

# **A Cognitively Inspired Architecture for Wireless Sensor Networks: A Web Service Oriented Middleware for a Traffic Monitoring System**

By,  
Sameer Vijay Tupe

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

APPROVED:  
Dr. Madhav Marathe, Chair  
Dr. Calvin Ribbens  
Dr. Chris Barrett  
Dr. Keith Bisset

June 08<sup>th</sup> 2006  
Blacksburg, Virginia

Keywords: Service Oriented Architectures, Web Services, UPnP, Traffic Monitoring  
System, Cognitively Inspired Architecture

© 2006, Sameer Tupe

# **A Cognitively Inspired Architecture for Wireless Sensor Networks: A Web Service Oriented Middleware for a Traffic Monitoring System: Sameer Tupe**

## **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 one to effectively respond to deductive as well as inductive queries by combining simulation based dynamic models with traditional relational databases. 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. CoSMo's architecture has a number of features that distinguish cognitive systems. This includes: dynamic internal models of the observed system, inductive and deductive learning and reasoning, perception, memory and adaptation.*

*This thesis describes the service oriented model and the associated prototype implementation. Two important contributions of this thesis include the following:*

- i. *The Generic Service Architecture - CoSMo's service architecture is generic and can be applied to many other application domains without much change in underlying infrastructure.*
- ii. Integration of emerging web technologies - Use of Web Services, UPnP, UDDI and many other emerging technologies have taken CoSMo beyond a prototype implementation and towards a real production system.

## Acknowledgements

I would like to thank my advisor, Dr. Madhav Marathe for giving me an opportunity to work with him on this intriguing and challenging thesis topic. I will always be grateful to him for all the assistance and help he has given me, for devoting so many hours of his own time, and finally, for his incredible kindness towards his students.

I would also like to thank Dr. Calvin Ribbens, Dr. Chris Barrett and Dr. Keith Bisset, for their help, and for taking the time to be on my committee. I have to thank Dr. Karla, Ms. Deepti Chafekar and all the other group members for taking active interest in my work and providing valuable suggestions from time to time. I would also like to give thanks to my thesis partner Ms. Kashmira Phalak for all the support and co-operation she provided to me over the past one year.

A special mention to all my friends back in India – I miss you and I wish I could see you more often, and the friends I have made during my time here at Virginia Tech; you have definitely made my journey worthwhile. Thank you for your incredible hospitality and good company, you have made me feel at home in the US.

Finally, I give the most sincere thanks to my parents and my sister for making it possible for me to come to the Virginia Tech, for supporting me throughout my studies, and for believing in me. Without you I would never have been able to achieve this.

# Table of Contents

<b>Abstract</b> .....	<b>ii</b>
<b>1. Introduction</b> .....	<b>1</b>
<i>1.1. Motivation</i> .....	<i>1</i>
<i>1.2. Our Contribution</i> .....	<i>3</i>
<i>1.3. Specific Contribution of this Thesis</i> .....	<i>3</i>
<i>1.4. Organization of the Thesis</i> .....	<i>5</i>
<b>2. Related Work</b> .....	<b>6</b>
<b>3. Overview of the CoSMo System</b> .....	<b>12</b>
<i>3.1. System Overview</i> .....	<i>12</i>
3.1.1. Client Side.....	12
3.1.2. System Core .....	13
3.1.3. Sensor Network and Transportation Network .....	13
<i>3.2. Brief description of Individual Components</i> .....	<i>13</i>
3.2.1. External World.....	14
3.2.2. Service Components .....	14
3.2.3. Data Processing Components .....	15
<i>3.3. Work-Flow Diagrams</i> .....	<i>20</i>
3.3.1. Sequence Diagram .....	20
3.3.2. Activity Diagram .....	21
3.3.3. Package Diagram .....	22
3.3.4. Deployment Diagram.....	23
3.3.5. Communication Diagram.....	24
3.3.6. Use Cases .....	24
<b>4. Service Components in CoSMo</b> .....	<b>33</b>
<i>4.1. Client Side</i> .....	<i>- 34 -</i>
4.1.1. Design .....	- 34 -
4.1.2. Input – Output for Client Side .....	- 36 -
4.1.3. Implementation Details.....	- 39 -
4.1.4. Summary of Module: Client Side .....	- 39 -
<i>4.2. External Service Component (ESC)</i> .....	<i>- 41 -</i>
4.2.1. Design .....	- 41 -

4.2.2. Input – Output for External Service Component .....	- 44 -
4.2.3. Implementation Details .....	- 44 -
4.2.4. Summary of Module: External Service Component .....	- 45 -
<b>4.3. Internal Service Component (ISC).....</b>	<b>- 46 -</b>
4.3.1. Design .....	- 46 -
4.3.2. Input – Output for Internal Service Component .....	- 47 -
4.3.3. Implementation Details .....	- 47 -
4.3.4. Summary of Module: Internal Service Component .....	- 48 -
<b>4.4. Service Manager.....</b>	<b>- 50 -</b>
4.4.1. Design .....	- 50 -
4.4.2. Input –Output for Service Manager .....	- 56 -
4.4.3. Implementation Details .....	- 56 -
4.4.4. Summary of Module: Service Manager .....	- 57 -
<b>4.5. Interaction between Service Components.....</b>	<b>- 58 -</b>
<b>5. CoSMo’s Service Oriented Architecture for Individual Components and Component Interactions.....</b>	<b>- 60 -</b>
5.1. CoSMo’s Generic Component .....	- 63 -
5.1.1. Design .....	- 63 -
5.1.2. Implementation Details .....	- 67 -
5.1.3. Summary of Generic Component Design .....	- 67 -
5.2. Communication Framework in CoSMo .....	- 69 -
5.2.1. Design .....	- 69 -
5.2.2. Implementation Details .....	- 75 -
5.2.3. Summary of Communication Infrastructure .....	- 75 -
5.3. Gateway Functionality.....	- 75 -
5.3.1. Design .....	- 76 -
5.3.2. Implementation Details .....	- 78 -
5.3.3. Summary of Gateway Functionality .....	- 78 -
5.4. Technologies used with Service Oriented Middleware.....	- 79 -
5.4.1. UDDI based communication within CoSMo .....	- 79 -
5.4.2. UPnP Service Publishing within CoSMo .....	- 80 -
5.4.3. Application Packages .....	- 81 -
5.4.4. Summary of technologies used in CoSMo.....	- 82 -
<b>6. Design Decisions and Future work.....</b>	<b>- 83 -</b>
6.1. External Service Component (ESC).....	- 83 -

6.1.1. Design Decisions .....	- 83 -
6.1.2. Future Work .....	- 84 -
6.2. <i>Service Manager</i> .....	- 84 -
6.2.1. Design Decisions .....	- 84 -
6.2.2. Future Work .....	- 85 -
6.3. <i>Internal Service Component (ISC)</i> .....	- 85 -
6.3.1. Design Decisions .....	- 85 -
6.3.2. Future Work .....	- 85 -
6.4. <i>Five Layered Service Architecture and Generic Component Design</i> .....	- 85 -
6.4.1. Design Decisions .....	- 85 -
6.4.2. Future Work .....	- 87 -
6.5. <i>Gateway Functionality</i> .....	- 87 -
6.5.1. Design Decisions .....	- 87 -
6.5.2. Future Work .....	- 89 -
6.6. <i>Inter-Component and Intra-Component Communication Design</i> .....	- 89 -
6.6.1. Design Decisions .....	- 89 -
6.6.2. Future Work .....	- 90 -
<b>7. Conclusion and Future Work .....</b>	<b>- 92 -</b>
7.1. <i>Conclusion and Summary</i> .....	- 92 -
7.2. <i>Future Work</i> .....	- 92 -
7.2.1. Addition of efficient algorithms.....	- 93 -
7.2.2. Support for Hybrid queries .....	- 93 -
7.2.3. Extension to the Grid environment .....	- 93 -
7.2.4. Interfacing with Hardware sensors and Simulator .....	- 93 -
7.2.5. Performance Evaluation .....	- 94 -
7.2.6. Creation of Secure Infrastructure.....	- 94 -
<b>8. Appendix.....</b>	<b>- 95 -</b>
8.1. <i>Web Services</i> .....	- 95 -
8.1.1. Web Service Architecture .....	- 95 -
8.1.2. Web Services benefits.....	- 96 -
8.1.3. Web Services Challenges.....	- 97 -
8.1.4. Technologies for Web Services and Web Services Stack.....	- 98 -
8.2. <i>Universal Plug and Play (UPNP)</i> .....	- 99 -
8.2.1. Motivation.....	- 99 -

8.2.2. UPnP Basics.....	- 101 -
8.2.3. UPnP Flow Diagram.....	- 102 -
8.3. <i>Universal Description Discovery and Integration (UDDI)</i> .....	- 104 -
8.3.1. Motivation.....	- 104 -
8.3.2. UDDI Basics .....	- 105 -
8.3.3. UDDI support in CoSMo .....	- 106 -
<b>9. References.....</b>	<b>- 108 -</b>

# List of Figures

Figure 1: CoSMo: System Overview. It shows three distinct parts namely, Client Side, External world comprising sensor and transportation network and CoSMo system core. 12

Figure 2: The CoSMo Architecture. It shows all the components in the CoSMo system. Broadly CoSMo is divided into two parts, data processing components and service components. .... 14

Figure 3: CoSMo: Sequence Diagram. It depicts the flow of a typical query as it passes through the different CoSMo components and returns back. .... 20

Figure 4: CoSMo: Activity Diagram. It depicts all the decisions taken and steps performed by internal components as the query passes through them. .... 21

Figure 5: CoSMo: Package Diagram. It outlines all the packages used in the CoSMo system. .... 22

Figure 6: CoSMo: Deployment Diagram. Typical system outlook when deployed in a real environment. .... 23

Figure 7: CoSMo: Communication Diagram. It depicts the control flow between the CoSMo components as a request is made between the components. .... 24

Figure 8: User Role Selection Screen. Two types of roles are provided, namely user and administrator. .... 25

Figure 9: Win-Forms based Client User Interface. Broadly it is divided into parts, namely, query special options, query attributes and query output part. .... 26

Figure 10: Google Map-based User Interface. Get Delay service prompting user to select two points on the map and specify the confidence and time range for the query..... 26

Figure 11: CoSMo: System Overview. The left panel shows CoSMo’s overall architecture; the right panel focuses on the service components. .... - 33 -

Figure 12: Input - Output for Client Side..... - 36 -

Figure 13: Win-Forms based Client User Interface. Broadly it is divided into parts, namely, query special options, query attributes and query output part. .... - 36 -

Figure 14: Google Map-based User Interface. Get Delay service prompting user to select two points on the map and specify the confidence and time range for the query..... - 37 -

Figure 15: UPnP-based User Interface. A device named GetDelay with service GetDelay is published over UPnP. The service supports all the parameters necessary to issue a delay query over UPnP. .... - 38 -

Figure 16: Internal details of External Service Component (ESC)..... - 41 -

Figure 17: CoSMo: Typical Application Services. Light shaded boxes show the services currently available within the CoSMo system. .... - 43 -

Figure 18: Input-Output for External Service Component ..... - 44 -

Figure 19: Input - Output for Internal Service Component ..... - 47 -

Figure 20: Composition of Module: Service Manager .....	51 -
Figure 21: Service Creation and Publishing within CoSMo.....	51 -
Figure 22: Authentication within CoSMo. Passwords are transferred in the encrypted format over the Internet. ....	52 -
Figure 23: Service Manager User Interface. It provides fields for entering the service specific information. ....	53 -
Figure 24: Automated Flow of Service Publishing in CoSMo. Most of the manual steps related with creating and publishing a UPnP device on the network are automated....	54 -
Figure 25: Input - Output for Service Manager .....	56 -
Figure 26: Interaction between Service Components and the Client Side. It shows how service components interact with each other and form a service infrastructure. Clients take advantage of this service infrastructure to issue queries into the CoSMo system. Numbering represents the following steps.....	58 -
Figure 27: CoSMo's Generic Component – Division based on Functionality .....	63 -
Figure 28: CoSMo's Component Architecture mapped to widely known 5-Tier Architecture.....	67 -
Figure 29: CoSMo: Inter-Component Control Flow. It shows how the control flows through different modules when a remote web service call is made. ....	70 -
Figure 30: CoSMo: Intra Component Control Flow. It shows how the internal modules of component are divided into sections to concentrate areas which are sensitive or insensitive to the domain specific changes. ....	71 -
Figure 31: CoSMo: Inter and Intra-Component Control Flow. Shows overall asynchronous query flow within the CoSMo system. ....	73 -
Figure 32: Generality of the CoSMo Service Architecture. The diagram shows what sections are most affected when the architecture is applied to a different application domain.....	74 -
Figure 33: CoSMo: Gateway Logic. It is divided into three layers, namely component layer consisting of components and criteria monitors, component selectors layer consisting of component selectors which observe components for specific criteria and gateway layer having gateway logic for selecting the best component out of all the best components given by component selectors.....	76 -
Figure 34: UDDI registry based Communication within CoSMo. The red circle within the diagram shows that entire communication infrastructure within the CoSMo system is centered on the UDDI registry. All components interact with the registry to find out about the remote component. ....	79 -
Figure 35: UPnP Service Publishing within CoSMo. It shows the internals of UPnP service publishing. ....	80 -
Figure 36: CoSMo: Technology Overview. It shows all the application packages and technologies used in all the components.....	82 -

Figure 37: Summary of Gateway Logic. It shows division of gateway logic into three layers, namely, Component Layer, Component Selectors and Gateway layer. .... - 88 -

Figure 38: Web Service Publish, Discover and Bind Operations. .... - 95 -

Figure 39: Web Services Stack..... - 99 -

Figure 40: UPnP: Interaction between Control Point and Services ..... - 101 -

Figure 41: UPnP Control Flow Diagram ..... - 103 -

Figure 42: UDDI Basic Data Types..... - 105 -

Figure 43: UDDI Support in the CoSMo system. It shows the change made to service storage to accommodate the gateway logic. .... - 106 -

## List of Tables

Table 1: Summary of Module: Client Side.....	- 40 -
Table 2: Summary of Module: External Service Component.....	- 46 -
Table 3: Summary of Module: Internal Service Component.....	- 49 -
Table 4: Steps Covered in Automated Flow of Service Publishing.....	- 55 -
Table 5: Summary of Module: Service Manager.....	- 57 -
Table 6: CoSMo: Device Capabilities and Available Access Methods. It shows the number of available methods to access the CoSMo system. ....	- 59 -
Table 7: Summary of Generic Component Design.....	- 69 -

# 1. 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, Map-Quest etc. Now imagine that you want to traverse the same path at 7:00 p.m. in peak traffic. If you query these route based systems, you would get the same static estimations for 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 in case of a multi-modal journey such as traveling a part of the journey on foot, part of it by bus and the remainder by car. Such systems lack the capability to provide dynamic results since they do not take into account the dynamic behavior of the transportation network [16] [34]. This is just one case in which a dynamic modeling of traffic networks is required. The data collected by a Traffic Monitoring application may find utility in improving performance of transport networks, to map "activity locations" based on travel patterns of members of each household, to plan the minimal cost routes for individuals, to calculate vehicle emissions [51], to provide a maintenance and diagnostic history of vehicles, detect events such as accidents, traffic jams, vehicle appearance, landslides on the road etc. [51]

The advent of wireless sensor networks (WSN) [46] and many supporting internet technologies have given rise to new solutions for monitoring such large complex systems. Wireless sensors are tiny devices capable of processing, communication and sensing capabilities which can be placed in large numbers in the environment. Wireless sensors have the ability to operate in unreachable or hostile environments and provide advantages in terms of cost, size, scalability, and distributed intelligence. [58][50] Because of the event monitoring and ubiquitous capabilities [50][66] of WSNs, they have found their use in many application domains including but not limited to Traffic Monitoring and Planning, Civil and Environmental Monitoring, Military, Medical and Space Monitoring etc. 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 [45].

Developing situational assessment and monitoring tools for such large complex socio-technical [45] system 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 measuring 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 query.

**Need for a Middleware:** To comprehend the complexities generated by the vast amount of data and the functionalities supported over this data gives rise to the requirement and motivation behind having a Middleware. The scope, functionality and challenges of this middleware in case of wireless sensor networks can be found in [18] [63] [40]. We can broadly summarize the responsibilities of such middleware as (i) Support for Sensor Network Management, (ii) Intelligent data aggregation, interpretation and presentation with the associated confidence level of the collected data, (iii) Support for Service Management (iii) Support for sensor data fusion. It should show some intelligence and adaptive capabilities [23] in responding to queries efficiently. Additionally it should also serve as an open infrastructure on which internal and external applications can be built upon without going through the internal details.

## 1.2. Our Contribution

Here and in the companion dissertation of Phalak [59], we describe the architecture and a prototype implementation of CoSMo, a Cognitively Inspired Model [47] and Service Oriented Architecture [57] [26] 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 be interested in various tasks relating to such infrastructure systems. Examples of such tasks include, finding out current traffic conditions, maintaining the sensor network so as to extend its lifetime, finding optimal routes, etc.

CoSMo is 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 [9] [43], a simulation based dynamic model as internal representation. 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 easily integrated with other dynamic models such as MITSims [43] so far as these models provide the required functionality.

## 1.3. Specific Contribution of this Thesis

As discussed earlier, the overall development of CoSMo was done jointly with Kashmira Phalak. The service oriented architecture is undertaken as a part of this thesis, while Phalak's [59] thesis was primarily responsible for the cognitively-inspired modeling environment. Integration of the architectures and systems that constitute CoSMo was undertaken jointly.

