Providing QoS in Ontology Centered Context Aware Pervasive Systems

Roman Arora, Vangelis Metsis, Rong Zhang and Fillia Makedon
Heracleia Human-Centered Computing Laboratory
Department of Computer Science and Engineering
University of Texas at Arlington
416 Yates St,
Arlington, TX, 76019
001-817-272-5459
{roman.arora, vangelis.metsis}@mavs.uta.edu, {rongz, makedon}@uta.edu

ABSTRACT
There has been significant research in adapting the Semantic Web technologies to create flexible context aware pervasive systems to enhance fields such as assisted living or smart environments. Several ontology based techniques have been proposed to simplify modeling knowledge and its relationships, and several ontology centered middleware tools are currently being developed to provide flexible and viable solutions for application developers. However, middleware built on the basis of Semantic Web generally suffers from drawbacks in performance, which limits its practical applications in the real world. This paper proposes a framework to facilitate Quality of Service (QoS) in ontology centered context aware pervasive middleware. Our approach suggests that context-aware middleware that operate by contracting mutual agreements with the client applications and provide controls over the amount of data to be processed by them can achieve predictable performance and response times. We also propose a service contract scheme that allows both client applications and middleware to participate in the decisions regarding the necessary data transformations required by the different system components in order to improve the overall system performance.

Categories and Subject Descriptors
D.4.8 [**PERFORMANCE OF SYSTEMS**]: Measurement techniques, Performance attributes.

General Terms
Measurement, Performance, Design

Keywords
Middleware, Pervasive Computing, Context aware applications, Quality-of-Service (QoS)

PETRA'09, June 9–13, 2009, Corfu, Greece.

1. INTRODUCTION
In recent years pervasive devices and services are increasingly becoming a part of our daily activities. Smart phones, PDAs, notebooks, medical devices for health monitoring and other smart devices construct an inextricable part of the of the typical modern professional. Those devices in cooperation with the corresponding applications that run on top of them have an aspiration of creating a state where services will be provided to people in a continuous and transparent way requiring as less of their attention as possible and at the same time they will enhance their productivity and quality of life by performing routine tasks or tasks that could not be performed by people easily without some external assistance. This state has been described in literature as Pervasive or Ubiquitous Computing [22].

In the pursuit of making pervasive computing as useful as possible in our everyday life we need to make the various computing units and applications communicate and cooperate with each other and with the surrounding environment. This introduces the problem of retrieving, storing and managing knowledge which will allow computing devices to make smarter decisions in a broader context and in a more macroscopic perspective. That means that we need to create larger context aware systems [4] which will handle heterogeneous types of information so that the decisions of specific components, whether devices or applications, will conform to a wider strategy plan.

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 challenging task, especially when we need to combine huge amounts of data from different sources which may also reside in different devices/machines. In addition to that, as we climb in the higher levels of the knowledge hierarchy the problem of the semasiological management of the data becomes even more difficult since there we do not yet have equally good solutions for data description and representation. In that level, the Semantic Web [2] technologies come into play and offer a promising solution for describing the data in a way that can be shared by machines and that allows for reasoning. Ontological representations [15] can be used to integrate knowledge from different domains into a common platform and (meta)data description methods such as RDF/XML [14] can be used to unify the communication channel among different devices and applications.

However, as it has been shown by previous studies, the ontological based reasoning is a computationally demanding task and it does not scale well with the amount of available data. Taking into account that many of the decisions in context aware systems have to be made in real-time we realize that the application of such a technology in real-life situations may be problematic. This arises the issue of Quality of Service (QoS) that those systems can offer to both the end users and to the underlying or cooperating applications. While most of the existing effort has gone in defining and exploring the different functionality aspects of such systems, little work has focused on the importance of efficiency and quality of service. However, QoS has been well-studied in other areas such as Networking and Multimedia applications [13] and some of the conclusions and methods that have been derived from there can be applied to context-aware pervasive applications as well.

In this paper we attempt to examine the QoS issues that arise from the use of Semantic Web technologies for knowledge representation and decision making in pervasive applications and we propose a framework that ensures QoS and therefore a smooth and consistent operation of systems that involve the cooperation of a number of distributed pervasive devices and computation nodes with a goal to provide integrated context aware time critical services. The framework can function as a part of a middleware system which operates between the data collection and application layers.

