of the supervisory committee, Dr. Manfred Huber and Dr. Vassilis Athitsos as without their knowledge and assistance this study would not have been successful.

Lastly, I would like to thank my parents, my friends, and my lab for their encouragement and support throughout the duration of my studies

November 19, 2009

ABSTRACT

ENABLING CONTEXT-AWARE APPLICATIONS

IN SMART ENVIRONMENTS

Roman Arora, M.S.

The University of Texas at Arlington, 2009

Supervising Professor: Fillia Makedon

Recognizing human activities is an important feature for the development of context-aware applications that are so fundamental to enabling assistive environments. Only once these applications are able to determine the activities that their inhabitants are performing can they assist the individuals and their special needs. In order to do this, it is necessary to build models that can accurately capture and recognize the observed patterns. Equally important is the need to manage and distribute the information that has been inferred, and to provide Quality of Service (QoS) guarantees so that context-aware applications can react effectively to emergent situations. In this thesis, we explore these two problems of knowledge inference and dissemination. We approach the issue of activity recognition from a new perspective, were our goal is that of mining rules to complement an existing algorithm's capability to recognize activities, and thus improve overall accuracy results. For knowledge dissemination, we propose a framework to facilitate QoS in ontology centered context aware pervasive middleware. We believe our work validates two potential ways in which to overcome some of the existing problems for the proper operation of context-aware applications.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

As the number of elderly people grows, the health care system struggles to find alternative ways to provide affordable health care to the aging population. According to the United Nations, Dept. of Economic and Social Affairs Population Division, by 2050, the percentage of people over the age of 60 is expected to double, and the percentage of those over the age of 85 will quadruple [1]. This trend motivates the development of solutions that reduce the dependence on healthcare and nursing personnel. By enabling the elderly and people with disabilities to live in their homes, it is possible to reduce health care facility and hospital dependency while at the same time enhancing an individual's self-reliance, self-esteem, and comfort by allowing that person to live a much more independent life style. In order to do so it is necessary to provide a means to help the individual in their home as they would be helped should they be in a care facility. This fact has created a great amount of research interest in the development of ubiquitous computing [2] environments capable of providing the infrastructure to monitor, recommend, and promote healthy life styles. However, before such realization of an assistive environment can be viable, it is necessary to develop the infrastructure and techniques that will allow inferring knowledge from the changes in the environment, and to manage and disseminate the acquired knowledge to interested agents.

The problem of inference is that of recognizing the actions or activities that the inhabitant is performing at a specific time. Researchers believe that providing pro-active care for their aging inhabitants by detecting medical conditions before they become critical can have a significant impact on the quality of their lives, and that this detection can be accomplished by analyzing variations in the inhabitant's capability to perform Activities of Daily Living (ADL) [3].

In order to enable an assistive environment to recognize the activities taking place, it is necessary to extract meaningful information from the low level information obtained from the sensors and devices within, and to combine this information to infer high level knowledge, such as the activities taking place. There are various challenges in solving the problem of activity recognition since there exists a wide set of heterogeneous sensors and devices As the scenarios explored increase in complexity, so do the algorithms proposed; each with its advantages and disadvantages.

Another problem is that of managing and distributing the high level knowledge to empower context-aware applications [4] and devices. For example, if a person required to have their medication three times a day, and the assistive environment is to appropriately operate, it is required not only that it detects the activity taking place, but also that the information being provided to applications is used as necessary by, for example, contacting a nurse, caregiver or family member about the emergent situation. There are again many challenges, as there are many possible architectures and frameworks that can be developed to allow applications and devices to make smarter decisions in a broader context and in a more macroscopic perspective. In many cases the decisions to be taken by such systems require real-time responses which may vary from a few seconds to a few nano-seconds. This is often a very difficult task, especially when we need to combine huge amounts of data from different sources which may also reside in different devices/machines.

Significant research has been conducted in order to handle the aforementioned problems and to build such systems [5]. However, the work is still fragile in real world conditions, requiring further research into creating more robust and viable solutions. For this reason, in this paper we propose our own strategies to tackle the problems of knowledge inference, storage, and dissemination. With respect to inference, we propose a two-level framework designed to maintain a low complexity in the algorithms involved in performing activity recognition. Instead of proposing ever more complex algorithms, we explore a different

approach in which our goal is to design an algorithm that can learn the cases that are consistently being misclassified by an activity recognition algorithm, and use this information to generate rules that can be applied to correct such cases. With respect to enabling context-aware applications so that they can operate with the knowledge that the environment has acquired, we propose a middleware architecture that relies on the Semantic Web for knowledge representation, yet is intended to facilitate Quality of Service (QoS) in order provide performance guarantees for applications that need to be responsive in emergent situations, such as those that can arise in an assistive environment.

The rest of the paper is structured as follows. In Chapter 2 we review existing work and trends on activity recognition and context-aware computing. In Chapter 3 we discuss our proposed two-level activity recognition framework; we present our experiments and evaluate the results. In Chapter 4, we propose a QoS-centered context-aware middleware framework for applications that need performance guarantees in order to operate smoothly in emergent situations. Finally, in Chapter 6 we provide some concluding remarks.

CHAPTER 2

OVERVIEW AND RELATED WORK

With the technological developments that have taken place in the last few years, the possibility of providing automated assistance to people with physical and mental challenges is becoming a reality. Sensor technology can be used to sample information, and automatic and robotic systems can use the information to aid human subjects. However, in order to build such autonomous systems that can monitor and detect medical or behavioral emergent situations we require a solution to the problems of knowledge inference, storage, and dissemination. Only then can the appropriate actions take place, such as alerting the inhabitant, the caregivers, or the family members about the emerging situation. In this chapter, we look into the different data collection techniques and algorithms that are being used for activity recognition as well as the ongoing efforts in building context aware applications for smart environments. We highlight some of the existing challenges and we discuss how we believe they can be overcome.

2.1 Data Collection Techniques

There are essentially two paradigms to activity recognition; these are video-based and sensor-based activity recognition. In video-based activity recognition, we rely on cameras and object recognition algorithms to determine the activities that a user is carrying out. Significant work has been done in the field of video-based activity recognition. Robertson, et al. [6] achieved activity recognition in video sequences by modeling the problem as a stochastic sequence of actions. Lymberopoulos, et al. [7] treated the problem of extracting a spatiotemporal human activity model from an assisted living environment using sensor networks. They used a data model consisting of *<location, time, duration>* tuples to derive

human activity patterns. Their experiment layout consisted of tracking cameras, door sensors, and passive infrared sensors to collect a data set on which they extracted a user's daily patterns. Hongeng, et al. [8] represented activities by considering them as a composition of action threads, where each thread belonged to a single resident. They exploited trajectory movements and shapes and used Bayesian methods to do inference. The key advantage of video-based activity recognition systems is that they are very informative sources of data. However, they are obtrusive and can introduce potential problems of privacy and security. These reasons have led to research using less invasive technologies to do activity recognition.

In sensor-based activity recognition, we rely on small cost-effective sensor technologies that provide small bits of information that when combined can be used to infer the high level activities taking place. In the case of wearable-sensors, sensors are attached to the body that can capture information regarding the motions different parts of the body are undergoing. Patterson, et al. [9] used a combination of RFID gloves and RFID sensors in objects to do object-interaction based activity recognition. In their experiment, they used over sixty RFID tags placed in kitchen objects to be able to capture the identity of the objects being manipulated. Ravi, et al. [10] explored using a single tri-axial accelerometer worn near the pelvic region to do activity recognition. Gu, et al. [11] carried out an experiment combining RFID technology and body sensors, thus capturing human-object interaction as well as human movement. They used a pattern-based data mining approach towards solving the activity recognition problem. Stikic, et al. [12] used a combination of RFID tag readers and accelerometers to infer activities. They explored different acceleration features and algorithms. Their experimental results show that fusing both types of information, RFID and body acceleration, significantly improves accuracy.

Another form of sensor-based activity recognition is the one in which sensor placement is within the environment and not on the inhabitants. Park, et al. [13] experimented with detecting human behaviors in an assisted living environment using the longest common subsequence algorithm. They define an episode as a sequence of events and compare the

current episode taking place against a dictionary of annotated episodes belonging to different activities. They used SunSpot [14] technology placed in a miniature smart-apartment. Singla, et al. [15] equipped an on-campus apartment with motion, temperature, and item sensors to perform activity recognition.

2.2 Activity Recognition

Many researchers have used probabilistic models to approach the problem of activity recognition. Their models differ widely depending on their data collection process, how they define activities, and the accuracy metrics they use. Patterson, et al. [9] used a combination of Hidden Markov Models (HMM) and Dynamic Bayesian Networks (DBNs). They initially explored a single state per activity HMM model. Later, they expanded their HMM to have multiple states per activity, however, this did not improve their accuracy results. Finally, they developed an object abstraction layer and used DBNs to combine the output of the HMM with their abstraction layer to obtain marginal improvements. Wu, et al. [16] explored the use of DBNs to infer the most likely activity and object labels in their work. Wilson, et al. [17] also used DBNs to exploit the spatial relationships between location and activity for simultaneous resident tracking and activity recognition. Singla, et al. [15] used HMMs to model sequential and interleaved activity recognition. They developed a data set from twenty volunteers performing a series of eight different ADLs in a campus apartment. Wu, et al. [18] used Factorial Conditional Random Fields (FCRF) to recognize multiple concurrent activities. Hu, et al. [19] proposed a two-level probabilistic framework called CIGAR (Concurrent and Interleaving Goal and Activity Recognition) for recognizing both interleaved and concurrent activities. They used Skip Chain Conditional Random Fields (SCCRF) to model interleaved tasks and a correlation graph to adjust inferred probabilities to model concurrent tasks.

Most of the research work done so far focuses on developing new models and algorithms to tackle the problem of activity recognition. They do so by introducing more powerful

and complex implementations to capture more spatial and temporal relationships, and thus produce more accurate results. Unlike their work, our focus is on developing an algorithm that can work over existing algorithms to improve their accuracy by learning their limitations. Our approach is fundamentally different in that we are developing a framework that allows us to use simple activity recognition models that have low complexity and low data set training requirements to perform activity recognition by complementing them with rules that overcome their shortcomings. The advantages of this approach are many, one of which is that we are not bound to work on one single algorithm, but that as developments take place in the field we can migrate from our base algorithm to another, while maintaining the overall framework structure.

2.3 Context-Aware Applications

There is significant research regarding the development of applications that use context information to make decisions; these are commonly known as context-aware applications. Developing such applications is challenging, as different smart environments are composed of different hardware and present their own architectural limitations. The solutions proposed have been to develop frameworks, middleware, that abstract the physical details of the smart environment and provide a high level API to enable applications to have access to context information. With the introduction of the Semantic Web [20] and semantic representation of knowledge in general there has been a significant turn into using such technologies for knowledge management, reasoning and decision making in smart environments and context aware applications in general [21]. Ontologies [22] have been used for formal representation of concepts and relations between them and structured data representation methods such as RDF/XML [23] have been used for description of resources and meta-data. Amongst the increasing number of ontology centered middleware, we highlight Construct [24], SOCAM [25], COBRA-ONT [26], and Semantic Context Spaces [27]. All these middleware address most of the issues identified as important for creating a functional environment, however they do not

dwell much in issues of efficiency and performance that arise due to the existing setup, whether in terms of resources or in computation costs. While the Semantic Web offers a great framework for information management and dissemination, performance issues are often left unaddressed. In this work, we approach context-aware applications from the perspective of performance and QoS, and we suggest methods that can address the arising issues.

Following, we explore what we believe are two key problems that must be addressed to enable context-aware applications in smart environments. First, we examine the problem of knowledge inference in the form of activity recognition; we discuss our proposed two-level framework and evaluate our results. Then, we examine the importance of QoS in context-aware middleware through examples, and present a framework to provide such functionality in middleware that rely on Semantic Web technologies.

CHAPTER 3

ACTIVITY RECOGNITION

Activity recognition is a well-known problem within the research community in which our goal is to determine the activity or set of activities that one or more inhabitants are performing within an environment. The definition of an activity itself however, is not as well defined, generally because human beings can conceive and perform activities in a very wide range of ways. Regardless of the interpretation or the variations introduced by specific individuals during the execution of an activity, spatial and temporal characteristics are shared amongst the different ways in which an activity can be performed. To develop a model that can address the problem of activity recognition, a common approach is to rely on learning algorithms that generate probabilistic models. The requirement, however, is that an annotated dataset be available to train the algorithm and create the model. In this chapter, we define key terms, explain a classification for activities based on their temporal characteristics, and describe metrics that are used to quantify the validity of a solution. We next describe a probabilistic model, and illustrate its shortcomings and the need for better solutions. Then, we present our proposed method to overcome those shortcomings and elaborate on its technical aspects. Finally, we describe our experimental setup and evaluate the results.

3.1 Problem Definition

When users interact with an assistive environment, they trigger sensor events. We define an *event* as a response of a sensor when a given condition is satisfied. A common way in which sensors produce events is when they exceed a certain threshold. An example would be a

light sensor that produces the “On” event when the light exceeds a certain light intensity value, and produces an “Off” event when the light intensity drops below a given intensity value.