Specific contributions of this thesis are:

- i. Design and conceptualization of a novel unified architecture for Traffic Monitoring which integrates four paradigms namely, cognitively-inspired, model-driven [60], data-centric [31] and service-oriented approaches [57] [26] 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. Design and Implementation of a web service oriented architecture which can be applied in many application domains.
- v. Design and Implementation of a Generic CoSMo Component which serves as a template for all of the CoSMo components.
- vi. Design and Implementation of a fully asynchronous communication infrastructure within CoSMo.
- vii. Design and Implementation of a gateway functionality illustrating the feasibility of CoSMo on a grid environment in the near future.

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

## **1.4. Organization of the Thesis**

Chapter 1 provides background and motivation for undertaking this project as our thesis and emphasizes the potential of our architecture. It outlines the scope of our work and our contributions in this area. Chapter 2 summarizes related work in Cognitive architectures, Model driven architectures and Service Oriented Middleware and covers some of the aspects of the technologies used for building a working prototype. Chapter 3 gives overview of CoSMo's unified framework. UML diagrams and use cases are also described in this section.

Chapter 4 covers the service components and their interaction within the system as well as with the external world. Chapter 5 covers web service oriented middleware CoSMo component, inter-component Communication and Gateway functionalities are discussed in detail. Chapter 6 covers design decisions and future work for the service components and service architecture. Chapter 7 outlines the future work and the conclusion for CoSMo. Appendix - Covers background concepts and technologies used in Service Oriented Middleware.

## 2. Related Work

CoSMo's unified framework consists of a Cognitively Inspired architecture coupled with Service Oriented architecture. It extends from traditional model driven and data centric aspects and has web service oriented service architecture. The presence of many paradigms makes CoSMo interesting from literature point of view, as it touches work from many other researchers in variety of domains.

In this section we will first discuss related work on Model driven architectures, Data centric architectures, Cognitive architectures and then covers Service Oriented Architectures along with some work done in UPnP middleware for Sensor Networks.

Model driven architectures [60] covers background and motivation for model driven architectures. It talks about Vision, Standards and current Technologies present in such architectures. It also emphasizes the use of core standards such as UML, XMI and CWM [55] [1] core standards for building coherent schemes for authoring, publishing, and managing models within a model-driven architecture.

"Model-Driven Data Acquisition in Sensor Networks" [28] concentrates on statistical models for interactive querying. It demonstrates use of statistical models for providing answers that are more efficient to compute in time as well as energy. Such models can help in variety of ways. They provide robust interpretations, help in identifying faulty sensors, optimize the acquisition of sensor readings etc. In the proposed architecture models are used to estimate sensor values in the current time period and the query is answered with very limited interaction with the sensors. It also underlines the importance of observing co-related attributes rather than actual attributes when it is not possible to monitor the actual attributes because of certain constraints. We have a similar architecture in which observation plan, confidences and probabilistic models storing traffic and sensors specific data are used. But our system extends beyond the model driven approach and encompasses a cognitive inspired paradigm, along with the strong

support for services. CoSMo also has place for a simulation of the real world which demonstrates the interplay between the real sensors and the simulator.

A Middleware-driven Architecture for Information Dissemination in Distributed Sensor Networks [17] presents a hybrid, multi-layered and agent-oriented architecture for distributed sensor networks. It proposes an agent-based middleware that sits in between the application layer, consisting of software agents and the physical layer consisting of wireless sensor devices. Some of the capabilities of the middleware are cooperative data mining, querying, self-organization and sensor network management. Application software is classified into sensor node applications and sensor network applications. Middleware is divided into Sensor Node, Region Management and Application agent platform middleware. Our middleware supports sensor network management but we do not have middleware at sensor or regional level. Our middleware is further characterized by cognitively inspired model driven part and web service oriented architecture which are not present in [17].

Data Centric storage in Sensornets [62] emphasizes the importance of content of the data from the sensors rather than the location of the sensor itself. It talks about geographic hash table in data centric storage approach. Keys are hashed into geographic co-ordinates and key-value pairs are stored at a sensor node which is geographically nearest to the hash of its key. Every sensor is treated as a tiny database. Unlike most of the routing protocols, [62] uses a strong geographic routing because of its geographic hash table approach. CoSMo at this stage does not support sensor databases. Traffic and Sensor specific information is stored and manipulated in a centralized database. [62] refers to events as certain pre-defined constellations of low level observations which get detected by processing the low level observations through collaborative information processing techniques. CoSMo also contains fixed predefined events which constitute low-level observations such as existence of car, its color, its make etc. Fundamental difference between our approach and [62] is the fact that [62] treats sensors as intelligent units with local data storage capabilities. In the future CoSMo can take the advantage of this fact.

Directed Diffusion [41] is an example of data centric routing, routing based on the name of the data as in [62] rather than the identity of the destination node. It differs from [62] because of the fact that it stores the event locally at the detecting node. While passing the data up the hierarchy, aggregation techniques are used by the intermediate nodes. Flooding is the normal technique used in their system. It [41] differs from our system in the way aggregation is performed and the location at which aggregation takes place. CoSMo does not support in-network aggregation at this stage. Our system aggregation is model driven and happens within the system.

PRESTO [29] addresses the two commonly used models for data processing in sensor network, direct sensor querying and data streaming. Both these approaches have advantages and disadvantages. PRESTO uses hybrid scheme which combines both of these approaches. It attempts to provide the interactivity of the data streaming approach with the energy efficiency of direct sensor querying. PRESTO supports hierarchical systems, archival queries and single logical view of data. PRESTO architecture is very similar to CoSMo. It has proxies which try to answer the queries from cache. On the similar lines CoSMo tries to search for the values locally. If the values are not found locally CoSMo tries to interpolate over the available values or might decide to query the sensors for fresh values. CoSMo has the added advantage because of existence of the Simulator, which increases the frequency of local hits within specified confidence levels.

RCS [11] was originally designed for sensory-interactive goal-directed control of laboratory manipulators. It consists of a multi-layered multi-resolution hierarchy of computation agents each containing elements of sensor processing, world modeling, value judgment, behavior generation and a knowledge base. Similar to other cognitive architectures, it represents procedural knowledge in terms of production rules and represents abstract data structures such as frames, classes and semantic nets. As RCS is a real time control system it maintains a tight coupling between iconic and symbolic data structures. It supports generation of plans by case based reasoning or search based optimizations. CoSMo has production rules representing procedural knowledge,

databases storing past knowledge, current knowledge and future predictions. Case based reasoning is used for selecting proper plan for the current scenario.

Cognitive Theory, SOAR [47] grew out of research on human problem solving, language understanding, computation linguistics, theorem proving and cognitive modeling. It is a computational theory of human cognition that takes the form of general cognitive architecture. Fundamental functional constraints in SOAR are flexible goal driven behavior, continuous learning and real time cognitive response. SOAR incorporates symbols, pattern structures, storage and manipulations over them. It has a fixed structure in terms of primitive processes, memories and control structures. Long term memory is stored as production rules where each production is a condition-action pair. SOAR represents all tasks as collections of problem spaces. Problem spaces are made up of a set of states and operators that manipulate the states. SOAR begins work on a task by choosing an initial state and a problem space. It represents the goal of the task as some final state in the problem space and repeats its decision cycle as necessary to move from the initial state to the current state [98]. CoSMo exhibits a subset of SOAR capabilities such as production rules; case based reasoning, short term memory and rudimentary learning in the form of addition of new rules at runtime. Some of the other work done in this area can be found at [13], DUAL [44], ADAPT [15].

Service Oriented Architecture [54] examines the history, business drivers and standards that are used in service oriented architectures. It presents localized definition of SOA along with discussion on SOA frameworks and patterns. Middleware Challenges for WSNs [63] examine the purpose, functionality and characteristics of middleware which support the development of applications for wireless sensor networks. It outlines the wireless sensor networks characteristics, issues of energy constraints and heterogeneous configuration and networking needs. It also describes direct sensor querying, in-network aggregation of sensor values and the corresponding role performed by middleware. Some of the other work done in services in sensor network can be found at Service Oriented Sensor Web [25] and Sensor Bean [52].

SOA for Sensor Networks [35] describes a service oriented software architecture for mobile sensor networks. The main objective is the simplified development of service applications in sensor networks. This architecture provides a scalable and flexible design for applications based on sensor networks. It addresses the sensor network requirements from dynamic system requirements of self organization, networking, co-operation, power efficiency etc. The paper also outlines the characteristics of middleware for sensor networks considering the above requirements. Some of the characteristics are scalability, generality, adaptivity and reflectivity. Web Services are used to further strengthen the architecture. Within CoSMo, design decisions were taken which increased the generality of overall architecture and extended its use to many application domains.

[56] describes Service Oriented Architectures (SOA) and Web Services technology. Advantages and issues in SOA and Web Services are outlined along with the new emerging standards in this area. “Developing a Software Engineering View” [7] describes two, three and five tier approaches for software development. [5] covers N-Tier architecture components and challenges involved in development based on these architectures.

UPnP based Sensor Network Management [64] proposes a framework for sensor network management based on UPnP. As sensors today do not incorporate UPnP standards, this framework includes a UPnP agent (BOSS) implemented in the base station and lies in between UPnP controller and non-UPnP sensors. Services are published by BOSS on behalf of non-UPnP based sensors. The technique used in this paper is similar as used in CoSMo, but in addition to the framework proposed by [64] CoSMo has web services, UDDI registry and many other advanced features. Another attempt to apply UPnP to sensor network is described in Sindrion [39] in which complex operations such as UPnP eventing and control are sourced out to external terminals with lots of power and processing capabilities. Only UPnP discovery, description and presentation are supported

within the sensor network. As sensors are highly resource constrained devices we believe this approach will not be suitable for long term use.

### 3. Overview of the CoSMo System

Following section describes CoSMo system overview. It follows with brief description of each of the modules within CoSMo. At the end we will cover different Work-Flow diagrams describing CoSMo along with the use cases.

#### 3.1. System Overview

Broadly CoSMo’s unified architecture is divided into three sections namely Client Side, Sensor and Transportation Network and the System Core.

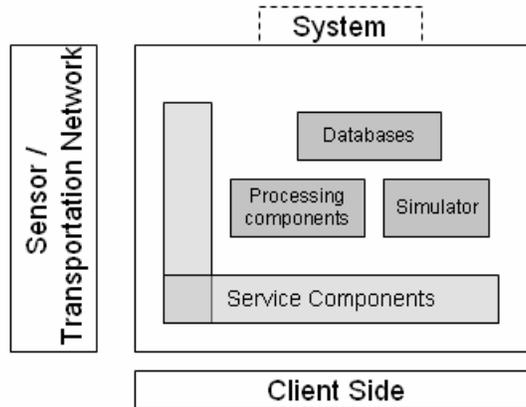


Figure 1: CoSMo: System Overview. It shows three distinct parts namely, Client Side, External world comprising sensor and transportation network and CoSMo system core.

##### 3.1.1. Client Side

The client side consists of different applications running on PDAs, laptops and handhelds which can issue the query to the system. Targeted users for our system are users querying the system for traffic information or traffic planners who perform short term congestion planning.

### **3.1.2. System Core**

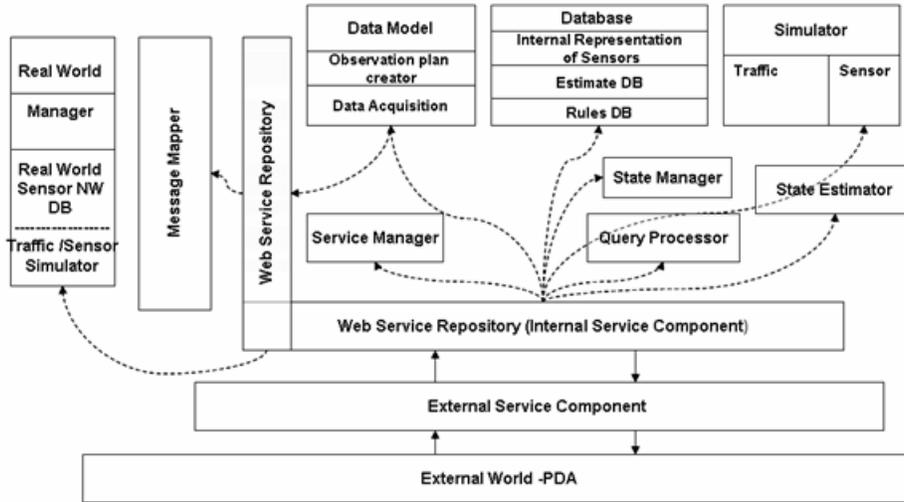
Broadly CoSMo core is categorized into processing components and service components. Processing Component mainly deals with processing the incoming query and service components deal with how to expose services and create a communication infrastructure within and around the CoSMo system. CoSMo has a databases and traffic simulator in terms of TRANSIMS [9] within our system which adds to the system capabilities. Database keeps information about the sensor network and traffic network. Processing components use the simulator or the database to satisfy queries. Thus effectively we have two real world networks (sensor and traffic) and two simulations (sensor simulation and traffic simulation) and there is interplay between them for satisfying the queries efficiently. The system optimizes its performance by selecting appropriate least cost option of interacting with the real world or implementing simulations or possibly both for satisfying queries efficiently. Service Oriented Architecture and Service Components act as a wrapper around CoSMo and provide a scalable and modular interface to the external world.

### **3.1.3. Sensor Network and Transportation Network**

CoSMo's architecture models sensors as the monitoring system and the transportation network as the monitored system. Sensors are used to monitor the traffic conditions and populate the internal database with traffic information. The overall aim of our cognitively inspired architecture is to treat the sensors as a critical resource and to extend their life time without much compromise on the quality.

## **3.2. Brief description of Individual Components**

CoSMo main architecture is as shown in the figure [2]. It is logically subdivided into functionalities related with client side, services core, cognitively inspired part, simulation, databases, and real sensor and traffic world.



**Figure 2: The CoSMo Architecture. It shows all the components in the CoSMo system. Broadly CoSMo is divided into two parts, data processing components and service components.**

The description is organized into the three parts, namely, external world, services related components and data processing components.

### 3.2.1. External World

The external world is the client side in our architecture. In the current scope, end users for our system are users who want to plan their daily trips, routes and traffic planners who need traffic specific information for efficient congestion planning. Two types of Roles are provided within the system, namely normal user role and admin role for carrying out the administrative tasks of system planning and monitoring.

Users are given access through web pages, UPnP and web services. In the current implementation CoSMo supports value based queries and event based queries.

### 3.2.2. Service Components

Service Components are responsible for creating a service infrastructure within and around CoSMo. These components act as a wrapper around the system and provide a scalable and modular interface to the external world. They represent and facilitate the interplay between (i) individual modules of internal representation, (ii) the external world

comprising of sensors and urban transport system and (iii) the individual users, including system managers, who would be interested in various tasks relating to such infrastructure system.

### **External Service Component (ESC)**

ESC acts as a gateway or entry point for the queries. It exposes the services to external world and passes the queries from clients to the Query Processor. Many technologies can be used to provide a richer user experience for the system. CoSMo currently supports windows forms, web pages and UPnP access to the system. In future if we get proper data, Google maps interface to CoSMo will become a reality.

### **Service Manager**

Service Manager is the core of the service publishing infrastructure within the CoSMo. It receives the service descriptions from service providers. After receiving the input, it publishes the UPnP [53] [64] [4] services on the network and it interfaces with the UDDI [10] registry to store the information related with the service providers and the services. UDDI stands for Universal Description Discovery and Integration. It is a web based service registry capable of storing service specific information. In the current implementation entire communication framework within CoSMo is based on the services published by the Service Manager.

### **Internal Service Component (ISC)**

Internal Service Component is a centralized UDDI web service repository in the system. It is a private UDDI node shared by all the components inside the system for service discovery and access. All the components interact with each other using the information provided by this web repository. Latest UDDI specification provides support for grid environment, hence in future if required we can have multiple copies of registries cooperating and coordinating with each other.

### **3.2.3. Data Processing Components**

Data Processing components form the functional core of the system. They are responsible for query processing and query output. Cognitively inspired, model driven and data driven approaches are incorporated into these components in order to improve system

performance, perform intelligent query processing and make CoSMo smart and adaptable.

### **Query Processor**

Query Processor is the first data processing component which handles user queries. It converts the user query into a fixed generic XML query template format and passes it on to the State Manager. In the current implementation, a generic query template was developed, which covers all the services provided by the CoSMo system. This template carries special options with the query such as type of query, service name etc. along with the attributes and criteria for query.

### **State Manager:**

State Manager parses the sub-queries and routes them appropriately to either the Interpolator or the Data-Model. In the current implementation, the routing decisions are based on a routing table which is setup at the system start-up from the Service Manager. CoSMo demonstrates preliminary intelligence in terms of dynamic addition of new rules to rules table, but in future CoSMo can be made much more intelligent by adding concrete learning mechanisms and changing the rules on the fly using the accumulated knowledge base.

### **Database Manager**

CoSMo contains three types of databases, namely, Internal Representation Database, Estimates Database and Rules Database.

### **Internal Representation Database**

Internal Representation Database maintains the internal representation of the sensor network. It includes a know-how of which sensors are currently present in the system, what attributes each one of them is sensing, their zone, their connectivity, battery levels, 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 be most suitable to observe the required attributes.