The rest of this article is organized as follows. Section 2 discusses related work. Section 3 describes the basic characteristics of ontology centered middleware architecture. In section 4, we examine a representative example to illustrate the need for QoS in smart environments. In Section 5, we define QoS as a maximization problem and develop a framework in order to be able to solve this problem. Finally, we present our conclusions in Section 6.

2. RELATED WORK

In most cases the ongoing efforts for building smart environments have concentrated on methods for, first, making the pervasive devices as sensitive and accurate as possible in the perception of environmental signals, and second, in the cooperation between the different nodes of a system which includes communication, data sharing and common decision making. To achieve those goals most researches have relied on a middleware architecture which attempts to bridge the communication gap among the different system components and provide a transparent layer to higher level applications [20].

With the introduction of the concept of Semantic Web [2] and semantic representation of knowledge in general there has been a significant turn into using such technologies for reasoning and decision making in smart environments and context aware applications in general [4]. Ontologies [7] have been used for formal representation of concepts and relations between them and structured data representation methods such as RDF/XML [14] have been used for description of resources and metadata. Amongst the increasing number of ontology centered middleware, we highlight Construct [6], SOCAM [9], COBRA-ONT [5], Semantic Context Spaces [8].

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. Our architecture on the other hand puts most of its effort in the concern of the QoS and performance, as we believe this to be the current most important issue restricting the existing middleware from becoming highly adopted by the community.

There are however significant challenges when addressing Quality of Service in a context aware pervasive systems. We have identified several aspects that need to be considered and that will affect significantly the design of middleware architecture. Some aspects have been identified in existing research such as [9], and how reasoning times grow disproportionally as the RDF triple store grows in size. [1] identifies the performance problems with reasoning and proposes a loosely coupled middleware architecture where ontological reasoning is mostly performed asynchronously.

QoS has also been extensively studied in other application domains such as Networking and Multimedia [10] [11] [21]. In those domains QoS has been formalized by QoS specification languages [13] and specific solutions have been proposed. However, although such solutions can be taken into consideration when designing QoS enabled smart environments they cannot be directly applied due the more complex nature of context aware applications which involve heterogeneous devices and data types.

From our point of view, in context aware pervasive systems the problem of QoS must be tackled in a higher level by creating a framework that will add the concept of "predictability" in behavioral patterns of the different system components in terms of efficiency and end-to-end delays. For that reason, in the next sections we propose a framework for strategic design of middleware applications 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 request service from the system.

3. ONTOLOGY CENTERED MIDDLEWARE

Ontology centered middleware focuses on providing a unified knowledge and data model. Several architectures that follow this principle have been proposed [5] [6] [8] [9] [17]. 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 Fig. 1

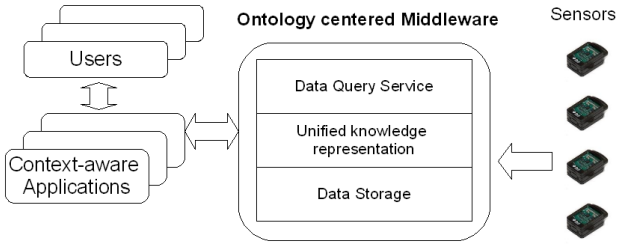

Fig.1: An example ontology centered middleware architecture

In these works, 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 [8], 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 [3] are used to store both the ontologies, and data, generally in Semantic Web standard formats such as OWL [15] and RDF [14] 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 [3], Pellet [18], and Racer [12].

(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 [16] 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 [19] 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 [1]. Other strategies proposed include data subscription by applications and maintaining the data store as minimal as possible [9].

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 [23] 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. QoS IN PRACTICE

4.1 Example problem that requires QoS

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. 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 Fig.1

We now consider a series of context-aware applications running on this environment, each of which have different QoS requirements and work by requesting data to 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 be of your interest. The application must execute in 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, for example 5-7 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 on 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.

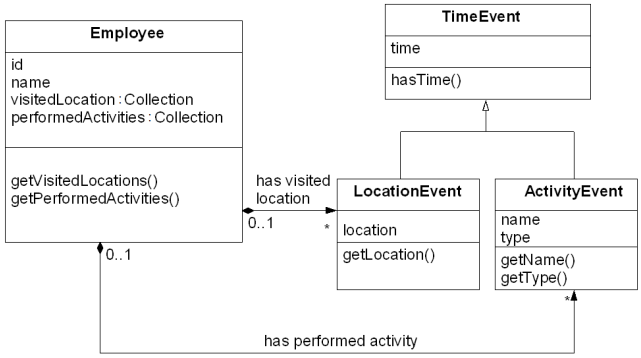

Fig. 2: Sample UML diagram displaying application ontologies

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

4.2 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 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 Fig. 2, 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 [6], COBRA [5], SOCAM [9], SCS [8] have focused their attention on non-time critical applications and demonstrating the flexibility and value of ontology centered architectures, showing little attention into 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.

