
Kashmira Phalak

Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science

In

Computer Science and Applications

Advisory Committee:

Dr. Madhav Marathe (Chair)  Dr. Christopher Barrett
Dr. Keith Bisset             Dr. Calvin Ribbens

June 8\textsuperscript{th}, 2006
Blacksburg, Virginia

Keywords: cognitive, data integration, model driven, service-oriented, middleware, wireless sensor networks

by Kashmira Phalak
Committee Chair: Dr. Madhav Marathe
Computer Science and Applications

Abstract

We describe CoSMo, a Cognitively Inspired Service and Model Architecture for situational awareness and monitoring of vehicular traffic in urban transportation systems using a network of wireless sensors. The system architecture combines (i) a cognitively inspired internal representation for analyzing and answering queries concerning the observed system and (ii) a service oriented architecture that facilitates interaction among individual modules, of the internal representation, the observed system and the user. The cognitively inspired model architecture allows effective deductive as well as inductive reasoning by combining simulation based dynamic models for planning with traditional relational databases for knowledge and data representation. On the other hand the service oriented design of interaction allows one to build flexible, extensible and scalable systems that can be deployed in practical settings. To illustrate our concepts and the novel features of our architecture, we have recently completed a prototype implementation of CoSMo. The prototype illustrates advantages of our approach over other traditional approaches for designing scalable software for situational awareness in
large complex systems. The basic architecture and its prototype implementation are
generic and can be applied for monitoring other complex systems.

This thesis describes the design of cognitively-inspired model architecture and its
corresponding prototype. Two important contributions include the following:

i) **The cognitively-inspired architecture**: In contrast to earlier work in model driven
architecture, CoSMo contains a number of cognitively inspired features, including
perception, memory and learning. Apart from illustrating interesting trade-offs
between computational cost (e.g. access time, memory), and correctness available
to a user, it also allows users specified deductive and inductive queries.

ii) **Distributed Data Integration and Fusion**: In keeping with the cognitively-inspired
model-driven approach, the system allows for an efficient data fusion from
heterogeneous sensors, simulation based dynamic models and databases that are
continually updated with real world and simulated data. It is capable of supporting
a rich class of queries.
Acknowledgement

I would like to take this opportunity to thank Dr. Madhav Marathe, my advisor and guide, for his support and direction through the course of this thesis. His unparalleled enthusiasm and practical outlook towards life has made working with him an absolute pleasure. Without his tireless and patient mentoring this project might not have been completed. I am grateful to Madhav for making this a really worthwhile experience for me. I would also like to acknowledge the support of Dr. Chris Barrett of my advisory committee who made me look at cognition and human behavior in a new light. I would like to extend special thanks to Dr. Keith Bisset for patiently churning out data sets until we finally hit the mark. In addition, I would like to thank Dr. Cal Ribbens for graciously agreeing to be on my advisory committee and providing me with invaluable feedback. Sameer Tupe, my colleague and friend with whom I have jointly produced this thesis, has been a great source of motivation. This project would not have been successful without our productive arguments and his relentless efforts. Also, it has been a distinct pleasure interacting with Deepti Chafekar who patiently helped me through the many iterations of this thesis. In addition, I am grateful to the faculty and students at Virginia Bioinformatics Institute (VBI) for attending our project talks and for their enthusiastic participation. Working on my thesis at VBI and interacting with everyone here has been a most fulfilling experience. I would also like to thank my brother, Vivek, for assiduously clarifying my doubts regarding the database aspects of this project. Finally, I extend a warm hearted gratitude towards my parents and friends who have touched me with their patience and understanding during the course of this thesis.
Table of Contents

1. INTRODUCTION .................................................................................................................. 1
   1.1 MOTIVATION .................................................................................................................. 1
   1.2 OUR CONTRIBUTION ...................................................................................................... 2
      1.2.1 A Cognitively-inspired Modeling Approach ............................................................ 3
   1.3 SPECIFIC CONTRIBUTION OF THE THESIS .............................................................. 4
   1.4 ORGANIZATION OF THE THESIS .............................................................................. 5
2. RELATED WORK .................................................................................................................... 6
3. OVERVIEW OF THE ARCHITECTURE ............................................................................. 12
   3.1 BRIEF DESCRIPTION OF CoSMo’S SYSTEM ARCHITECTURE .................................... 12
      3.1.1 High-level Architecture of CoSMo: ...................................................................... 12
      3.1.2 Overview of CoSMo’s System Internals:................................................................. 14
   3.2 WORK-FLOW DIAGRAMS ............................................................................................. 21
      3.2.1 Logical View ........................................................................................................... 21
      3.2.2 Process View ......................................................................................................... 23
      3.2.3 Development View ............................................................................................... 25
      3.2.4 Physical View ........................................................................................................ 27
      3.2.5 Scenarios ............................................................................................................... 29
4. QUERY MODEL ................................................................................................................... 39
   4.1 VALUE-BASED QUERY ............................................................................................... 40
   4.2 EVENT-BASED QUERY ............................................................................................... 41
   4.3 HYBRID QUERY .......................................................................................................... 41
5. QUERY PROCESSOR .......................................................................................................... 44
   5.1 OVERVIEW ................................................................................................................... 44
   5.2 DESIGN ......................................................................................................................... 44
   5.3 IMPLEMENTATION DETAILS ..................................................................................... 45
6. STATE MANAGER .............................................................................................................. 57
   6.1 OVERVIEW ................................................................................................................... 57
   6.2 DESIGN ......................................................................................................................... 57
   6.3 IMPLEMENTATION DETAILS ..................................................................................... 58
7. DATA MODEL ...................................................................................................................... 63
   7.1 OVERVIEW ................................................................................................................... 63
   7.2 DESIGN ......................................................................................................................... 63
   7.3 IMPLEMENTATION DETAILS ..................................................................................... 66
8. MESSAGE MAPPER INTERFACE ..................................................................................... 76
   8.1 OVERVIEW ................................................................................................................... 76
   8.2 DESIGN ......................................................................................................................... 78
   8.3 IMPLEMENTATION DETAILS ..................................................................................... 78
9. REAL WORLD SENSOR NETWORK MANAGER .............................................................. 79
   9.1 OVERVIEW ................................................................................................................... 79
   9.2 DESIGN ......................................................................................................................... 80
   9.3 IMPLEMENTATION DETAILS ..................................................................................... 80
Table of Figures

FIGURE 1: HIGH-LEVEL ARCHITECTURE OF THE COISMO SYSTEM ......................................................... 13
FIGURE 2: DETAILED SYSTEM ARCHITECTURE OF THE COISMO SYSTEM ................................. 14
FIGURE 3: LOGICAL VIEW OF THE COISMO SYSTEM ................................................................. 22
FIGURE 4: PROCESS VIEW OF THE COISMO SYSTEM ............................................................. 24
FIGURE 5: DEVELOPMENT VIEW OF THE COISMO SYSTEM ......................................................... 26
FIGURE 6: PHYSICAL VIEW OF THE COISMO SYSTEM ............................................................... 28
FIGURE 7: FLOW OF CONTROL THROUGH COISMO FOR QUERY 1 ........................................ 30
FIGURE 8: USER INTERFACE FOR USER QUERY INPUT .............................................................. 31
FIGURE 9: MAP-BASED USER INTERFACE FOR USER QUERY INPUT ........................................ 32
FIGURE 10: FLOW OF CONTROL THROUGH COISMO FOR USE CASE II ................................. 34
FIGURE 11: USER INTERFACE FOR USER QUERY INPUT ............................................................ 35
FIGURE 12: QUERY CLASSIFICATION .......................................................................................... 40
FIGURE 13: GENERIC XML QUERY TEMPLATE ............................................................................. 47
FIGURE 14: SERVICE ENUMERATION IN XML SCHEMA .......................................................... 50
FIGURE 15: OUTPUT XML QUERY FOR VALUE-BASED QUERY REQUESTING DELAY ON A LINK ............................................................ 51
FIGURE 16: OUTPUT XML QUERY FOR VALUE-BASED QUERY REQUESTING DELAY ON A PATH ............................................................ 53
FIGURE 17: OUTPUT XML QUERY FOR EVENT-BASED QUERY REQUESTING CAR DETERMINATION EVENT ON A LINK ......................................................................................... 55
FIGURE 18: THE RULE BASE ....................................................................................................... 59
FIGURE 19: FLOW CHART FOR THE STATE MANAGER COMPONENT: THE RULE-DRIVEN ENGINE FOR DECISION MAKING .................................................................................. 60
FIGURE 20: STATE DIAGRAM FOR THE DATA MODEL ............................................................... 65
FIGURE 21: XML OBSERVATION PLAN SCHEMA ........................................................................ 68
FIGURE 22: FLOW CHART FOR THE DATA MODEL COMPONENT: SELECTION OF THE MOST EFFICIENT SENSORS TO QUERY ......................................................................................... 70
FIGURE 23: OBSERVATION PLAN FOR AN EVENT-BASED QUERY ............................................. 73
FIGURE 24: GLOBAL SCHEMA FOR SENSOR-SPECIFIC QUERY PARAMETER INTERPRETATION ......................................................................................... 77
FIGURE 25: FLOW CHART FOR THE STATE ESTIMATOR COMPONENT: ESTIMATE CALCULATION FOR THE INPUT QUERY .......................................................................................... 85

TABLE 1: MAPPING OF CAPABILITIES OF A COGNITIVE ARCHITECTURE TO SYSTEM CAPABILITIES .... 98
Chapter I

Introduction

1.1 Motivation

Consider a situation wherein you plan to start on a trip at 8:00 A.M. in the morning. With today’s web application technology, one would probably look up the estimated time to complete the journey using a map-based system such as Google Maps, MapQuest etc. Now imagine that you want to traverse the same path at 7:00 p.m. in peak traffic. What would be the result of such a query if you wanted to judge the time taken the next day at 10:00 A.M.? If you query these navigational systems, you would get the same static estimates for routes and travel time. More cases can be thought of where such systems fail to give a dynamic update, such as, the best route and the time taken if you would like to travel part of the route on foot, part of it by bus and the remaining by car. Such systems lack the capability to provide dynamic results since they do not take into account the ever-changing behavior of traffic. This is just one case in which a dynamic modeling of traffic networks is requisite. The data collected by a Traffic Monitoring application may find utility in improving performance of transport networks [60], to map “activity locations” based on travel patterns of members of each household [11], to plan the minimal cost routes for individuals [10][11], to calculate vehicle emissions [11], to provide a maintenance and diagnostic history of vehicles [10], detect events such as accidents, traffic jams, vehicle appearance, landslides on the road etc. Such tasks require a model-based reasoning and analysis system in order to efficiently mine new data as well as process the gathered data. The otherwise extremely tedious task of data collection is facilitated by the deployment of wireless sensor networks. Wireless sensors have the ability to operate in unreachable or hostile environments and provide advantages in terms of cost, size, scalability, and distributed intelligence. Wireless sensor networks facilitate ubiquitous computing by embedding these tiny sensors into the road networks to be able to monitor the traffic patterns and other activities which spawn these patterns. Urban transportation networks are complex social-technical networks. Unlike physical systems,
these are not only characterized by physical laws but also affected by human behavior, regulatory agencies, network connectivity, etc [12].

Developing situational assessment and monitoring tools for such large complex socio-technical is challenging for a number of reasons. First, the size of the system implies that we need a large sensor network to monitor and collect data; this introduces the need to construct an internal dynamic model of the measured system. Second, for several queries of interest, it is necessary to construct internal models that are dynamic; this allows us to support inductive reasoning. The motivating example at the beginning of the chapter is an example of such a query. Third, computationally, it is infeasible to support every query by accessing sensors (beyond the fact that certain queries cannot even be supported in such a manner). This implies that a good situational awareness tool needs to have internal logic to decide when a query can be satisfied by using internal models and data and when it would have to activate sensors to collect data to answer the required query.

Need for a Middleware: Middleware is the grey area between the operating system and the application or in some cases even the bare hardware and the application. [49][50][53][61] precisely explain the scope, functionality and challenges of this middleware in case of wireless sensor networks. The raw data gathered by the sensors in a heterogeneous network needs to be collated, filtered, managed and fused in order to answer queries over this data. Since sensor networks provide only a close estimation of the phenomenon [21], a level of confidence in the aggregated result needs to be provided to the user. The middleware provides an ideal place to maintain an internal representation of the external real world. The tasks of data interpretation, aggregation, management and presentation of this information are among the core requirements for a practical situational awareness tool.

1.2 Our Contribution

Here and in the companion dissertation of Tupe [57], we describe an architecture and a prototype implementation of CoSMo, a Cognitively Inspired Service and Model
Architecture for monitoring vehicular traffic in urban transportation systems using a wireless sensor network. Wireless sensor networks offer an entirely new opportunity to design such systems. CoSMo integrates two distinct system architectures:

- a cognitively-inspired architecture to internally represent static as well as dynamic models of external world to support a rich class of user and administrative tasks, and
- a service architecture to represent and facilitate the interplay between the (i) individual modules of the internal representation, (ii) the external world comprising of the sensors and the urban transport system and (iii) the individual users, including system managers who would are interested in various tasks relating to such infrastructure systems. Examples of such tasks include, finding out current traffic conditions, maintaining the sensor network so to as to extend its lifetime, finding optimal routes, etc.

CoSMo is a practical middleware that in conjunction with appropriate dynamic models can support entirely new classes of queries and user requirements vis-à-vis the efficient monitoring and use of urban transportation systems. CoSMo is developed to specifically support situational awareness about such large complex socio-technical systems and uses TRANSIMS [11], a simulation based dynamic model as internal representation of the transport system. Nevertheless, a number of architectural features of CoSMo are generic and can be used to support monitor and reason about other socio-technical systems. Furthermore, although we use TRANSIMS, CoSMo can be integrated with other dynamic models such as MITSims [13] so far as these models provide the required functionality.

1.2.1 A Cognitively-inspired Modeling Approach

A novel feature of our work is the use of cognitively-inspired architectural elements. Examples of such features include perception, adaptation and learning. Our goal is not to develop cognitive architectures, but rather borrow key features in such architectures that can lead to a more practical and efficient modeling and service environment. We use the term *Cognitively-inspired* to convey the fact this viewpoint. We introduce several well
accepted cognitive features within CoSMo; these features among other elements facilitate the well known cognition cycle of Recognize, Plan, Decide and Act [47][48].

A key aspect of the CoSMo archetype is that it contains a static as well as a dynamic internal representation of the real world in the form of databases and simulators respectively. The complex interplay between the measured and the measuring system is also illustrated. An aspect of cognition done here is that of memory. CoSMo collates and fuses the raw sensor data pulled from different sources with the help of knowledge bases to generate the result of the issued query. The modeling and interaction between these knowledge bases brings out various aspects of a memory hierarchy. CoSMo exhibits situation awareness, decision awareness and knowledge awareness in its tasks. The database contains sensor data history while the simulator has the ability to predict sensor values and the real world sensor network gives the highest accuracy value of the phenomenon at current time. Chapter 12 gives a useful summary of the cognitive capabilities of our system.