Definition 1: An event e_i is a tuple $<id, time, value>$ where i is the unique sensor id.

An episode is defined as a collection of all the events that were triggered by the same person during a finite amount of time. Thus, if our time granularity is a day, we can say that an episode contains the sensor events that were triggered by a person throughout the course of his daily routine in an assistive environment.

Definition 2: An episode is a time ordered collection of events $P_K = \{e_1, e_1, e_2, e_2, \dots, e_x\}$ where x is the id of the last sensor event triggered.

Definition 3: A dataset D is a collection of episodes.

A contiguous series of sensor events present in an episode are usually triggered while performing one single activity. In training data sets, events are labeled according to the activity that triggered the event. Therefore, under the assumption that each event only has one activity label in the training dataset, we can define Q_i as a sequence of events all of which have the same activity label l_a. We define the functions L, E, and T, where L returns the label of a sequence Q_i, E returns a specific event of a sequence Q_i, and T returns the start time of sequence Q_i.

Definition 5: We define $L(Q_i)$ as the activity label l_a assigned to Q_i

Definition 6: We define $E(Q_i, j)$ as the j^{th} element in sequence Q_i

Definition 7: We define $T(Q_i)$ as the start time of sequence Q_i

Definition 8: We re-define an episode P_k as a sequence of sequences Q_i where:

$$P_k = Q_1 Q_2 Q_3 \dots Q_x$$

$$\forall_i |Q_i| \geq 1$$

$$\forall_i, \forall_j : 1 \leq j \leq |Q_i|, L(Q_i) \neq L(Q_{i-1}) \wedge L(Q_i) \neq L(Q_{i+1})$$

3.1.1 Activity Types

Activities can be classified into different types according to their temporal pattern of execution. We distinguish three types: sequential, interleaved, and concurrent. When a user performs an activity sequentially, it means the events triggered from the beginning to the end of an activity appear contiguously within an episode. People perform activities in a much more complex fashion, where they can stop during the course of an activity to start another one, and later at some point in time return to finish the previous activity. We refer to these cases as interleaving or interwoven activities. The third case is when a person is carrying out multiple activities at the same time. Concurrency can be regarded as a special type of interleaving pattern where the activities in question are interleaving very quickly and it would be in our nature to refer to them as happening concurrently. The concept of concurrency is important when the algorithms evaluating accuracy deal with time-slices rather than with single events.

3.1.2 Accuracy Metrics

There exist different metrics to evaluate the performance of an activity recognition algorithm. We highlight two metrics:

- *Event accuracy*: Label every event in a data set with one or more activity labels and then calculate the percentage of correctly labeled events.
- *Time-slice accuracy*: Divide all events in a data set into groups based on time. Then, label all events taking place in a time-slice and count the percentage of time-slices that were properly labeled.

A single event can have multiple labels in a data set depending on the researcher's criteria for the definition of concurrency. Multiple variants exist based on how they penalize the cases of partial or improper labeling and the size of the time window used.

Within these two metrics, three variables need to be measured; these are *precision*, *recall*, and *overall accuracy*.

- *Precision*: Proportion of the data labeled as activity n that actually was from data labeled as activity n in the training data (ground truth).
    - Formula: True Positive / (True Positive + False Positive)
- *Recall:* Proportion of the data originally labeled as activity n in the training data set that was correctly classified as activity n
    - Formula: True Positive / (True Positive + False Negative)
- *Overall accuracy:* Proportion of the data that was correctly labeled
    - Formula: True Positive + True Negative / Total number of events

Algorithms can evaluate their performance because they rely on an already labeled dataset. The common approach is to divide this set into two parts, one called the training set and the other one called the test set. A typical strategy to predict the accuracy of an activity recognition algorithm in a real scenario is to perform cross-validation during the analysis of the algorithm's performance.

3.2 Hidden Markov Models

A Hidden Markov Model (HMM) is a statistical model in which the system being modeled is assumed to be a Markov process with unobserved state. While the state is not visible, the output dependent of state, known as the observations, are visible. An HMM consists of a finite number of states, each of which is associated with a probability distribution over the possible observations. An illustration of the general architecture of a HMM is shown in Figure 1.

The arrows represent conditional dependencies. At a given time t, the random variable $x(t)$ is the hidden state and the random variable $y(t)$ is the observation.

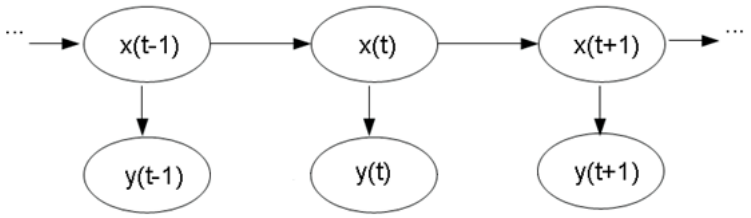

Figure 1 – General architecture of a Hidden Markov Model.

Two assumptions are made by the model. The first one is the Markov assumption, which says that the conditional probability distribution of any hidden state $x(t)$ at time t depends only on the value of its preceding hidden state $x(t-1)$ at time t - *1*. Thus, any states prior to $x(t-1)$ have no influence on the values of a state at time t. The second one is the independence assumption, which says that the value of the observation $y(t)$ depends only on the value of the hidden state $x(t)$ given at time t.

Hidden Markov Models (HMM) are known to perform well in cases where temporal patterns need to be recognized. In the case of activity recognition, sensor event values and their relationships are used to train the model. Figure 2 illustrates an example of a single-state per activity HMM.

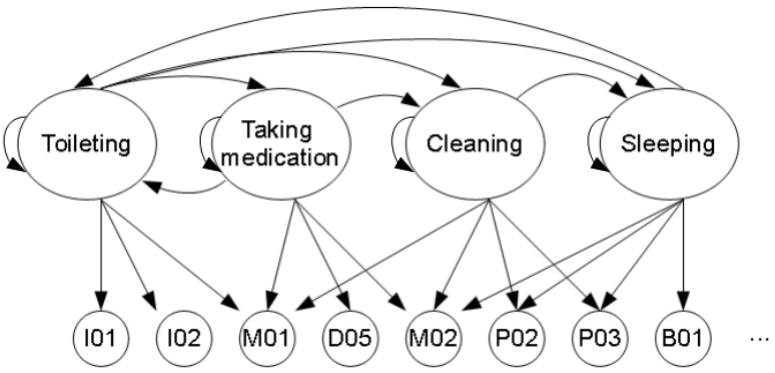

Figure 2 – Single state per activity HMM.

In Figure 2, the ellipses are states that represent activities. The small circles represent observations about the environment, such as events triggered by infrared, pressure, motion, and bed sensors. In the previous example, every activity is represented by a single state. In order to build the HMM, a typical strategy is to do statistical analysis of the training set to determine the initial probabilities, transition probabilities, and emission probabilities for each state. The initial probability of the states represents the likelihood that an episode would start with a given activity. The transition probabilities represent the likelihood that given a current activity is taking place, another one would follow it. The emission probabilities represent the likelihood of observing a given sensor event in a given activity.

Typical implementations of HMMs for the problem of activity recognition use either a single state [9, 15] or multiple states [9] to represent each activity. Determining a strategy to divide a single activity into multiple states such that the overall accuracy of the model actually improves is a non-trivial problem. Attempts [9] to improve accuracy by increasing the number of states do not automatically guarantee an accuracy improvement. Many factors can influence the effectiveness of a multiple state per activity model, such as the data or data collection process, and the design logic followed to develop the HMM.

In order to determine the sequence of activities that took place in an episode, it is necessary to determine the most likely states that caused the observations. Two common algorithms are listed:

- *Forward-Backward* [43]: Algorithm for finding the most likely state s at a given time t for a given observation.
- *Viterbi* [28]*:* Algorithm for finding the most likely sequence of states that correspond to a sequence of observations.

While the most probable state obtained with the Forward-Backward algorithm is the most likely to be correct at a given point in time, the sequence of individually most probable

states is not likely to be the most probable sequence. This is because the probabilities for each point are calculated independently of each other and thus do not take into account transition probabilities between states. The Viterbi algorithm however, does take into account state transition probabilities.

3.3 Proposed Method

In order to model complex relationships and dependencies between events in an activity, it is required to increase the number of states and thus the complexity of the Hidden Markov Model. Adding such relationships can make the model grow exponentially and be no longer tractable. For this reason, we believe the solution is a two-level approach were we allow an algorithm to do the first estimation of the data, and then we allow a second algorithm to correct more complex event relationships which the first algorithm could not handle. By separating the problem of activity recognition into a multi-level problem, we simplify the complexity of the algorithms involved in estimating the correct labels for the events observed. However, it is important for the second level estimation to correct only the cases that are consistently being missed by the first level estimation. Thus, the goal of the second level algorithm is to identify frequently incorrect estimations by the first algorithm and execute a strategy to re-label the mislabeled events. Figure 3 illustrates the overall architecture of the two-level framework.

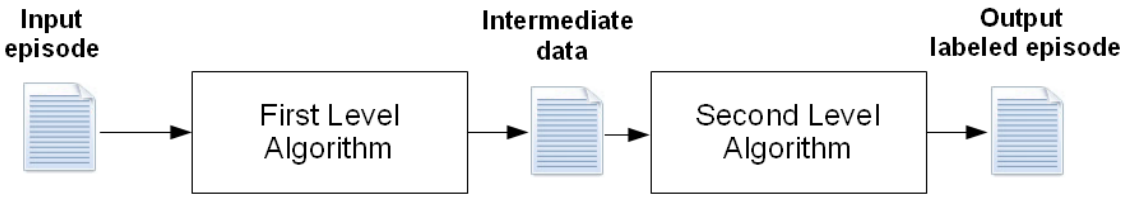

Figure 3 - Data flow in the two-level framework for activity recognition.

3.3.1 Mining Association Rules

Our second level algorithm operates by mining rules between the training data and the estimations produced by the first level algorithm on the same training data. Once the mining process is complete, a dictionary of the most valuable rules is kept and used them to re-estimate the output produced by the first level algorithm on the test data. We define association rule and invalid sequence as follows.

Table 1 – Symbols.

e_i	A tuple <*id, time, value*> generated by a sensor, where i is the unique sensor id
l_a	A label for an activity a
P_k	A time ordered collection of events $P_k = \{e_1, e_1, e_2, e_2, ..., e_x\}$ where x is the id of the last sensor event triggered.
D	A collection of episodes
Q_i	A sequence of events all of which have the same activity label l_a
$L(Q_i)$	The activity label in sequence Q_i
$E(Q_i, j)$	The j^{th} element in sequence Q_i
$T(Q_i)$	The start time of sequence Q_i

Definition 9: Given a sequence of events Q_i with activity label l_j, and the sets of events s_a (antecedent set) and s_c (consequent set) we define an association rule r as an implication of the form: $(L(Q_i) = l_j \Lambda (\forall x \in s_a)[(\exists m)(x = E(Q_i, m)] \Rightarrow (\exists y \in s_c)[(\exists n)(y = E(Q_i, n)]$.

Definition 10: Given a sequence of events Q_i with activity label l_j, we define the sequence as *invalid* if it violates the rule r.

In order to mine rules in an efficient way, we propose the following algorithm. For each episode belonging to our training dataset D, and for each activity a we create two lists. One containing a series of sequences of events that we know belong to the given activity L_v =

$\{Q_1, Q_2, Q_3, \ldots, Q_x\}$, and another list containing the series of sequences of events that were improperly labeled as belonging to activity a by the first-level algorithm; we call the second list $L_i = \{Q_1, Q_2, Q_3, \ldots, Q_y\}$. We take each sequence and convert it into a set, so we now have two lists of sets $L_{vs} = \{s_1, s_2, \ldots, s_x\}$ and $L_{is} = \{s_1, s_2, \ldots, s_y\}$. For each set s_i in the invalid list, we use s_i as the antecedent set s_a of a rule, and we use the subtraction of a valid set s_j from the invalid set s_i as the consequent set s_c of the rule. If the consequent is an empty set, no rule is created. We call the resulting rule a *candidate rule*, since we have not yet evaluated whether it will be used or not on the test set. The intuition behind these operations is to derive event dependencies, where the presence of a set of events s_a strongly correlates with the presence of an event $y \in s_c$. An example follows.

Table 2 - Example candidate rules.

Invalid sets for activity a in data set d	Valid sets for activity a in data set d	Candidate rule sets generated	
		s_a	s_c
{1,3,20,36}, {2,6,13, 20}, {4,10,20,30}	{2, 6, 8, 10}, {6, 8}	{1,3,20,36},	{2,6,8,10}
		{1,3,20,36}	{6, 8}
		{2,6,13,20}	{8, 10}
		{2,6,13,20}	{6,8}
		{4,10,20,30}	{2,6,8}
		{4,10,20,30}	{6,8}

All candidate rules generated are stored for future use; we refer to them as being in the *candidate rule pool*. Whenever a new candidate rule is to be placed in the pool, we attempt to combine it with all existing candidate rules already in the pool. This combination process is explained in detail later. In the process, we evaluate the combined rules with a fitness function and keep the fittest combined rule. This one then becomes a valid rule and is kept for future use in the *valid rule pool*. The methodology for combining candidate rules, evaluating fitness, and combining valid rules is described below. A flowchart description of the process for generating rules follows.