### **Estimates Database**

This database maintains estimates of the sensor data. It is a brief representation of the traffic network. To begin with, it contains the data extracted from Transims output file which contains the values at a coarser granularity of time intervals. Over the time the estimates database gets populated with finer granularity values as State Estimator estimates time instances where values are not available. Confidence levels are also stored with the estimates. In the current implementation CoSMo does not support validity period for these confidences. If the values are not directly available then they are temporally or spatially interpolated. Those values are returned back with the calculated confidence values.

In the current implementation we send back the values from State Estimator to State Manager even if they do not satisfy the required confidence level. But in the future we can have a control flow between the Estimator and the Data Model in case the Estimator results in the lower confidence values. It will result in more queries getting satisfied with higher confidence as well as we can support different class of queries including hybrid queries [59].

### **Rules Database**

The Rules Database helps in making the query routing decisions. They contain attributes, plans and query destination information. Different combination of attributes forms plans. Different plans along with the special conditions detected in a query form rules which help in determining the query destination.

### **Data Model**

State Manager passes the queries to the Data Model when fresh data with the current time is required. Within CoSMo, Data Model is given the functionality to talk with the sensor network. Data Model parses the XML query and creates an Observation Plan [28]. Creation of an Observation Plan is dependent on the many factors including location of sensors, the attributes they are monitoring, power constraints etc. Data Model passes this observation plan to the Message Mapper which acts as the sensor network interface.

## **State Estimator**

The State Estimator interpolates the values present in the Estimates Database at a finer granularity of the time intervals. In broader sense State Estimator is responsible for two kinds of interpolations:

**Temporal Interpolation** - If the value at a particular time is not present in the database then temporally closer values are chosen. Some algorithm is performed on these values to find out the values at an intermediate time instance.

**Spatial Interpolation** - Spatial interpolation is possible in this application domain as traffic network has a spatial dimension attached to it. Links and Zones have a geographic connectivity and one can relate values over multiple links if we know nearby traffic and connectivity information.

In the current Implementation State Manager routes all path based queries as well as link based queries in which values for past and future are required to the Estimator. When a query is routed to the State Estimator, it first checks whether the value is available in the estimates database. If the value is found, then that value is picked up. The confidence level is calculated and answer is returned. If the values are not readily available then they are temporally interpolated. The values are returned back with the confidence achieved. The Estimator returns calculated value along with its associated confidence interval to the state manager irrespective of whether it satisfies the required confidence or not. In the current implementation we do not interpolate the values spatially. Also in future we can implement a path from the Estimator to the Data Model for the cases where Estimator cannot find an answer with the required confidence.

## **Simulator**

We are using output from the TRANSIMS [9] simulator. In the current implementation we are getting a data file containing a complete 24 hour feed from the TRANSIMS. Whenever required we estimate over these values to satisfy the queries. In the future,

CoSMo can be interfaced directly to the TRANSIMS simulator, which can dynamically update the internal traffic database.

### **Real World Manager**

The Real World Manager simulates the wireless sensors. Real World Manager talks to the internal representation database and is responsible for mimicking the sensor behavior of sensing the values and sending the results back to the Data Model.

### **Message-Mapper**

Message Mapper is another abstraction level added in the system which isolates system from sensor specific properties and output format details. It exposes a uniform interface to the sensors hiding heterogeneity of sensors in terms of different vendors, query format, different attributes they are sensing for and. In the current implementation we assume sensors are capable of understanding SQL queries.

### 3.3. Work-Flow Diagrams

#### 3.3.1. Sequence Diagram

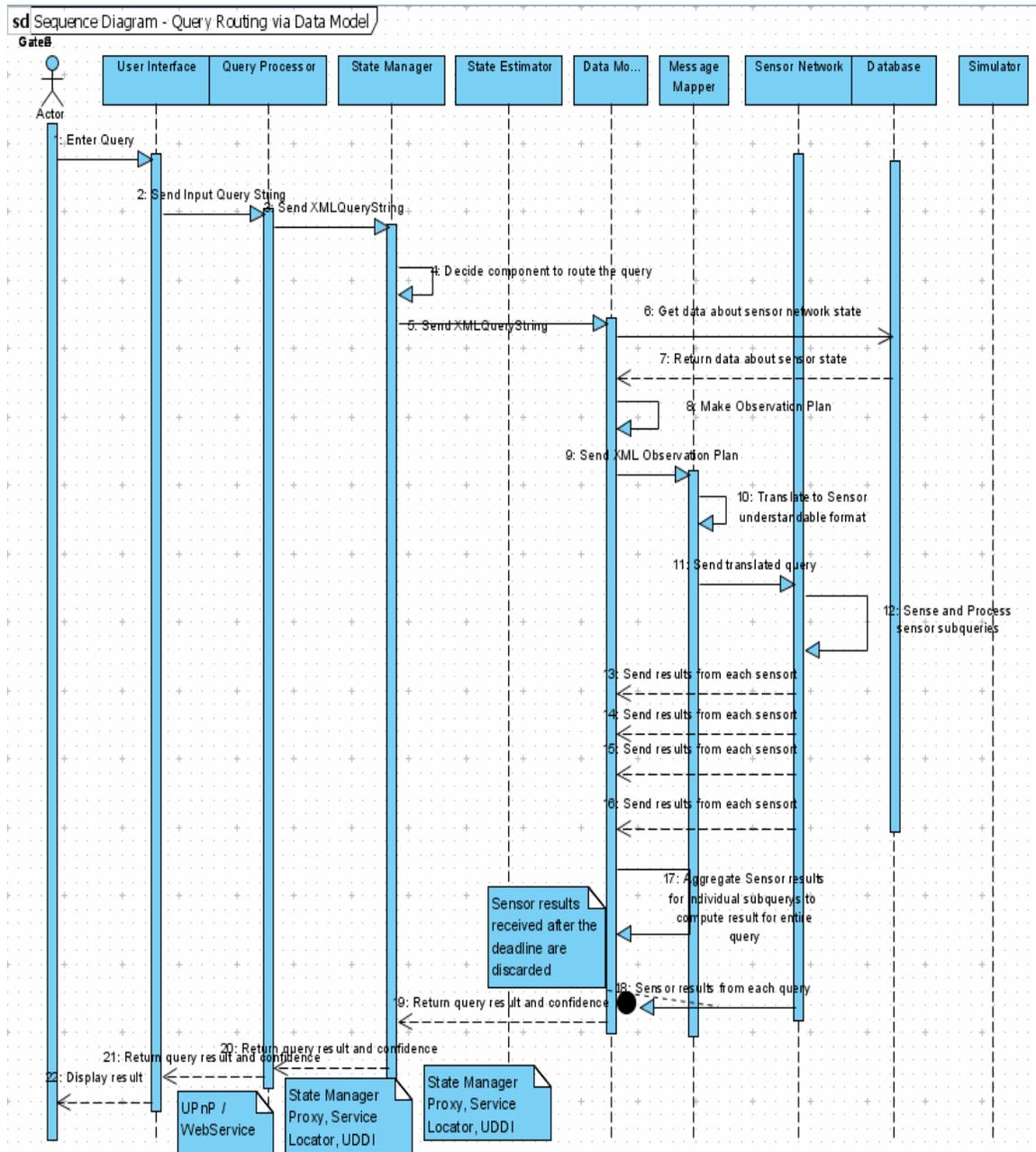


Figure 3: CoSMo: Sequence Diagram. It depicts the flow of a typical query as it passes through the different CoSMo components and returns back.

### 3.3.2. Activity Diagram

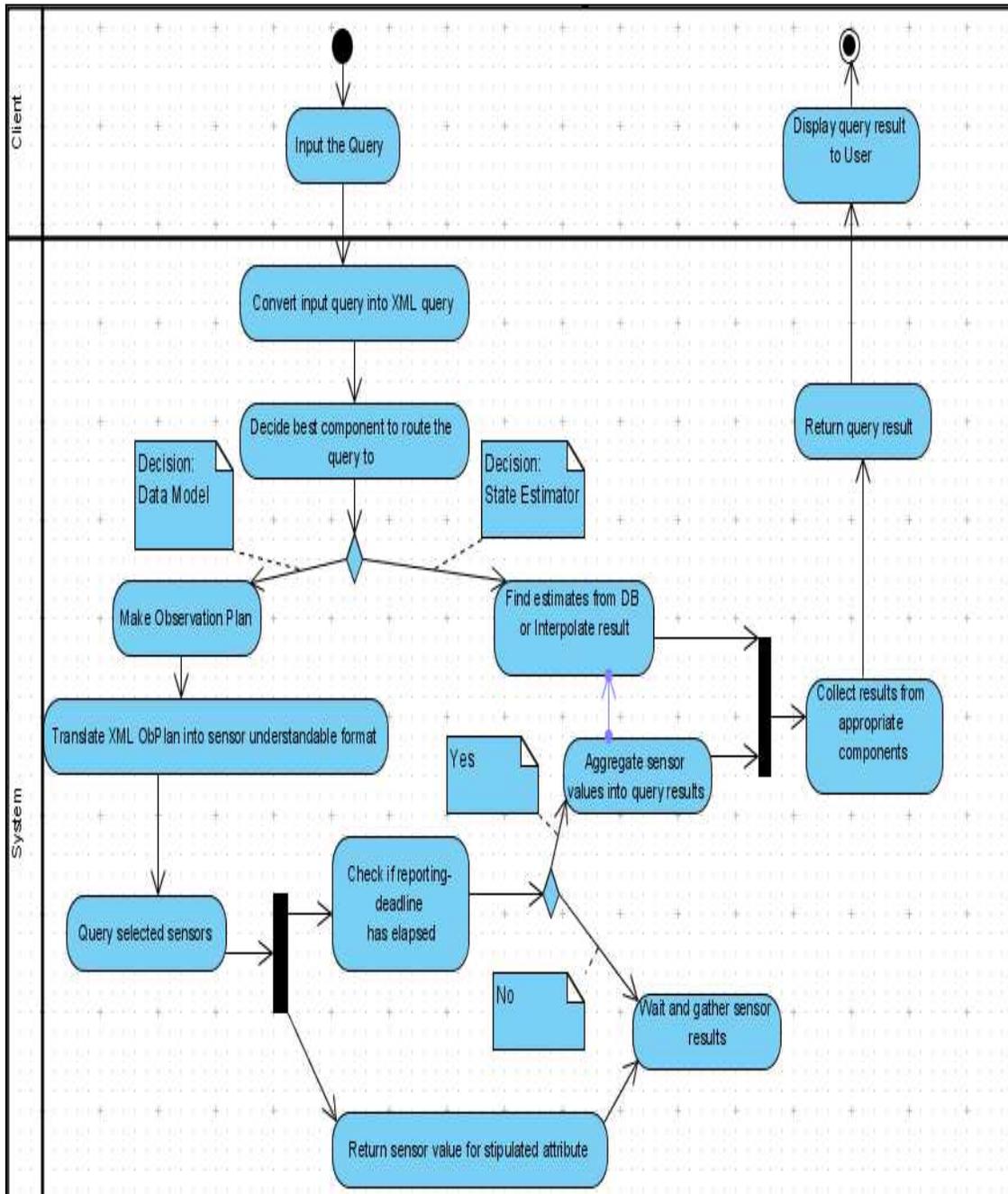


Figure 4: CoSMo: Activity Diagram. It depicts all the decisions taken and steps performed by internal components as the query passes through them.

### 3.3.3. Package Diagram

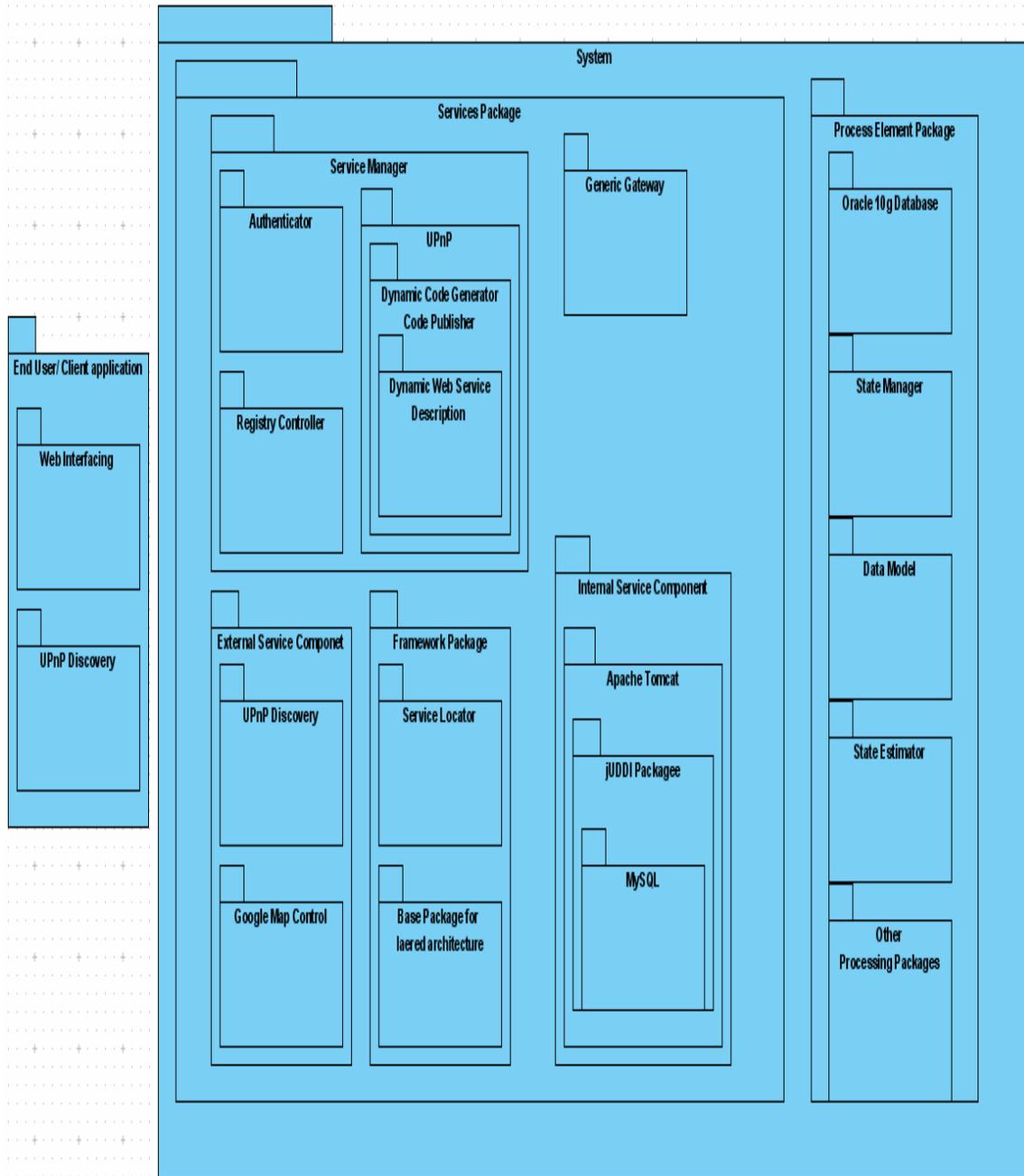


Figure 5: CoSMo: Package Diagram. It outlines all the packages used in the CoSMo system.