### 1.3 Specific Contribution of the Thesis

As discussed earlier, the overall development of CoSMo was done jointly with Sameer Tupe. The cognitively-inspired modeling environment is undertaken as a part of this thesis, while Tupe’s thesis [56] was primarily responsible for the service oriented architecture. Integration of these architectures and systems that constitute CoSMo was undertaken jointly. Specific contributions of this thesis include the following:

i. Design and conceptualization of a novel unified architecture for Traffic Monitoring which integrates four paradigms namely, cognitively-inspired, model-driven, data-centric and service-oriented approaches to build an intelligent, adaptive, systematic, efficient, loosely coupled, easily modifiable, extensible and pluggable infrastructure.

ii. Design and implementation of a Traffic Monitoring Information System:
   a. A practical system to monitor an Urban Transportation Network.
b. Fairly detailed representation and task management of two complicated systems namely, the heterogeneous sensor network and the integrated middleware.

iii A working prototype of the system modeling a set of real world queries.

iv Implementation of a cognitively-inspired model-driven architecture incorporating dynamic internal models of the observed system, inductive and deductive learning and reasoning, perception, persistent memory and adaptation.

v Distributed data fusion of sensor data obtained from various sources internal and external to the system.

vi Defining and supporting interesting query classes.

As of now, we do not have an archival data warehouse for sensor data; we handle traffic planning on a day to day basis and at a road level.

1.4 Organization of the thesis

The thesis is organized in the following manner: Chapter 2 presents the work performed in the field of wireless sensor network middleware with a focus on different architectures and their implications. Chapter 3 gives a high-level view of CoSMo’s architecture and documents the architecture using the “4+1” approach and thus presents a visual modeling of the System. Chapter 4 defines the Query Model which describes the query classes that can be handled by the System. Chapters 5 to 10 offer a detailed explanation of the core building blocks of the System. These chapters are sectioned into 3 main parts: the overview of the component, the design details and the implementation details. Chapter 11 offers an insight into the System Database and data tables. Chapter 12 summarizes the cognitive capabilities of the CoSMo System. Finally, Chapter 13 consists of conclusions and directions for future work.
Chapter 2
Related Work

The industry is geared towards the deployment of wireless sensors networks that can operate in hostile environments and provide advantages in terms of cost, size, scalability, and distributed intelligence. A variety of applications such as traffic monitoring, environment monitoring, surveillance etc. have been inspired by these sensor networks. These sensors typically have very limited resources such as computational power, power levels, lifetime and communication capability. Considerable effort has been made to design middleware which tries to tune various parameters of wireless sensors such as energy-efficiency, reduction in communication costs, data acquisition costs, accuracy of readings, security etc. The choice of an architecture involves the analysis of various trade-offs, dynamism of the system characteristics and the goal of the system. This choice ultimately reflects in the performance of the system. The work related to the design and development of cognitive middleware for wireless sensor networks includes a broad spectrum of areas such as Artificial Intelligence, Sensor Networks, Databases and Software Engineering. This thesis can truly said to be at the interface of these general areas. This section particularly looks at some classic systems which may be model-driven, data-driven, service-oriented, adaptive or cognitive.

Model-Driven Data Acquisition in Sensor Networks [21] by Deshpande et al breaks away from the paradigm which treats the sensor network as a database. It is based on the principle that a database is a store of true data; hence a sensor network which can only approximate the sensed phenomena cannot be treated as a definitive source of this data. This model-driven architecture aims to present richer and higher confidence rendition of the sampled set of physical phenomena by augmenting sensor network query processing with statistical models of interpretation. This architecture is also driven by the need to reduce the cost of executing a query in the sensor net by optimizing the data acquisition from sensors. This particular model depicts a single flow irrespective of the query type, response time and confidence level requirements. We have advanced this bare model
with support for intelligent decisions within the System which is capable of satisfying multiple goals of balancing between not only accuracy and communication cost, but also accuracy and speed of query execution. PRESTO [41] by Li, Ganesan and Shenoy is a system based on a feedback-based model-driven push approach directed towards capturing unusual data trends, supporting archival queries and incorporating an adaptive design to user and data needs. It has the ability to support queries on current sensor data as well as on historical data using interpolation, prediction and indexing into local repositories. Active feedback between the proxy to the sensors is used to adapt to long-term changes in data and queries and re-configure the model parameters. PRESTO lies in the same genre as our System, but CoSMo is sentient since it is endowed with the ability to learn, decide, react and reconfigure itself. The System is equipped with a simulator component to take over predictions and more importantly with dynamic and static representations of the measured and measuring entities. This aspect among others introduces an element of cognition in our System and lends to the System the capacity to handle a much richer class of queries as demonstrated in Chapter 4.

Directed Diffusion [35] by Estrin et al is one of the seminal paradigms of the data-centric architecture for wireless sensor networks. Directed Diffusion is data-centric, in that, it uses named data in the form of attribute value pairs and gives importance to what the data is rather than from where it originates. The architectural style to which this paradigm conforms is rather different from the IP-style layered communication wherein the endpoints have globally unique IDs. It suggests an application-aware paradigm to facilitate efficient aggregation, and a robust data delivery technique back to the sink. The goal of this design is to address some of the challenges involved in software development for wireless sensor networks. It aims to reduce the average energy dissipated per node, the average delay seen by the inquiring sink and improve the distinct event delivery ratio. In the realm of data-centric architectures, DSWare [42] by Li et al presents an interesting perspective on handling event data. The main goal of this architecture is to design a common data service platform for various applications. It is directed towards integrating real-time data services by providing a database-like abstraction of services to sensor network applications. It provides an abstraction of common data services to applications.
instead of having the applications to implement the entire stack of application-specific data services. In general, a complete System deployed over a wireless sensor network would require more components than just compound event detection. Our System can complement a system like DSWare by providing application-level support for event and non-event data handling. Other oft-quoted examples of a Data-centric architecture are Cougar [14] and Directed Diffusion [35], which espouse the notion of sensor network as a database. Cougar was succeeded by TinyDB [43]. It is mainly a query processing system built over a wireless sensor network. It promotes the use of a variant of in-network data aggregation in order to increase the energy efficiency. TinyDB treats the entire sensor network as a single large data table wherein columns specify all the sensor attributes and rows subsume sensor data. This constitutes its Data Model. The Query Model features an SQL based query language which is extended to include event processing, aggregation operators and long running queries. This scheme though flexible and powerful enough to support a large class of sensor queries needs to be augmented with services which process and act on these queries. Cougar [14] and TinyDB [35] both lack adaptability while constructing the query execution plans. These projects take a global view of the network and require the copying of sensor meta-data from every node to the root node periodically. This is an expensive operation. Our System eliminates this cost by maintaining a localized internal copy of the sensor network which can be queried time to time in order to gauge the network state. In our System, data messages, transformation of this data and its flow are intelligently chalked out. The System has explicitly maintained data stores and knowledge bases, dynamically planned data flows and deterministic data transformations. It thus gives data a first class status and addresses the challenges of integration of data obtained from various sources and stored in different data stores.

Taking into account the distributed nature of sensor applications deployed on a grid as well the inherent difficulty in deciphering raw sensor data, there is a huge potential for employing a Service-oriented approach in a sensor environment. This potential is just recently being tapped with architectures such as SensorWeb [16][17], IrisNet [27], GeoSWIFT [57] etc.
The SensorWeb paradigm set a trend towards making sensors, other measuring devices as well as sensor data stores discoverable, accessible and to a certain extent controllable via the internet [16]. The Open Sensor Web Architecture (OSWA) [16] looks into the confluence of service-based SensorWeb and Grid technologies. GeoSWIFT [57] by Tao et al is based on Sensor Web and intends to build a geo-spatial infrastructure to connect distributed sensor networks for the sharing, access, exploitation, and analysis of sensing information. GeoSWIFT Sensor Service offers a portal of sensors to provide web services interfaces for sensor networks, its observations and the related geo-spatial information as well. IrisNet [27] by Gibbons et al, a well-cited architecture for worldwide SensorWeb, has been envisioned with a purpose of providing a service framework which allows users to query distributed and heterogeneous sensors along with very large sensor data reserves. It extends ease of service authorship and addresses the challenges of providing integrated services for data-collection, query answering and filtering sensor readings dynamically. It allows several different services to be deployed on a single IrisNet software infrastructure. The proposed architecture [29] by D. Gracanin et al focuses on providing a flexible and unified framework which would help in embodying and assessing different wireless sensor network models. The design is directed towards mapping the application level purpose to the hardware specific capabilities while maintaining a clear separation between the two. The objective of this design is to create a stratum for development of service-centric software architecture. The design choice of arranging the wireless sensor network in terms of layers and functional planes enhances the QoS, modifiability, usability, availability, security and performance of the system. Middleware encompassing such a service model must keep intact the mapping between layers by careful design of interfaces. Linear or non-linear programming techniques maybe used to establish the global optimum for the mission objective. Ultimately, a set of services needs to be provided to the user wherein the services at mission layer is a composition of atomic services provided by individual sensors in a region. Visualization tools can be incorporated in the middleware and provided as services to expose the details of these layers. The communication paradigm may be based on web services or messaging. This model suggests that middleware software should incorporate protocols
and algorithms for data gathering and processing and also maintain repositories to record a history of sensor knowledge.

Another important class of middleware for wireless sensor networks is that of Adaptive Middleware. Adaptive middleware generally focus on offering flexibility and ease of accommodation of the dynamic network conditions. Proactive adaptation allows the system to define the parameters and conditions under which the adaptation takes place and most importantly bears the capability to actively affect the network [34]. Reactive adaptation refers to a system’s ability to monitor the network and system performance and react accordingly. Along with adaptivity, Quality of Service or performance issues are of primary importance in this type of an architecture. Three notable systems which may be classified under this tag are MiLAN [34] by Heinzelman et al, AutoSec [33] by Han and Venkatasubramanian and TinyCubus [46] by Marron et al. All three systems show a trend of proactive adaptation while TinyCubus [46] incorporates an element of reactive adaptation. MiLAN [34] allows the sensor application to specify the policies which govern the quality needs and proactively fine-tunes network parameters to guarantee QoS, increase the sensor network lifetime and balance the energy-efficiency. Adaptation can be said to be the first step towards cognition. It recognizes the need to bring about a change and takes a step towards providing better QoS. CoSMo depicts reactive adaptation, in which it monitors the system and network state and takes decisions regarding the selection of sensors which would most efficiently satisfy the query. It also allows, to a certain extent, users to define the policy regarding quality needs via the confidence level mechanism and the best query handler component in the system may be selected on this basis.

Current research is directed towards adding intelligence and learning in a System which will take it beyond mere adaptive behavior. This trend has seen the emergence of Cognitive systems for wireless sensor networks. Inspite of the work done in the field of Artificial Intelligence, Cognition is very difficult to precisely delineate, let alone realize completely in a working system. SOAR [6][40] is a well-known cognitive system which attempts to incorporate learning and intelligence into its design. SOAR is a general
cognitive architecture to support systems which portray intelligent behavior. This architecture aims to support all the capabilities that would be necessary for a general purpose intelligent agent. This system embodies cognitive capabilities such as Parallel, associative memory, Belief maintenance, Preference-based deliberation, Automatic subgoaling, Decomposition via problem spaces, Adaptation via generalization of experience, Emphasis on efficiency and performance etc. SOAR aims to mimic human cognition, while our System selectively substantiates the cognitive capabilities. While SOAR looks at building a general purpose intelligent agent, CoSMo focuses on supporting multiple goals, planning, learning, decision making and execution specific to wireless sensor networks. CoSMo is not completely cognitive, but is termed as cognitively-inspired since it embodies a subset of the cognitive capabilities. Mitola’s research work on Cognitive Radios [47][48] put forth a novel paradigm for wireless sensor which aims at building a fully reconfigurable system which modifies its communication functions, reception and transmission parameters to adapt to the changes in the network, real-time spectrum changes or user demands. It also has the capability to determine its location, sense spectrum and frequency use. Thus, the cognitive radio, carries a level of cognition or intelligence that permits decision-making and learned patterns of behavior. It augments the software radio with radio-domain knowledge, model-based logic. One of the interesting aspects of this system related to CoSMo, is that it defines a powerful and flexible language which represents knowledge of radio etiquette, devices, software modules, propagation, networks, user needs, and application scenarios [47]. This research is significant in that, it one of the pioneering efforts in bringing cognition into the realm of wireless communication. While Cognitive Radios focus on cognition in the lower layers of the OSI stack, we bring this cognition into the upper layers.
Chapter 3
Overview of the Architecture

First we provide a sketch of CoSMo’s system architecture which briefly explains the basic processing and service blocks which constitute the system. Then we formally document this architecture using standard UML diagrams.

3.1 Brief Description of CoSMo’s System Architecture

3.1.1 High-level Architecture of CoSMo:
Figure 1 shows a High-level Architecture diagram for the CoSMo System. This diagram gives a birds-eye view of the system. It shows the component placement within the framework.
The High-level Architecture has 3 basic components, namely, the Client side, the Data Processing Components and the Service Components. The Client Side depicts users with PDA, laptops or PCs. The Client Side is responsible for issuing the desired query through a User Interface. The external world also consists of the real world sensor network which is a mesh of sensors which will be ultimately queried for the measurement of the desired phenomenon. Processing components are the core of the system which handles the transformation and interpretation of the query within the System. Some of the constituent components are situation and knowledge aware and add to the cognition of the System. The Simulator component simulates queries input to it and also populates the Database with sensor data estimates. The Database component is a repository for storing sensor data, sensor network state representation and production rules used for query navigation.
3.1.2 Overview of CoSMo’s System Internals:

Figure 2 exposes more detail of the overall architecture of the system shown in Figure 1. It consists of the following basic components:

**Data Processing and Reasoning Components:**

The Data Processing components of the system are primarily responsible for the analysis and transformation of the query and data as it flows through the system. These components provide the core functionality and services by making intelligent decision
regarding query processing and data extraction or fusion. The database provides static modeling of the external traffic and sensor world while the simulator provides a dynamic modeling of the same.

3.1.2.1 Query Processor

The CoSMo system is capable of handling three types of queries namely, Value-based query, Event-based query and Hybrid query. The query processor maintains query templates in form of XML schemas. It takes the user query and on-the-fly converts it to an XML query according to the XML query template which defines the attributes, structure, constraints and semantics that the query needs to conform to. Communication between modules takes place in XML due to the platform independence and expressivity provided by it. This module extracts sub-queries from the user query and groups them as a compound query or a collection of simple sub-queries within the main XML query.

3.1.2.2 State Manager

State Manager takes processed XML query from query processor and has a rule based engine for judging the best component to handle the query. Depending upon the type of query and other special conditions it will invokes either the data model or the State Estimator or both. The State Manager adds to the cognitive element of the system. This component maintains tables which contain rules for routing the query to the appropriate component according to the attribute combination along with the ensuing decision, tables consisting of plans which depict legal combinations of the attributes, a table to record the confidence with which the query was satisfied and tables containing generic context of the query. These tables and others form a knowledge-base in the system from which the state manager learns and refines its decisions.

3.1.2.3 Database Manager

This component contains three main repositories:
Internal Representation Database:

This database maintains an internal representation of the sensor network. The model updates it whenever it receives updates from the sensor network about the network topology. The internal representation maintains a know-how of which sensors are currently present in the system, what attributes each one is sensing, their zone, their connectivity, battery levels, maybe the shortest route to each sensor with the current topology, delays in getting data back from the sensor, communication cost etc. This block collaborates with the Data Model which decides a set of sensors which would most efficiently observe the required attributes.