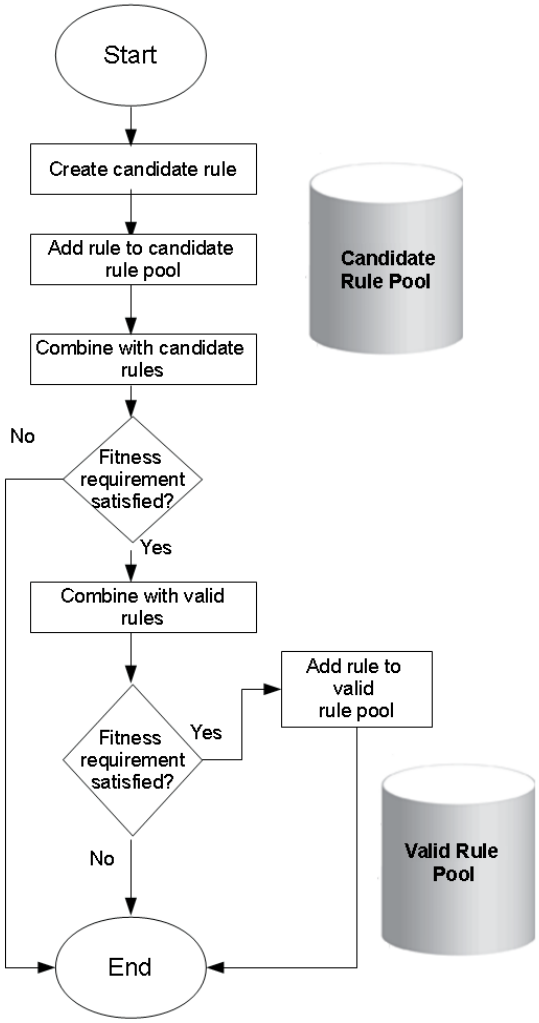

Figure 4 – The process of generating rules.

The combination process of the new candidate rule with existing candidate rules is as follows: For each candidate rule already existing in the pool, we perform an intersection of the antecedent sets s_a of the rules, and a union of the consequent sets s_c of the rules. The logic behind these operations is that when we do an intersection of the antecedents, we are guaranteeing that the rule can be applied to the original invalid sequences that gave place to the rule. It also helps determine the common events that appear in invalid sequences. By doing a union of the consequent sets we make sure that the application of the rule would not violate

valid sequences that have all the elements of the antecedent set because they should contain at least one of the elements in the consequent set. The candidate rule combination algorithm is shown below where s_a is the antecedent set of a rule and s_c is the consequent set of a rule.

Algorithm 1: Combine Candidate Rules Algorithm

Input: A new candidate rule x for activity a,
a set C of existing candidate rules for activity a
Output: The rule r with greatest fitness value
that can be produced by combining x with C

```
1:  Create a rule r and initialize to null;
2:  for-each candidate rule c in set C do
3:      Create new rule t;
4:      Set t.s_a equal to x.s_a ∩ c.s_a ;
5:      Set t.s_c equal to x.s_c ∪ c.s_c ;
6:      if r equals null or Fitness(t) > Fitness(r)
7:      then r ← t;
8:  end for
9:  return r;
```

The fitness algorithm is responsible for evaluating the expected performance of a rule or candidate rule. In order to do so, we run the first level algorithm on the training dataset excluding the episode we used to derive the rule and count the number of events that do not belong to activity a but where assigned a label for activity a (false positives) and the number of events that belong to activity a but were not assigned a label for activity a (false negatives). We now execute a rule we have created for activity a on the same training dataset and repeat the label counting process. If the difference between invalid count prior to the execution and after the execution of the rule is positive, the rule is valid, and we return this difference as the fitness value of the rule. An algorithmic description follows.

Algorithm 2: Calculate Rule Fitness Algorithm

Input: A rule *r* for activity *a*, and a dataset *D*
Output: An integer value representing the fitness of the rule

```
1:    fitness = 0;
2:    for-each episode d in the data set collection D do
3:        invalidSum = sum of all false positives for
4:        activity a plus all false negatives for activity a;
5:        Execute(r) on episode d;
6:        nowInvalidSum = sum of all false positives for
7:        activity a plus all false negatives for activity a;
8:        fitness += invalidSum – nowInvalidSum ;
9:    end for
10:   return fitness;
```

Once the rule is created, it will be stored in the valid rule pool, however, before this is done, it will be compared against all other existing rules that have the same antecedent set s_a or whose antecedent set is a superset of the new rule's antecedent set. If there is any other rule that matches the criteria, the rules will be combined together. The algorithm for combining rules is presented below.

Algorithm 3: Combine Rules Algorithm

Input: A new rule *t* for activity *a*,
a collection of valid rules *R* for activity *a*
Output: An updated collection of valid rules *R*

```
1:  for-each rule r in the collection R do
2:  if r.s_a ⊇ t.s_a then
3:      t.s_c ← t.s_c ∪ r.s_c ;
4:      replace r with t in R;
5:      return R;
6:  else if r.s_a = t.s_a
7:      Create rule m ← mostFit(r, t) ;
8:      Create rule l ← leastFit(r, t) ;
9:      foreach event e ∈ l.s_c do
10:         Create rule c ← m ;
11:         c.s_c += e ;
12:         if Fitness(c) > Fitness(m)
13:         then m ← c;
14:     end for
15:     Set r ← m ;
16:     return R;
17: end for
18: add t to R;
19: return R;
```

In the event a new rule is a subset of an existing rule, we combine together their consequent sets and replace the existing rule with the new rule. The intuition is that wherever the existing rule would apply, so would the new rule, therefore, it is not necessary to have both rules. However, we need to keep all elements of the consequent set of the existing rule, so that the resulting rule does not re-label sequences that would have otherwise been left unmodified by the original rule.

Once we have a set of valid rules, we can apply them to the output of the first level algorithm to identify invalid sequences.

3.3.2 Applying Association Rules

In practice, it is often the case that a sequence Q_i is missing either the leading or trailing events due to incorrect labeling by the first-level algorithm. For this reason, a perfectly valid

sequence of events might resemble an invalid sequence and violate a rule r that we have determined must apply to sequences of a given activity a. To overcome this situation, we relax our association rule to include another set of events on the right hand side. We call this new set s_n, the *neighbor set*. The neighbor set is determined during the execution of a rule by obtaining preceding and following events with any activity label that are within a time t of the sequence being evaluated. An illustration follows:

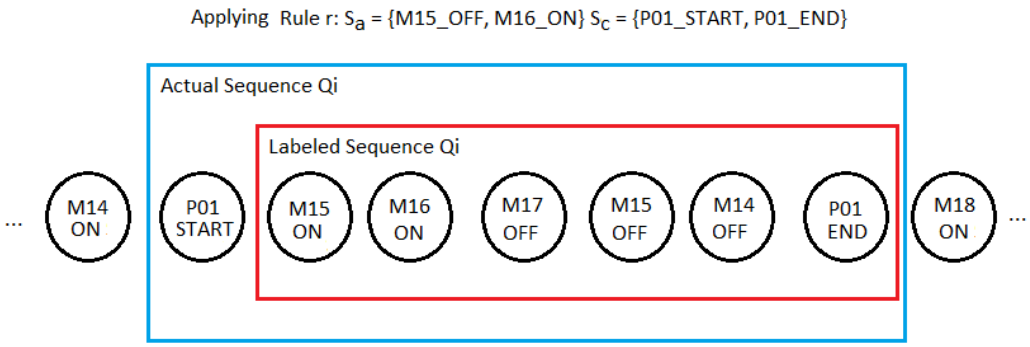

Figure 5 – Potential issues when applying a rule.

In Figure 5 we can observe part of an episode in which several sensor events have triggered. These events are represented by circles and contain in the middle a descriptive name that indicates the source and type of event that took place. These two pieces of information are generally used to create a corresponding unique event *id*. For example, sources M14-M18 represent motion sensors located in different parts of the assistive environment, and "On" and "Off" event types represent whether they have detected movement or whether they have stopped detecting movement. The sensor P01 represents the phone sensor, and the "START" and "END" identifiers refer to whether the phone was activated or deactivated. As illustrated in Figure 5, a sequence of events starting at M15_ON and ending in P01_END have been labeled with the same activity label and thus are a sequence Q_i. However, the first level algorithm failed to determine that the event P01_START also belongs to the same sequence and therefore should have the same label. If our rule where to specify that the consequent set S_c of the rule

for this activity requires the event P01_START, the sequence might accidentally be re-labeled even though it is valid for the given activity. To overcome this problem, we rely on the neighbor set S_c. In order to determine the neighbor set, we use a time argument t that is determined by the fitness function. The resulting rule is as follows:

$$(L(Q_i) = l_j \Lambda (\forall x \in s_a)[(\exists m)(x = E(Q_i, m)] \Rightarrow (\exists y \in s_c \cup s_n)[(\exists n)(y = E(Q_i, n)]$$

3.3.3 Mining Temporal Rules

Another form of rule explored is the one that imposes a restriction between the temporal precedence of a sequence Q_i with activity label l_a and an event e_j; we call this a *temporal rule*. It is of the form: $(L(Q_i) = l_a) \Rightarrow [\, T(e_j) > T(Q_i) + t)]$

Working with all the different types of events that exist in the dataset would require the mining process to be very complex and exhaustive. However, limiting the mining process to focus only on a small set of events can be accomplished. In order to determine the events to consider for the mining process, we rely on a threshold Tc to determine when an event is considered frequent in a given activity. Once we have identified a frequent event, we mine its time relationship with respect to the valid sequences and determine the value of t.

Once we have determined potential violations of our rules, we can use this information to further improve on the accuracy results. By determining sequences of events that violate rules and re-evaluating them as part of another activity's episode, we can improve recognition accuracy. However, this poses a challenge, as while detecting invalid combinations might be more intuitive, finding a valid activity label to substitute the current one is a non-trivial problem

3.3.4 Re-labeling Invalid Sequences

Once the first-level algorithm has estimated the labels for the events in the test dataset, we use the second-level algorithm to determine invalid sequences that violate the rules we have previously mined from the training dataset. At this moment, we can now proceed and re-label

these invalid sequences with a new label with the expectation that the new label assigned will be correct and thus that the overall accuracy of the final estimation will be greater. In order to determine this new label, we propose various strategies. The first one is to use the Forward-Backward algorithm to determine for each specific event e_j at time t that belongs to an invalid sequence Q_i, the most likely state that caused this event. Once we know the most likely state, we can infer the corresponding activity, and thus the activity label to assign to e_j. In order to determine the most likely state, we compute the Forward-Backward probability of the event e_j at time t for each state, and we pick the state with highest probability that does not correspond to the activity of the original invalid label. However, as noted before, the Forward-Backward algorithm can determine two states s_1 and s_2 for times t and t+1 such that the transition probabilities between these two states would make such a combination unlikely or impossible. In order to overcome this issue, we propose a different variation in which for an invalid sequence Q_i, we label all events in the sequence with the same activity label, which is the mode of the labels that were determined following our previously discussed strategy. Finally, our last variation is to choose an activity label only from those activities we believe precede or follow the invalid sequence. In order to determine what activity label is most suitable of the preceding or following activities, we pick the one whose state returned the highest probability for the given event e at time t by based on the Forward-Backward algorithm.

3.4 Experimental Setup

In order to evaluate our algorithms, we test them using data collected from volunteer participants performing activities in our assistive environment testbed. This testbed is a miniature apartment located inside the Heracleia laboratory; it includes a kitchen, a dining room, one bedroom, and one bathroom. The environment is equipped with motion, acceleration, temperature, pressure, switch, light, infrared, and item sensors; placed in a non-obtrusive manner. The layout of the apartment is shown in Figure 6.