### 3.3.4. Deployment Diagram

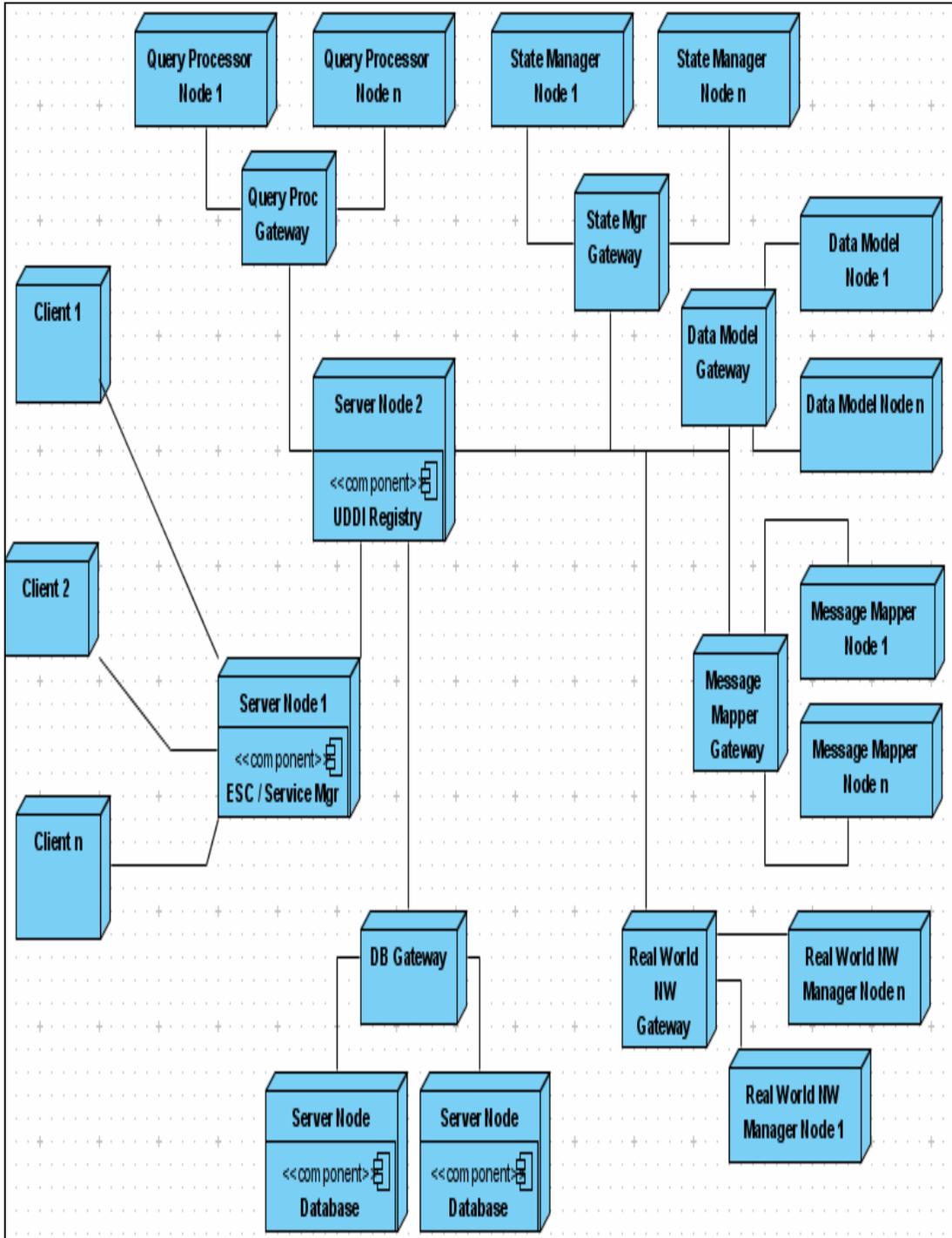
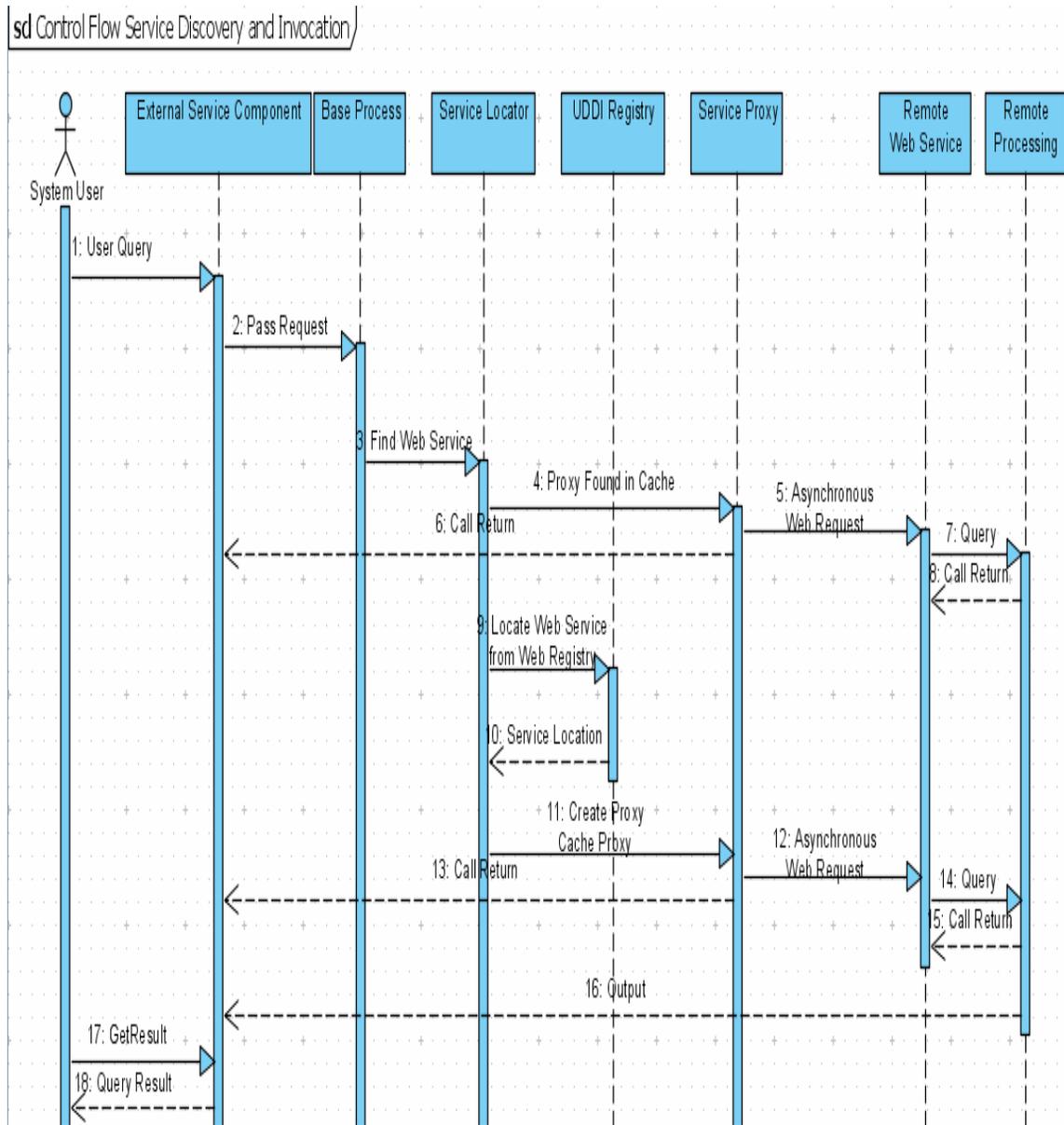


Figure 6: CoSMo: Deployment Diagram. Typical system outlook when deployed in a real environment.

### 3.3.5. Communication Diagram



**Figure 7: CoSMo: Communication Diagram.** It depicts the control flow between the CoSMo components as a request is made between the components.

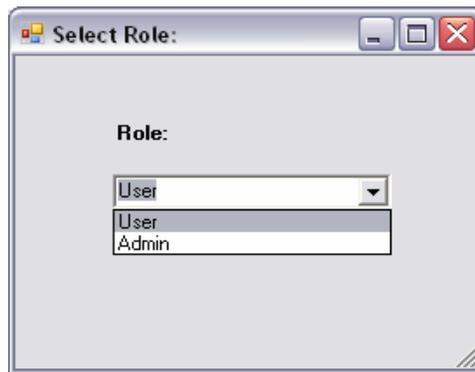
### 3.3.6. Use Cases

Following use cases consider two scenarios. Use case 1 depicts the query flow wherein the State Manager selects the State Estimator as the best component to further process the query. Use case 2 demonstrates the case where the query is forwarded to the Data Model and hence out to the sensor network.

### Use Case I:

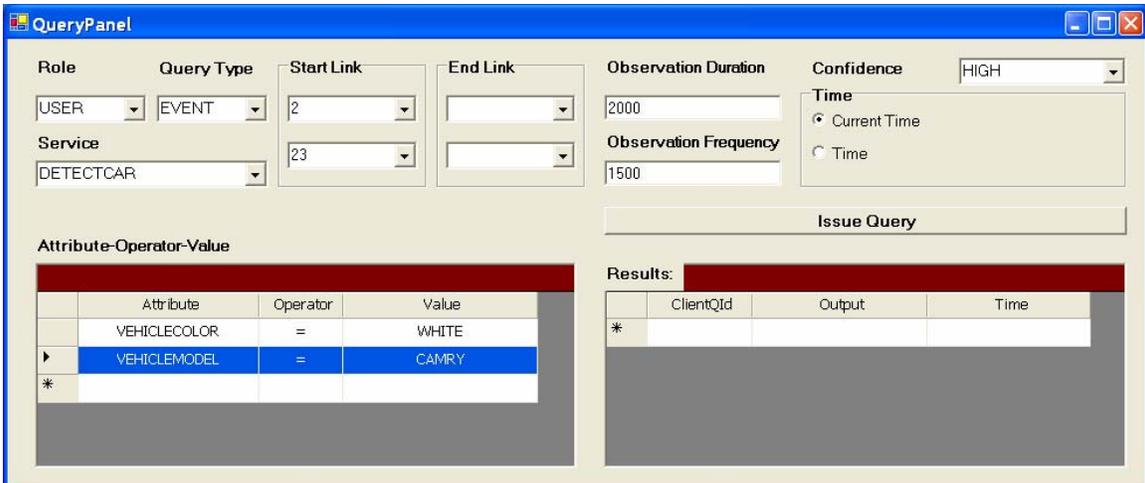
**User Query:** Get delay on link 12 between time 8:50 A.M. and 1:20 P.M and get the result with a Medium confidence level.

- i. 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.



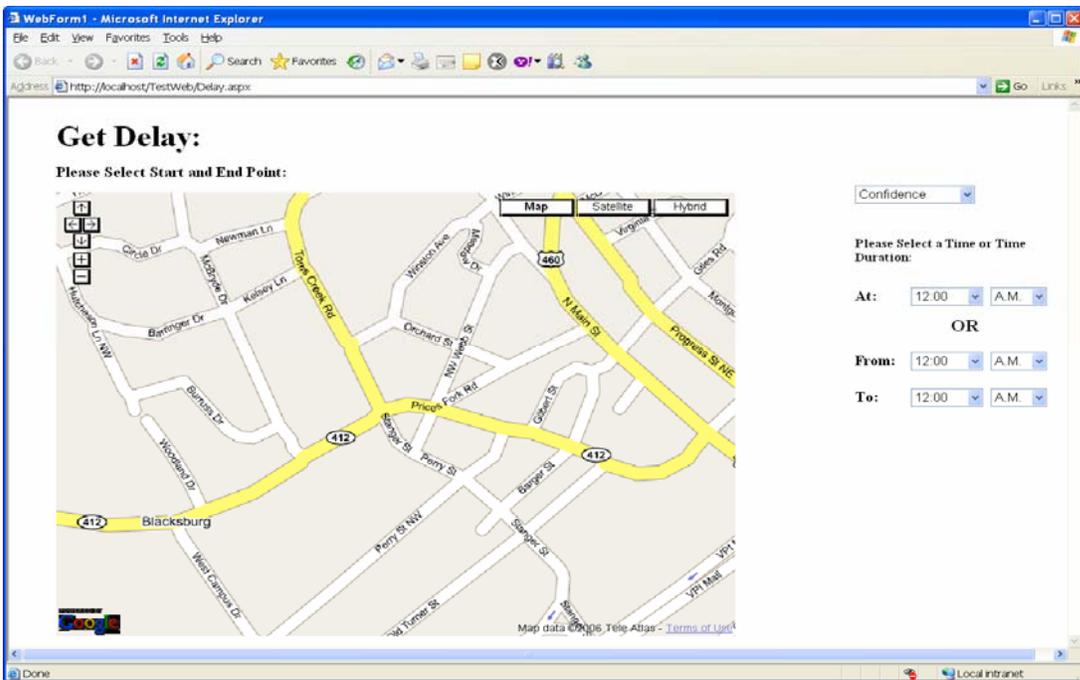
**Figure 8: User Role Selection Screen. Two types of roles are provided, namely user and administrator.**

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 9: Win-Forms based Client User Interface. Broadly it is divided into parts, namely, query special options, query attributes and query output part.**

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 10: Google Map-based User Interface. Get Delay service prompting user to select two points on the map and specify the confidence and time range for the query.**

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

- ii. 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.

- iii. 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.
- iv. The Query Processor formats this message string into tagged a XML stream according to a query template specified by an XML schema. Details of the query template can be found in Chapter 6. The Query Processor assigns query ids and forwards this XML to State Manager via access point lookup in the UDDI registry.
- v. The web service access locations point to the gateway module for the succeeding component. Gateway module is used to illustrate that the CoSMo system can be easily ported to a grid environment in future. The Gateway selects the best

- component from available distributed and duplicated components. The selection process depends on component workloads, business constraints and quality parameters.
- vi. 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.
  - vii. In this case, the compound query is routed to the State Estimator.
  - viii. 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.
  - ix. The State Manager passes back the results to the Query Processor which in turn sends it to the ESC.
  - x. 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.

- i. 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.

- ii. 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.*

- iii. 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.

- iv. 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 6. The Query Processor assigns query ids and forwards this XML to State Manager via web service access location lookup in the UDDI registry.
- v. 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.
- vi. 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.
- vii. In this case, the compound event query is routed to the Data Model.
- viii. 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 the Sensor Network in the Database which maintains knowledge 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 sub queries contain one main attribute each, such as VehicleColor and VehicleModel, their operators and the required values.

- ix. 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.
- x. 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.
- xi. 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 the same values, etc. For the same query-id, the data model aggregates the values for the encompassed attributes by taking into account the conditions of AND, OR, etc.

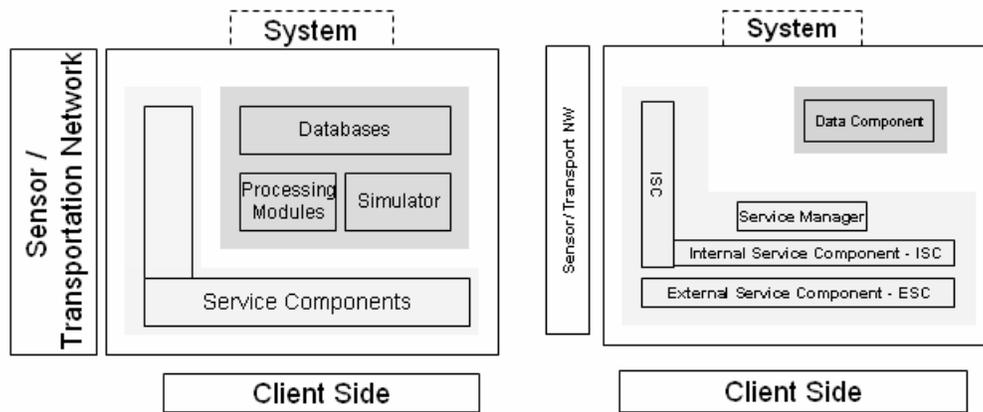
In this case, we have multiple attributes namely, VehicleColor and VehicleModel. The detection of these individual attributes is indicated by the sensors and the aggregate result suggesting the recognition of a white color Camry depends on whether the individual sub-queries detected their attributes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

- xii. The Data Model periodically returns the result of the detection of the event to the State Manager for the specified time duration.
- xiii. The State Manager sends this result to the Query Processor which in turn returns the values to the ESC.
- xiv. 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.

## 4. Service Components in CoSMo

Broadly, the CoSMo system architecture is divided into two parts (i) Cognitively inspired data components, and (ii) Services related components. This thesis focuses on the Service Architecture and Service Components. We have briefly described other modules while describing the use cases. For all the internal implementation details pertaining to these data components please refer to Phalak's [59] thesis.

A schematic block diagram of CoSMo is shown in Figure 11. For the rest of the thesis we will use the block diagram in the right panel.



**Figure 11: CoSMo: System Overview. The left panel shows CoSMo's overall architecture; the right panel focuses on the service components.**

As discussed previously system is divided into three main parts namely Client Side, External World consisting of Sensor and Transportation Network and System Core consisting of Service and Data Components. The rest of this section will focus on Service Components as shown in Figure [11].

The Following acronyms are used in representing the service components:

(i) Service Manager – SM: Core of Service Publishing infrastructure within CoSMo,

(ii) External Service Component – ESC: Acts as a Gateway for external client to interact with the system and

(iii) Internal Service Component – ISC: It is a Web based UDDI web service registry which stores information about all the web services available in the system.

Let us start the description from client side.

## **4.1. Client Side**

Client Side consists of different application running on devices such as Laptops, PDA's, and Blackberries etc. making queries to CoSMo. Typical users of the system include, end users who are interesting in obtaining the traffic information and traffic planners, who are interested in understanding the overall traffic conditions throughout the city.

### **4.1.1. Design**

CoSMo supports for two types of users, namely, normal user and administrators. These users can issue event or value based queries within CoSMo. Apart from web pages and web service CoSMo also supports queries issued over UPnP.

#### **Types of User Roles within CoSMo**

To strengthen our system further, we provide two types of Roles for accessing the system from outside.

- i. **User Access** – Normal users make queries related to a particular application of our system. It represents the business/ application side of CoSMo. Typical examples are queries issued by end user for short term route planning. For example, Get Delay, Get Best Route etc.
- ii. **Admin Access** – Along with the normal users, we envision administrators playing some role in managing the system from outside the system, by giving them access to the core functionality from outside. General administrative capabilities which one can expose are sensor network management, monitoring, notification capabilities and some level of parameter controlling capabilities. Administrator

can monitor system for its correct behavior and optimize its performance. Services with finer granularity are required to allow such additional flexibility.

To illustrate administrative role at the client side currently CoSMo has a simple power management service in which it can get and set the power usage criteria within a zone. Along with this, system administrators can randomly sample the link delays and other relevant information. This information can then be compared with the actual values obtained from surveys or other live data acquisition systems.

In the current implementation CoSMo stores client identification and the corresponding query results at the server side. These results get polled from the clients for output. This information can be used effectively for future enhancements. Using this information one can find the context in which the queries were issued by the client. User preferences and activity patterns can be saved and user experience can be further enhanced by intelligently using this information. This extraction though interesting is beyond the scope of this thesis.

### **Types of Queries**

CoSMo currently supports two types of queries, namely, value-based queries, event-based queries and hybrid queries. Further information about these types can be found in Phalak's [59] thesis.

### **Link and Path Based Queries**

In the current implementation, the user interface supports issuing link based and path based queries. This distinction helps us in illustrating the point of smart devices and selective services at the client side. User might not ask for information over the entire path he is traveling. Because of cost constraints, confidentiality or availability of other sources he can just be interested in knowing conditions at a segment of his journey. He

might use his experience, judgment or even other methods such as a smart application on his handheld helping him plan his entire journey using selective queries over our system.