5. 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 5.2 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 5.3. 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:

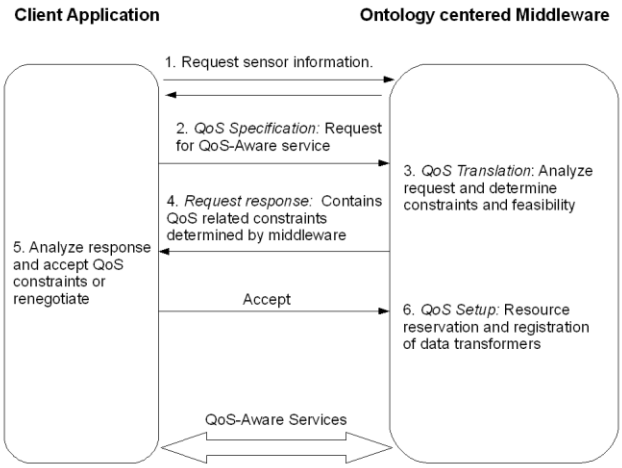

Fig.3: A proposed QoS negotiation mechanism

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 [8], but they go beyond such functionality, their details are described in Section 5.2
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 5.1. 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.

5.1 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:

Cardinalities: $X = \{x_1, x_2, ..., x_n\}$, $X \in V = \{v_1, v_2, ..., v_k\}$

x_i*: cardinality value assigned to client i.*

V: predefined set of allowed cardinality values.

Constants: $C = \{c_1, c_2, ..., c_n\}$

c_i*: 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.*

Time constraints: $T = \{t_1, t_2, ..., t_n\}$

t_i*: time constraint specified by the client i.*

Maximize: $c_1x_1 + c_2x_2 + ... + c_nx_n$

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

Note: 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.

5.2 QoS Specification - An XML representation of the client's QoS requirements

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.
- Similarly, 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

```
<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>
```

Fig.4: An example Xml QoS specification file

optimize performance.

Figure 4 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.

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.

5.3 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.

5.4 Prototype Experiment

We develop a prototype experiment to study the proposed framework. We imagine a scenario in which there are hundreds of employees, all of which have PDAs running a context-aware Activity Reminder program. We've built a simple QoS-aware Activity Reminder application that requires a response time of 2s from the middleware. The application can operate in three different modes, each of which takes advantage of a specific data transformer. The data transformers generate and handle activity statements, the first one handles up to 1000 statements (Mode 1), the second one 500 statements (Mode 2), and the third one 100 statements (Mode 3). We allow an increasing number of instances of the Activity Reminder application to run on the framework, from 50 instances up to 400. Each instance represents a different employee's application. The middleware is in charge of adjusting the operation mode of the instances to maintain QoS requirements. A chart of the middleware's adjustment on the applications is presented in the figure bellow

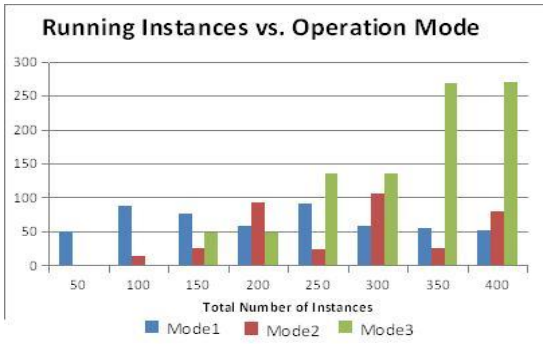

Fig. 5: Snapshot of Activity Reminder instances vs Operation mode.

Clearly, as the load on the system increases, the middleware relies on the availability of the application to operate at different modes with different amounts of data to reduce overall load and maintain QoS constraints. Evaluating the quality of the recommendations produced by the Activity Reminder under decreased amounts of data can be accomplished through user feedback. Alternatively,

we intend as future work to study the optimality of allowing applications to prioritize the data sources and running these through the data maximization algorithm proposed.