Estimates Database:

This database maintains the estimates of sensor data. To begin with, it contains training data from the simulator. Thus, this database contains values at a coarser granularity of time intervals. These estimates may be refined and data at interpolated time intervals is inserted into this database by the State Estimator. If the query requires sensor data at a particular time interval and the estimate for that particular time is already present in the estimates database, then in that case, the data value is picked up from this estimates database. The estimates database is also accessed by the Data Model. This database is updated by it as it receives fresh data from the sensor network.

Rules Database:

This database contains a record of all the attributes sensed by the sensors currently residing in the sensor net. It also encompasses different plans which are legal combinations of these attributes. It further includes production rules which are based on certain special conditions which may occur within a query and the matching plan as explained above. This database enlists the decisions which are a consequence of triggering the most appropriate production rule. New plans and rules may be added to this database on the fly as the system analyzes new queries and discovers new plans and rules.
3.1.2.4 Data Model

The State Manager routes the XML query to the Data Model according to certain rules. For example, in case of our traffic monitoring system, the query will be routed to this component if the user query is link-based and requires current value of sensed value at current time or if the query is an even-based query. The Data Model is responsible for creation of an “Observation Plan” [17]. This involves selecting particular attributes which will increase the confidence in the answer to the issued query at minimal cost. It also decides which nodes to be queried by accessing the Internal Representation Database for getting state information of the network. Quality (confidence/accuracy) and communication cost plays important role in selecting the sensors and attributes to query. It updates the internal representation and estimate database when it gets fresh information from selected sensors. The observation plan as well as the XML query is sent to the Message Mapper. Multi-source data aggregation takes place at the Data Model according to the sub-queries within the timeframe specified.

3.1.2.5 State Estimator

The State Manager routes the XML query to the State Estimator in case it is a path/link-based query requiring values within a time range or past or future values with medium and low confidence. This component converts the XML query into SQL query and queries the Estimate Database. It gets the range of values (for example, traffic delay values) and finds average. It interpolates in case the above returns a null set and finds the confidence. It may direct the query to the simulator in case this confidence level is lower than required. It then routes the result back to the State Manager.

3.1.2.6 Simulator

A simulator, in our case TRANSIMS [11], may be used to simulate the query when the state estimator cannot answer the query with required confidence by querying the
estimates database or by interpolating on the values from the database. The TRANSIMS Network representation renders detailed information about streets, intersections, signals, and transit in a road network [11]. This helps maintain the precision of the result at the cost of increased response time to answer the query. TRANSIMS also populates the Estimates database with sensor data values.

3.1.2.7  Message Mapper

This component exposes an interface to the hardware sensors from system software point of view. It handles message format conversion tasks between sensors and systems. This is an ideal place to perform format conversions for vendor-specific, multi-attribute sensors in a heterogeneous sensor network.

3.1.2.8  Real World Sensor Network Manager

The sensor network consists of nodes which sense particular attributes. It maybe a heterogeneous network which contains different sensor nodes sensing different attributes. The role of this sensor net would be passive in case of a pull based query. In case of an event-based or push-based query, the query is initially pushed into the network by the data model. When the sensor node receives the event from outside world, it will push the readings to the data model. Since we do not deploy hardware sensors, the Real World Sensor Network Manager takes over the task of simulating the request, execution of the query and response within the sensor network.

Service Components:
The service framework provides the foundation for the CoSMo system. The service components facilitate interaction within the system as well as that between the system and the external world. It characterizes CoSMo with scalability, reliability and genericity.
3.1.2.9 External Service Component (ESC)

This component exposes multiple interfaces to communicate with various client devices. Communication with these external devices can take place over HTTP or UPnP. The ESC acts as a gateway for the internal system. It publishes services provided by the system to the external world. It also interfaces with the service manager to receive service information and service change information once the system is up and running. It gets the entry point to the Query Processor component from the Internal Service Component or the UDDI registry and passes the user query to it.

3.1.2.10 Service Manager

Service Manager is the core for Service Middleware. Service Manager knows the system capabilities and it helps Internal and External Service Components to create a service database in which all the internal and external services are registered. It supports a user interface wherein the administrator can feed in service information. This information is used to populate the UDDI service registry. Using these services different processing components can communicate with each other. The Service Manager also publishes services with UPnP for clients which are within the intranet.

3.1.2.11 Internal Service Component (ISC)

The ISC maintains an internal private UDDI registry [19][20]. Service providers populate the UDDI service registry with services offered by their components. These components can then perform a lookup for a particular service. Detailed information about the service name, description of the service, type of service, calling semantics and access rules for each service is stored here.
External World:

3.1.2.12 Client Application in External World

The client side application consists of a client UI which allows the user to issue queries to the sensor networks which are intercepted by our middleware. The client application typically runs on a PDA or a PC. The client has two ways to access the services provided by the sensors: i) using a web-interface by connecting to the internet ii) discovering services using the UPnP protocol. The first choice requires the user to have access to the internet while the second choice requires the user to be within the intranet domain. The external world clients are classified as either normal users or administrator which determines the services that are visible to them.

3.1.2.13 Wireless Sensor Network

The sensor network is part of the measuring system. The sensor network consists of sensor nodes which sense particular attributes. We envision the CoSMo system to interact with a heterogeneous sensor network which contains a large scale deployment of sensor nodes sensing various attributes. The traffic network being a complex and dynamic environment requires the deployment of a large number of sensors for accurate representation. The otherwise tedious task of data collection in such a dynamic and complex environment is facilitated by wireless sensor networks. These tiny sensors are embedded into the road networks. They can then monitor the traffic patterns and other activities which spawn these patterns. Wireless sensors, though resource constrained, have the ability to operate in seemingly unreachable or hostile environments and provide advantages in terms of cost, size, scalability, and distributed intelligence. Currently, we do not deploy hardware sensors but provide a simulation of these sensors [36].

3.1.2.14 Traffic Network

The traffic network is the observed or measured system. The mesh of roads, vehicle demographics, activity locations etc. constitute a Traffic Network. These networks are part of socio-technical systems. Unlike physical systems, these are not only characterized
by physical laws but also affected by human behavior, regulatory agencies etc which adds to its complexity.

Communication between all components takes place using web services with SOAP over HTTP[19][31][55]. This thesis will explain the cognitively-inspired processing elements in the system while the companion thesis by Tupe [56] explains the service components of the CoSMo System.

3.2 Work-flow Diagrams

The architecture of the system which is a novel combination of the Model-driven, Data-driven, Cognitive and Service Oriented Architectures is an important artifact of this thesis which needs to be well documented to capture its complex structures and accurately model its behavior. Taking this into account, this chapter shows the “4+1 View Model” of the system architecture [39]. The 4+1 View Model allows various stakeholders to find what they need in the software architecture. System engineers can approach it from the physical view, then the process view; end users, customers, and data specialists can the logical view most useful; and project managers and software-configuration engineers can get a better insight into the system via the development view [39]. UML has been used to model the interaction between the system components due to its standardized notations and ease of visual representation.

3.2.1 Logical View

The 4+1's logical view supports behavioral requirements. It depicts how the system is decomposed into a set of abstractions and how relationships are modeled within the system. The following sequence diagram is useful in illustrating the flow of logic in the system, the relationship of the system elements and their behavior from a logical view [18][39].
Figure 3: Logical View of the CoSMo System.

This Sequence Diagram depicts the flow of control, relationship between System elements and in general the behavior of the CoSMo System.
The rectangular boxes at the top of the diagram represent the interacting entities, namely, objects, classes, use cases or actors. The dashed-lines are called object lifelines, representing the life span of the object. The thin stretched boxes on the lifelines are activation boxes which indicate processing is being performed by the target object/class to accomplish an interaction. Messages are depicted in the UML sequence diagrams as labeled arrows. Return values are indicated using a dashed arrow with a label indicating the return value or message. UML notes facilitate the addition of free form of text.

The interaction between the processing components of the CoSMo system as well as the repositories can be seen in Figure 3. For example, the Data Model sends a message to the Database requesting it for data regarding the sensor network state. The Database returns the required data which is modeled in form of the label dashed arrow. The self loop indicates that the Data Model processes this data to create an Observation Plan and in the next interaction sends it to the Message Mapper. The communication between these components takes place over web-services and its mechanisms. The communication between the actor once he/she enters the desired query with the system entry point component, i.e. the Query Processor, takes place over either UPnP or Web-services.

3.2.2 Process View

The process view describes the System’s processes and how they communicate [18][39]. It helps in gaining an understanding of the System’s concurrent processes and threads and their interactions. The following Activity Diagram depicts this view of the System:
Figure 4: Process View of the CoSMo System.
This Activity Diagram describes the Communication between different processes and threads within the CoSMo System.
The filled circle depicts the starting point of the activity diagram and the filled circle with a border represents the ending point. The rounded rectangles show the processes or activities that are performed. Diamonds represent decision points. Arrows represent transitions between activities and model the flow between the various activities. The text on the arrows represents conditions that must be satisfied to proceed along the transition. The thick bars represent the start and end of parallel processes. Thus, Figure 4 depicts the task flow within the system. For example, the user input query is converted into an XML query. The next task decides the destination of the query, whether it should be routed to the Data Model or the State Estimator. If the condition for Data Model is satisfied, the query will be routed to the Data Model over the transition shown and the “Create Observation Plan” task is activated. If the condition for State Estimator is satisfied, then the transition to the “Find Estimates” activity is triggered.

### 3.2.3 Development View

The Development view [18][39] describes the modules within the System and how they are deployed within the System. The following Package diagram gives an insight into this view:
Figure 5: Development View of the CoSMo System.
This Package Diagram exposes the modules within the CoSMo System.
UML Packages are a grouping of objects into sets of objects that provide related services. Packages are depicted as file folders. Figure 5 shows the Development view represented by a Package diagram for the CoSMo System. The System is mainly split into two main packages: the Service Package and the Processing Elements Package. The Service Package couples all objects related to the communication paradigm and provision of web services and UPnP services. The Process Element Package group together the processing components of the system which are responsible for transforming and handling the query and data flow through the system. These include components and libraries for the State Manager, Query Processor, Data Model, State Estimator, Simulator and Oracle System Database.

3.2.4 Physical View

The Physical diagram depicts the physical mapping of the components to the development environment. It describes the execution of the application on a cluster of computers [18][39]. This view takes into account nonfunctional requirements like availability, reliability, performance, and scalability [39]. The following Deployment diagram depicts this view of the System:
Figure 6: Physical View of the CoSMo System.
This Physical Diagram describes the physical mapping of CoSMo's system components with the development environment.
The cubes in the physical diagram represent the concept of a node. A node represents either a physical machine or a virtual machine node. Software components are deployed on these nodes and are indicated in the diagram. We envision CoSMo to be modeled as shown in Figure 6 when it will be physically distributed on a grid. For example, the UDDI registry may be deployed on a separate server while the processing component may be mapped to multiple nodes for increasing efficiency as well as availability of the system.

3.2.5 Scenarios

Completing the above description of the System, the following use cases walk through two scenarios. Usecase 1 depicts the query flow wherein the State Manager selects the State Estimator as the best component to further process the query. Usecase 2 demonstrates the case where the query is forwarded to the Data Model and hence out to the sensor network.

**Use Case I:**

**User Query:** Get delay on link 12 between time 8:50 A.M. and 1:20 P.M and get the result with a Medium confidence level.

Figure 7 shows the flow of the control through the System for this query:
1. As the first step, a user interface for query input is presented to the client. The user can select either an Admin or Normal User role. The user will be authenticated according to the role selected. On successful authentication, the user is prompted to select from authorized services and other query parameters such as the type of query, the start and the end link of interest, the zones that the chosen links are part of, the required confidence of the result, whether the query is for a value at current time or it is a time-range based query, the time range if relevant and the attribute-operator and required value of interest.
Figure 8: User Interface for user query input.

The screenshot of the User Interface shows the pertinent fields and the values entered in case of a Value-based Query requesting for the Delay Service on a link within a specified time-range.

Alternatively, the client may be provided with the likes of a Google map and asked to select the desired start and end locations. He/she may then pick the time or time range within which the delay on that link or path is required. The following screenshot demonstrates the UI in the form of a road map:
Figure 9: Map-based User Interface for user query input.
This screenshot depicts an alternative map-based UI to the CoSMo System, wherein the source and destination points can be selected by clicking on the map.

The latitude and longitude of the locations user has selected are calculated and posted to the server.

2. This user query is converted into a format understandable by the Query Processor which captures information particular to the user query. The above specified query gets converted to the following format:

User, Value, Delay, Link=12 And LinkZone=23 And Time > 31800 and Time < 48000 and Confidence=Medium.

Here, the time is specified in seconds elapsed after midnight.

3. The ESC passes the query string to the Service Manager. The Service Manager is responsible for routing the query to the appropriate component depending on the sensor
application. In the current implementation, the Service Manager invokes the Query Processor via a look up with the UDDI registry and passes the message string to it. Another entry point to the System is provided over UPnP services published by the Service Manager.

4. The Query Processor formats this message string into tagged XML stream according to a query template specified by an XML schema. Details of the query template can be found in Chapter 5. The Query Processor assigns query ids and forwards this XML to State Manager via access point lookup in the UDDI registry.

5. The web service access locations point to the gateway module for the succeeding component. The Gateway selects the best component from available distributed and duplicated components. The selection process depends on component workloads, business constraints and quality parameters.

6. The State Manager picks the best component to route the query to. It has a knowledge base in persistent memory which stores trigger rules and plans which aid the State manager in deciding between the Data Model and the State Estimator.

7. In this case, the compound query is routed to the State Estimator.

8. This compound query is converted into an SQL query which runs over the Estimates database. The query requires a value that holds true between 8:50 A.M. and 1:20 P.M. An average is taken over these values and returned to the state manager as the result of the compound query. If such a direct range is not present in the database, the State Estimator will interpolate or extrapolate over the available values in the database and calculate the confidence level of the result. If this confidence level is lower than the required value, then the query may be sent to the simulator. The simulator will simulate this query and send out the results. Currently, even if the confidence is lower than that required, the result and its confidence will be sent back to the State Manager as its best estimate.
9. The State Manager passes back the results to the Query Processor which in turn sends it to the ESC.

10. The external service component sends the query answer to the client.

**Use case II:**

**User Query:** Detect the appearance of a white color Camry with number plate JWG 1234. Send updates for the next one hour at the frequency of 15 minutes.

Figure 10 shows the flow of the control through the System for this query:
1. As the first step, a user interface for query input is presented to the client. The user can select either an Admin or Normal User role. The user will be authenticated according to the role selected.

On successful authentication, the user is prompted to select from authorized services and other query parameters such as the type of query, the start and the end link of interest, the zones that the chosen links are part of, the required confidence of the result, whether the query is for a value at current time or it is a time-range based query, the time range if relevant and the attribute-operator and required value of interest.

![User Interface for User Query Input.](image)

This screenshot of the User Interface shows the pertinent fields and the values entered in case of an Event-based Query requesting for the DetectCar Service on a link with the required car demographics. The UI allows specification of the requisite car demographics in the Attribute, Operator, Value fields, while facilitating the user to specify the frequency at which he/she requires updates and the duration for which the query should be active.