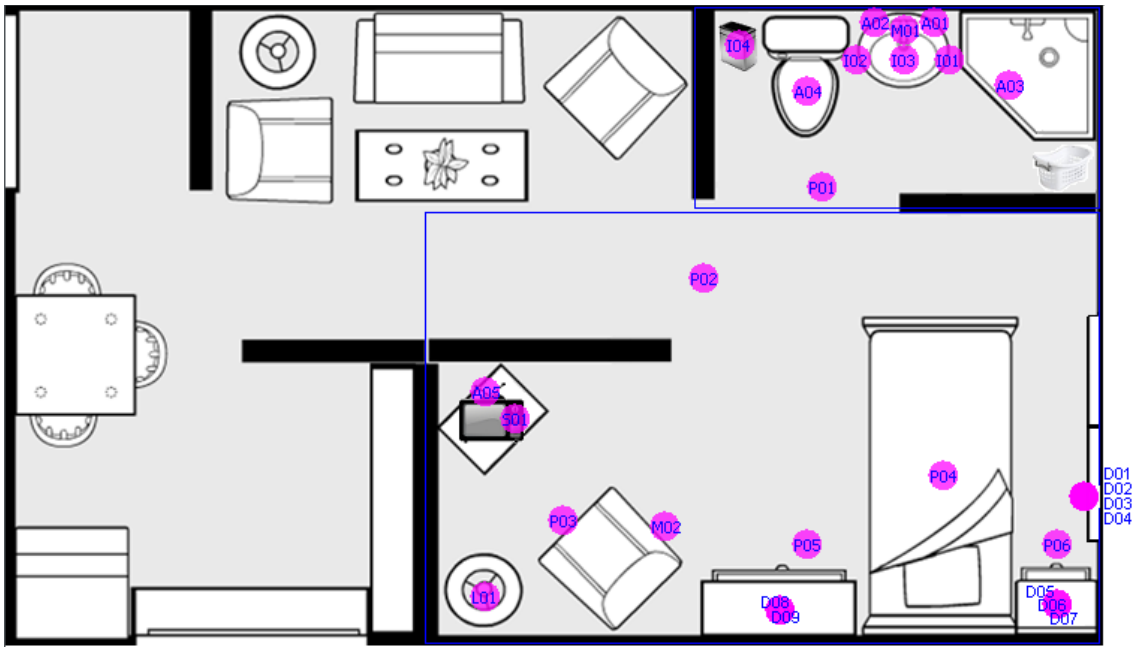

Figure 6 - Heracleia AtHome apartment testbed (@Home).

Most of the sensors provide analog readings; we use different algorithms to convert the raw data into events (i.e. On/Off). The sensor key legend is shown in Table 3.

Table 3 - Sensor key legend.

Abbreviation	Sensor Type
M	Motion Sensor (On/Off)
I	Infrared Sensor (On/Off)
P	Pressure Sensor (On/Off)
L	Light Sensor (Sunspot board- Analog)
A	Acceleration (Sunspot board)
T	Temperature (Sunspot board - Analog)
S	Switch (Sunspot board - On/Off)

We use two main types of hardware to handle the sensors in our apartment. The first type is SunSpot devices [14], which are wireless miniature computers with an embedded sensor board. The second type is Phidgets interface kits [29], which are embedded boards to which a wide variety of sensors can be attached. In order to operate with the Sunspots and the Phidgets interface kits, we rely on two different software applications that have been developed at the Heracleia laboratory. The first program is the *ApartmentBase* [30] data collection tool; the second program is the *Phidgets sensor tool*. Additional information regarding the Phidgets sensor tool can be found in Appendix B.

Additionally, we have placed cameras within the apartment for validation purposes. The cameras are not used for activity recognition but for validating the data set annotations. Furthermore, participants are instructed to say out loud whenever they are starting a new activity, this audio/video is recorded by the cameras to help data set annotations be as accurate as possible.

3.4.1 Participant Activities

Twelve volunteers were instructed to perform eight different activities all belonging to either the category of ADLs or instrumental ADLs. They were provided with a description of the activities and the layout of the apartment. Their task was to execute all eight activities throughout a combination of three different scenarios; morning, evening, and night. Participants were allowed to repeat activities so long as these were not within the same scenario. This was done in order to be able to differentiate interleaving from cases where a participant performed again an activity that was previously completed. In order to guarantee a certain level of complexity, participants were requested to perform some interleaving and concurrency throughout the execution of the activities. The activities performed are described below.

1. *Taking medication:* Participant opens a drawer containing medication and consumes a pill with or without water. He can obtain the water from either the bathroom or the kitchen.

2. *Toileting:* Here the participant performs hygiene and toileting activities, such as washing hands, brushing teeth, grooming, or taking a shower.

3. *Answering phone*: Participant responds to a phone call or initiates one.

4. *Sleeping*: Here the participant rests on the bed located in the bedroom.

5. *Watching TV*: Participant turns on the TV switch, interacts with the TV and remote, and turns off the TV switch at the end.

6. *Reading books*: Here the participant opens any of the book drawers, extracts books, and proceeds to read them in any desired location. At the end, the participant must return the books to their original place.

7. *Choosing outfit*: Participant changes clothes by opening the clothes drawer. At the end, the participant must close the drawer

8. *Cleaning*: Participant uses a broomstick to clean the bedroom and/or toilet areas.

The results were recorded through video cameras for annotation and validation purposes. Volunteers were requested to mention throughout the course of a scenario whenever they were starting or resuming an activity. A researcher was in charge of labeling all events collected based on the video and user feedback. A sample data set is shown in Figure 7.

Event date	Event time	Event source	Event type	Event value	Activity label
2009-10-21	12:29:00.000	B04	ON	1	4
2009-10-21	12:29:00.000	B05	ON	1	4
2009-10-21	12:29:29.000	B01	OFF	1	4
2009-10-21	12:29:30.000	B02	OFF	1	4
2009-10-21	12:29:30.000	B04	OFF	1	4
2009-10-21	12:29:30.000	B05	OFF	1	4
2009-10-21	12:29:30.370	M02	ON	712	7
2009-10-21	12:29:31.025	B03	OFF	0	7
2009-10-21	12:29:31.025	P03	ON	312	7
2009-10-21	12:29:33.591	D09	Drawer	0.31	7
2009-10-21	12:29:34.067	P03	OFF	312	7
2009-10-21	12:29:34.110	D09	Drawer	0.20	7
2009-10-21	12:29:34.471	D08	Drawer	0.07	7

Figure 7 – Annotated sensor events example.

3.5 Experimental Results

In order to validate our algorithms, we tested their accuracy results on two different datasets. The first dataset was collected in the Heracleia apartment testbed where a total of 10 participants performed three different scenarios, on average each 12 minutes long. The second dataset belongs to the CASAS Smart Apartment testbed [15]. This data set consists of twenty trace files (episodes), each for one volunteer conducting a series of eight different activities. A full description of their activities, the average times taken by participants to complete the activities, and the average number of sensors triggered per activity are available in [15].

First, we evaluated which of the two algorithms Viterbi or Forward-Backward would produce a more accurate labeling for the Hidden Markov Model on the two datasets. Figure 8 illustrates the results. The Forward-Backward algorithm relies on two different probability values; these are called alpha and beta. In order to have beta probabilities, we assigned non-zero probabilities to all states. When a state had a zero probability observation, we assigned it a

small delta value δ = 0.0001. For evaluating the Viterbi algorithm on the dataset, we did not modify observation probabilities to be non-zero.

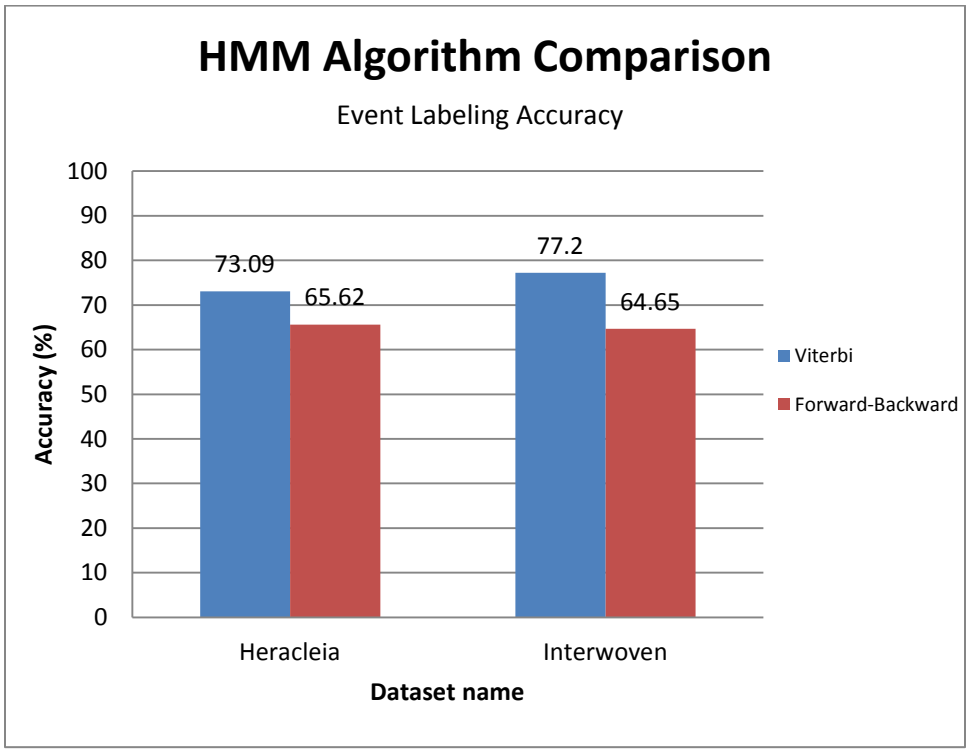

Figure 8 – Accuracy comparison for two HMM related algorithms.

Our findings suggest that the Viterbi algorithm outperforms the Forward-Backward algorithm in finding the sequence of states that best fits the observed events. We believe this is the case because the Viterbi algorithm also captures the likelihood of transitions amongst states. In any event, findings suggest that using the Forward-Backward algorithm for re-labeling events is a possibly good strategy.

We evaluated the performance of the single state HMM algorithm, the combination of HMM with association rules (HMM + AR) and the combination of HMM with temporal rules

(HMM + TR) on the CASAS “Interwoven” data set. For our charts, we measure the percentage of labels correctly labeled for a given activity vs. the total number of labels assigned for the given activity. For overall accuracy, we use the event-accuracy metric. Figure 8 shows the results we obtained and Table 4 has the overall accuracy for each algorithm.

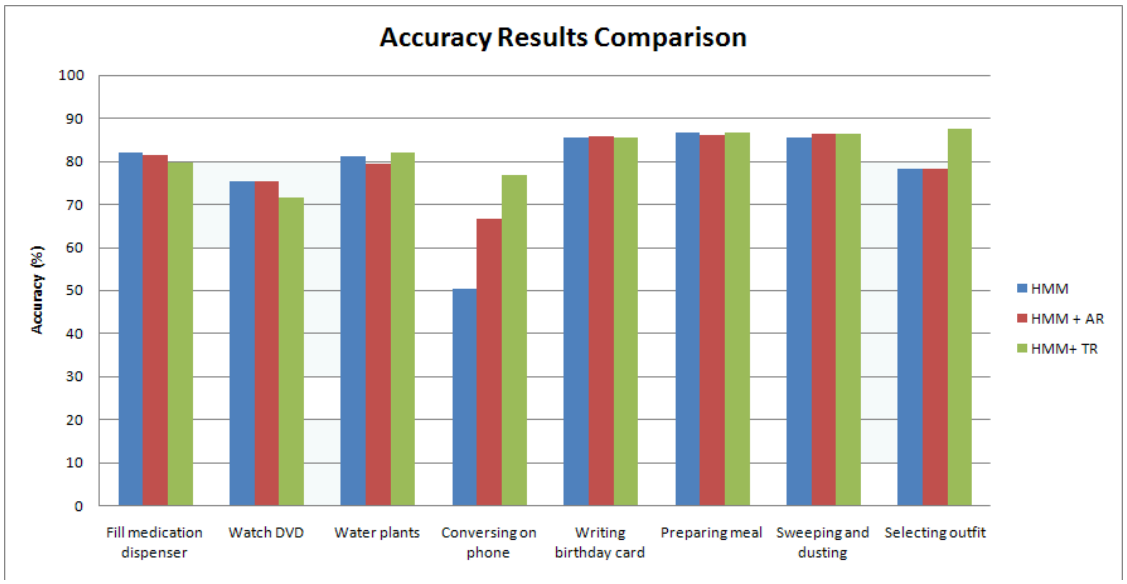

Figure 9 – Accuracy results on “Interwoven” dataset.

In the chart above we can see the average recognition results obtained testing the “Interwoven” dataset using leave-one-out cross validation. The activity names are listed in the x-axis and the individual activity accuracy is listed on the y-axis. The activities that improved most when using the two-level framework approach are “Conversing on Phone” and “Selecting Outfit”.

Table 4 - Overall accuracy on “Interwoven” dataset.

Algorithm	Overall Accuracy	σ
HMM	77.2%.	7.93
HMM + AR	78.2%	7.36
HMM + TR	80.4%	6.21

From the report produced by our *Assistive Room Activity Analyzer* tool, described in Appendix A, we can observe that rules have been generated, however, only rules for activity ‘Conversing phone’ have survived the strict criteria of fitness, and have been applied. A total of 11 rules were created, however only 7 were applied, these are listed in Table 5. The association rules applied are exclusively for sequences labeled as ‘Conversing phone’, whose activity label is the value 4. A list of the valid rules generated is presented below, where events are represented with a name consisting of the source and type of event, which are the two types of identifiers that are used to form the event’s unique id.

Table 5 - Files modified by rules in “Interwoven” dataset.