6. CONCLUSION

In this paper, we have examined the problem of providing QoS in smart environments which are built using an ontology centered middleware architecture. According to our view, the best way to ensure QoS in such environments is that the middleware system should be able control the amount of data that processes and distributes to each client application. 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 grated 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.

7. REFERENCES

[1] Alessandra Agostini, Claudio Bettini, and Daniele Riboni. Loosely coupling ontological reasoning with an efficient middleware for context-awareness. In *MOBIQUITOUS '05: Proceedings of the The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pages 175–182, Washington, DC, USA, 2005. IEEE Computer Society.

[2] T. Berners-Lee, J. Hendler, O. Lassila, et al. The Semantic Web. *Scientific American*, 284(5):28–37, 2001.

[3] Jeremy J. Carroll and Dave Reynolds. Jena: Implementing the semantic web recommendations. pages 74–83, 2004.

[4] G. Chen and D. Kotz. A Survey of Context-Aware Mobile Computing Research, 2000.

[5] Harry Chen, Tim Finin, and Anupam Joshi. An Ontology for Context-Aware Pervasive Computing Environments. *Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review*, 18(3):197–207, May 2004.

[6] Lorcan Coyle, Steve Neely, Graeme Stevenson, Mark Sullivan, Simon Dobson, and Paddy Nixon. Sensor fusion-based middleware for smart homes. *International Journal of Assistive Robotics and Mechatronics (IJARM)*, 8(2):53–60, 06/2007 2007.

[7] T.R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *INTERNATIONAL JOURNAL OF HUMAN COMPUTER STUDIES*, 43:907–928, 1995.

[8] Tao Gu, Hung Keng Pung, and Daqing Zhang. A semantic p2p framework for building context-aware applications in multiple smart spaces. In *EUC*, pages 553–564, 2007.

[9] Tao Gu, Xiao Hang Wang, Hung Keng Pung, and Da Qing Zhang. An ontology-based context model in intelligent environments. In *In Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference*, pages 270–275, 2004.

[10] X. Gu and K. Nahrstedt. An event-driven, user-centric, QoS-aware middleware framework for ubiquitous multimedia applications. In *Proceedings of the 2001 international workshop on Multimedia middleware*, pages 64–67. ACM New York, NY, USA, 2001.

[11] R. Guerin. Specification of guaranteed quality of service. In *Integrated Services WG Internet Draft (work in progress*, 1997.

[12] V. Haarslev and R. Mφller. Racer: A Core Inference Engine for the Semantic Web. In *Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools*, pages 27–36, 2003.

[13] J. Jin and K. Nahrstedt. QoS Specification Languages for Distributed Multimedia Applications: A Survey and Taxonomy. *IEEE MULTIMEDIA*, pages 74–87, 2004.

[14] O. Lassila, R.R. Swick, et al. Resource Description Framework (RDF) Model and Syntax Specification. 1999.

[15] D.L. McGuinness, F. van Harmelen, et al. OWL Web Ontology Language Overview. *W3C Recommendation*, 10:2004–03, 2004.

[16] E. Prud'Hommeaux, A. Seaborne, et al. SPARQL Query Language for RDF. *W3C Working Draft*, 20, 2006.

[17] Anand Ranganathan and Roy Campbell. A middleware for context-aware agents in ubiquitous computing environments. page 998. 2003.

[18] E. Sirin and B. Parsia. Pellet: An OWL DL Reasoner. In *2004 International Workshop on Description Logics*.

[19] Y. Theoharis, V. Christophides, and G. Karvounarakis. Benchmarking Database Representations of RDF/S Stores. *LECTURE NOTES IN COMPUTER SCIENCE*, 3729:685, 2005.

[20] S. Vinoski and I.T. Inc. CORBA: integrating diverse applications within distributed heterogeneous environments. *Communications Magazine, IEEE*, 35(2):46–55, 1997.

[21] Z. Wang and J. Crowcroft. Quality-of-service routing for supporting multimedia applications. *Selected Areas in Communications, IEEE Journal on*, 14(7):1228–1234, 1996.

[22] M. Weiser. Some computer science issues in ubiquitous computing. *ACM SIGMOBILE Mobile Computing and Communications Review*, 3(3), 1999.

[23] D. Xu and B. Li. QoS-Aware Middleware for Ubiquitous and Heterogeneous Environments. *IEEE Communications Magazine*, page 2, 2001.