2. This user query is converted into a format understandable by the Query Processor which captures information particular to the user query. The converted query format for the above query is as shown below:
User, Event, DetectVehicle, Link=12 And LinkZone=23 And VehicleColor=WHITE And VehicleModel=CAMRY And ObservationFreq=900 And ObservationDuration=3600 And ReportingDeadline=200 And Confidence = High.

3. The External Service Component (ESC) passes the query string to the Service Manager. The Service Manager is responsible for routing the query to the appropriate component depending on the sensor application. In the current implementation, the Service Manager invokes the Query Processor and passes the message string to it. Another entry point to the System is provided over UPnP services published by the Service Manager.

4. The Query Processor formats this message string into tagged XML stream according to a query template specified by an XML schema. Details of the query template can be found in Chapter 5. The Query Processor assigns query ids and forwards this XML to State Manager via web service access location lookup in the UDDI registry.

5. For all the components, these web service access locations point to the gateway module for the succeeding component. The Gateway selects the best component from available distributed and duplicated components. The selection process depends on component workloads, business constraints, quality parameters etc.

6. The State Manager picks the best component to route the query to. It has a knowledge base in persistent memory which stores trigger rules and plans which aid the State manager in deciding between the Data Model and the State Estimator.

7. In this case, the compound event query is routed to the Data Model.

8. When the Data Model receives the compound query, it creates an “Observation Plan” which consists of choosing the attributes which need to be observed, the sensors which will observe these attributes, time frame within which the results should be received in order to be valid, the route that the data needs to follow from source back to Data Model
etc. All these parameters are computed by looking up the Internal Representation of Sensor Network Database which maintains know-how of the sensor type, sensor attributes, power-level, location, response time for each sensor and sensor connectivity. Thus, in case of the specified DetectVehicle query, the compound query is sent to the data model comprising of the sub-queries and their relationships specified in the conditions. The subqueries contain one main attribute each, such as VehicleColor and VehicleModel, their operators and the required values.

9. The observation plan in XML format is sent to the Message Mapper which is a rich interface between the Data Model and the Software Sensors. The Message Mapper converts the XML observation plan to a sensor understandable query format. We assume that our software sensors are capable of processing SQL queries. Thus, the observation plan is converted into an SQL query which will select the specified sensors and query them. When a sensor is queried, its power level is reduced by a fixed amount. This is reflected in the Internal Representation of Sensor Network database.

10. Since we do not have hardware sensors, the Real World Sensor Network Manager simulates the issuance and execution of the query in the sensor network.

11. Once the sensors send the data back to the Data Model, data aggregation takes place. This data also contains the associated queryid. The Data Model has a time frame specified in each observation plan within which it should get back the response from the selected sensors. The data model waits for this time period and gathers the values. The confidence level is a function of number of sensors returning the values, how many sensors out of the responsive ones have returned same values etc. For the same query-id, the data model aggregated the values for the encompassed attributes by taking into account the conditions of AND, OR etc.

In this case, we have multiple attributes namely, VehicleColor and VehicleModel. The detection of these individual attributes is indicated by the sensors and the aggregate result
suggesting the recognition of a white color Camry depends on whether the individual sub-queries detected the same vehicle.

12. The Data Model periodically returns the result of the detection of the event to the State Manager for the specified time duration.

13. The State Manager sends this result to the Query Processor which in turn returns the values to the ESC.

14. The Query Processor finally sends the result of the query to the client. For example, in case of the specified query, the client periodically receives status updates on the detection of the car.
Chapter 4
Query Model

In our architecture, users of the System input queries via a restrictive UI at the client machine. This query is transformed into a simple language that describes the situation and data the client wishes to collect and information on how to coalesce it. This conversion is performed before it is transmitted to the server. This query is then converted into a system understandable format, namely XML. The transformed query conforms to a generic query template structured as an XML schema. Details of this schema can be found in Chapter 5.

Our query language and the grammar implied by the XML query template, is capable of supporting queries in the Conjunctive Normal Form. The CNF can contain only propositional connectives such as ‘and’, ‘or’, and ‘not’. But logically, in case of compound events, our query language implicitly models the ‘+’ operator connecting atomic event symbols. At this stage of the prototype, nested queries are not supported, but grouping of the atomic symbols is certainly provided for.

At a coarser level, the queries can be classified into User and Admin queries depending on the type of service and the authorization level required. The User and Admin queries can be further split into Value and Event based queries as shown in Figure 12:
Thus, the queries supported by the query language can be assigned to the following query classes:

### 4.1 Value-based Query

This query takes as input a user query and returns a set of sensors satisfying the query or value reported by these sensors after computing a function $f$ on them. Thus, this class of queries request simple values over a certain group of sensors observing the same attribute or lying within the same area of interest. They may also request for a set of sensors satisfying a particular criteria as in the case of a diagnostic query. Depending on the time step at which the value for the phenomenon needs to be postulated, the query is either directed to the State Estimator or to the Data Model. In case the user requires the value at current time with high confidence, it will be routed to the Data Model in order to acquire fresh and accurate results. When the query requests for historical sensor data, requires a prediction into the future or is a time range based query, it will be routed to the State Estimator. Also, currently path-based queries are forwarded to the State Estimator. The State Estimator is capable of satisfying such requests with a combination of archival database indexing, interpolation and simulating predictions using a simulator.
A few examples of value-based queries are given below:

1. Get the Delay at Link 12 between 9:00 A.M. And 4:00 P.M.
2. Retrieve the Delay at Link 12 at 10:00 A.M. on the same day, the current time being 2:00 P.M.
3. Retrieve the Delay at Link 12 at 6:00 P.M. on the same day, the current time being 2:00 P.M.
4. Get the Delay on Path having Start Link 12 and End Link 15 at current time.
5. Get the Delay on Link 12 at current time with High confidence.
6. Get the set of sensors which have power levels greater than 0.5.

4.2 Event-based Query

In certain cases, it may be difficult to delimit the meaning of an “event”. There may be a grey area wherein the query may seem to lie between event-based and value-based classes. In our System, such queries form a separate class, termed as a Hybrid Class. Hence, to simplify the notion of an event, we define an event as “an atomic occurrence which can be answered by the sensors with an affirmation or negation”. Compound events are a combination of these atomic events wherein the final detection of this query depends on the detection of the atomic events. In our System, we define atomic events as certain attributes measured by the sensors. Detection of vehicle color, vehicle count, vehicle model, vehicle number plate are the atomic events currently defined for our System. Compound events can be composed using these atomic events. The detection of a car on the road which has white color and model Camry and number plate JWG7698 is considered as a compound event.

4.3 Hybrid Query

As explained previously, in case of certain queries the classification into value-based or event-based classes may not be evident. We group these queries into a Hybrid Class
according to the navigational patterns inherent in the query. CoSMo’s System architecture provides support for these queries, but currently we do not implement them. Two examples of the Hybrid class of queries are:

i) **Event Prediction:**

This category subsumes a class of queries which may be initially classified as event-based and routed to the Data Model but responsibility for processing a part of the query may be delegated to the simulator. In case of Event Prediction, the detection of the event will occur in the real world sensor network model, but the task of prediction of the event may be taken over by the Simulator model. Thus, this type of query illustrates a multi-branch flow within the System. Though this query is not particularly implemented in the System, we provide place-holders in the implementation and integrated reasoning in the design to be able to solve this situation.

ii) **Hypothesize-Verify:**

The *Hypothesize-Verify* class as we term it is an interesting class of queries. It demonstrates the potency of our architecture in handling queries which cannot be satisfied by a single diversion to either the Data Model or the State Estimator. For example, in a traffic jam detection scenario, the real world sensor network hypothesizes the occurrence of a traffic jam by judging the traffic patterns observed by selected sensors. This detection result may be sent to the Simulator model to simulate the best sensors which would help verify the detection of the event. This information is feedback to the real world sensor model to corroborate the occurrence of that event. This query class thus leverages CoSMo’s symmetric architecture to exemplify and solve a complex situation.

The query template has been designed with a view to implement the above queries. This template describes the query in terms of its sub-queries. Just like the higher level query, each of the sub-queries is assigned a unique identification number in the context of the enclosing query. The division of the query at this granularity allows individual sub-
queries to be routed to an appropriate component, independent of the other sub-queries. Thus, the State Manager may decide the best component by triggering rules for individual sub-queries. This component maintains the context of the query and the combination conditions for the sub-queries. It would then be responsible for integrating the results for each of the sub-queries to adjudicate the final query solution. Thus, this design consideration provides the hook for satisfying the above mentioned hybrid queries. More details about the query template and the flexibility it offers in modeling queries can be found in Chapter 5.
Chapter 5
Query Processor

5.1 Overview

The Query Processor is the point of entry for the query into the system. It is responsible for mapping the input query which is entered via a restrictive UI into an XML query. This component validates the query before it is disseminated to the rest of the System. It may be argued that this bridge between the client and the System may be a bottleneck single point of failure. But we choose to maintain a simple System design wherein the query needs to be validated only once at its port of entry. We can provide for redundancy and availability by replicating this component across the grid. The Query Processor component is rule-based wherein these rules enable the conversion of the input query into a XML query. The input query is transmitted from the UI in a format understandable by the Query Processor. The input query is parsed in order to extract its key properties. These properties are appropriately inserted into an instance of an XML schema or serialized to generate the output XML query. This generic XML schema which enforces a structure onto the output query is hereon termed as the Query Template.

5.2 Design

Goal: Convert user query into System understandable format with a minimum redesigning requirement for different queries.

The Query Processor transforms the input query into a System understandable XML format. It does so with the help of an XML query Template. The XML query template is derived from an XML schema. An XML Schema consists of components such as type definitions and element declarations. This schema is used to specify the constraints on
these components and validates the XML document which references this schema by checking whether the elements and attributes of the document are in conformance with these constraints.

XML is chosen as the language of communication between the individual components and has the following advantages over other data exchange languages:

i) XML is platform and vendor independent.

ii) XML is portable. Any language which can parse XML can use the information represented in XML.

iii) XML is already widely supported by most of the vendors.

iv) XML is extended to include tags and attributes specific to the system.

v) XML is easily readable and understandable.

The query template designed for our system is generic in nature and rich enough to encapsulate all the structural information required to represent value-based, event-based, user or admin queries. In the future, the query processor may select and load different query templates on the fly by creating dynamic assemblies. As of now, all our queries are adequately represented by a single generic template.

5.3 Implementation Details

Within the Query Processor component, serializable classes are created from the XSD or XML Schema Definition specified by the generic Query Template. Serialization is the process of converting the state of an object into a form that is suitable for transportation over web-services or across different processes/machines. XML serialization serializes the public fields and properties of a class or the parameters and return values of methods into an XML stream. When an instance of a class generated from an XSD is serialized, the generated XML conforms to the XML Schema [5]. The .NET XmlSerializable class is chosen to generate the required XML stream which adheres to the query template/XML Schema.
The Query processor thus parses the user input query injected into the system according to the attributes, conditions and special tags such as user role, query type and service type.

The generic query template, examples of input queries and the resulting XML queries are illustrated below.

**Query Template:**

A visual representation of the generic XML query template is as shown below in Figure 13:
This figure presents a schematic representation of the XML Query Template depicting its structure and the constituent processing tags. It also shows the one-to-one and one-to-many composition of tags specifying which tags can occur in a multitude and those that are strictly single occurrences. It represents a hierarchy in the tag structure with the Query tag as the encompassing root of the XML document. This template is generic enough to map all the System supported queries.

The Query Template is a generic XML schema in the sense that it has the ability to represent more than one query without having to be redesigned or the query processing engine to be re-programmed. Every input query enters the system in a fixed format which can be structured into this format for a complete specification of the query without any loss of information. For example, each query is specified using a role, a query type, the
service required, the location of interest and as a set of sub-queries. For different queries, only the values for these tags vary while the query composition remains essentially the same. Hence, the data within the query may change but the meta-data remains the same.

**Special Processing Tags:**

The query template defines the structure of the XML query that is generated from the input query. Figure 13 shows this generic Query Template. The boxes in the above illustration represent the XML tags. The Query tag is at the top-most level of the hierarchy and is the root tag of the XML schema. The XML Query Template thus allows the specification of special processing tags for the system such as the user Role, the query Type, and Service requested. These tags will at a latter stage help the System decide on the work flow for the query. The Query Processor generates a unique query id for identifying the query. It is important for the query to retain its identity as it traverses the system, undergoes processing and splitting into sub-queries to be diverted to different components of the system. This query id is the key in accessing the query context of any query currently being processed by the system.

The location tag subsumes the specification of the links and their corresponding zones for which the query is pertinent. As of now, the link ids as well as zone ids are specified in the user interface, but we visualize a sophisticated mapping component as part of our UI. Web based maps of the city such as Google Maps or Yahoo Maps may be used as a user interface. Infact, our system does have an alternative front end in the form of the Google Map, but it has not yet been fully configured to the requirements of our system. The user can specify the starting point as well as the destination by clicking on the city map. The latitude/longitude input format is then translated into UTM format. This choice of location is then represented in the form of link ids and zone ids via an internal database mapping. A snapshot of this map-based interface is shown in Chapter 3. For a link based query, the location tag will contain a single link and the zone to which it belongs. In case of a path-based query, as shown in case 2 in the next sub-section, the path will be
represented in the form of connected links forming the path. The connecting links included in the path specified by the starting link and ending link is essentially extracted from a data table that maintains information about the road links and their start and end nodes.

The QueryType tag is used to indicate whether the query is “simple” containing a single or no sub-query or if it is “compound” and thus if the system should expect multiple sub-queries and conditions. A compound query is that query which contains more than one attributes connected by a predicate such as AND, OR etc. This tag encloses the sub-queries that the user query is broken into as well as the conditions with which these sub-queries as associated with each other. The sub-query has a unique identifier in order to trace these sub-queries, as in the future we envision each of these sub-queries being split and individually processed by the most suitable component. These sub-query identifiers are also used while specifying joins on the sub-queries in the Condition section of the XML query. Each individual attribute gets assigned to a separate sub-query. Here, the Attribute tag indicates the sensor phenomenon to be measured or estimated, Operator assigns the arithmetic operator to act on the attribute and the Value tag is used to specify the required value for that attribute. Similarly, the Condition tags have a unique Condition Id, a PredicateType tag can be any conjunctive predicate which serves to colligate the sub-queries expressed within the Operand1 and Operand2 tags. We have restricted each condition to having only two operands, but a nested or associative query can be definitely specified with this level of association, for example, by specifying one of the operands as a condition and other as a sub-query, both operands as sub-queries or both operands as previously defined conditions.

For each of these tags, the Query Template specifies the set of allowed values that each tag can take. This is specified in the form of enumerations within the query tags. For example, the following snippet shows the allowed values of Service that the input query can request:
As seen in Figure 14, line numbers 118 to 123 denote a restriction on the value that the Service element of the XML query can specify. The only services permitted at the current stage are Delay, CountVehicles, DetectCar, DetectTrafficJam, GetPower and GetTopology. Similarly, other element values filled into the XML query are validated against this template. The valid XML query stream thus generated is transferred over the web service to the State Manager for further handling.

**Query Transformation Examples:**

Examples of the query transformation from user input format to XML format are given below for various types of queries supported by our System. A brief explanation will be provided for the first case and the remaining cases follow the same guidelines.