### 4.1.2. Input – Output for Client Side

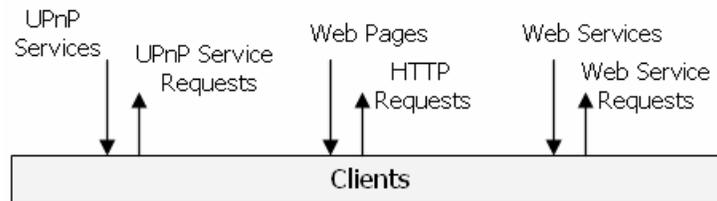


Figure 12: Input - Output for Client Side

There are three ways of accessing the system from outside. The clients can use UPnP service invocations, web pages and UDDI web services to interact with the CoSMo. Input to the client side is in the form of the services published by the system and output is the query issued by devices over these published services.

### Win-Form based application

Following diagram shows Win-Form based application used for interacting with the system.

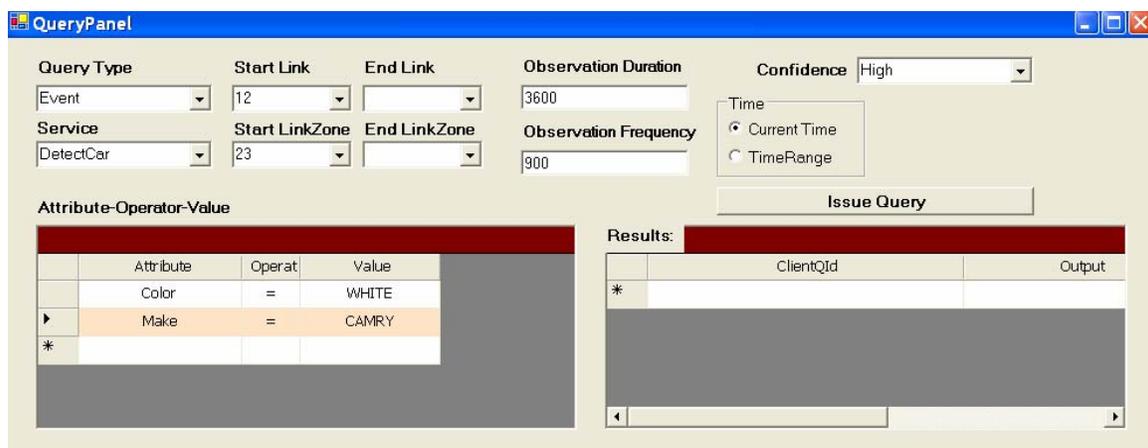


Figure 13: Win-Forms based Client User Interface. Broadly it is divided into parts, namely, query special options, query attributes and query output part.

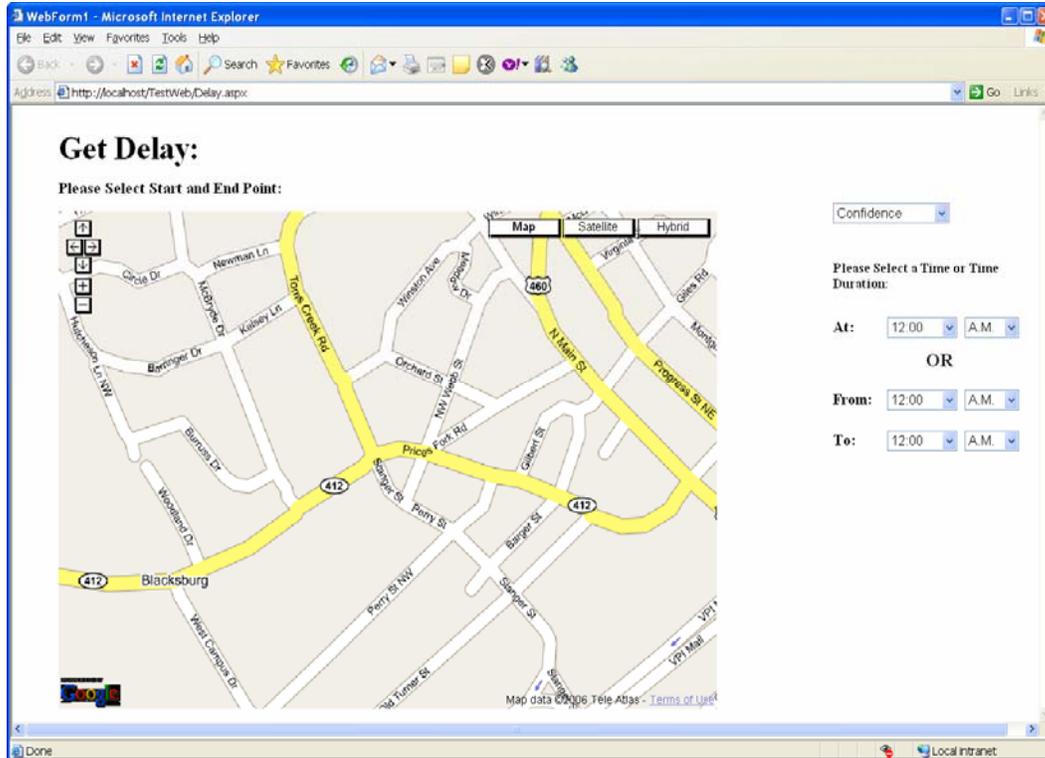
Various fields listed on the user interface are Query Type, Links to observe, Service, Confidence and Time at which the service is required. It also contains a grid for specifying observation attributes i.e. Attribute-Operator-Value pair and it has a Query Result grid displaying the client query id verses its output information.

Typical example of a client query is:

*“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”*

## Google Maps

Apart from Win-Form based application, CoSMo has a Google Map interface. Currently we do not have Google Map control enabled since as we do not the data that map user clicks to the actual locations. Sample prototype with Google Maps is as shown below. This data can be purchased from the Google. The system is designed so as to be integrable with it.

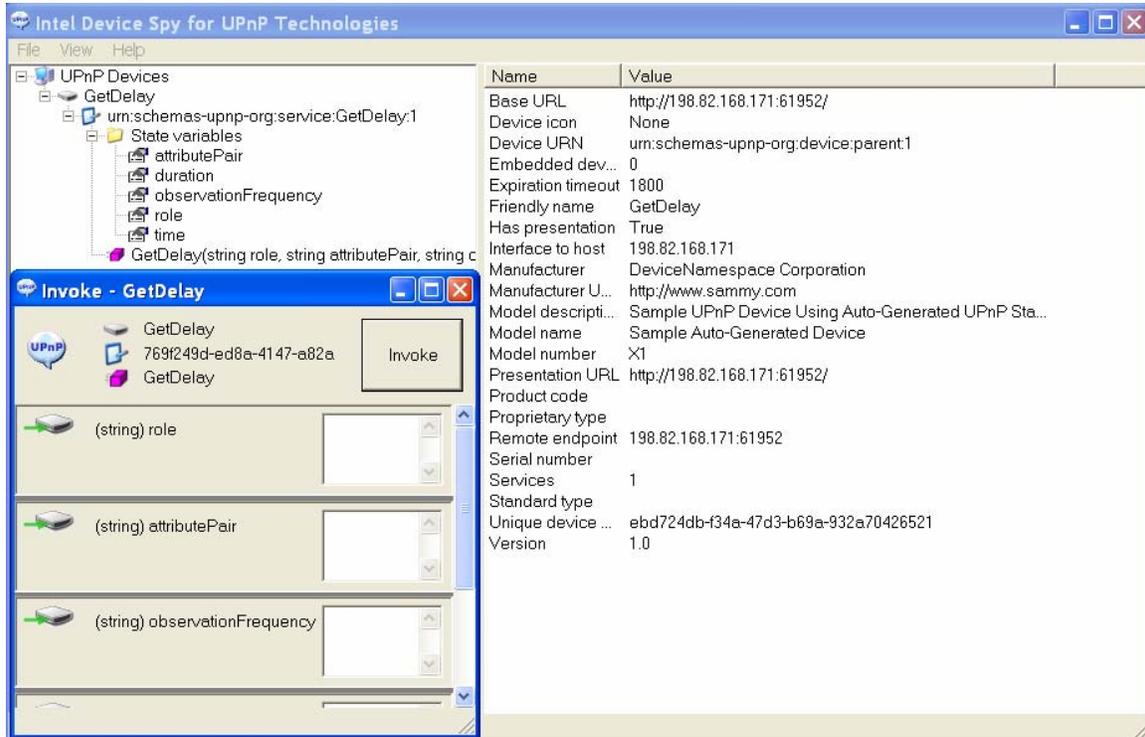


**Figure 14: Google Map-based User Interface. Get Delay service prompting user to select two points on the map and specify the confidence and time range for the query.**

## UPnP based clients

As CoSMo supports UPnP, smart clients capable of UPnP discovery capabilities can also interact with the system.

Following figure [16] shows Intel's UPnP Device Spy tool which can discover UPnP devices and services on the network.



**Figure 15: UPnP-based User Interface.** A device named GetDelay with service GetDelay is published over UPnP. The service supports all the parameters necessary to issue a delay query over UPnP.

## Motivation for using UPnP

UPnP stands for Universal Plug and Play. UPnP defines architecture for pervasive peer to peer computing. It is a fairly light weight protocol, widely implemented in handhelds and it has strong industry support from many companies including Microsoft. Using UPnP, devices can dynamically join a network, discover the existing services as well as start publishing their own services. UPnP is designed to work efficiently with Ad-hoc networks [61][42].

Consider an end user with a UPnP enabled PDA driving down the road. If the PDA is not connected to the Internet, then there is no way for the user to interact with the system. UPnP on the other hand can provide access to the system without any Internet infrastructure. This feature increases CoSMo's use and adds further value to the CoSMo system, thus justifying the decision to provide UPnP support within CoSMo.

### 4.1.3. Implementation Details

In the current implementation CoSMo supports access over window forms in C#. Google Map interface is currently disabled and UPnP access is available over UPnP presentation web pages. One can easily write a client side application which will communicate with the CoSMo system over direct UPnP rather than using presentation web pages.

The system currently allows the selection of Event based queries such as 'Detect Car' and 'Traffic Jam' and Value based queries such as 'Get Delay' and 'Count Vehicles'.

### 4.1.4. Summary of Module: Client Side

<b>Module Name</b>	Client Side
<b>Our Contributions</b>	(i) Implemented windows forms client application in C#.  (ii) Implemented UPnP discovery application at the client side and tested the UPnP functionality.  (iii) Support for UPnP in CoSMo increases it's usability by addressing a wide variety of users.
<b>Purpose/ Capabilities</b>	(i) Denotes client side. Includes laptops, PDA's and smart devices.

(ii) Queries the system for output. (Value based or Event based queries).

(iii) Browser capabilities, UPnP or .NET enabled devices.

**Input/ Output**

Input: null

Output: A user initiated query from browser, UPnP or .NET application

**Technologies Used**

ASP.NET, UPnP, Win-Forms, C#, Google Maps

**Table 1: Summary of Module: Client Side**

Now we will move on to the next component, namely, External Service Component.

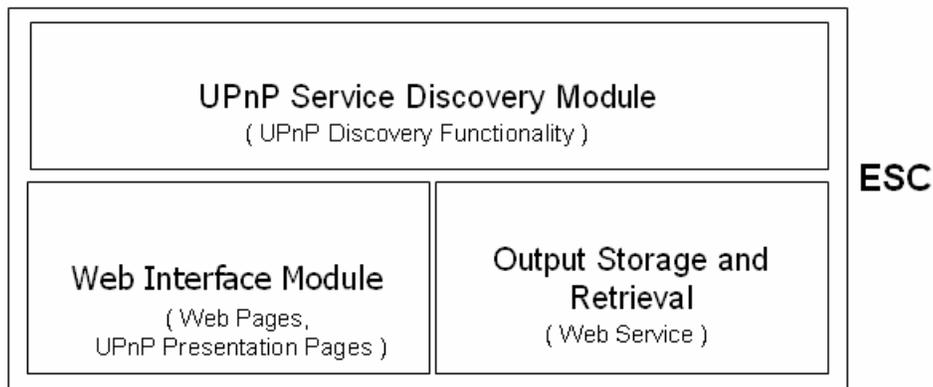
## 4.2. External Service Component (ESC)

The External Service Component acts as the entry point for external world for interacting with our system. It provides a clean interface for clients to issue the query and it can be used to make initial routing decisions for the received query. Currently we route all queries to Query Processor. Client can access the system over web pages, UPnP or web services. All the web pages are stored at External Service Component and exposed to the external world.

### 4.2.1. Design

CoSMo separates External Service Component from the Service Manager as it provides a cleaner interface to the outside world. It also gives us a chance to add on functionalities at the interface to improve the user experience.

CoSMo also supports a secondary UPnP control point within the system to expose UPnP services to users which are not UPnP enabled. Another important functionality added at the External Service Component is the storing of client specific information which can be used in the future for data mining over user navigational patterns [16].



**Figure 16: Internal details of External Service Component (ESC).**

ESC consists of three sub-modules (i) Web Interface Module, (ii) UPnP Service Discovery Module and (iii) Output Storage and Retrieval. A brief description of each of the parts is as follow

- i. Web Interface Module - It contains normal web pages as well as the UPnP presentation web pages. These web pages expose system functionality to the outside world.
- ii. UPnP Service Discovery Module – CoSMo has a secondary control point within the system. It makes available UPnP functionality to those client devices who do not understand UPnP. This secondary control point discovers UPnP devices published by Service Manager and exposes presentation web pages to external world clients. UPnP presentation web pages are the web pages provided by the service providers for controlling UPnP services. Most of the times these web pages expose very restricted access to the service.

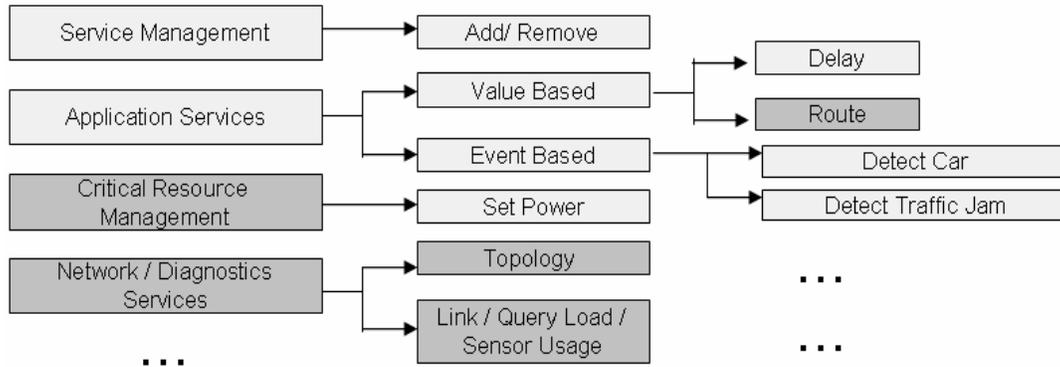
Incorporation of the UPnP discovery capability was an interesting and challenging idea. It has addressed those devices which can not talk UPnP but are willing to use services provided by the UPnP service providers.

- iii. Output Storage and Retrieval - External Service Component also holds the results for the issued query if it can not be passed to the client directly. Typical examples would be event based queries in which system needs to send back the results after some time interval to client. In such cases it is better for clients to poll the values rather than keeping the connection open all the time or making calls to the client device. As discussed previously, one can use this information intelligently to enhance user experience future by extracting user preferences from the stored data. Another scenario where this approach would be useful is temporary disconnection of client from the system.

UPnP discovery module internally passes on UPnP presentation web pages to web interface module. All these modules make External Service Component an important and indivisible part of CoSMo architecture. In the next section we describe different types of services which can be exposed to the outside world.

## Typical System Services

Typical Services exposed by the ESC can be categorized into different types depending on the purpose they serve.

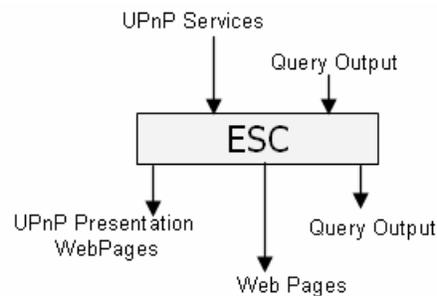


**Figure 17: CoSMo: Typical Application Services. Light shaded boxes show the services currently available within the CoSMo system.**

- i. **Service Management** – Provides functionality to control addition of or removal of existing services.
- ii. **Application Service** – It includes all the services in which a normal user would be interested in. It can be further divided into Value based and Even based services. Typical examples are Temperature Service, Route Service, Detect Traffic Jam etc.
- iii. **Critical Resource Management** – This is particularly useful for administrators at the client side. They can manage the critical resources within CoSMo. For example, they can manipulate sensor power consumption levels from outside as well as they can change the sleep/ wake times of sensors from outside.
- iv. **Localization/ Topology Service** – It provides administrators with location specific information of sensors. Topology information can also be given out for administrative purposes.
- v. **Maintenance or Statistical Service** – Administrators can invoke maintenance calls or statistical report generation queries using this service.

Not all the services are provided at this time. Some of the services and functionalities illustrating the strength of our architecture are implemented. Addition of any more services can be added with ease because of the existing infrastructure. Services marked with light shade are currently implemented.

#### 4.2.2. Input – Output for External Service Component



**Figure 18: Input-Output for External Service Component**

- i. We have taken the decision to place all the web pages providing services to the outside clients inside External Service Components.
- ii. Within CoSMo, ESC has a secondary control point which acts as UPnP discovery component within the system. It discovers UPnP services published by Service Manager and publishes UPnP presentation web pages to the external world. This functionality makes UPnP services available to those devices which are not UPnP enabled. Presentation URL's (presentation web pages) are another way in which UPnP functionality can be exposed. These web pages expose limited, striped down functionality for interacting with the UPnP services.
- iii. Query output is also stored at ESC. Clients poll ESC and fetch the results.