File name	**Activity**	**Rules applied**	
	l_j	s_a	s_c
p24	4	{M13_ON}	{P01_END,P01_START}
p28	4	{M13_ON}	{P01_END}
p29	4	{M13_ON}	{P01_END}
p32	4	{M15_OFF} {M13_ON}	{P01_END,P01_START} {P01_END}
p34	4	{M13_ON} {M14_ON}	{P01_END,P01_START} {P01_END,P01_START}

In Table 6, we display the events that were re-labeled by the HMM + TR algorithm on the “Interwoven” dataset. The left column lists the name of the episode (trace file) modified.

Table 6 – “Interwoven” dataset relabeling results.

Data set	Events with invalid label	Events tagged invalid	Events relabeled correctly
p04	10	10	8
p13	55	55	44
p14	96	96	52
p15	64	68	46
p17	0	19	0
p18	18	18	2
p19	6	6	6
p20	36	36	0
p23	23	27	20
p24	66	66	41
p25	27	29	2
p26	0	0	0
p27	43	46	37
p28	16	16	15
p29	5	5	5
p30	6	6	6
p31	39	61	30
p32	17	22	17
p33	76	76	7
p34	46	46	1

We conducted a one tailed paired t-test to compare the performance of the single-state per activity HMM and the HMM + TR algorithm. We obtained a p value < 0.002. We can observe from the results that temporal rules performed significantly better than the association rules. Also, they were able to generate temporal rules for multiple activities that violated temporal constraints. We have a 3.5% improvement with temporal rules vs. a 1% improvement with association rules.

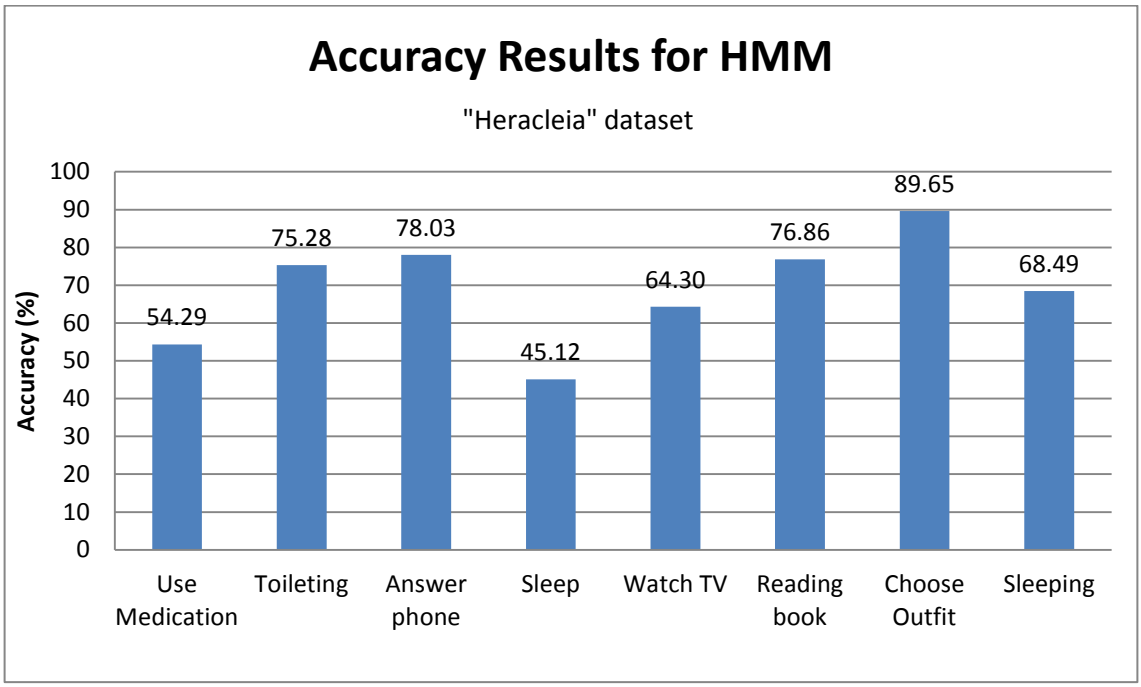

Figure 10 – Accuracy results using single-state HMM for “Heracleia” data set.

When experimenting with the Heracleia dataset, the use of neither association generated very few rules and an overall improvement smaller than 1%. No temporal rules were generated. Our conjecture for the small number of association rules generated is that the data set involved too few sensors that were common to many activities. As for the lack of temporal rules, we believe this to be because of two reasons. The first one is that the duration of activities did not reflect the natural pattern of execution of an activity, as participants were requested to keep a similar duration between activities. The second reason is that unlike in the “Interwoven” dataset, participants performed activities multiple times per episode, thus, the data has greater amount of temporal variation for one single participant’s execution of each activity. Thus, we believe that the algorithms were not capable of exploiting association or temporal relationships as effectively as in the case of the “Interwoven” dataset.

CHAPTER 4

QoS IN ONTOLOGY-CENTERED MIDDLEWARE

The Semantic Web [20] and its technologies have widened research and developments in many different fields. Context-aware computing has found in ontologies [22] a well-defined structure for knowledge representation and reasoning. Numerous architectures have been proposed [24,25,26,27] to enable context-aware applications in smart environments, many of which use semantic web technologies at their core in order to provide knowledge storage, reasoning, and dissemination. These middlewares have been designed to address issues identified as important by their authors for creating a functional architecture to enable context-aware applications. However, they do not dwell much on issues of efficiency and performance. In this chapter, we propose an ontology-centered architecture focused on the idea of Quality of Service (QoS) and performance. We believe this feature to be critical in order to enable the operation and reliability of context-aware applications in assistive environments, where emergent situations arise and quick application responses are critical.

QoS has been extensively studied in application domains such as Networking and Multimedia [32,33,34]. In those domains, QoS has been formalized by QoS specification languages [35] and specific solutions have been proposed. However, although such solutions can be taken into consideration when designing QoS enabled assistive environments, they cannot be directly applied due to the more complex nature of context aware applications which rely on heterogeneous devices and data types. For that reason, in the next sections we propose a framework for the design of middleware that will provide QoS by controlling the behavior of the client applications that the system supports using a protocol for in-advance agreements which are achieved at the time each client requests service from the system.

4.1 Ontology Centered Middleware

Ontology centered middleware focuses on providing a unified knowledge and data model. Several architectures that follow this principle have been proposed [24,25,26,27]. They attempt to simplify application development by providing a useful set of APIs and a robust framework for knowledge sharing and derivation. A general architecture design is illustrated in Figure 11.

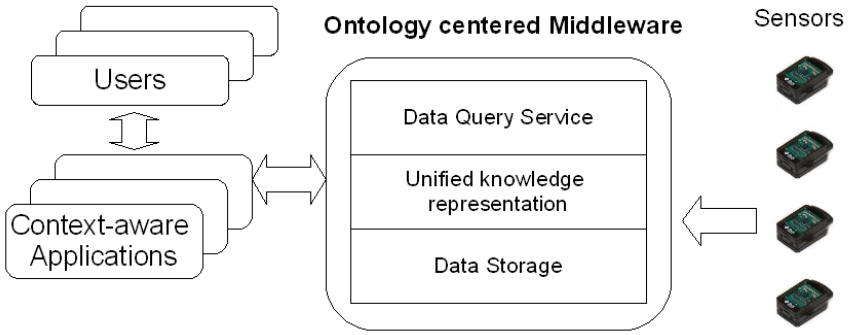

Figure 11 – An example ontology-centered middleware architecture.

Following, we define a few key terms related to ontologies [37] that will be used throughout our work.

- A *class* defines a group of individuals that belong together because they share some properties.
- A *property* is a binary relation that specifies class characteristics.
- An *individual* is an instance of a class; it has a unique identifier and actual property values.
- *Cardinality* is a restriction on the number of values an instance of a class can have for a given property.

- An *ontology* is a set of axioms which place constraints on a set of classes, their properties, instances, and operations, and the types of relationships permitted between them.
- *Inference* is the process of using ontology axioms to derive additional information based on the available data.

In these works [24,25,26,27], authors have identified the following major components that a comprehensive middleware should cover:

(i) Sensor discovery and sensor data collection. Any context-aware architecture will operate with sensors transmitting data. The protocols used for sensor discovery and data collection will play an important role in the overall flexibility and adaptation that the system offers to integrate and work with existing hardware.

(ii) Inter-operable model for creating, accessing, and storing ontological data. A common model that can be shared by all software applications and middleware services. This is accomplished through the use of ontologies, which help define concepts and their corresponding properties. In [27], Sensor Wrappers are proposed as a mechanism to convert raw data into ontology data while separating sensor hardware specifics from applications. Frameworks such as Jena [36] are used to store both the ontologies, and data, generally in Semantic Web standard formats such as OWL [37] and RDF [23] triples.

(iii) Ontological inference. The use of ontological reasoning through the language constructs available, combined with rule-based reasoning allow for a flexible mechanism to derive knowledge. Some widely used inference engines include Jena [36], Pellet [38], and Racer [39].

(iv) Ontological data access and dissemination. Data routing, synchronization, and dissemination algorithms differ significantly depending on whether the architecture is centralized, distributed, or P2P. The query language of preference has become SPARQL [40] since its adoption as a W3C standard. Queries are represented as a series of RDF triples *<Subject, Predicate, Object>* where some triple values are left as query variables.

(v) Efficiency. Many strategies are proposed. Efficient persistent data storage in schema-aware [41] databases is one of the proposals, where a table is created for every single *<Subject, Predicate>* combination. This increases performance time when accessing information that is maintained in a database. To reduce inference time, strategies focus on minimizing the amount of on-demand reasoning that needs to be carried out and establish mechanisms to take advantage of off-line reasoning [31]. Other strategies proposed include data subscription by applications and maintaining the data store as minimal as possible [25].

The previous architectures, however, do not propose methods to allow for QoS and the required resource management techniques to accomplish such. Given that inference is a major deterrent in performance and limits the viability of such infrastructure for time-sensitive applications, we model inference execution time as a maximization problem and provide a framework to enable solving the underlying problems in inference time. We look into the required aspects [42] to allow for QoS in such architectures. Next, we illustrate examples of time-sensitive applications and the problems they face in the traditional ontology centered architectures that have been proposed so far.

4.2 QoS in Practice

Following we describe an example of a context-aware scenario in which applications require QoS in order to function properly. Our intentions are to highlight some of the existing problems with context-aware applications, and later address these in our proposed QoS-aware architecture.

4.2.1 Example problem that requires QoS

We consider the scenario of a smart building used by a company and equipped with sensors that are capable of detecting the employee's location and current status. Furthermore, the building has an ontology centered middleware architecture in place responsible for collecting, transforming and distributing data to satisfy application queries, like portrayed in Figure 11. We now consider a series of context-aware applications running in this environment, each of which have different QoS requirements and work by requesting data from the middleware. The middleware is in charge of managing and distributing the data in appropriate formats such that the advantages of a unified representation for concepts and data are maintained.

The first application is an activity reminder; it attempts to periodically update the employee with possible activities he wants to carry out. The activities that the employee wants to do are initially programmed into his PDA at a given frequency, for example every day, then the PDA application will query the middleware to obtain the person's current and past location as well as status information to determine how best to organize and execute his day to day activities. The reason why this application is time-sensitive is because as you carry out your daily activities, you will visit locations and might deviate from the proposed routes, in those cases, the application needs to recalculate new routes or schedules that might meet your interests. The application must execute in a short period of time, because if it takes time the user is not likely to wait, and instead will think by himself what activity to carry out next and

ignore the recommendation. For this reason, we think a short response time, like for example less than ten seconds, might be acceptable; anything longer might prompt the user to put down his PDA and plan his own schedule.

The second application is an energy efficiency adjustment program; it retrieves current location information of all people in the building as well as past location information to establish what regions should reduce their energy consumption in the building. As the current location information for the employees changes and deviates from the predicted model computed from past location information, the energy efficiency program needs to adjust itself. This application is also real-time, as updates in the application's behavior should happen in very short periods of time so as to avoid energy settings from disrupting users in their day to day activities. We consider for example, that this application's response time might need to be around 2 seconds, so that it can adjust energy settings in different regions fast enough so as to not trigger a negative reaction from employees, or force them to have to do manual adjustments to energy related devices.

A possible representation of the concepts involved in these example applications is represented bellow. We distinguish four different ontologies, these are: *ActivityEvent*, *LocationEvent*, *TimeEvent*, and *Employee*.

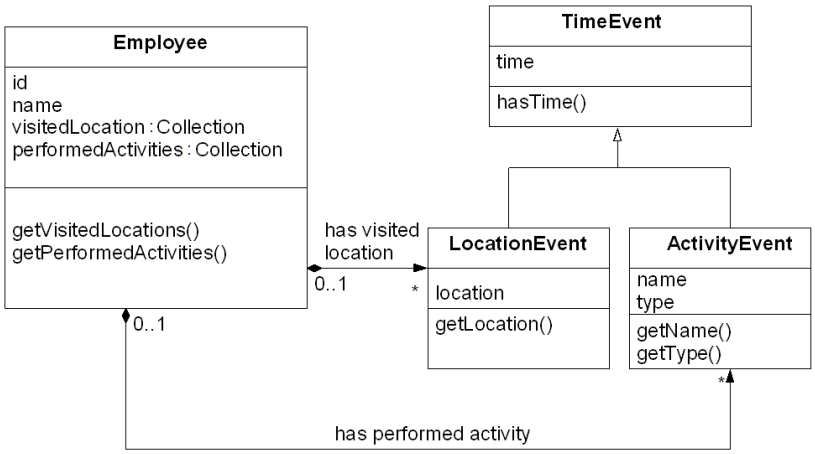

Figure 12 – Sample UML diagram displaying application ontologies.