1. Case: Value-based query for delay on a single link:

   Input query string containing information submitted by the user:

   **User, Value, Delay, Link=12 AND LinkZone=23 AND Confidence=HIGH AND Time=900 AND ReportingDeadline=1200**

   Output XML query conforming to the Query Template is shown in Figure 15:
<xml version="1.0"?>
<Query xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Role>User</Role>
  <Type>Value</Type>
  <QId>34055</QId>
  <Service>Delay</Service>
  <Location>
    <LinkId>12</LinkId>
    <ZoneId>23</ZoneId>
  </Location>
  <QueryType type="Compound">
    <CQId>38323</CQId>
    <Subquery>
      <Id>Subquery.21290</Id>
      <Attribute>Confidence</Attribute>
      <Operator>=</Operator>
      <Value xsi:type="xsd:string">HIGH</Value>
    </Subquery>
    <Subquery>
      <Id>Subquery.42408</Id>
      <Attribute>Time</Attribute>
      <Operator>=</Operator>
      <Value xsi:type="xsd:string">900</Value>
    </Subquery>
    <Subquery>
      <Id>Subquery.3843</Id>
      <Attribute>ReportingDeadline</Attribute>
      <Operator>=</Operator>
      <Value xsi:type="xsd:string">1200</Value>
    </Subquery>
  </QueryType>
  <Condition>
    <Id>Condition.21290</Id>
    <PredicateType>AND</PredicateType>
    <Operand1>Subquery.21290</Operand1>
    <Operand2>Subquery.42408</Operand2>
  </Condition>
  <Condition>
    <Id>Condition.42408</Id>
    <PredicateType>AND</PredicateType>
    <Operand1>Condition.21290</Operand1>
    <Operand2>Subquery.3843</Operand2>
  </Condition>
</Query>

Figure 15: Output XML Query for Value-based Query Requesting Delay on a Link
This is the simplest kind of query in the system. The user enters the query information into the User Interface as shown in Chapter 3. This information is converted into a format as shown in the input query string specified at the beginning of the current case. The input format specifies that the query is issued by a normal user, it is a value type of a query, the service requested in that of delay, the location of interest is Link number 12 which lies in Zone number 23 and that the confidence of the result should be high. It also specifies the time parameters, namely, the time step at which the delay value is required and the deadline for reporting back this value. The above string created by the client side application is transmitted to the Query Processor. The XML Query Template schema is implemented in the Query Processor in the form of a serializable class. The input query is parsed according to rules based on the expected pattern in the input query. The parsed values are aggregated to populate an instance of this Query Template class and the object is finally serialized into an XML query string. Thus, in this example, we see a direct mapping of the input query to the output query shown in Figure 15. Line 3 shows the value of the Role XML tag is inserted as ‘User’, line 6 shows the value of the Service XML tag as ‘Delay’ etc. The attributes or observation conditions of Confidence, Time and Reporting-deadline are split into two sub-queries as indicated in lines 13 to 30. The conditions connecting these sub-queries are set in the Condition tags shown in line 31 to 42. In this query, the predicate of type ‘AND’ connects two subqueries as indicated in lines 33 to 35.

The next query cases follow the same procedure and explanation of each query is omitted for the sake of brevity.

2. Case: Value-based query for delay on a path consisting of multiple links

Input query string containing information submitted by the user:

```
User,Value,Delay,StartLink=1;EndLink=4 AND Confidence=HIGH AND Time=900 AND ReportingDeadline=1200
```

Output XML query conforming to the Query Template is shown in Figure 16:
Figure 16: Output XML Query for Value-based Query Requesting Delay on a Path
3. Case: Event-based query to detect a car with specified Vehicle Color and Model.

Input query string containing information submitted by the user:

\[ User, \ Event, \ DetectCar, \ Link=12 \ AND \ LinkZone=23 \ AND \ VehicleColor=WHITE \ AND \ VehicleModel=CAMRY \ AND \ Confidence=HIGH \ AND \ Time=900 \ AND \ ReportingDeadline=1200 \ AND \ ObservationFrequency=1000 \ AND \ ObservationDuration=6000 \]

Output XML query conforming to the Query Template is shown in Figure 17:
Figure 17: Output XML Query for Event-based Query Requesting Car Detection Event on a Link
Thus, the Query Processor acts as a port of entry into the CoSMo System and performs the function of translating the user input query into a common language agreed upon by the internal components. The Query Processor supports the Query Model described in Chapter 4 via the means of a flexible Query Template. As seen in this chapter, the Query Template contains special processing tags which help in query interpretation and serve as a basis for intelligent decisions later in the CoSMo System. This feature of the Query Processor facilitates inclusion of new queries in the Query Model with minimum possible re-designing.

The XML Query is then sent to the State Manager which is one of the key components in learning incorporated in our System. The next chapter describes this State Manager component in detail.
Chapter 6

State Manager

6.1 Overview

The State Manager is the driver module within the system. The State Manager is responsible for selecting the best model to process the input query from those available with our System. It is a rule-driven engine based on production rules. Decisions regarding the internal routing of the query are based on production rules stored in persistent memory.

6.2 Design

Goal: Maintain a balance between accuracy and cost of satisfying the query by making intelligent decisions about query routing.

The State Manager is a context and situation-aware component which is responsible for deciding the future course of the query; whether it should be handled by the State Estimator or the Data Model. This choice affects the accuracy of the result as well as the communication cost of obtaining the result. Navigational strategies for this component are built into its architecture by applying a rule-based model. The State Manager is driven by these production rules stored as a part of the persistent memory. This component facilitates cognition in the form of recognition and categorization of situations modeled by the queries, demonstrates the ability to make decisions and select among alternatives and has an element of planning built into its design. It also supports remembering and reflection by logging traces of its activities. As a side effect of using a database to store the production rules, it deploys the single learning method; new tasks maybe performed without having to be redesigned or reprogrammed. It simply entails discovery and
addition of new rules either manually by the Admin or dynamically by the System itself. The State Manager is designed to learn from experimentation by comparing the confidence levels of the results it obtains from both the components. Table 1 describes the cognitive features present in the current version of the State Manager.

### 6.3 Implementation Details

Figure 19 shows a flowchart of the logic used by the State Manager to route queries. Before we look at this flow, it would be useful to discuss some of the key concepts used in the State Manager, namely, Attributes, Plans and Rules.

**Attributes, Plans and Rules:**

The Query processor sends the constructed XML query to the State Manager. The State Manager extracts the required attributes from the XML query. This component indexes into the ATTRIBUTEINTEGERREP data table to find the corresponding integer representations for the extracted attributes. The structure of this table is explained in Chapter 11. The integer representations are powers of 2. When represented in binary, each bit can represent one sensor attribute. For example, Power is represented by integer 2 or binary 0000 0010 whereas Delay is represented as integer 1 or binary 0000 0001.

Plans are effectively a combination or a logical OR of the sensors attributes which may occur in a query. For example, a plan containing Delay and Power will have a value of 3 or 0000 0011. These plans are stored in persistent memory in the PLANATTRIBUTE data table described in Chapter 11. Plans are retrieved by applying a logical AND to check for presence of the required attributes. In case that the attribute combination specified within the query, is a subset of another plan, more than one matching plans may be found. In this case, the perfect match if present or then the next best match depending on the attributes is considered. In case no matching plan is found in the data table, a new plan is constructed and written to the database. The State Manager is rule-driven wherein the production rules are stored in a persistent data store for ease of addition, deletion or modification. A production rule is a concatenation of the user role, query type, service, a plan as stated previously and some special conditions such as whether the query is link or
path based, whether it needs high confidence or low or whether it expects values at current time or future/past time etc. A sample of the Rules data store is shown in Figure 18:

<table>
<thead>
<tr>
<th>RULESTRING</th>
<th>DECISION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Admin@Command@GetPower@Plan9</td>
<td>DataModel</td>
</tr>
<tr>
<td>Admin@Command@SetPower@Plan9</td>
<td>DataModel</td>
</tr>
<tr>
<td>Admin@Command@GetPower@Plan5</td>
<td>DataModel</td>
</tr>
<tr>
<td>User@Value@Delay@Link@Plan1 @(Confidence=LOW)</td>
<td>Interpolator</td>
</tr>
<tr>
<td>User@Value@Delay@Link@Plan1 @(Confidence=MEEDIUM)</td>
<td>Interpolator</td>
</tr>
<tr>
<td>User@Value@Delay@Link@Plan1 @(Confidence=HIGH)</td>
<td>Interpolator</td>
</tr>
<tr>
<td>User@Value@Delay@Link@Plan1 @(Confidence=LOW, TimeRange=True)</td>
<td>Interpolator</td>
</tr>
<tr>
<td>User@Value@Delay@Link@Plan1 @(Confidence=MEEDIUM, TimeRange=True)</td>
<td>Interpolator</td>
</tr>
<tr>
<td>User@Value@Delay@Link@Plan1 @(Confidence=HIGH, TimeRange=True)</td>
<td>Interpolator</td>
</tr>
<tr>
<td>User@Value@Delay@Link@Plan1 @(Confidence=HIGH, CrtTime=True)</td>
<td>DataModel</td>
</tr>
<tr>
<td>User@Value@Route@Path</td>
<td>Interpolator</td>
</tr>
<tr>
<td>Admin@Command@SetPower@Plan9</td>
<td>DataModel</td>
</tr>
<tr>
<td>Admin@Command@GetTopology@Plan5</td>
<td>DataModel</td>
</tr>
<tr>
<td>User@Value@Delay@Link@Plan3 @(Confidence=LOW)</td>
<td>Interpolator</td>
</tr>
<tr>
<td>User@Value@Delay@Link@Plan3 @(Confidence=MEEDIUM)</td>
<td>Interpolator</td>
</tr>
<tr>
<td>User@Value@Delay@Link@Plan3 @(Confidence=HIGH)</td>
<td>Interpolator</td>
</tr>
<tr>
<td>User@Value@Delay@Link@Plan3 @(Confidence=HIGH, CrtTime=True)</td>
<td>DataModel</td>
</tr>
<tr>
<td>User@Event@CountVehicles@Plan2</td>
<td>DataModel</td>
</tr>
<tr>
<td>User@Value@Delay@Plan1 @(Confidence=HIGH, CrtTime=True)</td>
<td>DataModel</td>
</tr>
<tr>
<td>User@Event@DetectCar@Link@Plan11 @(Confidence=HIGH, CrtTime=True)</td>
<td>DataModel</td>
</tr>
</tbody>
</table>

**Figure 18: The Rule Base.**

This cut shows the format of the production rules for the Rule-driven Engine which are a concatenation of the user role, the query type, the service, extracted attribute plan and special conditions. The rightmost column presents the navigational decision corresponding to the rule.

The rule strings can be seen in the left column while the consequent routing decisions are listed in the right column. Thus, these rules are activated when a particular situation or query is recognized and the corresponding decision is triggered. The rule-driven engine and the matching of these rules is explained next.
The Rule-driven Engine:

The following figure shows the Rule-driven Engine implemented within the State Manager.

The State Manager communicates with a Rule base which is a persistent data store for housing production rules. The initial set of rules is populated into the Rules database at System start up time by the administrator. The production rule has a pattern which reflects those features which distinguish one query class from another. The rule matching
The algorithm beginning from the point that the XML query enters the State Manager can be described as given below:

i) The State Manager maintains a working memory in which the XML query is parsed using the XQuery language in order to extract the key distinguishing features of the query. Along with the processing directives in the query such as role, query type, service, link or path, the attributes specified within the query are extracted.

ii) The integer representations for each of these attributes are obtained from the corresponding data store and are combined with a logical OR.

iii) The integer obtained depicts a Plan string consisting of a combination of these attributes. Matching Plan(s) are selected from the data store containing various attribute Plans.

iv) If more than one Plans is matched, check if there is a perfect match of all attributes. If yes, select this Plan as the matched Plan, else apply a Conflict Resolution Strategy such as the First Applicable or the Most Specific Plan strategy. The First Applicable Strategy fires the first applicable rule and allows control over the order in which rules fire. This is the strategy we have implemented [25]. The Most Specific Strategy is based on the number of conditions of the rules. From the conflict set, the rule with the most conditions is chosen [25]. We support the Most Specific Strategy via the logical OR mechanism. The matching of the Plan uses the logical AND mechanism which allows us to check which other Plans contain the required attributes.

v) If a perfect match is not found in the above step, then the best plan according to the above listed strategies is picked up and the exact new Plan is written to the Plans data store.

vi) A Rule string is constructed from the extracted processing directives and the Plan thus obtained in the working memory.

vii) The matching rule(s) are selected from the Rules base. This may result in a conflict set.

viii) The best match is found by applying a Conflict Resolution Strategy.
ix) If a match is found, then the rule triggered governs which component (State Estimator or Data Model) should further handle the query.

x) In case no matching rule is found, the query is by default sent out to both, the Interpolator as well as the Data Model for further processing. The rule is added to the Rules data table on-the-fly. The component which returns the query result with higher confidence will be recorded in the data store as the decision for this rule string.

In general, the event-based queries, link based queries and queries which needs to be answered with 100% confidence are directed to the Data Model whereas value-based queries which require future values, need to index into the past or path based queries are diverted to the State Estimator. The intersection set of these queries can be sent to both the components in case the corresponding rule-decision pair has not yet been discovered.

Thus, the State Manager has rudimentary reactions to each type of query built into its architecture. These built-in reactions help this component formulate a strategy for directing the queries to the most appropriate component [1]. Once again, here the focus is not on building out impressive algorithms, but rather on developing a pluggable rule and case based approach for demonstrating intelligent navigational strategies, modifiability and learning.

The next chapter describes the Data Model, which is instantiated as a result of the State Manager’s decision to send the query out into the real world sensor network. We shall look at the State Estimator component which reflects the State Manager’s alternative decision in Chapter 10.
Chapter 7
Data Model

7.1 Overview

The architectural model depicts a symmetrical design wherein the Data Model interacts with the real world while the Interpolator is closely associated with the internal simulation of the real world. The Data Model is responsible for fetching fresh data from the sensor network. The Interpolator has been mainly designed to save on communication costs of sensor network, reduce the response time of the query and give an estimate of the sensor values. On the other hand, the Data Model focuses on fidelity of the results by retrieving data from the actual sensor network itself. Thus a balance of communication cost against accuracy is maintained in the system.

7.2 Design

Goal: Reduce the cost of communication and increase accuracy of sensor readings by making intelligent selection of sensors and by maintaining the current state of the sensor network.

The Data Model handles queries which require 100% confidence results. Sensors sample the physical phenomena being measured at a discrete interval of time. A database is a store of true data; hence a sensor network which can only approximate the sensed phenomena cannot be treated as a definitive source of this data. [21] Thus, when we say that the Data Model can handle queries which require 100% confidence we imply that these queries need to be satisfied with the best possible values which can be obtained from the real sensors themselves. As of now, sensors have limited or no storage and hence we assume that results from past queries cannot be obtained by querying the
network. Hence only queries for values at current time are routed to this component. Another case that the Data Model handles is that of Event-based queries wherein the sensor network keeps sensing the phenomena for the requested duration of time and sends updates at the desired frequency. These are typically modeled as long running queries in the sensor network.