#### 4.2.3. Implementation Details

In the current implementation External Service Component provides access over ASP .Net web pages, C# webs services and UPnP presentation web pages. It has a UPnP

discovery module implemented which discovers UPnP services published by Service Manager and displays it to the external world.

External Service Component also stores client and query specific data. Clients can poll this output storage to retrieve the final output for the issued query. In the future, we envision distributed External Service Components spread geographically away from the system core and exposing system functionality to large number of users. This approach is similar as having a wireless base station for serving a limited set of users and will help us in localizing most of the information specific to users at these peripheral locations.

#### **4.2.4. Summary of Module: External Service Component**

<b>Module Name</b>	External Service Component (ESC)
<b>Our Contributions</b>	<p>(i) Design and Implementation of a secondary UPnP discovery control point. This was an interesting and novel idea since it allowed large number of handhelds to talk to the system.</p> <p>(ii) Query output storage at ESC has opened up a complete new domain of intelligent data mining over user collected data.</p> <p>(iii) Design and implementation of the web pages.</p> <p>(iv) Google Map control was successfully hooked into CoSMo. Currently disabled because of lack of location specific data mappings.</p>
<b>Purpose/ Capabilities</b>	<p>(i) Denotes external world interface. Entry point to the system.</p> <p>(ii) Client can explore and invoke the services exposed by</p>

ESC.

(iii) Internally divided into three parts namely UPnP Service Discovery component, Web Interfaces and Output Storage and Retrieval module which stores client specific output information.

**Input/ Output**

Input: User Query or null

Output: Passes the query to Service Manager for further routing. Carry outs initial authentication of the user with the help of Service Manager.

Output for the issued query

**Technologies Used**

ASP.NET, UPnP, C#, Google Maps

**Table 2: Summary of Module: External Service Component**

Now we will move on to the next service component namely Internal Service Component.

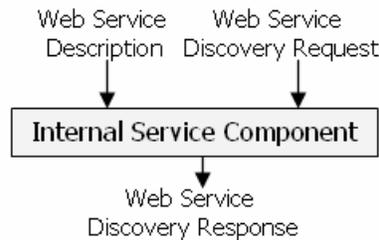
### **4.3. Internal Service Component (ISC)**

Internal Service Component is a centralized Universal Description, Discovery and Integration (UDDI) registry, a place where service information is stored. It is a web based registry that maintains details about service providers, service implementations and service metadata. It consists of a standards-based set of specifications which allow service description and discovery and improves system accessibility [33].

#### **4.3.1. Design**

CoSMo has a centralized private UDDI registry for storing service specific details. The entire communication infrastructure is centered on this service registry. We changed the way CoSMo services are stored and interpreted within the system. This change was logical in nature and did not entail any change in the physical storage of the service details or UDDI specification. The change was motivated by incorporation of Gateway Logic in the code. Details about this change can be found in section 8.3.3.

### 4.3.2. Input – Output for Internal Service Component



**Figure 19: Input - Output for Internal Service Component**

A UDDI entry actually contains more than a WSDL interface and implementation; it can also include further metadata such as quality of service parameters, payment mechanisms, security, and keywords for resource discovery [33]. Service providers register their services as well as they perform a lookup for particular service. Detailed information about the Service Names, Description, Type, Calling and Access semantics for each service are published here. This registry is similar to an online UDDI database previously hosted by Microsoft and IBM. We have opted for a private in-house registry for our system as it adds advantages in terms of performance, self-sufficiency and security.

UDDI adds redundancy to the web services architecture by providing a centralized repository of services. It exploits the advantages offered by web services to the full extent. We have made a slight modification to the way services are stored in the system. These details and advantages are discussed in detail in the appendix. Also, the latest UDDI specification has support for grid environment. In the future if required we can have multiple copies of registries co-coordinating with each other. In the current implementation, redundancy is in terms of multiple web services and components performing similar tasks.

### 4.3.3. Implementation Details

Though current version of CoSMo runs on .Net and Windows, a lot of effort has been made in making the CoSMo system run on an open platform. Decisions were made to which will make this future transition less cumbersome. One example of this would be

the decision to go for open source implementation of UDDI standards. i.e. 'jUDDI'. It is a web application implementing UDDI standard. jUDDI is hosted in Apache Tomcat Server. Though CoSMo uses Oracle 10g as system database, we decided to add a light weight and local data storage in terms of My-SQL database for managing services. In the current implementation, use of C# is affecting portability, but soon it will be overcome as substantial progress has been made in porting C# code on Linux systems.

#### **4.3.4. Summary of Module: Internal Service Component**

<b>Module Name</b>	Internal Service Component (ISC)
<b>Our Contributions</b>	<p>(i) CoSMo modified the way CoSMo components register their services within the UDDI registry. This change (discussed later in Appendix) was well thought of and proved crucial in adding Gateway functionality within the CoSMo system.</p> <p>(ii) This change was related with service interpretation which is stored in standard structures in UDDI database. It was made possible because redundancies defined by UDDI standards in terms of Service Categorizations and it will not affect normal functioning of the UDDI registry.</p> <p>(iii) Rather than using default UDDI registry support provided with Windows Server 2003, we decided to use open source technologies and integrated Apache Tomcat Server, a Java implementation of the UDDI standards ( jUDDI ) and My-SQL database into a single package which collectively acts as CoSMo's Internal Service Component.</p>

<b>Purpose/ Capabilities</b>	Private Web Service Repository.
<b>Input/ Output</b>	Input: Service publishing request or service locate request Output: Publish services in the database, Give out service details.
<b>Technologies Used</b>	UDDI, My-SQL, C#, Apache Tomcat

**Table 3: Summary of Module: Internal Service Component**

Now we will move on to the next service component, namely, Service Manager.

## **4.4. Service Manager**

Service Manager is designed to handle the entire service management within CoSMo. It is the core of the CoSMo's service publishing infrastructure. It knows about the capabilities of the individual components and interfaces with Internal and External Service Components to create a service infrastructure in which all the services are registered and discovered. Service Manager User Interface is used to enter service specific information.

### **4.4.1. Design**

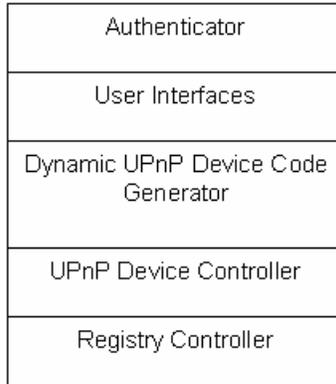
Service Manager was designed to take care of all internal and external service publishing infrastructure. In the current implementation Service Manager User Interface is the only way to enter the service details in the system.

Service Manager interacts with Internal and External Service Component to setup the service infrastructure at the start of the system. One can dynamically change the available services using Service Manager User Interfaces.

#### **Service Manager Modules**

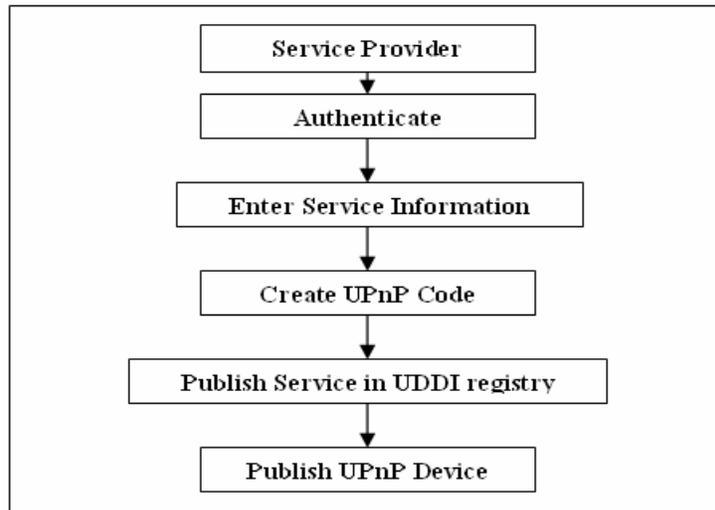
Service Manager controls service discovery and service publication functionalities within CoSMo. It is divided into

- i. Authenticator to authenticate the user
- ii. User Interfaces to enter service specific information
- iii. UPnP Dynamic Device Code Generator to create UPnP C# device stack
- iv. UPnP Device Controller to control UPnP Devices
- v. Registry Controller to manage UDDI registry



**Figure 20: Composition of Module: Service Manager**

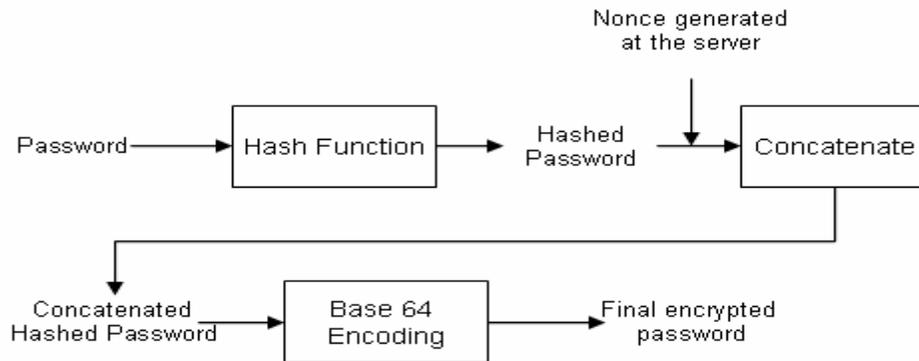
We will describe individual modules shown in Figure 20 in depth along with the service publishing flow as shown in Figure [22].



**Figure 21: Service Creation and Publishing within CoSMo**

### **Authenticator**

CoSMo uses a preliminary authentication scheme, in which, we transfer passwords in an encrypted format over the Internet. Within the system, passwords are stored as hashed values. For security purposes we are using nonce mechanism i.e. a random number is generated per login attempt, which, along with the hashing, is used for authentication. Overall mechanism is as follows:



**Figure 22: Authentication within CoSMo. Passwords are transferred in the encrypted format over the Internet.**

- i. First the password entered by the user is hashed.
- ii. For each new login, server generates a nonce which is a random number. Hashed password is concatenated with this nonce and base 64 encoding is performed over this concatenated string.
- iii. The output of this base64 encoding is transferred over the network and is validated at the server.
- iv. Server code performs the same operation at the server side and matches the string received from the client. If both the strings match then user is authenticated and given access to the system.

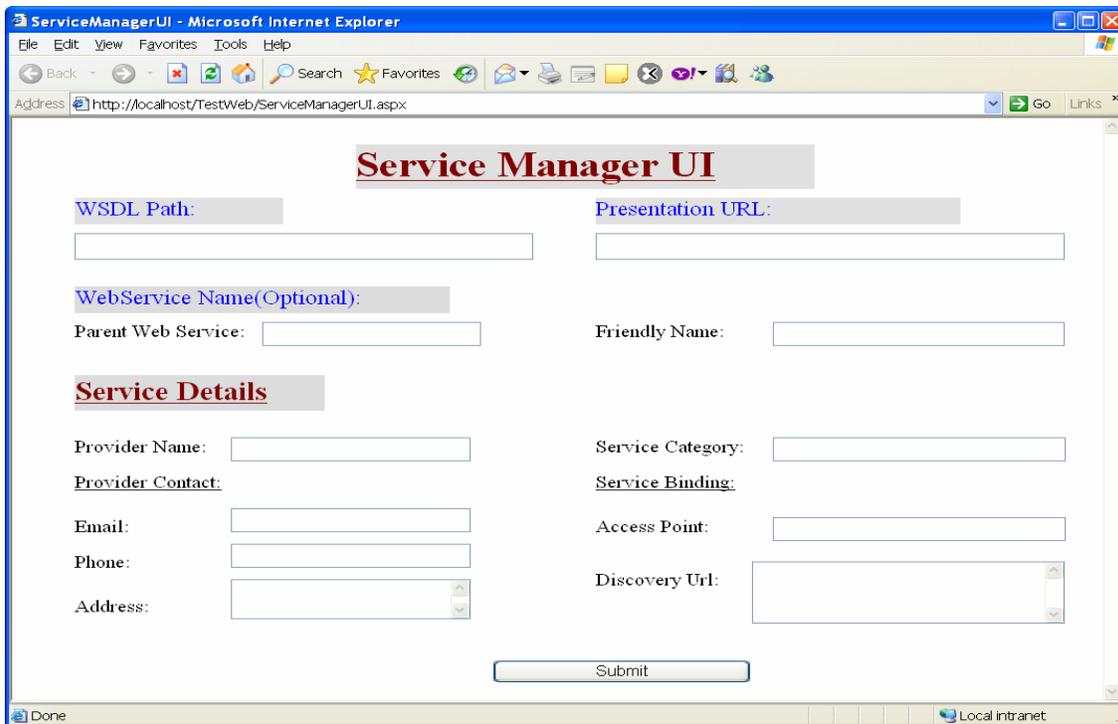
Once a user is authenticated, it can use Service Manager User Interface to enter service and business specific details. Service Manger uses this information to discover, create and publish these services within the network

### **User Interfaces**

Within the current implementation Service Manager User Interface is the only way of entering service information within the CoSMo system. Figure [23] shows Service Manager User Interface. It contains fields for all the necessary information required to publish a web service. It includes service specific details along with the service provider details.

Some of the important fields are:

- i. WSDL Path - Points to the Web Service Description format for the web service.
- ii. Parent Web Service – We can specify parent-child relation between the services with this field. Used in case of UPnP services. This information can be used to display services hierarchically.
- iii. Service Details – It contains Provider Name, Email, Address and Phone number details of the Service Provider. One can also specify the category to which that service belongs, access point or location where that service can be found etc.



**Figure 23: Service Manager User Interface. It provides fields for entering the service specific information.**

After successful login, businesses feed web service descriptions and other service/business specific details to the Service Manager User Interface. This information is then passed to the Dynamic UPnP Device Code Generator which contains web service discovery capabilities.

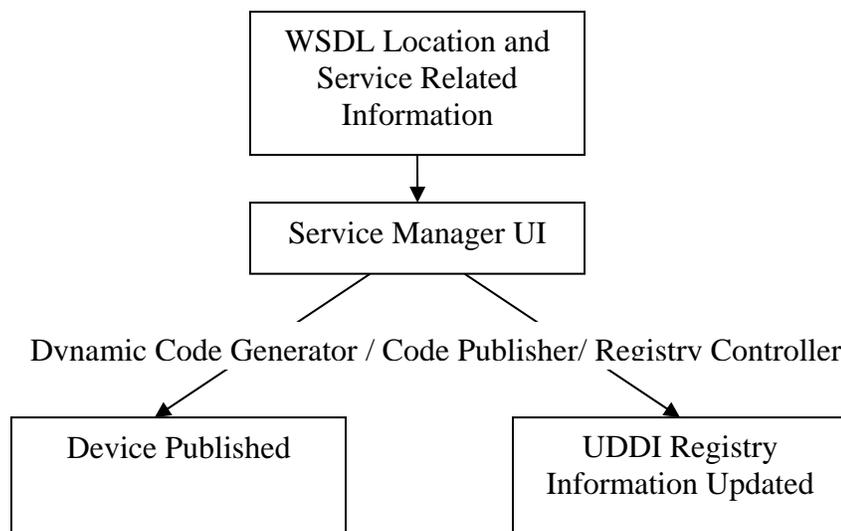
## Dynamic UPnP Device Code Generator

Dynamic Code Generator is a UPnP specific module. It creates the UPnP device stack in C# on the fly. Once it receives the web service description from the Service Manager, it performs a dynamic discovery of service contents and creates a memory representation of all the web methods and state variables for that web service. Using this input it creates a UPnP device code stack at runtime.

## UPnP Device Controller

Device Controller compiles the code generated by UPnP Code Generator and activates the devices and starts publishing these devices on the UPnP network. It keeps track of all available devices and provides functionality to start and stop device publishing at any time. We refer to Intel UPnP Code stack [8] for dynamic UPnP code generation.

Automated Flow for UPnP device and Web Services publishing:



**Figure 24: Automated Flow of Service Publishing in CoSMo. Most of the manual steps related with creating and publishing a UPnP device on the network are automated.**

Automated flow consists of following steps for creation and publication of services

- i. Web Service description creation.
- ii. Input of all the methods and parameters in the system.

- iii. Use of service authoring tool for creation of Service Description files (e.g., Service Author from Intel).
- iv. Use of device building tool for creation of device stacks code. (E.g. Device Builder from Intel).
- v. Compilation and Activation of the generated device.
- vi. Entry of Web Service description and associate accessing information in UDDI registry.

	<b>Steps</b>	<b>Description</b>
1)	Web Service Description Language	Create Web Service and Associated description
2)	Service Author tool from Intel	Read WSDL, List down variables and methods manually
3)	Service Description File	Service Author creates service_description.xml
4)	Device Builder Tool from Intel	Device Builder takes service description file and creates a C# UPnP code
5)	UPnP stack is compiled and run to publish the device	Services offered by the device are available on the UPnP network
6)	UDDI registry entries	Web Services are published in UDDI registry.