The idea behind these ontologies is that both applications share the same knowledge model, they submit queries requesting the same type of ontology data.

4.2.2 Existing QoS related problems

We distinguish two main problems in the applications defined in the previous section when used in the currently existing ontology centered middleware architectures. The first one is that the existing architectures have no information regarding the access patterns and requirements of the different applications; therefore they are unable to provide Quality of Service. We believe the solution to this problem is to introduce a QoS negotiation mechanism between client application and middleware where the client can agree to certain access patterns, and the middleware can agree to carry out multi-resource reservations to guarantee query execution times. On the other hand, we want to introduce as minimal a complexity as possible to the design of client applications.

The second problem, and the main reason why existing work does not dwell in QoS is that ontological inference is a computationally expensive task, generally considered intractable, and in order to be able to guarantee execution times, it is necessary to impose limitations on the

amount of data being processed. For example, in Figure 12, both applications will request for *LocationEvent* and *ActivityEvent* data. The size of the retrieved data by the client applications varies depending on the nature of the data, the amount of data that has been collected so far by the sensors, and the requirements of the client application. The solution is indeed to limit the size of the data set, and we propose a strategy that allows for great flexibility with the expense of introducing the client into the process of sensor data transformation into ontological data.

There is a conflict of interest by different applications running on the middleware as to their expectations on how data should exist in the system to provide their QoS requirements. For example, the previously discussed applications will require the *ActivityEvent* and *LocationEvent* data set to be of different sizes, matching their desired response times. One possible solution to this problem is to introduce the client in the process of transforming raw sensor data into ontological data. By allowing the client to participate in the decisions on how data should be created, updated, and deleted at the initial point of service negotiation between client and server, this issue can be addressed.

Because these two context-aware, time-sensitive applications have the same interests for data and different QoS requirements, they are relevant examples to our proposed work and will help illustrate better the solutions proposed. Both, deriving activities and location relationships require the use of an inference engine, and thus both suffer from the time complexities of doing inference on large data sets.

There are many factors that influence the end-to-end delay in either centralized or distributed service oriented architectures, generally these are the resources involved in providing the services, like network bandwidth, memory, CPU, I/O, and storage space. These have been studied in depth by the community as computers and their purposes have evolved. Different heuristic algorithms have been proposed for issues such as multi-processor real-time task scheduling, resource reservation, and multimedia distribution. However, problems arising from the calculations performed in ontology centered middleware architectures have received

very little attention, and existing architectures such as Construct [24], SOCAM [25], COBRA [26], SCS [27] have focused their attention on non-time critical applications and demonstrating the flexibility and value of ontology centered architectures, showing little attention for time related issues. The development and execution of the proposed two applications would lack QoS support in any of the above mentioned architectures, and thus would not operate as desired. For this reason, we present a different perspective of an ontology centered middleware architecture, where it is possible for time-sensitive applications to function.

We believe there is significant work on resource management, thus, our focus will be to study and understand the complexities that arise from inference and its computational requirements and how they could be addressed in ontology centered middleware architectures.

4.3 Methodology and Architecture

In order to allow an ontology centered middleware architecture to provide QoS support to context-aware applications, we need to provide an infrastructure to allow applications to describe their structure and query patterns; this is generally referred to as QoS specification. In Section 4.3.2 we explore our QoS specification format in detail. We characterize an application as ultimately consisting of queries, which have end-to-end delay requirements. There are many factors that influence the end-to-end delay of an application's queries. Most of them can be handled through heuristics and multi-resource reservation, such as those to manage network bandwidth, memory usage, task scheduling, and I/O. However, ontology centered middleware requires the use of an inference engine, where it is not possible to determine the inference time unless the size of the data set used is known. In order to know how much data will be used by a query, it is necessary to establish restrictions on how data is generated for the ontologies, and their corresponding properties. Each property in an ontology can have a restriction on how many data entries can be associated with that property. We call this the cardinality of a property. This posses a conflict of interest, as the data restrictions that are necessary for one

context aware application might not be suitable for another context-aware application. A possible solution would be to have both applications rely on a different set of ontologies with different cardinality constraints for their properties, but that would defeat the purpose of an ontology centered middleware architecture, whose greatest value is a unified model for knowledge and data representation. To solve this problem we propose a trade-off where we relax the unity of the data in order to allow some level of QoS support. This is accomplished by using Data Transformers, which are further described in Section 4.3.3. This component allows the client to have some level of participation on the process of converting raw data into ontological data. This is done in order for different applications to be able to modify the same sensor data and produce different data while storing it using the same knowledge representation. While this might seem counter-intuitive, the goal is to make a fine-grained distinction in the different ontological properties used by the applications, where a single property can be treated as a set of different *<property, cardinality>* couples by the middleware's inference engine. The challenge is to make this process completely transparent to the client application. We elaborate more on this idea in the following sections. The basic flow of our proposed middleware would be as follows.

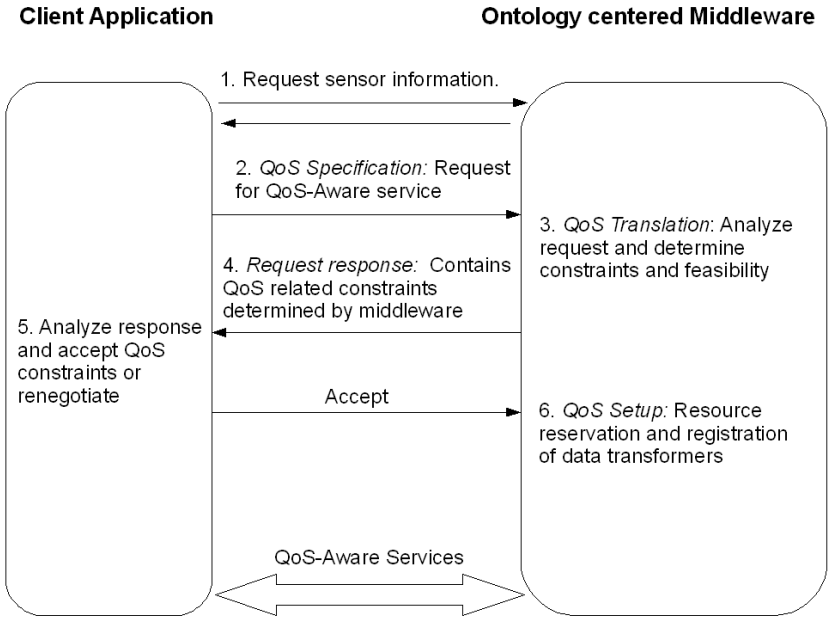

Figure 13 – A proposed QoS negotiation mechanism.

1. The client application requests sensor information to the middleware. The sensor information describes what sensors are available to the middleware and also provide specifics about the sensors, such as for example, unit representation, accuracy, and frequency. The client application needs to determine if it understands the data that the sensor is producing and if it has in its library of “drivers”, we call this a Data Transformer, one that will be able to convert the sensor's raw data into RDF. Data Transformers share some common functionality with the Sensor Wrappers introduced in [27], but they go beyond such functionality, their details are described in Section 4.3.2

2. Using its own QoS requirements, the client application builds a QoS specification file that includes its query access pattern information, details are described in Section 4.3.1. Additionally, the specification file includes a list of “Data Transformers” it will need the server to install to convert the data appropriately. We imagine the Data Transformers to be written in a

format that restricts their access in the server to read appropriate sensor data and to write only to the data store, so as to avoid security issues.

3. The server will now perform QoS translation, to analyze and determine if the client application requirements can be met. In the case of distributed systems, this will involve coordinating with other servers and reserving resources such as CPU time in the task scheduler, and other resources such as network bandwidth or disk space. The server will also determine further QoS constraints that the client is not able to determine initially, since it does not know the current middleware's utilization and resource availability. The major factor that it will determine is what cardinality values are acceptable for the different ontologies used by the client application, and will govern the creating/update/deletion of ontological data.

4. The server will provide a response either accepting or rejecting the client's request and informing him of further constraints he must follow. These constraints are restrictions in the cardinality of collections in properties of ontologies. These are not a natural restriction, but rather are imposed by the server.

5. The client will now determine if the cardinality constraints imposed are acceptable or not, it can then accept the service or attempt to renegotiate with the server, say by introducing different QoS requirements.

6. Once the client has accepted, the middleware will go ahead and perform the required resource reservations, in terms of allocating and reserving CPU time in its task scheduler, and other reservations such as bandwidth, memory, and disk space.

4.3.1 Data Use Maximization

At this point we consider inference execution time as a maximization problem where we are given application requirements and the set of ontology properties used by each of its queries and we attempt to determine the cardinality constraints that each ontological property should have in order to satisfy the QoS. In our approach we assume that we have a set of predefined cardinality values for property. This is done so that applications do indeed reuse the same type of data, by using an existing property with established cardinality.

In our system each client application submits queries to the middleware and expects to get a reply within a specified time constraint. Also each query response must have been calculated using a certain minimum amount of data, in order to be considered a valid response. The minimum amount of data and time constraints are agreed during the client-middleware agreement negotiation. Because we are using ontologies, we define the minimum amount of data in terms of the ontology properties that will be used by the query. The way we define the minimum amount of data is through the properties/predicates involved in the query. Ontologies consist of different types of properties, and when we query for them, we call them predicates. Therefore, each query consists of a number of predicates. The delay of the system's response depends on the cardinality of the predicates to be processed, as they affect the amount of data. We represent the amount of data as the variable x, and we use the constant value c to represent the type of data for the property. There are generally many types of properties, for the same cardinality value, the processing of different property types will take different amounts of time. We can define the problem of providing QoS by using maximum system resource utilization as a maximization problem as follows.

For a given class c and a given property p, let $X = \{x_1, x_2, \dots, x_n\}$ be a set of cardinality values where x_i is a cardinality value assigned for the class property p to client i. And let $V = \{v_1, v_2, \dots, v_k\}$ be a set of discrete predefined allowed cardinality values where we enforce

the constraint that $(\forall x \in X)[x \in V]$. Let $C = \{c_1, c_2, \ldots, c_n\}$ be a set of constants where c_i is a constant describing the amount of resources used by the middleware to serve client *i* using one unit of data. The value of c_i is related to each <*client, query, predicate*> triplet. Finally, let $T = \{t_1, t_2, \ldots, t_n\}$ be a set of time constraints where t_i is the time constraint specified by the client *i*. Then:

Maximize: $c_1x_1+c_2x_2+\ldots+c_nx_n$

Subject to: $time(c_i,x_i) \leq t_i$

Although we define the cardinality variables as variables that can take discrete values v_i, and thus our problem appears to be an instance of Integer Programming, it is not intractable because the number k of different possible values is finite and relatively small.

Each client has been dedicated a specific amount of resources depending only on the total number of clients that are being served. Therefore the amount of time that the system needs to respond to the client for each specific combination of queries and predicates is affected only by the cardinality of the amount of data that the client will be using.

The goal is to provide as better service as possible to the clients as long as a minimum QoS is guaranteed for all the clients. In cases where the number of clients is small and their requirements for a minimum QoS is smaller than what the system can provide the system can optionally offer an even better service by allowing them to access more data. When new clients are added then the system recalculates the amounts (cardinalities) of the data that can be accessed by each client up to a point where a minimum amount of data (according to the initial agreement) can be accessed by each client and the time constraints are met. After that point the middleware system rejects any requests for new client subscriptions.

4.3.2 QoS Specification – An Xml Representation

In our architecture, client applications are required to send QoS specification files when establishing an agreement for service. These specification files consist of varied types of information, that help the middleware calculate the application's requirements and carry out the necessary multi-resource reservations. We define the following concepts that will be used in the specification file.

- A context-aware application consists of a series of tasks.
- Each task consists of a number of states and the corresponding transitions.
- In each state, a specific set of queries will be executed.
- Transition between a task's states happens when a series of query result values are satisfied.
- Each query has its own QoS requirements.
- A state's QoS requirements are satisfied when the QoS requirements of all queries taking place in that state are satisfied.
- A task's QoS requirements are satisfied when the transition between the states that this task involves happens within a predefined amount of time, i.e. the QoS of each state is satisfied.
- An application's QoS requirements are satisfied when all the application's states' are met.

Thus, each application is described as consisting of Task nodes. In our examples, these could map to for example an activity reminder or a route organizer. Each task has its own independent behavior and QoS constraints. Task behavior is described by a collection of states, and each State consists of a series of queries with their QoS constraints, (i.e. end-to-end delay).