The different states during the execution cycle of the Data Model are as depicted in figure. When the query enters the Data Model, it extracts the subqueries and enters the Processing state wherein it creates an Observation Plan for each subquery by selecting sensors for each sub-query. Once the plan is created, it is sent out into the network via the respective interfaces. After sending out the subquery plans into the network, the Data Model enters a Wait state for the query and goes into the Data Acquisition Phase wherein it collects values from the sensors. After a particular deadline, the Data Model moves out of the Data Acquisition phase into the Data Aggregation phase wherein it combines the results of all sensors for each of the subqueries to generate a cumulative result for the query as a whole. It enters the Confidence Calculation phase and updates the databases in the next phase and sends the result back to the State Manager. It re-enters the Start Phase and waits for the next query to be issued.
Figure 20: State Diagram for the Data Model.
This graph illustrates the sequence of legal states that the Data Model may reside in and the transitions which lead to the state change.

Thus, from the design standpoint, the Data Model satisfies the aim of maintaining accuracy versus communication balance by:

- Maintaining an updated internal copy of the real sensor network state which helps in deciding the sensors which should be queried.
- Constructing an Observation Plan for the attributes included in the query. The Observation Plan specifies the sensors which will be responsible for measuring the phenomenon specified in the sub-query.
7.3 Implementation Details

As enumerated above, the Data Model handles queries which need 100% confidence, are issued for current time sensor values and event based queries. Here we assume that only link-based queries will be routed to the Data Model via the State Manager. Path based queries are handled by the Interpolator component.

The Data Model is responsible for creation of the “Observation plan” [21]. This involves selecting particular sensor attributes which will increase the confidence in the answer to the issued query at minimal cost.

The State Manager passes the XML query string to the Data Model as given to it by the Query Processor component. It is worth noting here, that the intelligent extraction of relevant data from the XML query/document is easily facilitated by the XQuery language. XQuery is based on XPath expressions. XPath models an XML document as a tree of nodes. The benefits of XPath are as follows [4]:

- Queries are compact.
- Queries are easy to type and read.
- Syntax is simple for the simple and common cases.
- Query strings are easily embedded in programs, scripts, and XML or HTML attributes.
- Queries are easily parsed.
- You can specify any path that can occur in an XML document and any set of conditions for the nodes in the path.
- You can uniquely identify any node in an XML document.
- Queries return any number of results, including zero.
- Query conditions can be evaluated at any level of a document and are not expected to navigate from the top node of a document.
- Queries do not return repeated nodes.
- For programmers, queries are declarative, not procedural. They say what should be found, not how it should be found. This is important because a query optimizer must be free to use indexes or other structures to find results efficiently.
XPath is designed to be used in many contexts. It is applicable to providing links to nodes, for searching repositories, and for many other applications.

When the compound query (or in the future, a sub-query not satisfied by the Interpolator) is received, it goes into the “Make Observation Plan” state where an “Observation Plan” is generated. This plan is created taking into account the Role of the user (Admin/User), the Type of the query (Value/Event/Command) and the Service that is required. A routine corresponding to the service is invoked in order to create a customized observation plan. Currently the services supported by the Data Model include GetDelay, GetVehicleCount for User Value type of queries, GetVehicleCount for Event-based User queries, GetPower and GetTopology for Command Admin type of queries.

Figure 21 shows the diagrammatic representation of the Observation Plan Schema:
This schematic representation depicts the structure and composition of the XML Observation Plan. It represents a hierarchy in the tag structure with the ObservationPlan tag as the encompassing root of the XML document. This Observation Plan specifies the group of most efficient sensors to query for each attribute, the observation conditions and time deadlines along with zone information.

The above plan is customized according to the query that needs to be sent out into the network. Creation of this observation plan involves: i) choosing the attributes which need to be observed to satisfy the particular query, ii) the sensors which will observe these attributes, iii) extracting time frame within which the results should be received in order to be valid, iv) the route that the sensor data needs to follow from source back to the base station. To begin with, the Data Model chooses the attributes to be observed for the specified service and specified query from a table of primary attributes and finds the
supporting conditions for the primary attribute from a conditions table and fills it into the observation plan. Once the attributes are selected, the best sensors to observe these attributes are selected from the Internal Network Representation Data Table. As its name suggests, this table maintains a know-how of which sensors are currently present in the sensor network, the attributes each one is sensing, the sensortype, their zone, connectivity and battery levels, the shortest route to each sensor with the current topology, delays in getting data back from the sensor, communication cost etc. Based on this information, sensors are selected to observe these attributes using the following rule-based algorithm:
The flow-chart given in Figure 22 can be explained as below:
Select sensors on the given link which have minimum delay and have power above threshold

   If the above selection returns a null-set then
   Select sensors on the given link by removing the battery-constraints
   If no results are returned, it means all sensors for that particular attribute
   (for the particular link) are dead, then
   Select alternative links to query
   For the time being we try querying all sensors lying in
   the same zone
   If above set still returns zero sensors, then
   Choose an alternate attribute from the alternate
   attribute table is available and repeat the algorithm

Only a subset of sensors defined by a sensor number threshold is selected from the number of sensors returned. The query for fresh data will be selectively issued to only these chosen sensors out of all the eligible sensors. Thus, a balance is maintained between accuracy as well as communication cost. The query is then sent to the Message Mapper component which handles the conversion of this XML Observation Plan into a sensor understandable format. This sequence of steps which contribute towards the selection of the best sensors to query is depicted in Figure 22.

The data model spawns off an asynchronous per-query thread along with the observation plan and goes into a Wait state for a period of time specified by the reporting deadline. During this time, the Data Model keeps gathering resulting values of the subqueries sent back to it by the sensor network. Once this deadline expires, the Data Model thread handling the query comes out of its wait state and moves into the Aggregation Phase. It begins aggregating the multi-sensor values collected for each of the subqueries into the final result for the query. Sensors observing different attributes or phenomenon return their results in any order. The values arriving at the Data Model after the deadline elapses are discarded. Only sensor results which arrive within the reporting-deadline are considered in the aggregate and the remaining ones are discarded. Multiple queries are
concurrently processed in the Data Model and issued to the sensor network. Hence we assume that the sensor has the ability to know the id of the encompassing query for the subquery it is currently processing. Thus, for the same query-id, the Data Model fuses the sensor values for different attributes by combining them taking into account the conditional operators such as AND, OR etc. This is possible since the Data Model maintains a hash of the query context once the query has been sent out into the network and thus is able to reconstruct the query from its subqueries. Once the result for the query is obtained, the variance and the standard deviation for the aggregated sensor results are calculated. Confidence is then calculated for a value based query using the Chebyshev’s formula for Confidence interval calculation. This confidence is thus based on the number of sensors which were supposed to return values, number of responsive sensors which actually returned the value for the sensed phenomenon and the degree to which the sensor values agree with each other. The final aggregated result of the compound query as well as its confidence level (classified as LOW, MEDIUM or HIGH) is sent back to the State Manager. As of now, the result is sent back to the State Manager irrespective of the required confidence level. The implementation makes provision for a future extension wherein, if the obtained confidence level is lower than the required level then the result may be discarded and the cycle can be started again with a different choice of sensors or with caching of results returned by the sensors which replied after the stipulated time frame. The fresh data values so obtained are written to the Internal Representation of Sensor Networks and tagged as updated by Data Model.

The handling for Event-based queries, though consistent with that of Value-based queries, needs explicit handling in terms of the subscription time for the event as well as the frequency with which the user requires updates. These time frames are in addition to the reporting deadline for individual subqueries within the event. The observation plan provides for specification of these time frames in milliseconds within the following tags:

- `<reporting-deadline>5000</reporting-deadline>`
- `<observation-frequency>1000</observation-frequency>`
- `<observation-duration>3000</observation-duration>`

An example of the Observation plan for an event-based query is shown in the figure 23:
Figure 23: Observation Plan for an Event-based Query.

This XML Observation Plan output illustrates the structure of an Event-based query with the corresponding values inscribed within these tags which suggest the attributes to be observed and the group of sensors which will sense these attributes.
The Event-based query is sent out just once into the Sensor Network, but the Data Model keeps a track of the time periods at which it needs to aggregate the sensor results. The Sensor is assumed to have enough memory and processing power to keep track of the query-id, the duration for which it needs to observe the specified attribute as well as the frequency at which it needs to send back the results of the measured phenomenon. It must be noted that, only sensors which detect the particular phenomenon send back the results. For example, if the subquery is “VehicleColor = WHITE”, only the sensors which detect a white color vehicle will notify the Data Model about the detection. The Data Model then processes the sensor results valid within the stipulated time frame and sends the decision back to the State Manager along with the confidence in the answer. The confidence calculation in this case is currently taken as a simple probability depending on the number of sensors which were employed in observing the phenomenon and the number of sensors which actually detected the sub-event. Confidence of the occurrence of the event can be computed as a function of the presence of the detected sub-events, their relative importance in the query. This is computed based on pre-assigned weights assigned to individual sub-events when they occur in a combination. For example, for a query “Detect the appearance of a car with Color=White And Model=Camry And NumberPlate=JWG7698”, there are three sub-events specified by Color, Model and Number Plate. The weights are pre-assigned to this combination with Number Plate having the highest weight, Model having an intermediate weight and Color having the least weight out of the three according to the importance of the attribute (sub-event) in deciding whether the event was detected. If a perfectly matching combination is not found, then in the future a search can be implemented through the structure maintaining the attribute combination against weight hash, to find the closest match. The total confidence of the event query is then computed as a mutually exclusive product of the individual sub-event confidence levels. As in the case of Value-based queries, the result along with the obtained confidence is sent back to the State Manager.

Thus, the Data Model is a rich component depicting intelligence, planning and perception among other cognitive capabilities, which partakes in creating an internal representation of the real world sensor network while bringing in fresh data into the System. It does so
in a situation-aware manner with a goal to save on communication cost while maintaining the accuracy of the result.

The next chapter elaborates on the Message Mapper Interface which is responsible for translating the XML query into a format understandable by the sensors.
Chapter 8
Message Mapper Interface

8.1 Overview

The Message Mapper acts as an interface between the Data Model and the Sensor Network. The basic task of this interface is to translate the XML query or observation plan into a language which is understandable by the sensors. Thus, the Message Mapper may translate the XML query into a Tiny DB query. In our System, we do not deploy hardware sensors but have created a simulation emulating software sensors. We assume that these software sensors process SQL queries. This component thus exposes an interface to the current software sensor simulation and hardware sensors in the future. It handles message format conversion tasks between the system and the sensor network and would be an ideal place to handle message format conversions in case the sensors are supplied by different vendors. The query may also be appended with more information which would ultimately help individual multi-capability, cross-platform, vendor-specific sensors to interpret the XML query in terms of their own dialect. An example of such a global scheme is shown below in Figure 24:
Figure 24: Global Schema for Sensor-specific Query Parameter Interpretation
8.2 Design

*Goal: Maintain a clean interface between the System and the Sensor Network.*

The emphasis in building this component is not so much on the implementation as much as that of illustrating the purpose of a component to map between entities communicating in different languages. It demonstrates the importance of having a clean interface to translate queries as they traverse across the boundaries of the CoSMo System into the sensor network.

8.3 Implementation Details

The Message Mapper takes an XML Observation Plan string as its input. This component employs the XQuery language to extract data from the Observation Plan. The XML Observation Plan is split into sub-queries and the data from the query is converted into parameters in order to bind to parameterized SQL queries. A sub-query is created for each of the sensor groups specified within the Observation Plan. For each of these sub-queries, examples of parameters would be the attribute observed by this sensor group, the operator which needs to be applied to this attribute, the attribute value, the list of sensors, various time parameters, observation conditions etc. These SQL queries are then virtually injected into the Sensor Network. Once again, we do not have hardware sensors, but have setup a simulated environment. The sub-query parameters constituting the query are then forwarded to the Real World Sensor Network Manager which simulates and manages the sensor network.

Thus, the Message Mapper is a rich interface between the CoSMo System and the real world sensor network which provides a clean place to convert queries into sensor understandable format and also addresses the vendor-specific nature of these sensors. The next chapter looks at the Real World Sensor Manager in detail.
Chapter 9
Real World Sensor Network Manager

9.1 Overview

The sensor network is a heterogeneous mesh consisting of sensor nodes with diverse capabilities. The sensor nodes are pre-selected by the Data Model. The Data Model knowing the network topology may decide on the best route to be taken to propagate the query to the sensors and route the data back. The sensors on interrogation may also send updates to the model about their connectivity in case it has changed. This is one way the data may be routed back to the model: under complete control of the model. Another way is for the model to delegate the responsibility for route selection to the sensors which may maintain the best paths in their routing tables or according to any standard routing algorithm. In-network Data aggregation [45] may also take place to save on the number of data messages sent back to the model.

In case of an event-based or long-running query, the query is initially pushed into the network by the System. When the sensor node receives the event from outside world, it will push the readings back to the model. There may be some events the prediction of which can be taken over by the Data Model and some which require pushed updates from the sensors. For example, in case of a query by the police department to find the location of a particular car if it appears on the streets and track the route of the car, a combination of both can be used. This may be an event which may happen at any time and the reading is expected by the user when it actually happens. In this case, the value based query strategy of the model may not work as this event with all the combinations of attributes may not be monitored by the sensors. For example, there may be some sensors monitoring color, some others vehicle ids and vehicle make. The appearance of a white color car of the particular make is an event, details of which need to be pushed back to
the model. Now knowing the zone, speed, direction of the car and also the network topology, the model may be able to predict the future routes.
But there may be cases where the event cannot be predicted by the model. For example, if there is an accident on the street or a traffic jam. Such events have to be pushed by the sensors to the system.
As mentioned earlier, since we do not deploy a hardware sensornet for the purpose of this project, a simulator has been employed to mimic the distribution of the queries to the sensor network. TRANSIMS, a traffic simulator is used to emulate the software sensors. The Real World Manager is mainly responsible for issuing the query received from the Message Mapper out to the currently simulated sensornet and for collecting the sensor values from the network and forwarding them to the Data Model.

9.2 Design

Goal: Model a real sensor network to the closest possible degree.

The complexity of this particular component arises due to the fact that we do not deploy a real Sensor Network and we need to mimic the asynchronous nature of the queries and sensor values on the System side as well as the network side. This component needs to provide robust handling for multiple queries, their asynchronous arrival and query issuance. It facilitates the emulation of the asynchronous and interleaved manner in which the sensors process these multiple queries and send back values of the measured phenomenon. This component focuses on providing a close modeling of the actual sensor behavior with careful design of response times, as well as simulating the real world sensor accuracy and transmission losses.

9.3 Implementation Details

TRANSIMS, is part of the multi-track Travel Model Improvement Program. TRANSIMS is a set of analytical and simulation models and related databases which provides
information about synthetic population. This information includes household structure and demographics, Activity locations, times, and durations and the number of trips between activities, including route plans and execution of the route plans in the transportation network [11]. The sensor data which populates the database is thus sourced from TRANSIMS.