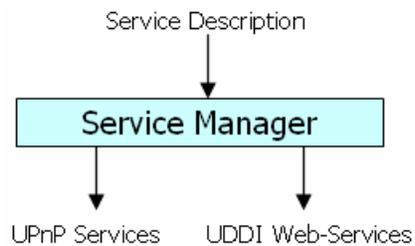
**Table 4: Steps Covered in Automated Flow of Service Publishing**

### **Registry Controller**

Finally the discovered service information along with the specific service details from Service Manager User Interface is written into the UDDI registry. Registry Controller module interfaces with the UDDI private registry node and populates this service information. Registry Controller is used by all the components to talk to the UDDI registry. UDDI registry is maintained in My-SQL database and jUDDI is used to talk to the registry. Registry controller directly talks to jUDDI application which is running

under apache tomcat server. Users are checked for proper access rights before making any changes to the registry.

#### 4.4.2. Input –Output for Service Manager



**Figure 25: Input - Output for Service Manager**

Service Manager is fed with the service descriptions at the start of the system. It creates UPnP, Web Service infrastructure and is responsible for dynamic service management within the system. Currently CoSMo supports Web Service Definition Language (WSDL), a widely known and used web service description format for discovering the services.

Responsibilities of Service Manager can be broadly summarized as:

- i. Create and Publish UPnP devices on the intra-net.
- ii. Expose services within the system into the UDDI registry, i.e., data model, interpolator, simulator, sensor network, databases, etc., query this information and talk to each other.

#### 4.4.3. Implementation Details

Service Manager dynamically discovers the web service description and creates a UPnP C# code stack on the fly. This code is compiled and published on the network by UPnP Device Controller. At the current stage CoSMo supports exposing UPnP presentation web pages and cannot directly create event handling code within UPnP. One needs to write the actions to be performed when a client invokes the service over direct UPnP. We

believe addition of event handling code will not be difficult as we have all the necessary information available to create the event handling code automatically.

#### 4.4.4. Summary of Module: Service Manager

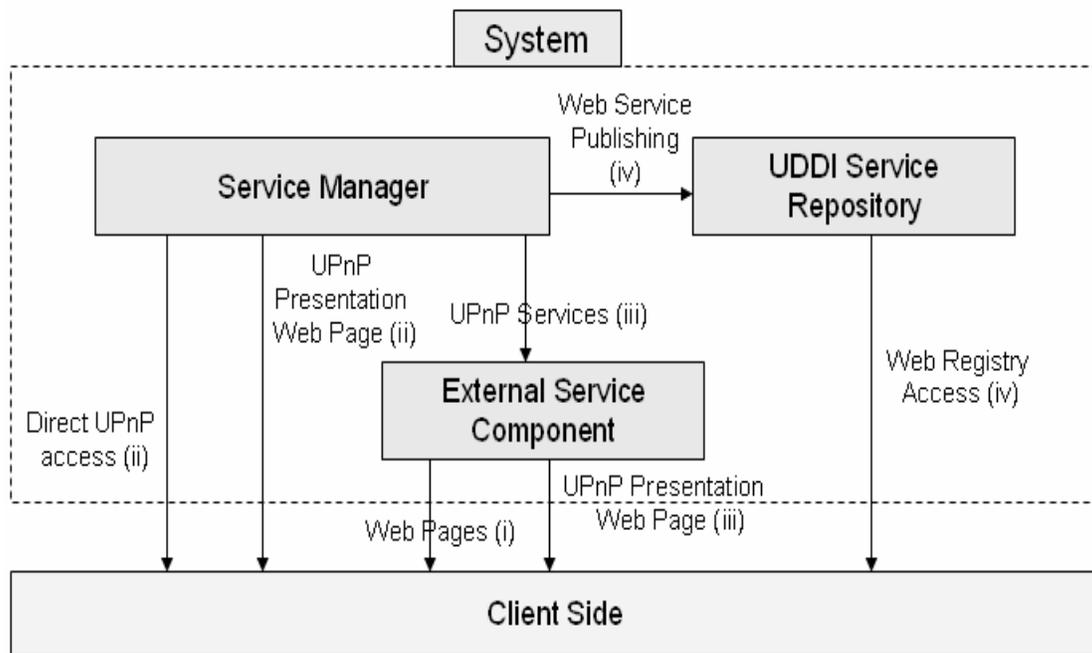
<b>Module Name</b>	Service Manager
<b>Our Contribution</b>	(i) Implemented on the fly generation and publication of UPnP devices in C#.  (ii) Creation of the service infrastructure which can be used by internal components as well as by external applications to build upon.
<b>Purpose/ Capabilities</b>	Talks to Internal UDDI service registry.  Creates and Publishes UPnP devices on the fly and exposes them to outside world.
<b>Input/ Output</b>	Services Input: Web Service descriptions.  Services Output: Makes UDDI registry entries, Create UPnP devices, Publishes UPnP services.
<b>Technologies Used</b>	ASP.NET, UPnP, C#, Google Maps, UDDI, Dynamic Code generation.
<b>Comments</b>	Query flow might not always goes through ESC. We have UPnP, UDDI mechanisms in place to facilitate external client interaction with the system. For example, a UPnP enabled client with no internet connection, might on the fly discover the UPnP services published by Service Manager and invoke them.

**Table 5: Summary of Module: Service Manager**

Now we will move on to the next section which describes how all service components interact with each other and provide services to the client side.

## 4.5. Interaction between Service Components

Figure [27] summarizes the interaction between Service Manager, External Service Component and Internal Service Component.



**Figure 26: Interaction between Service Components and the Client Side.** It shows how service components interact with each other and form a service infrastructure. Clients take advantage of this service infrastructure to issue queries into the CoSMo system. Numbering represents the following steps.

- i. External Service Component exposes system functionality over web pages to the client side.
- ii. Service Manager publishes UPnP Services on the networks which are directly available over UPnP network. These UPnP services are found by UPnP enabled devices. Access is also available over UPnP Presentation web pages.

- iii. These UPnP service are also found by External Service Component. It provides UPnP access to non-UPnP devices over UPnP presentation web pages. We have a secondary control point which performs UPnP service discovery and exposes UPnP presentation web pages to clients which are not UPnP enabled.
- iv. Service Manager publishes these services in the UDDI service registry which can be made accessible to the clients.

Thus we can summarize different types of accesses available to the external clients as –

	<b>Device Capabilities</b>	<b>Available Access</b>
1)	<b>No Internet, No UPnP</b>	No Access
2)	<b>Internet, No UPnP</b>	Web Pages, Web Services and UPnP Presentation Web Pages
3)	<b>UPnP, No Internet</b>	Direct UPnP and UPnP Presentation Web Pages
4)	<b>UPnP and Internet</b>	Web Pages, Web Services, Direct UPnP and UPnP Presentation Web Pages

**Table 6: CoSMo: Device Capabilities and Available Access Methods. It shows the number of available methods to access the CoSMo system.**

At this point we have looked into all the service related components and now we will move on to the Service Architecture which is more focused towards design aspect of the CoSMo system.

## **5. CoSMo's Service Oriented Architecture for Individual Components and Component Interactions**

Till this point we have covered all the Service components in the CoSMo system. Now we will move on to CoSMo's Service Architecture which covers software engineering aspect of our work. We will first describe service architectures along with the web services and advantages gained by using them, and then we will move on to the specific design details of CoSMo components.

### **Service Oriented Architectures and Web Services**

CoSMo's cognitively inspired architecture for model based data integration is coupled with a Service Oriented Architecture (SOA). It represents and facilitates the interplay between (i) individual modules of internal representation, (ii) the external world comprising of sensors and urban transport system and (iii) the individual users, including system managers who would be interested in various tasks relating to such infrastructure systems. It provides a modular and scalable interface to the world.

Some of the design goals taken into consideration while building CoSMo's Service Architecture are:

- i. Build a loosely coupled, robust, extensible infrastructure for Traffic Monitoring:  
It should be flexible and scalable in such a way that addition of any new functionality or removal of any of the existing systems should not cause too many changes in the existing system.
- ii. Design a Generic architecture which different application domains can utilize:  
The architecture should be generic enough to put into use into a completely new application domain without much change in the service infrastructure.

- iii. Provide all the necessary services for Sensor Network Management, Service Management and it should provide an open platform for internal components or other applications to build upon.

Some of the driving factors in using Service Oriented Architecture are [21] [14]:

- i. **Legacy application support** – Legacy systems can be encapsulated and accessed via Web service interfaces. A business service can be constructed as an aggregation of existing components, using a suitable SOA framework and made available online.
- ii. **Common runtime environment** – Existing service oriented architecture acts as a common run time environment for current services as well as facilitates the development of new services over existing infrastructure environment.
- iii. **Faster development and entry in the market** – Once the existing infrastructure is setup and service libraries are available it becomes very easy to reuse and build over them. It leads to faster development and expedited entry of the product in the market and business can take advantage of the peak period.
- iv. **Iterative Prototyping Model** – SOA evolve iteratively over time with addition of new functionality over existing functionality. It leads to relatively stable and refined system as new development happens on a time tested optimized platform.
- v. **Cost Effectiveness** – SOA help reduce the cost of new application development over the existing infrastructure. Most of the savings are from collaboration among the divisions across the business, reduced development and setup time
- vi. **Pipelined Architecture Support** – Most of the business processes are pipelined and they go through step by step processing. SOA suit such environment as each processing step can be treated as a separate service and chain of services performs the required task in much more effective manner without added complexities.

We are using Web Services as an implementation technology for our Service Oriented Architecture. It benefits from all of the advantages of using of Web Services [Appendix]

such as interoperability, loose coupling, ubiquity, extensibility, etc., along with the advantages of Service Oriented Architectures discussed above.

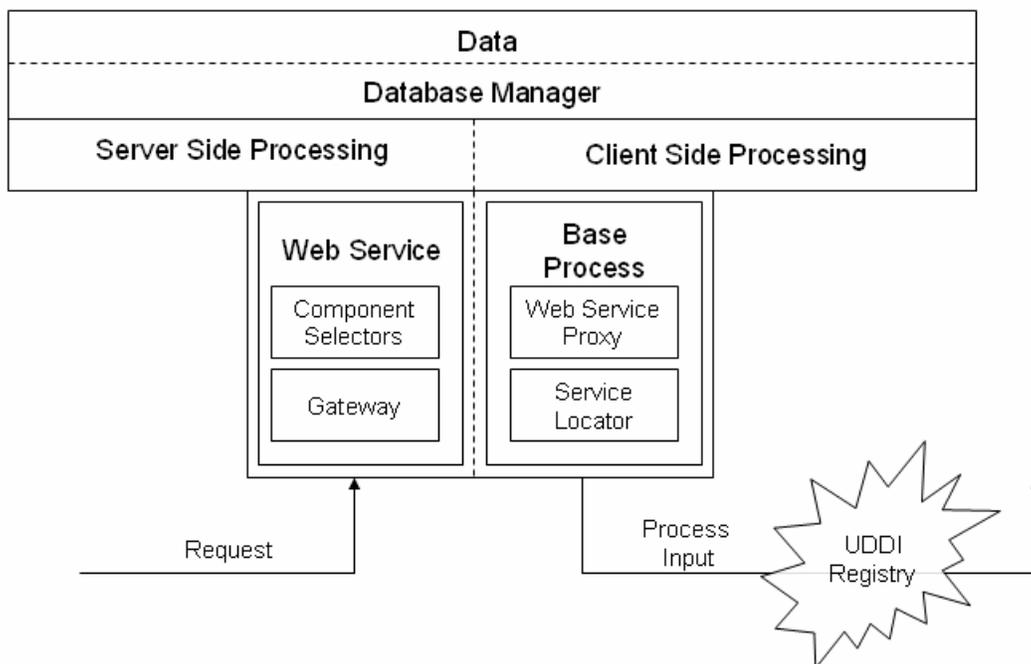
In the next section first we first describe CoSMo's generic component and outline its similarity with the widely used 5-Tier architecture [5] in industry today.

## 5.1. CoSMo's Generic Component

In CoSMo, being a service centric architecture, every component is expected to act as a service requestor as well as a service provider. Effectively every component can have user interfaces, web services, client and server side validations, query specific processing logic and functionality for accessing the databases.

### 5.1.1. Design

The following diagram shows the generic design followed by individual components. Components are divided into different functionalities. All these functionalities and components are designed and implemented in-house.



**Figure 27: CoSMo's Generic Component – Division based on Functionality**

Description of Individual Modules is as follows. Figure [27] shows the mapping of our component modules to the well known five tier architecture.

## **Client Side Processing**

User interfaces and associated client side processing is covered in this section. Within CoSMo, we have a Win-Form as well as HTML web page interface. Users make queries over these web pages or win-forms. These interfaces can be interactive in nature in the sense that they can dynamically respond to the user actions and help the user choose from available alternatives. Once the query is entered it is validated for a particular format and attributes values.

The responsibilities of this module can be summarized as:

- i. It provides access over web interfaces.
- ii. It interacts with all the client side applications including C# Win-Form, Browser and UPnP enabled devices.
- iii. It performs first level query validations.

## **Base Process / Web Service Proxy / Service Locator**

Base Process is the base class functionality provided at application process layer for all the services. It is used to provide common functionalities such as client side caching, service discovery and service invocation. It contains Service Locator and Web Service Proxy creation and invocation functionality.

The responsibilities of this module can be summarized as:

- i. Abstracts client code from directly interacting with the web services.
- ii. Contains CoSMo's Service Locator functionality for finding web services and instantiating a proxy object which will talk to the remote web service.
- iii. Performed client side caching. For example CoSMo's Web Service Proxy objects caching.

## **UDDI Registry**

UDDI is a web service database which stores service details from all the providers and allows access to that information to the users. UDDI stands for Universal Description Discovery and Integration, an open industry initiative enabling businesses to publish

services, discover them and describe how the services and applications interact over the Internet. The UDDI registry maintained in CoSMo is a private node controlled by Service Manager. Service Manager acts as centralized UDDI controller module.

### **Gateway / Component Selector / Web Service**

Gateway functionality is implemented to illustrate the feasibility of CoSMo's grid deployment. Component Selector helps Gateway to select the best component out of available list of components.

Web Services expose the functionality of the component to the outside world. We have taken the decision to have a standard structure of web services in all the components. Every web service has a standard 'Process' method which invokes appropriate internal method out of all available methods. Such standardization has pros and cons but we believe this approach will help in maintaining the system. Changes in one of the modules need not be visible to the other modules. One can easily change the internal implementation of a component to add more functionality without worrying too much about the ill effects of such modifications on other components.

### **Server Side Processing Logic**

Web Service Oriented Architecture is designed for an environment in which many service providers individually or collaboratively offer services. These services perform server side processing on received query and give out the results to the requesting module.

Responsibilities of this section are:

- i. Perform business level validations of the request; authenticate the component and apply other business criteria specific to the services exposed etc.
- ii. Handle business and transactional logic.
- iii. Support server side caching (Not available currently in CoSMo).

## **Database Manager**

It acts as an abstract layer between business processing modules and actual databases. It provides methods to connect to, read and write to different databases. It also does the job of formatting the query string in proper database understandable format. For example if it is a stored procedure that is getting invoked then we need to populate and aggregate all parameters for the stored procedure before making the call.

This abstraction will help in a completely distributed environment where we have distributed databases of different types getting accessed from heterogeneous applications.

All the CoSMo components including the CoSMo data processing components described in Kashmira's thesis [59] should follow this design. The only section which changes is the query processing logic within the component as per the responsibility assigned to that individual component.

Next we will see how our design maps to the widely known 5-Tier architecture used in industry

### **CoSMo Component mapping with 5-Tier architecture**

CoSMo's component design can be easily mapped to the 5-Tier architecture widely used in the industry. Following figure [29] depicts mapping of component functionalities with five layered architecture.

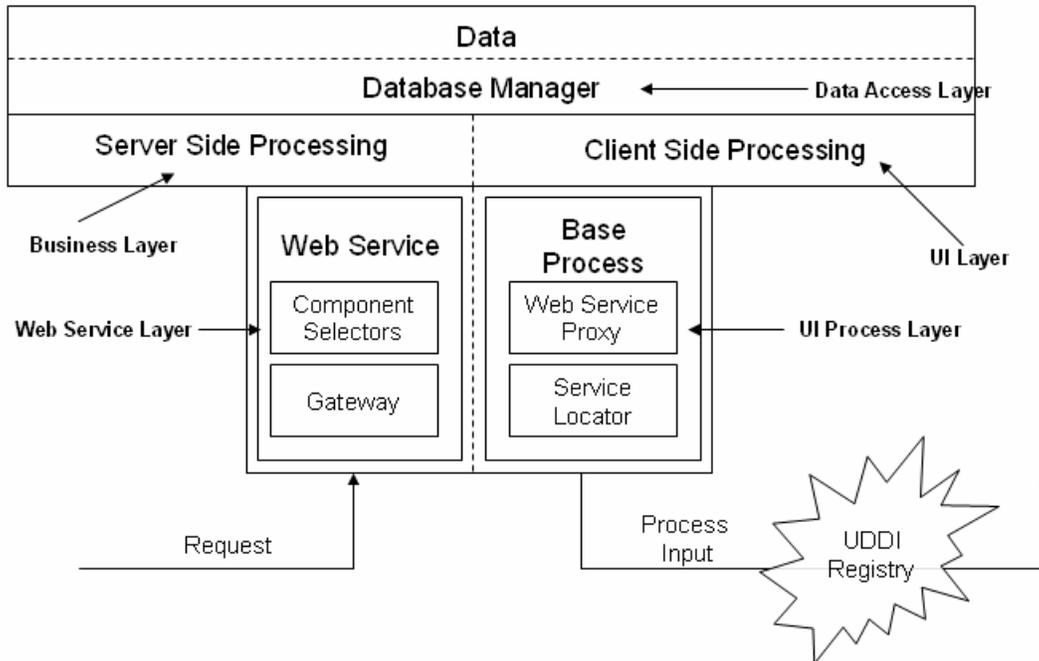


Figure 28: CoSMo's Component Architecture mapped to widely known 5-Tier Architecture

### 5.1.2. Implementation Details

We have designed and implemented all the modules, namely, client side processing, base process module with service locator and web service proxy functionalities, web services, server side query processing logic and database manager in house. Entire implementation was done in C# .Net and ASP .Net

### 5.1.3. Summary of Generic Component Design

We have taken design decisions and added abstractions to create a generic CoSMo Component which can be easily applied to many other application domains. Following diagram summarizes the different sections of CoSMo component along with their responsibilities.