Typical application behavior and query pattern change depending on its internal state. Our interest is to capture and provide this information to the middleware.

Generally speaking it is very hard for an application developer to know exactly how big or small should the cardinalities be. Initially, property cardinalities will be set according to some predefined values, and then, based on statistical observations on the data and usage patterns, these values will be adapted to optimize performance.

Figure 14 illustrates how the QoS information is specified in an XML file. We can observe the hierarchy relationships and we can distinguish QoS requirements at the query level

```
<ApplicationRequirements>
<Task>
  <State num="1">
    <Query>
     <SPARQL><![CDATA[SELECT ?x ?y
          WHERE ?x <http://heracleia.uta.edu/
          Employee:hasLocationEvents> ?y]]/>
     </SPARQL>
     <QOS>
       <ResponseTime unit="s">5</ResponseTime>
          <CardinalityConstraint ontology="Employee"
          property = "hasLocationEvents">500</Cardinality
        </QOS>
    </Query>
    <Query>
      <SPARQL><![CDATA[SELECT ?x ?y
          WHERE ?x <http://heracleia.uta.edu/
          Employee:hasActivityEvents> ?y]]/>
      </SPARQL>
      <QOS>
            <ResponseTime unit="s">2</ResponseTime>
          <CardinalityConstraint ontology="Employee"
          property = "hasActivityEvents">250</Cardinality
      </QOS>
    </Query>
  </State>
  <State num="2">
  </State>
</Task>
<Task>
  ...
</Task>
</ApplicationRequirements>
```

Figure 14 – An example XML QoS specification file.

However, in order for a client application to be able to operate effectively with a given cardinality constraint, it needs to be able to determine how sensor data is mapped onto ontological data. For this purpose, we present the concept of the Data Transformer.

4.3.3 The Data Transformer Architecture

Proposed research on ontology centered middleware focuses on many issues, but puts little stress on how data generation affects performance. The assumption is generally that all applications share a big set of data that has been gathered by sensors, and then transforms this data and distributes it to the client application when needed. Our approach deviates tangentially from such architectures, we consider that data generation is the key problem to reasoning performance and thus we propose a completely different mechanism on how data should be generated. In order to do this we introduce in the QoS specification file a complex node called the “Data Transformer” whose goal is to control the creation, update, and deletion of statements from the data store. The data transformers are provided by the client and act as “drivers”. Once the client application has queried the middleware for sensor types, it identifies from the list of data transformers if it knows how to manipulate the sensors available so as to generate the desired ontology statements. If this is possible, it will then send these Data Transformer nodes in its QoS Specification, else, application developers are responsible for obtaining and possibly developing these “drivers” from sensor descriptor files. The Data Transformer node consists of the following elements; first, it contains a list of all the ontologies that will be used to map raw data to statements. For each ontology, it will retrieve from the middleware the established cardinality constraints and use these to determine how it should create/update/delete ontological data.

We present a mechanism by which for example, the two previously discussed applications could have different cardinalities for *LocationEvent* data, one requiring there to be at least 500 instances, and another requiring at least 5000 instances, where instance here

refers to when an object is created out of an ontology and corresponds to data entries. Clearly, if both applications where to share the same data, this would present a conflict, since the data does not map as well for both applications and will create processing delays that might be acceptable by one application and not by the other one. The purpose of the data transformer is to be able to isolate both uses of the same property *LocationEvent*, in such a way that we do not need to make distinctions, so as to be able for both to use the same ontologies, yet at the same time to have both applications use a different data set for this property. We accomplish this by using a schema-aware database in which tables are generated for every RDF *<Subject, Predicate>* combination. Additionally, different tables are created for the same *<Subject, Predicate>* combination when this predicate has multiple cardinalities.

First, the Data Transformer is registered for a given type of sensor data. Upon collecting such data, the middleware runs the data transformer. The transformer is thus responsible for converting the raw data into statements. Then, at the point where persistent storage is going to be carried out, the data transformer uses its configuration to select the appropriate table under which the data will be stored. For example, the first application could use for its inference the table <*Employee, hasLocationEnvet#500*> while the second application will store it in a table called <*Employee, hasLocationEvent#5000*>.

The advantages of having two different data transformers on the same type of data is that both of them have full access and rights on the creation/deletion/update of the data without having to overlap or cause conflicts. The disadvantage is that the overall amount of data we store is much greater and that we sacrifice some unity in the way data is stored for a given ontology. This is required though, as it is not possible to expect all applications to have same purposes for the same data. Nonetheless, any other approach in which there would be an interest in distinguishing the data between these two applications would eventually require more statements and therefore more data. So we believe this is a trade off that will always be necessary.

CHAPTER 5

CONCLUSION

In this thesis, we have examined two problems related to developing context aware applications for assistive environments. The first one is that of inferring the context in the environment to allow for proper actions to take place in emergent situations. This problem of activity recognition has been widely explored in the past. However, not much work has approached the problem from the same perspective as we have. While experimental results show small improvements in activity recognition accuracy, we believe that the two-level framework provides a potential unlike other approaches in that it can be used to leverage algorithm complexity in the lower layers of inference. We also believe that with the expansion of rules and their structure it will be possible to achieve further accuracy improvements and that the result will encourage the use of such a type of framework.

With regards to the problem of providing QoS in smart environments which are built using ontology-centered middleware architectures; we believe that the best way to ensure QoS in such environments is to enforce a mechanism to be able to control the amount of data that is being distributed to client applications. We believe providing differentiated services, where a client application can participate in the decision of how data is to be generated and can negotiate its requirements in terms of QoS is the key to enabling prediction of system's behavior and response times. For that reason we propose a framework that uses a negotiation mechanism between the client applications and the middleware system which ensures that all applications that have been granted service will have at least a minimum QoS. Furthermore,

when the minimum QoS for each application has been met but the system has not consumed the total amount of available resources we suggest a method to distribute those resources to the client applications in a way that maximizes the system's utilization and provides an improved service to the clients. Finally, we give an example for the formulation of the negotiation method between the client applications and the middleware system and we explain how the client applications can participate in the data transformation from the initial raw formats to ontological representations in order to improve efficiency. Preliminary experimental results on a prototype [44] of the architecture proposed suggest the viability of such a design.

We believe that this research lays the groundwork for two new approaches to enabling context-aware computing. We hope that the research community finds insights in the work we have conducted and that the collaboration and further investigation can bring us closer to making context-aware assistive environments a viable and robust technology.

APPENDIX A

ASSISTIVE ROOM ACTIVITY ANALYZER TOOL

Assistive Room Activity Analyzer

The Assistive Room Activity Analyzer is a tool developed throughout the course of this research with the goal of presenting a simple and intuitive interface for the development of assistive environment experiments. It incorporates a series of algorithms and functionality to allow researchers to concentrate on the algorithmic work, and at the same time to have valuable visuals and UI features to evaluate their work quickly and efficiently. This application is publicly available for use by the research community.

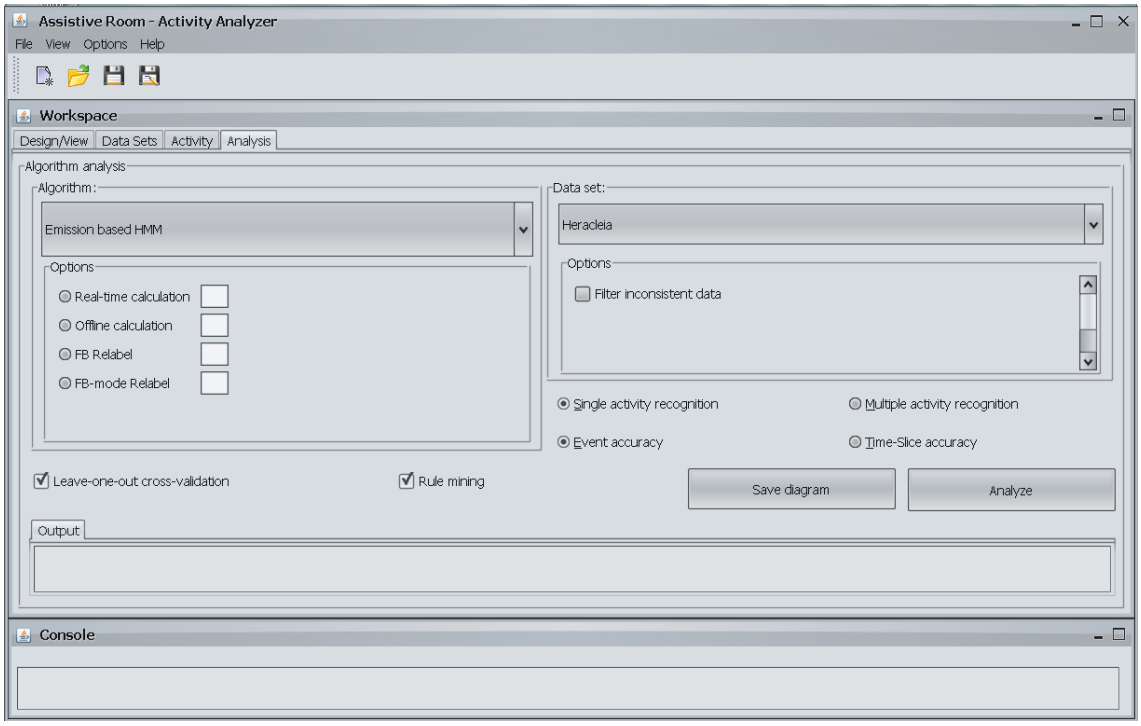

Figure A1 – The Assistive Room Activity Analyzer Tool.

The application is written in the Java programming language and consists of several modules. The *core* module provides basic functionality to create and evaluate algorithms based on HMM implementations. The *plugins* module is designed to enable developers to create their own algorithms and just plug them as a separate jar file into the application. In order to develop

a plugin that will work with the application, it is necessary to implement a series of Java interfaces. The *structures* module provides a series of class files and convenience methods that abstract the specifics of different assistive environment designs and data types, allowing the developer to operate conceptually with a high level abstraction of the environment's functionality. Finally, the *ui* module provides different predefined frames and panels that facilitate the visual representation and inspection of algorithm results.

In order to use the application, it is only necessary to have a Java Runtime Environment 1.6+ installed in a computer system and to follow these steps:

1. Execute the Assistive Environment Activity Analyzer tool
2. Using the application's menubar, load an existing experiment configuration file. Two different experiment configuration files are available by default, these are CSH and HAH. These represent the CASAS Smart Home and the Heracleia Apartment @Home
3. Once the experiment is loaded, the design panel will display an image of the assistive environment and the sensor configuration setup
4. In order to modify the environment, interact with the different buttons and shapes available in the different panels.
5. To evaluate an activity recognition algorithm, it is necessary to load a dataset. Go to the menubar, and select the "Load dataset…" menu item, then select a any dataset configuration file available in the experiment folder.
6. In order to evaluate an algorithm, click on the "Analysis" tab, select the algorithm options, and click on the "Analysis" button.

APPENDIX B

ATHOME APARTMENT TESTBED

Heracleia AtHome

The Heracleia AtHome apartment testbed (@Home) is a miniature apartment located within the Heracleia laboratory and consisting of one bedroom, one bathroom, a kitchen, a dining area, and a living room. In order to collect the dataset used for the experiments in this research, the environment was fitted with a variety of sensor technologies. Following, we provide illustrations for the hardware used.

Figure B1 – SunSpot minicomputers equipped with temperature, light, and acceleration sensors.

The above image shows one of the main components used for collecting data in the apartment testbed. Sunspots are small computers equipped with radio and sensor technology that can transmit wireless information. In our setup, we used SunSpots in various places, such as the bedroom cabinets and drawers, the TV, the toilet, and the shower curtain. These devices were programmed with thresholds to detect events of different types based on their location or placement.

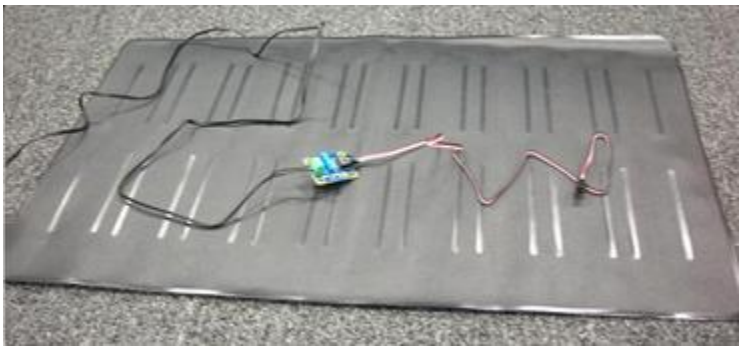

Figure B2 – Pressure mat.

Pressure mats were used to detect the physical presence of participants within the miniature environment. Five pressure mats were placed, two near the bed, one next to the TV chair, one at the entrance of the toilet, and one next to the entrance of the bedroom.