The Real World Network Manager component receives the SQL query from the Message Mapper interface. This component communicates with the System Database containing a Real World Sensor Network Data Table and supporting tables specifying information about links, sensor nodes as well as vehicle demographics. This data table is populated by TRANSIMS with values obtained at particular time-steps. As of now, since TRANSIMS is a batch processing utility, the data tables can be updated occasionally to reflect fresh data. The queries are issued out to the database over a Generic Data Access Layer. Once the values are extracted from the representative data tables, the values are emitted back to the Data Model. This emission of values to the Data Model is modeled by implementing a scheduler based on a priority queue ordered on time values picked by each sensor. First, sensor values from the returned records are randomly selected and a random time value is picked from a time range which is marginally outside the reporting deadline. The scheduler maintains a global priority queue which enqueues all the sensor values received for all the issued sub-queries belonging to the same query. A threaded timer takes a peek at the time value chosen by the record at the front of the queue. If the timer decides that it is time or past the time to send that sensor value, the Real World Network Manager will immediately forward this value to the Data Model. This serves as an emulation of the asynchronous nature of the sensors as well as sends some values after the reporting deadline elapses, thus simulating unreliable sensors and lossy data transmission. As mentioned before in Chapter 7, the Data Model does not aggregate values for sensors which send these results after the reporting deadline. This reduces the number of sensors reporting the same result and hence lowers the confidence level. This implementation thus attempts at imitating a real, lossy sensor network. In case of Event-based queries, values for the sub-events are fetched from the Real World Network data table representation at the required observation frequency specified and transmitted to the Data
Model. This is done for a time-span equal to the life-time of the long running query. This life-time is extracted from the observation duration field specified within the original query. The Data Model constructs the bigger picture by aggregating the results of these sub-queries to get the result of the final enclosing query. For example, if the query requests the detection of a white color Camry, the result of the detection of a white color car and that of detection of a Camry are combined to answer the compound query. Here the Real World Sensor Network Manager is responsible for emitting the marked car ids fitting the criteria to the Data Model. It emulates software sensors and observes the phenomenon at the requested observation frequency and transmits it to the Data Model. Thus, the client receives multiple answers for the same query, but at the required time period.

Thus, the Real World Sensor Network Manager tries to model the asynchronous nature of the sensor network to the closest possible degree. From Chapter 7 to 9, we have seen the flow of control within the CoSMo System in the case that the State Manager chooses the real world sensor network model to satisfy the query. In the next chapter, we shall look at the flow of control when the State Manager chooses Estimation Model as the means to fulfill the query instead of routing it to the real world sensor network. The next chapter gives a deeper insight into the design and implementation of the State Estimator.
Chapter 10
State Estimator

10.1 Overview

The State Estimator provides the System with the ability to predict future values, index into the past as well as interpolate over a range of values. The State Estimator is supported by TRANSIMS, a traffic simulator which can be used to emulate software sensors. This component complements the Data Model by generating estimates instead of querying the sensors for fresh data. It thus helps reduce the power consumption, by decreasing the number of queries that are actually sent out to the sensor network. Hence, a balance of accuracy and cost of communication is maintained in the system.

10.2 Design

*Goal: The goal of the State Estimator is to provide the ability to index into past values, predict future values and interpolate over a range of values and give the closest possible estimate of the required phenomenon.*

The State Estimator has three basic design elements, namely, the Interpolator, the Estimates Database and the Simulator. These three elements are incorporated to maintain a certain level of accuracy, to reduce the response time required to answer a query and to generate and predict values without having to send the query out into the sensornet. Thus, this component incorporates cognitive architecture capabilities such as *prediction* and *monitoring*. *Planning* is an integral part of the State Estimator, depicted via rules for selecting between the Estimates database, Interpolator, and the Simulator in order to satisfy the query with required accuracy and response time [1].
10.3 Implementation Details

The State Estimator references two main database tables: PathEstimates and LinkEstimates. The schema of these data tables is explained in Chapter 11. These tables are initially populated by TRANSIMS and over the course of time can be refined by the State Estimator and Data Model. The XML query is routed to the State Estimator in case the query requires values of a phenomenon on a path (collection of links), past values or requires a prediction of these values. If the query is link-based, the State Estimator queries the LinkEstimates table in order to extract the values required at the specified time step. If this value is not found in this knowledge base, then a linear interpolation is run on the values in the vicinity of the missing time step and is returned to the State Manager as the query result. This interpolated value is stored into the estimates table. If the query specifies a time-range for the estimates, then a simple average of the values constitutes the query result. As mentioned before, simple mechanisms are used in this System and better algorithms and functions can be plugged into its place in the future. The confidence for the estimated query is calculated and sent to the State Manager as well as stored into the data table in case of interpolation. The value is flagged as updated by the State Estimator. Another nuance to be noted here, is that the System considers the values given by the Data Model to have a higher confidence or preference over those estimated by the State Estimator due to the freshness of the data. The Data Model during the course of its run, may also update the estimates table and flag it appropriately. In the estimates table, only the Data Model is allowed to update values written to by the State Estimator but not vice-versa.

The same procedure is followed in case of path based queries. The difference is that in case of path-based queries, not only temporal interpolation, but also spatial aggregation over all the constituent links is performed to get the final result for the query. Similarly, the confidence of the result is an aggregate of individual link confidences. In the future, if the confidence is lower than required confidence, then the interpolator can command the Simulator to generate the values at the required time step. The Simulator provides the
System with an important ability to simulate values which could hold true at a future point in time.

Figure 25 summarizes the sequence of steps followed by the State Estimator on receiving an input query. These steps ensure the best possible estimates for the given query.
The flow seen in Figure 25 and the inherent logic in following this particular sequence of steps is explained below:

Check if the value (delay or vehicle count) at the required time step is present in the Estimates data store

   If the value at the required time step is found, then
      Extract that value and its corresponding fields giving the model which last updated the value and the confidence value
      Check if the confidence value maps to a confidence level equal or greater than the required confidence level
         If the confidence level is lesser than required, then
            Pick a range of values at past and future time-steps to interpolate
            Find the confidence level of the obtained value
            Check if this value maps to a confidence level greater than or equal to the required confidence level
               If the confidence level is lesser than required, then
                  Send the query to the Simulator (in the future)
               Calculate the confidence level obtained
         Else
            Send the value (delay or vehicle count) and confidence level back to the State Manager and write the refined value to the Estimates Database.
   Else if the value at the required time-step is not present, then
      Pick a range of values at past and future time-steps to interpolate
      Find the confidence level of the obtained value
      Check if this value maps to a confidence level greater than or equal to the required confidence level
         If the confidence level is lesser than required, then
            Send the query to the Simulator (in the future)
         Calculate the confidence level obtained

Initially, the State Estimator picks the Estimates Database with a view to satisfy the query. Sending a query into the real world sensor network or directly to the Simulator, is a time consuming process. The State Estimator thus picks the fastest means of answering the query by first querying the Estimates Database. This ensures a decent query response time. Since, the State Estimator aims to maintain a balance between accuracy and response time, the confidence level of the extracted data is calculated as the next step. If this confidence level is lesser than the required level, then the Interpolator function is set into action. The interpolator performs a temporal interpolation from the data available in the data store. Currently, this data store initially contains values at a time-step of thirty seconds and is refined over time. A simple average over a range of temporally close sample points serves as our interpolation function. This step is also executed if the value is not directly found in the database. Once again, in a bid to maintain the accuracy of the estimate, the confidence level is calculated for the estimate obtained via interpolation. If this confidence level is lower than expected, then in the future the query will be routed to a traffic and sensor Simulator such as TRANSIMS. This decision supports the goal of maintaining accuracy while saving on the communication cost that would have been incurred in delegating the query to the real world sensor network. Finally, the obtained estimate along with the result is sent back to the State Manager.

Thus, the State Estimator maintains a balance between accuracy, response time and communication cost within the CoSMo System through the interplay of the three elements: the Estimates Database, the Interpolator and the Simulator. These provide the System with Dynamic and Static Models of the real world sensor network capable of indexing into the past, interpolating over a range of sensor values as well as predicting future values.

With this chapter, we complete a detailed view of the processing components of the CoSMo System. The next chapter discusses the schemas for the System Database and the supporting data tables that we have seen within the preceding chapters.
Chapter 11
The System Database

The System Database is deployed using Oracle 10g. The Oracle 10g database is specifically designed for Grid Computing. The advantages of adopting Oracle 10g over other databases are as follows [3]:

- Oracle Database 10g is compliant with all industry-standard platforms and specifically addresses concerns regarding database migration to a cluster.
- It can be migrated from a single instance to a grid without changing a single line of code.
- Oracle 10g provides availability and scalability by providing competent response times and reduction in downtimes with Real Application Clusters. It also has advanced security features.
- Oracle 10g provides lower costs with the self-managing database which allows the DBA to concentrate on business strategies.

The Database contains Data Tables which can be classified according to their access by the System components. The main data tables are described below.

The tables accessed by the State Manager to obtain a decision regarding the routing of the query to different System components are as follows:
1. AttributeInteger Table: This table maintains the integer representation of each attribute observed by the sensors.
2. PlanAttribute Table: This table stores the integer representation of all allowed attribute combinations constituting a plan and their corresponding plan names.
3. ModuleRoutingDecision Table: This table acts as a repository for production rules depicting the query situation and the corresponding decisions. The use of this Rule-base is explained in detail in Chapter 6.
The tables accessed by the Data Model to obtain a decision regarding the sensors that should be queried according to the sensor network state are as follows:

1. **InternalRepSensorNw Table**: This data table maintains an internal representation of the sensor network. The data model updates it whenever it receives updates from the sensor network about the network topology. The internal representation maintains a know-how of which sensors are currently present in the system, what attributes each one is sensing, their zone, their connectivity, battery levels, maybe the shortest route to each sensor with the current topology, delays in getting data back from the sensor, communication cost etc. It thus assists the Data Model in creation of an effective Observation Plan.

The tables accessed by the Real World Sensor Network Manager component to simulate the working of a sensor network model are as follows:

1. **RealWorldSensorNw Table**: This data table acts as the sensor network for our System as we do not have a deployed net of hardware sensors. It holds the following information about each sensor: sensor-id, its role in a cluster (master or slave), the attribute or phenomenon sensed by the sensor in a heterogeneous network, the power-level, the zone-id to which the sensor belongs, the link-id on which the sensor is placed and type of the sensor.
2. **LinkVehicleCount Table**: This table holds data about the road link and the number of vehicles observed on that link at each time step.
3. **LinkCarId Table**: This table contains records pertaining to the cars which were observed on each link at each time step.
4. **CarDemographics Table**: This table holds car demographics indexed by the car identifiers. It contains the car color, car model and number plate as part of its information base on cars.

The above mentioned data tables are populated by the data assimilated and generated by TRANSIMS. As per the current state, TRANSIMS will periodically need to refresh the data in these tables.
The tables accessed by the State Estimator to get estimates for sensor values, index into past values as well as predict them are listed below. These tables are also written to by the Data Model:

1. LinkDelayEstimates Table: This table contains information about the road links and the delays and vehicle count observed on those links at each time step for duration of 24 hours. It also contains the confidence levels for the estimated values and a field to keep track of the component which has updated the estimate. We generally consider the values filled in by the Data Model to be fresh and in sync with the real world sensor values. Hence the confidence of estimates updated by the Data Model is considered higher than that given by the State Estimator.

2. PathDelayEstimates Table: The State Estimator handles path based queries which may index into the past or need a prediction. We maintain a separate table for path based estimates. This table relays information regarding the link-id, the starting and ending sensor nodes on the link and the estimates of vehicle count and delay for that link at each time step. This information about the constituting individual links is utilized to compose a result for the required path.

3. LinkVehicleCountEstimates Table: It follows the exact structure as the LinkDelayEstimates table reported in point 1, but has records of the number of vehicles seen on the link at particular time steps instead of delay values.

4. PathVehicleCountEstimates Table: It follows the exact structure as the PathDelayEstimates table reported in point 1, but has records of the number of vehicles seen on the path at particular time steps instead of delay values.

The data table accessed by the Query Processor in case of path-based queries is as follows:

1. NodeLinkMap Table: This table maintains a map of road links and their corresponding start and end sensor nodes. This table is used by the Query processor to extract a path between the start and end link specified in the path-based input query.
The database is accessed by a Generic Data Access Layer. The Data Access Layer has been designed to be a generic layer which could be easily extended to support any database system such as IBM’s DB2, Microsoft’s SQL server etc. As mentioned before, currently Oracle 10g has been chosen as the System database due to its flexibility in migrating the database to a grid. The interactions of various components within the System with the database are described in detail in the chapters dedicated to these modules.
Chapter 12
Summary of System’s Cognitive Capabilities

In the previous sections, we described the goals, design details and the implementation of the CoSMo System. CoSMo’s architecture is cognitively-inspired. In particular, a number of features directly derive their motivation from the work in cognitive architectures [1][41]. To evaluate our System’s cognitive capabilities, we use the SOAR [6][41] model as our guideline. SOAR is a widely known and accepted cognitive architecture in AI. It serves as a benchmark for our System and aids us in precisely playing out the cognition in our System. We do not claim that CoSMo is a cognitive architecture; rather it is a cognitively-inspired architecture which chooses to reflect a subset of the cognitive capabilities in its design and implementation. The cognitive capabilities described in [1] and found in CoSMo are summarized below in Table 1:

<table>
<thead>
<tr>
<th>Cognitive Capability/Property</th>
<th>Architecture</th>
<th>System Capability/Property</th>
</tr>
</thead>
</table>

**Recognition and Categorization**
- Ability to recognize situations or events as instances of known or familiar patterns
- Ability to identify when a situation matches a stored pattern
- Ability to learn new patterns

*Module: State Manager:*

1. Patterns and situations are represented in the form of production rules and plans representing a combination of attributes. These are stored in persistent memory.
2. These rules are triggered when a particular situation or query is recognized. The matching rule(s) are selected and the best match is found depending on the number of attributes and conditions that are
3. Learning is facilitated by addition of new plans and rules on the fly when a particular query is not recognized. This query will be initially sent to the Data Model as well as the Interpolator and the module which gives a result with higher confidence is recorded as a decision.

<table>
<thead>
<tr>
<th>Decision Making and Choice:</th>
<th>Module: State Manager:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Ability to make decisions and select among alternatives</td>
<td>1. Implements the recognize-act cycle</td>
</tr>
<tr>
<td></td>
<td>2. Offers a process for selecting among alternatives, specifies conditions under which the decision is legal, e.g. whether the query is link or path based, whether it needs high confidence or low, whether it requires , whether it requires values at current time or future/past time.</td>
</tr>
<tr>
<td></td>
<td>3. Provides conflict resolution when more than one plan of attributes is matched</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module: Data Model:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Chooses the best sensors and attributes to observe to get high quality and low cost results.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Perception</th>
<th>Module: Data Model:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Ability to involve integration of results from different modalities</td>
<td>1. Integrates results from different sensor types in a heterogeneous sensor network into a single result (Aggregation).</td>
</tr>
<tr>
<td>- Ability to extract knowledge from the</td>
<td></td>
</tr>
</tbody>
</table>

93
<table>
<thead>
<tr>
<th>Environment</th>
<th>Prediction and Monitoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Ability to handle faulty sensor readings</td>
<td>- Ability predict what the state of the world</td>
</tr>
<tr>
<td>2. Maintains a database containing a snapshot of the real world sensor network which helps in decisions regarding the sensors and attributes which would give better results.</td>
<td>- Ability to utilize predictions to monitor environment</td>
</tr>
<tr>
<td>3. Deals with the issues of noisy sensors by aggregating the results and calculating confidence levels. In the future, the data model may maintain confidence levels given by these sensors and decide whether a different set of sensors should be selected to satisfy the query.</td>
<td></td>
</tr>
<tr>
<td>4. Event and situations from the sensor network can be detected by aggregating the sensor values.</td>
<td></td>
</tr>
</tbody>
</table>

*Module: State Estimator*

1. The TRANSIMS simulator provides the ability to predict future situations and values.
2. State Estimator maintains a database of estimates which could be used to give an estimate of the past or future value with minimum response time.