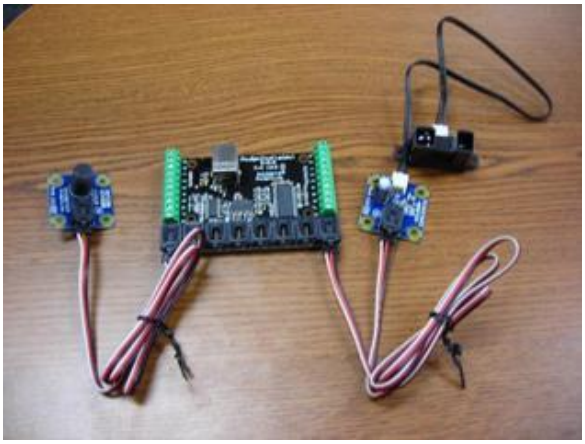

Figure B3 – Phidgets interface kit with motion and infrared sensors.

Three infrared sensors were placed in the toilet, in positions to detect a user's presence next to the sink, and another to detect interaction with the toilet towel stand. Motion sensors were placed near the sink and by the TV chair.

In order to convert the raw sensor values from the sensors attached to the Phidgets device into events, we developed the *Phidgets sensor tool*; illustrated in Figure B4.

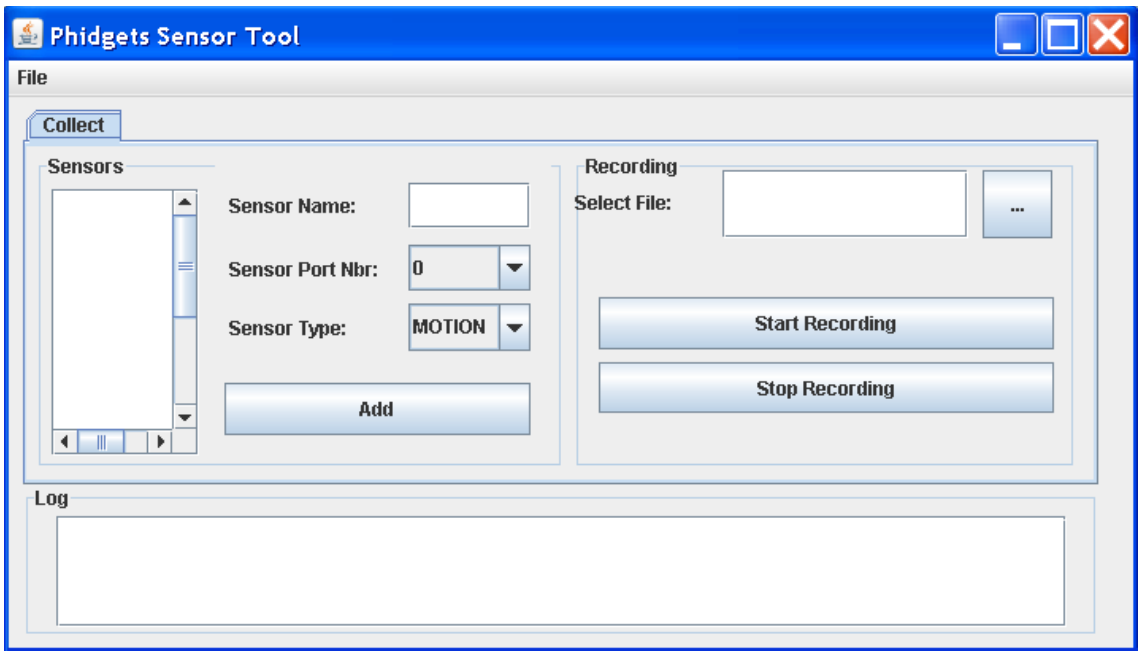

Figure B4 – Phidgets sensor tool.

The Phidget's sensor tool is a java based program that relies on third party libraries in order to read the incoming data from the sensors. The program uses a series of predefined threshold values and logic to determine when an event should be triggered based on the raw sensor data. The application is designed for quick operation; it includes functionality to create and load sensor setup configurations, and to save the resulting events into files. Furthermore, it provides a logging panel to report in the event of unexpected errors or faulty situations.

REFERENCES

[1] United Nations, “World Population Prospects: The 2008 Revision, Highlights,” *Department of Economic and Social Affairs, Population Division (2009)*, Working Paper No. ESA/P/WP.210

[2] M. Weiser, “Some computer science issues in ubiquitous computing,” *Commun. ACM*, vol. 36, 1993, pp. 75-84.

[3] S. Iwarsson, V. Horstmann, and U. Sonn, “Assessment of dependence in daily activities combined with a self-rating of difficulty,” *Journal of Rehabilitation Medicine*, vol. 41, Feb. 2009, pp. 150-156.

[4] B. Schilit, N. Adams, R. Want, and others, “Context-aware computing applications,” *Proceedings of the workshop on mobile computing systems and applications*, 1994, pp. 85–90.

[5] V. Metsis, Z. Le, Y. Lei, and F. Makedon, “Towards an evaluation framework for assistive environments,” *Proceedings of the 1st international conference on PErvasive Technologies Related to Assistive Environments*, 2008, p. 12.

[6] N. Robertson and I. Reid, “A general method for human activity recognition in video,” *Computer Vision and Image Understanding*, vol. 104, 2006, pp. 232–248.

[7] D. Lymberopoulos, A. Bamis, and A. Savvides, “Extracting spatiotemporal human activity patterns in assisted living using a home sensor network,” *Proceedings of the 1st international conference on PErvasive Technologies Related to Assistive Environments*, 2008, p. 29.

[8] S. Hongeng, R. Nevatia, and F. Bremond, “Video-based event recognition: activity representation and probabilistic recognition methods,” *Computer Vision and Image Understanding*, vol. 96, Nov. 2004, pp. 129-162.

[9] D. Patterson, D. Fox, H. Kautz, and M. Philipose, “Fine-grained activity recognition by aggregating abstract object usage,” *Ninth IEEE International Symposium on Wearable Computers*, 2005, pp. 44–51.

[10] N. Ravi, N. Dandekar, P. Mysore, and M.L. Littman, “Activity recognition from accelerometer data,” *Proceedings of the National Conference on Artificial Intelligence*, 2005, p. 1541.

[11] T. Gu, Z. Wu, X. Tao, H.K. Pung, and J. Lu, “epSICAR: An Emerging Patterns based approach to sequential, interleaved and Concurrent Activity Recognition,” *Proceedings of the 2009 IEEE International Conference on Pervasive Computing and Communications-Volume 00*, 2009, pp. 1–9.

[12] M. Stikic, T. Huynh, K. Van Laerhoven, and B. Schiele, “ADL recognition based on the combination of RFID and accelerometer sensing,” *2nd International Conference on Pervasive Computing Technologies for Healthcare*, 2008.

[13] K. Park, E. Becker, J.K. Vinjumur, Z. Le, and F. Makedon, “Human behavioral detection and data cleaning in assisted living environment using wireless sensor networks,” *Proceedings of the 2nd International Conference on PErvsive Technologies Related to Assistive Environments*, 2009, p. 7.

[14] R.B. Smith, “SPOTWorld and the Sun SPOT,” *Proceedings of the 6th international conference on Information processing in sensor networks*, 2007, p. 566.

[15] G. Singla, D.J. Cook, and M. Schmitter-Edgecombe, “Tracking Activities in Complex Settings Using Smart Environment Technologies.”

[16] J. Wu, A. Osuntogun, T. Choudhury, M. Philipose, and J.M. Rehg, “A scalable approach to activity recognition based on object use,” *Proceedings of Int’l conference on computer vision*, 2007.

[17] D.H. Wilson and C. Atkeson, “Simultaneous tracking and activity recognition (STAR) using many anonymous, binary sensors,” *The Third International Conference on Pervasive Computing*, 2005, pp. 62–79.

[18] T. Wu, C. Lian, and J.Y. Hsu, “Joint recognition of multiple concurrent activities using factorial conditional random fields,” *AAAI Workshop PAIR 2007*, 2007.

[19] D.H. Hu and Q. Yang, “CIGAR: Concurrent and interleaving goal and activity recognition,” *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI)*, 2008, pp. 1363–1368.

[20] T. Berners-Lee and J. Hendler, “Scientific publishing on the semantic web,” *Nature*, vol. 410, 2001, pp. 1023–1024.

[21] G. Chen and D. Kotz, “A survey of context-aware mobile computing research,” *Dartmouth College*, 2000.

[22] T.R. Gruber and others, “Toward principles for the design of ontologies used for knowledge sharing,” *International Journal of Human Computer Studies*, vol. 43, 1995, pp. 907–928.

[23] O. Lassila and R.R. Swick, “Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation 22 February 1999,” *W3C Recommendation. See http://www. w3. org/TR/REC-rdf-syntax*, 1999.

[24] L. Coyle, S. Neely, G. Stevenson, M. Sullivan, S. Dobson, P. Nixon, and G. Rey, “Sensor fusion-based middleware for smart homes,” *International Journal of Assistive Robotics and Mechatronics*, vol. 8, 2007, pp. 53–60.

[25] T. Gu, X.H. Wang, H.K. Pung, and D.Q. Zhang, “An ontology-based context model in intelligent environments,” *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference*, 2004.

[26] H. Chen, T. Finin, and A. Joshi, “An ontology for context-aware pervasive computing environments,” *The Knowledge Engineering Review*, vol. 18, 2004, pp. 197–207.

[27] T. Gu, H.K. Pung, and D. Zhang, “A semantic p2p framework for building context-aware applications in multiple smart spaces,” *Lecture Notes in Computer Science*, vol. 4808, 2007, p. 553.

[28] A. Viterbi, “Error bounds for convolutional codes and an asymptotically-optimum decoding algorithm,” *IEEE Transactions on Information Theory*, 1967, 13(2):260-269

[29] S. Greenberg and C. Fitchett, “Phidgets: easy development of physical interfaces through physical widgets,” *Proceedings of the 14th annual ACM symposium on User interface software and technology*, 2001, pp. 209–218.

[30] E. Becker, Z. Le, K. Park, Y. Lin, and F. Makedon, “Event-based experiments in an assistive environment using wireless sensor networks and voice recognition,” *Proceedings of the 2nd International Conference on PErvsive Technologies Related to Assistive Environments*, 2009, p. 17.

[31] A. Agostini, C. Bettini, and D. Riboni, “Loosely coupling ontological reasoning with an efficient middleware for context-awareness,” *Proceedings of the Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous 2005)*, 2005, pp. 175–182.

[32] X. Gu and K. Nahrstedt, “An event-driven, user-centric, QoS-aware middleware framework for ubiquitous multimedia applications,” *Proceedings of the 2001 international workshop on Multimedia middleware*, 2001, pp. 64–67.

[33] S. Shenker, C. Partridge, and R. Guerin, *Specification of guaranteed quality of service*, RFC 2212, September 1997, 1995.

[34] Z. Wang and J. Crowcroft, “Quality-of-service routing for supporting multimedia applications,” *IEEE Journal on Selected areas in communications*, vol. 14, 1996, pp. 1228–1234.

[35] J. Jin and K. Nahrstedt, “QoS specification languages for distributed multimedia applications: A survey and taxonomy,” *IEEE MULTIMEDIA*, 2004, pp. 74–87.

[36] J.J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, “Jena: implementing the semantic web recommendations,” *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, 2004, pp. 74–83.

[37] M. Dean, G. Schreiber, S. Bechhofer, F. Van Harmelen, J. Hendler, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, and L.A. Stein, “OWL web ontology language reference,” *W3C Recommendation February*, vol. 10, 2004.

[38] E. Sirin, B. Parsia, B.C. Grau, A. Kalyanpur, and Y. Katz, “Pellet: A practical owl-dl reasoner,” *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, 2007, pp. 51–53.

[39] V. Haarslev and R. Möller, “Racer: A core inference engine for the semantic web,” *Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools*, 2003, pp. 27–36.

[40] E. Prud’Hommeaux, A. Seaborne, and others, “SPARQL query language for RDF,” *W3C working draft*, vol. 4, 2006.

[41] Y. Theoharis, V. Christophides, and G. Karvounarakis, “Benchmarking database representations of rdf/s stores,” *Lecture notes in computer science*, vol. 3729, 2005, p. 685.

[42] K. Nahrstedt, D. Xu, D. Wichadakul, and B. Li, “QoS-aware middleware for ubiquitous and heterogeneous environments,” *IEEE Communications Magazine*, vol. 39, 2001, pp. 140–148.

[43] G.D. Forney Jr, “The forward-backward algorithm,” *Proceedings of the 34th Allerton Conference on Communications, Control and Computing*, 1996, pp. 432–446.

[44] R. Arora, V. Metsis, R. Zhang, and F. Makedon, “Providing QoS in ontology centered context aware pervasive systems,” *Proceedings of the 2nd International Conference on PErvasive Technologies Related to Assistive Environments*, 2009, p. 8

BIOGRAPHICAL INFORMATION

Roman Arora started his bachelor’s degree in Computer Science Engineering at the University of Texas At Arlington in the year 2002 and completed it in the year 2006. He returned to the University of Texas At Arlington to pursue a master's degree in Computer Science Engineering on January of 2008 and completed his degree on December of 2009.