*Module: Data Model*

1. The Data Model may be able to monitor the environment since it maintains a copy of the sensor network which contains the sensor power levels, delays, connectivity.
<table>
<thead>
<tr>
<th>Planning</th>
<th>etc.</th>
</tr>
</thead>
</table>
| - Tasks are expressed as goals and the model consists of a plan to carry out these tasks through a series of actions. | **Module: State Manager:**
|                                  | 1. Plans out the further course of a query based on production rules. |
|                                  | **Module: Data Model**
|                                  | 1. Creation of observation plan reflects the ability to plan for better use of resources |
|                                  | **Module: State Estimator:**
|                                  | 1. Implements a plan to extract the value at required intervals from either the simulator or with interpolation. |
|                                  | In each of the above modules, the plan is represented as an ordered set of actions and specifies the effects. The individual plans are constructed from components available in short term or persistent memory (InternalRepSensorNw DB, Estimates DB, AttributePlan DB or ModuleRoutingDecision DB). Process for search and selection of appropriate plan is in place. |

<table>
<thead>
<tr>
<th>Reasoning</th>
<th>Reasoning was incorporated as part of the architecture based on the scenarios constructed. It was implemented in the form of rules and plans and the system if</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Reasoning can be utilized to draw inferences based on assumptions and knowledge already present with the System</td>
<td></td>
</tr>
<tr>
<td>Remembering, Reflection and Learning</td>
<td>Module: State manager:</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>- Remembering: Ability to store results of processing and retrieve them later</td>
<td>1. Remembering: All newly learnt rules and decisions are stored in a persistent store and can be retrieved at any run of the system.</td>
</tr>
<tr>
<td>- Learning: Ability to learn new rules from observations</td>
<td>2. Reflection: System maintains traces of activity within the module for analysis by an external entity.</td>
</tr>
<tr>
<td></td>
<td>3. Learning: Case-based, rule-based learning is observed in this component.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Support of Multiple, Simultaneous Goals</th>
<th>The architecture of the system supports multiple simultaneous goals, namely balance between accuracy and cost of communication, balance between response time and accuracy, flexibility and</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Support the achievement of many top-level goals.</td>
<td></td>
</tr>
</tbody>
</table>
| Modifiability (web services) | Single Learning Method:  
- Ability to perform new tasks without having to be redesigned or reprogrammed |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Module: Query Processor</strong></td>
<td>1. The Query Processor maintains the flexible Query Template for describing diverse queries.</td>
</tr>
<tr>
<td><strong>Module: State Manager</strong></td>
<td>1. The State Manager maintains attributes, plans and rules in a persistent store for ease of addition, modification and deletion by admin as well as the System itself</td>
</tr>
<tr>
<td><strong>Module: Data Model</strong></td>
<td>1. The Data Model has the capability to find best sensors to query inspite of changes in queries as well as the sensor network state.</td>
</tr>
<tr>
<td></td>
<td><strong>Module(s): Data Model and State Manager</strong></td>
</tr>
<tr>
<td></td>
<td>The Data Model may learn the best sensors to observe for a particular attribute or the best alternative attribute to observe after trying combinations of sensors for different types of queries.</td>
</tr>
<tr>
<td></td>
<td>The State Manager may learn from experimenting, the most optimal model to choose in case of certain queries.</td>
</tr>
<tr>
<td>Navigational Strategies:</td>
<td><strong>Module: State Manager</strong></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Agents constructed under the hypothesis of situated action often have rudimentary reactions built into the architecture. These built-in reactions give rise to the strategy that the agent will take under certain environmental conditions.

**Real-Time Execution**
- System is guaranteed to behave within certain time constraints as specified by the task and the environment

1. Implements query routing decisions

2. Decisions are taken while building the system to ensure timeliness such as implementing time deadlines for returning sensor results back to the client.

2. Simulator lends to the real-time execution property of the system.

**Responding Intelligently to Interrupts and Failures**
- Ability to handle asynchronous behavior.
- Ability to continue operation in case of failures

1. Web Services provide handling of asynchronous behavior.

2. Internal error-handling mechanisms are provided.

| Table 1: Mapping of capabilities of a Cognitive Architecture to System capabilities |
|---|---|

Thus, the System is instilled with the above summarized cognitive capabilities, which makes it sentient and sets it apart from other adaptive systems.
Chapter 13
Future Work and Conclusion

13.1 Future Work

With the CoSMo system we have taken the first step towards an intelligent, scalable, adaptive and easily integrative system for Traffic Monitoring. Even though, the design and implementation of CoSMo’s framework required substantial amount of effort, there remains a wide scope for future work. We list a few of them here:

i) Quantitative evaluation of the System: The CoSMo though built on strong fundamental design, an evaluation needs to be undertaken in order to prove the gains brought on by this cognitively-inspired system. A detailed analysis of the system performance needs to be done.

ii) Deployment on hardware sensors: The CoSMo system needs to be deployed on real hardware sensors.

iii) Deployment on the grid environment: CoSMo’s design and functionality can be truly leveraged when it is migrated onto a scalable grid environment. We forsee changes in implementation rather than design for this task. Nevertheless, the challenges in migration of CoSMo onto the grid need to be tackled in the future.

iv) Comparative Study: Once the system is evaluated for its quality and performance and deployed on a live hardware sensor network, the future work entails a comparative study between our system and the existing systems.

v) Efficient Algorithms: The last but the most important work which needs to be addressed as part of future work is the implementation of more efficient and elaborate algorithms in the System. We have provided CoSMo with simple algorithms which go towards demonstrating the practical nature of our system. These algorithms implemented in a modular form can be easily replaced by
more powerful algorithms. For example, sophisticated algorithms can be implemented to enhance learning in the System, the query processor can be made adaptive, a more detailed interpolation algorithm can be incorporated etc.

vi) Security Mechanisms: A System such as CoSMo which has its services compartmentalized according to the user role, needs certain security mechanisms to be in place in order to provide a protected environment for the sensor network and the application.

vii) Application of the System to other domains: Due to the scalable and generic nature of CoSMo, it would be exciting to see it applied to other application domains.

13.2 Conclusion

In this thesis and the companion thesis of Tupe [56], we have described CoSMo, a Cognitively Inspired Service and Model Architecture for monitoring Urban Transportation Systems. The System architecture is novel in its blend of (i) a cognitively inspired internal representation for analyzing and answering queries concerning the observed system and (ii) a service oriented architecture that facilitates interaction among individual modules, of the internal representation, the observed system and the user. This thesis presents the motivation for building a Middleware for Traffic Monitoring based on wireless sensor networks, the design of cognitively-inspired model architecture and the practical implementation of distributed data integration and fusion based on this paradigm. It suggests potential benefits in incorporating internal dynamic models of the observed system and making the system capable of cognition.
References


Appendix A

Design Factors

Some of the requirements and design factors that drive the design of middleware for wireless sensors are as follows [52][59][50]:

1  Power constraints

The wireless sensor is a battery operated device and the power is depleted as a result of its sensing, processing and transmission operations. The replenishment of the battery is at most times very difficult and when the power is completely drained, the sensor becomes inactive. Since each sensor plays a significant role in the network topology, it is necessary to design power-aware middleware.

2  Lifetime

The lifetime of a sensor is related to the limited supply of power with each sensor. A group of dead sensors can cause significant topological changes and cause re-routing of packets [7]. Hence energy-consumption is a major design constraint.

3  Heterogeneity

The sensor network maybe composed of sensors differ in their hardware, software and phenomena sensed. For example, in a traffic monitoring system, there maybe video sensors, temperature sensors, speed sensors etc. The complexity of the application middleware is to an extent determined by the degree of heterogeneity as it also has to take into account the varied services exposed by such a network.
4 Dynamic network topology

The wireless sensor nodes are deployed in dynamic environments which require them to reconfigure themselves to adapt to their changing environments. The sensors may be mobile which leads to changes in topology and connectivity. The middleware has to take into account these topology changes for power management, error control and to provide the application with robustness, flexibility etc.

5 Computational power

The computational power is related with the amount of energy available. Sensors are constrained in their processing capability and more restricted in that, out of the total energy, more energy is consumed by communication.

6 Scalability

The network should be scalable to a large number of nodes and the middleware should not be affected by the change in size or topology of the network.
Appendix B

Quality Attributes of CoSMo

The CoSMo System has been designed with an eye to model the Quality Attributes mentioned below. A more quantitative evaluation of this prototype in terms of these attributes and fine tuning different parameters which would affect them presents scope for future work.

1 Performance:

According to this middleware model, it is no longer necessary to send the query out to the sensor network. In most of the cases the query could be fulfilled with database access (Estimates database component), interpolation (State Estimator component) or simulation (Simulator component) all of which take much lesser time than sending the query to the sensors, the results being routed back to the model and aggregating the data for all sub-queries to fulfill the compound query. Hence, the latency that the user experiences between issuing his/her query and receiving a response will be reduced.

Another aspect of performance is the concurrency of the processes within the system. Web services are inherently concurrent with the ability to handle multiple query flows at a time between the components. Thus, here the choice of publish/subscribe paradigm in web services and UPnP contributes to concurrency of the system increasing the number of queries that can be handled at a time. To supplement this, the individual modules are implemented with threads as well.

2 Accuracy:

This model has a tradeoff between the communication cost and accuracy. The data collected by the sensors is not an accurate representation of the physical phenomena [21],
but it is as precise as it can get with the currently available technology. When a decision is made by the model to query the estimates database instead of directly querying the sensors, there is bound to be some error in the readings. This is where the property of confidence comes into play, wherein the certainty of the resulting values is calculated and the query answered by the estimator/simulator only if the confidence is as high as that specified by the user or higher, else the query is routed to the data model to be sent out to the sensornet. Hence, a balance between communication cost and accuracy is maintained.

3  **Modifiability:**

One of the design goals is to make the system generic enough to be pluggable and extensible. To meet this goal, we have designed the communication between modules to be over web services. Web service is a platform and implementation independent software component, can live anywhere on the network, internet or intranet [31][56][19]. Though, this system is currently built for traffic monitoring, we have provided mechanisms which would make it easily extensible to other applications when the underlying sensor types and capabilities are changed. The query template fed to the query processor, the database containing query routing rules that can be augmented with additional rules according to application need, the entire learning framework can rework with other applications with minimum tweaking.

4  **Security:**

Authentication and authorization mechanisms are in place at the gateways in this model.

5  **Availability:**

The ultimate aim of this middleware is to deploy it on the grid to enhance its availability by replicating the modules on multiple nodes. Web services which are essentially distributed in nature can scale very easily to the grid environment.
6 Usability:

The system has services which can provide system and network state reports to the administrator. The database components which maintain a snapshot of the network and system help generate these reports. The services are easily accessed by the user either via internet or by discovering the services via UPnP. This enhances the usability of the system.

7 System Cognition:

The system integrates the data which is pulled in from different sources: database, simulator, sensornet in a cognitive manner. This process relies on three functional qualities associated with cognitive data integration [37]:

i) Situation awareness: Use cases are chalked out which walk through the data flow paths through the system according to the situation. Rules for the system are based taking into account the spatio-temporal patterns [37] of requested query and the response sensors attribute values.

ii) Decision Awareness: Decisions based on situation analysis and also on taking a resource-centric view are implemented in the form of rule-based component interactions. The model deliberates on its actions taken and the confidence of the results it received and refines its actions as time progresses.

iii) Knowledge Awareness: The model maintains repositories to maintain history of system state, decisions and outcomes of decision.

A summary table of the cognitive abilities of our System has been discussed in this thesis.
Vita

EDUCATION:

- **Master of Science, Computer Science and Applications**
  
  (graduated – Summer I, 2006)
  
  Virginia Polytechnic Institute & State University, Blacksburg, VA
  
  GPA: 3.83 / 4.0

- **Bachelor of Engineering, Computer Engineering**
  
  (graduated – May 2003)
  
  University of Pune, India
  
  Honors in First Class

COMPUTER SKILLS:

- **Computer languages / Technologies:**
  
  C, C#, C++, VC++, J2ME, BREW SDK, Visual Basic, .NETcf, JSP, HTML, DHTML, XML, SMIL, VRML, JavaScript, SQL, Python, Shell scripting, AWK

- **Operating Systems:**
  
  Windows, Linux, Synergy

- **Other Tools:**
  

PROFESSIONAL EXPERIENCE:

- **Intern, VeriSign Inc.**
  
  (June 2005 – August 2005)
- Security enhancement, testing for Cross-site scripting, Middle-man attack, SQL injection, Phishing, secure error handling etc. and automating web application testing using Canoo on Unified Authentication product.

- **Graduate Assistant, College Alcohol Abuse Prevention Center, Virginia Tech** (Fall 2004 – Current)

  - Development of website for CAAPC and an adaptive online course based on Adaptive Hypermedia

- **Software Engineer, Ruksun Software Technologies Pvt. Ltd., India** (5th September 2003 – 5th July 2004)

  - Developed an Instant Messenger Client providing Instant Messaging and Presence Service features for mobile phones in BREW SDK in Visual C++ environment
  - Implemented email services for the Motorola Synergy platform in Visual C++ 6.
  - Developed Instant Messaging clients for Wireless Village compliant servers for mobile phones using J2ME

**ACADEMIC PROJECTS & PRESENTATIONS:**

- Independent Study project laying the foundation for building share allocation policies over the VTRR scheduler with an emphasis on application behavior analysis and modeling process scheduling in a discrete event simulator
- Paper on “Evaluation of Software Architectures that Support Middleware for Wireless Sensor Networks”
- Implemented and analyzed marking scheme for IP traceback
- Developed UPnP devices using C# and Intel UPnP tools
• Developed wireless “hot spot” services considering routing, IP firewalls, IP masquerading, authentication aspects
• Designed, simulated and analyzed a typical office network, the performance of random access in a LAN and the operation of Internet Application protocols using OPNET
• Designed concept maps for an adaptive online course to facilitate interaction between the designer and developer
• Developed an Intelligent Character Recognizer using OCR for the Tamil script in Microsoft Visual C++ 6.0
• Presented seminar on Voice-over-IP solutions in Ruksun Software Technologies

AWARDS:
• Second place at national-level paper presentation competition for paper on Multimedia Search Engines

PROFESSIONAL ORGANIZATIONS:
• Association for Computing Machinery (ACM)

REFERENCES:
• Dr. Madhav Marathe
Deputy Director, Basic Simulation Science and Computer Sc, Virginia Tech
Email: marathe@vt.edu Ph:540-231-8832
• Dr. Steven Clarke
Director, College Alcohol Abuse Prevention Center
Email: sclarke@vt.edu
Ph: 540-231-2233
• Mr. Mangesh Joshi
QA Manager, VeriSign Inc.
Email: mjoshi@verisign.com
Ph: 650-426-3100