	<b>Module within CoSMo component</b>	<b>Responsibilities</b>
1	Client Side Processing	<p>1) Provides user interface to interact with the component.</p> <p>2) Performs client side validations</p>
2	Base Process Module	<p>1) Abstracts user code from directly interacting with the web services.</p> <p>2) Contains service locator functionality for finding the web services and creating the web service proxy objects.</p> <p>3) Client Side caching of web service proxy objects</p>
3	Web Service Module	<p>1) Contains all the web service exposed by the component</p>
4	Query Processing Module	<p>1) Server side query validation.</p> <p>2) Server side query processing.</p>
5	Database Manager	<p>1) Abstracts code from presence of different databases and different underlying accessing mechanisms.</p> <p>2) Provides uniform interface to the data.</p>

6	Data	1) We assume existence of local or remote or both types of data storage per component.
---	------	--

**Table 7: Summary of Generic Component Design**

In the next section we describe the communication infrastructure within CoSMo.

## **5.2. Communication Framework in CoSMo**

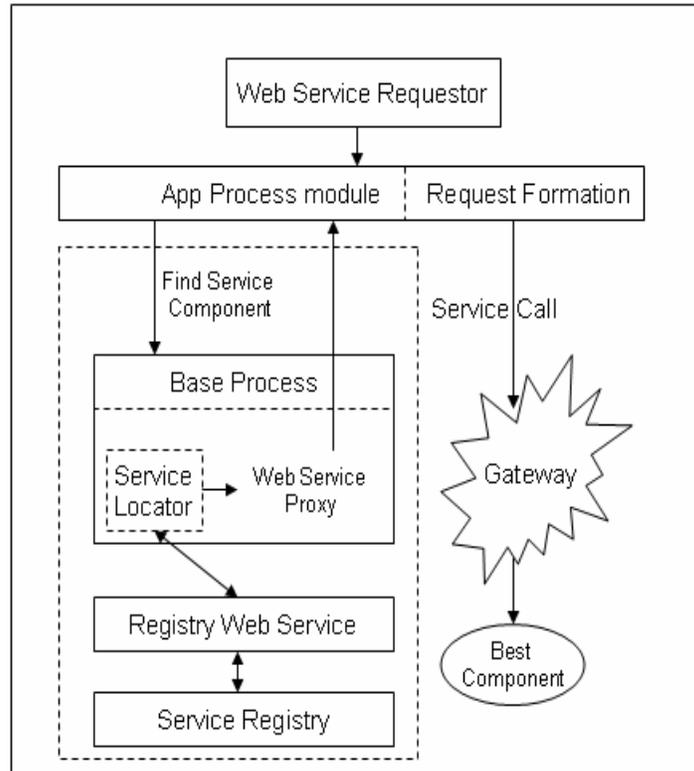
CoSMo was built to support fully asynchronous communication paradigm. We started with a generic framework in our mind and as a result, decisions were taken to build a strong generic CoSMo communication infrastructure at the end.

### **5.2.1. Design**

#### **Inter-Component Communication within CoSMo**

CoSMo's base process module contains service locator and base web service proxy functionalities. Service Locator and Base Web Service proxy are used in finding and invoking appropriate web services. In this section we shall briefly discuss the web service request control flow.

Inter Component web service request control flow is as follows:



**Figure 29: CoSMo: Inter-Component Control Flow. It shows how the control flows through different modules when a remote web service call is made.**

Steps for Web Service Discovery and Invocation are as follows:

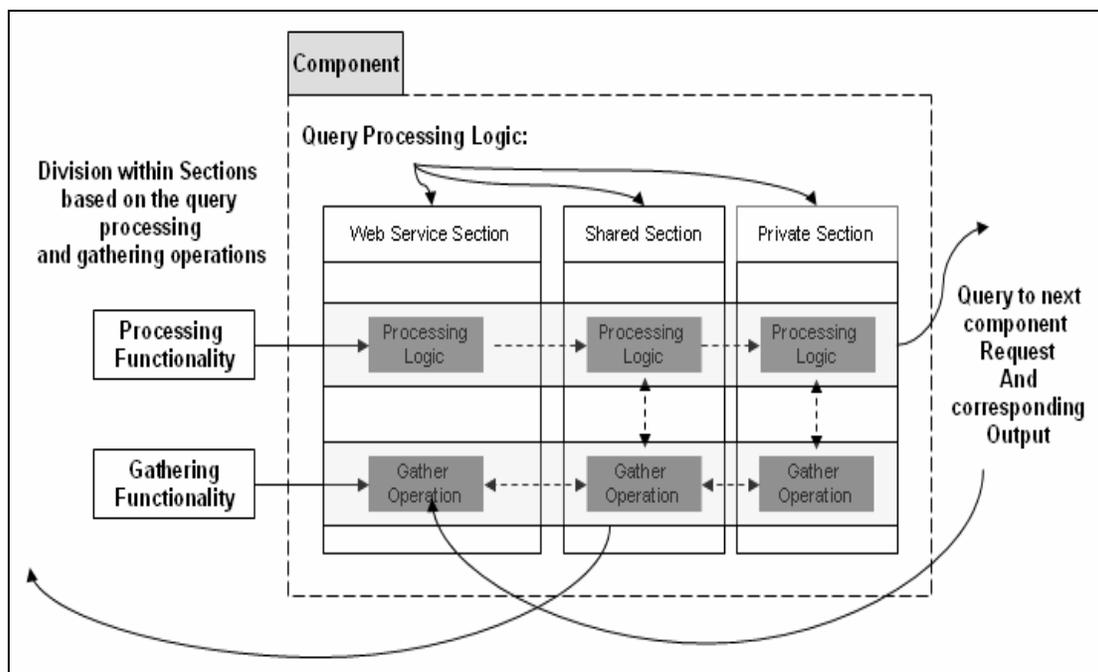
- i. From the requesting module a web service request goes to Base Process Module.
- ii. Base Process module abstracts requestor code from directly interacting with the web services. It contains base contains base web service proxy and service locator functionality.
- iii. Service Locator makes an attempt to locate required web service proxy object in the available web service proxy cache. If the web service proxy is found then web service request is formed and remote web service call is made.
- iv. If the cache does not contain web service proxy, Service Locator will talk to UDDI web service registry and locate the remote web service. After locating the service a web service proxy object is created and cached for later use. This proxy object is used to make remote web service calls.

Sequence diagram for this web service control flow is as shown in Figure [7].

After describing Inter-Component Communication we will move on to Intra-Component Communication within CoSMo.

### Intra-Component Communication within CoSMo

Before moving on to intra component communication we will first go through the query processing logic within the component. In generic component design we divided the component into different parts depending upon the functionality within the component. Along the same lines we can divide query processing logic within the component into three sections i.e. web service section, shared section and private section as shown in the Figure [31].



**Figure 30: CoSMo: Intra Component Control Flow.** It shows how the internal modules of component are divided into sections to concentrate areas which are sensitive or insensitive to the domain specific changes.

- i. **Web Service Section** - It contains all the web services published by the component. These web services serve multiple queries at a time and are primarily

responsible for initiating the internal processing of that query. Since we designed a fully asynchronous communication paradigm, service calls are returned back as soon as they are done with initiating further processing sequence within the component. This ensures that web service resources are not blocked for longer period of time, increasing the availability of web services.

- ii. **Shared Section** - It contains all the context independent information while processing for a query. This processing remains same irrespective of the type of query. Some of the query independent business validations can be placed here. In CoSMo, component's shared sections holds the references to the private sections, i.e., which process is serving which query. This information is very useful because we need a way of retrieving back the query context once the control goes out of scope of current component.
- iii. **Private Section** - This section is designed to handle query specific processing. For example in Data Model component we are creating observation plan which is specific to a query, such processing is done in the private section.

These sections are further divided into processing and receiving functionalities. We can call these functionalities as Processing and Gathering operations.

- i. **Processing functionality** – Processing functionality contains tasks which happen when the query is first received by the serving component. All the tasks related with query validations and initial query processing are done here.
- ii. **Gathering functionality** – Gathering functionality contains tasks which should be performed when query results in some value which should be passed back

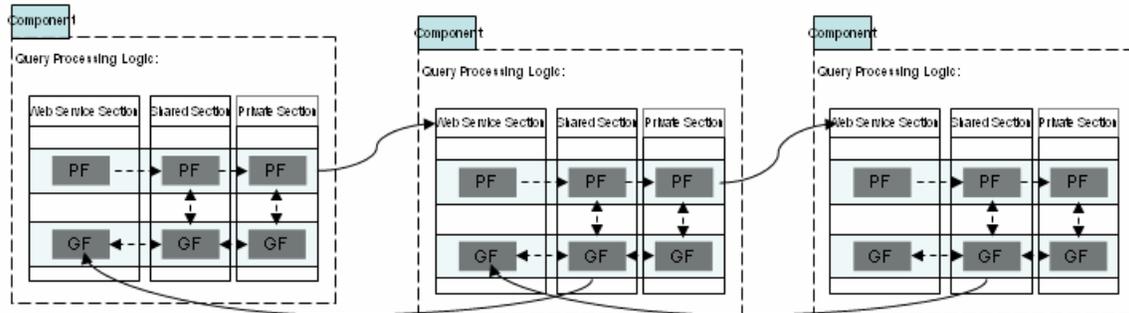
Gather functionality in Web Services is particularly useful as CoSMo is based on a fully asynchronous communication infrastructure. Once a remote web service call is made, requestor component needs to stop further query processing operations. Serving

component initiates a separate web service call in Gathering section of web services and passes the output to the requestor. Gathering functionality in all the sections can coordinate and form the final answer. If this web service call was initiated by another component which is further back in the chain, then the value is again passed back to the previous component and cycle is repeated till it reached the ultimate client.

Now let us look at how these intra-component and inter-component functionalities are put together and form communication infrastructure within the CoSMo system.

### Inter and Intra Component Communication in CoSMo

Following figure illustrates the fully asynchronous communication between multiple components. It shows how intra-component and inter-component functionalities are put together to form communication infrastructure within the CoSMo system



**Figure 31: CoSMo: Inter and Intra-Component Control Flow. Shows overall asynchronous query flow within the CoSMo system.**

This figure once again stresses the fully asynchronous communication infrastructure used in CoSMo. Multiple components are shown which work together towards satisfying a query.

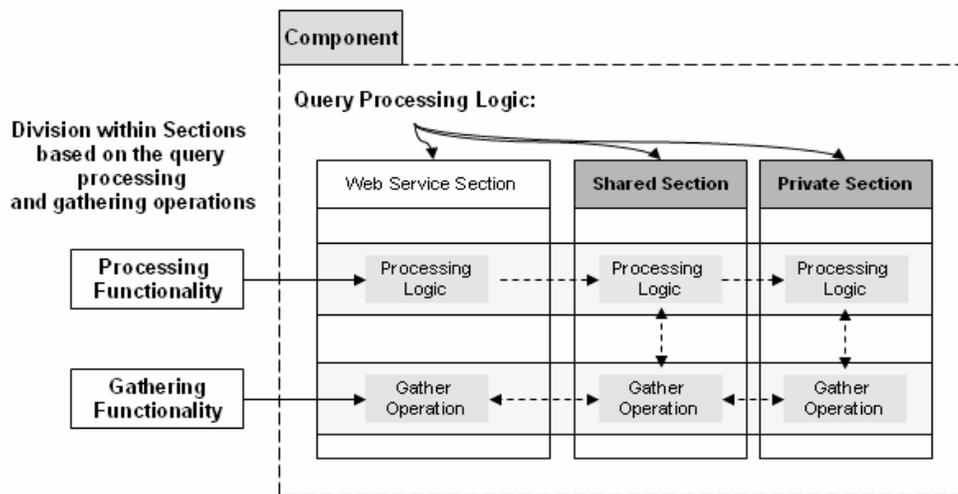
Now we will try to verify that we have achieved our goal of generic service oriented architecture.

## Generic Framework

Let us quickly summarize different parts of CoSMo's service framework.

It has

- i. Component Design – Based on Functionality,
- ii. Component Design – Based on Query Processing Logic,
- iii. Intra-Component communication mechanism
- iv. Inter-component communication mechanism.



**Figure 32: Generality of the CoSMo Service Architecture. The diagram shows what sections are most affected when the architecture is applied to a different application domain.**

Our analysis shows the Component Design – Based on functionality, Intra and Inter-Communication, web service discovery and calling mechanism shall remain same in most of the application domains. Only the processing parts in Shared and Private Sections of Intra-Component design show some changes as per the query logic. Thus Figure [33] shows those changes highlighted in dark. Hence we feel that we have achieved our design goal of building fairly generic service architecture and currently it is one of the biggest strength of the CoSMo system.

### **5.2.2. Implementation Details**

C# was the primary language for coding. Between the components, CoSMo uses asynchronous web service calls and within the component it uses threads, timers, events and callbacks to reduce the blocking of resources.

Query Processing Logic was divided into sections and care was taken to pull together areas into sections according to the sensitivity or insensitivity of the query logic within them. It resulted in concentrated areas within the code where changes take place as per the application domain and made overall maintenance and enhancement a comprehensible task.

### **5.2.3. Summary of Communication Infrastructure**

- i. Current CoSMo implementation supports fully asynchronous communication infrastructure. Design and implementation of this fully asynchronous infrastructure was very intriguing and challenging. It stands as one of the most important functionality within the CoSMo system.
- ii. Division of individual component within the CoSMo system based on query Processing Logic is very generic and we believe it is sufficient to address most of the business scenarios one can encounter in today's systems.

Now we will move on to the Gateway functionality implemented within CoSMo.

## **5.3. Gateway Functionality**

We have incorporated a Gateway functionality to illustrate the feasibility of CoSMo on a grid environment. We assume existence of distributed components, which are located geographically in different regions. Efficiency of such Gateway logic lies in its ability to choose the best resource with least cost and minimum time delay. This resource selection can range from a very simple to a very complex task. Some of the criteria for selection are connection bandwidth, physical location, processing load etc.

### 5.3.1. Design

Our Gateway works on this generic theme but with a slight modification. Generally Gateways do not include complete computational logic present in the duplicated resources, they route the queries directly to the appropriate component. We designed our Gateway with complete computing functionality for serving the request. In other words, Gateway consists of functional logic within component as well as gateway logic.

Such computing redundancy in Gateway enables on the fly decisions to choose between remote components or it self. We have a threshold which will help gateway component decide the ultimate serving component. If the current gateway load indicator value falls above the threshold then it will act purely as a controller gateway. It will pass all the requests to the appropriate remote components. Otherwise it satisfies all the requests locally.

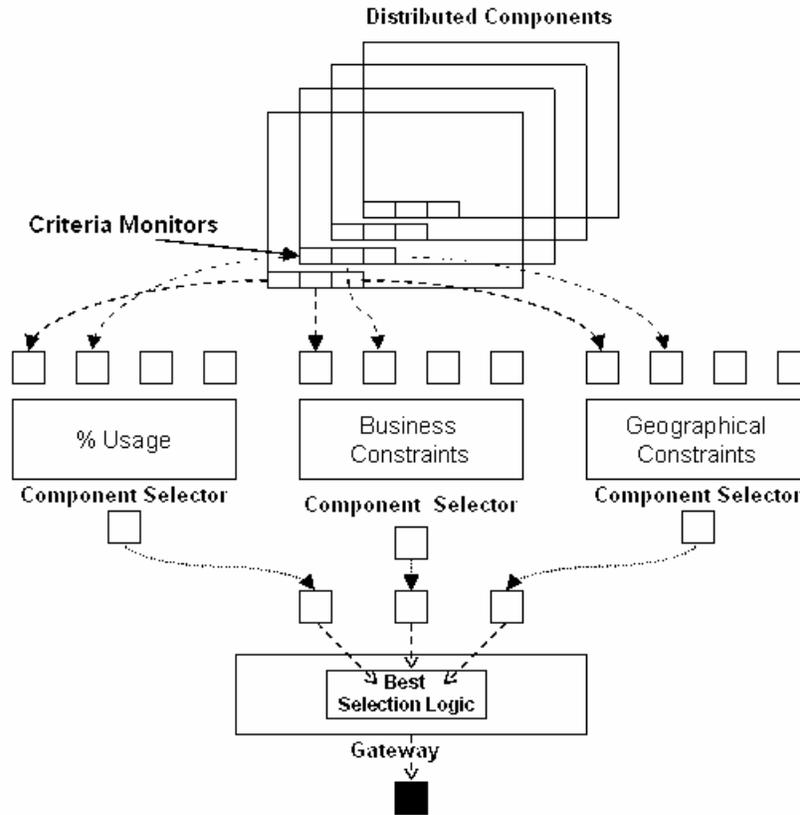


Figure 33: CoSMo: Gateway Logic. It is divided into three layers, namely component layer consisting of components and criteria monitors, component selectors layer consisting of component selectors

**which observe components for specific criteria and gateway layer having gateway logic for selecting the best component out of all the best components given by component selectors.**

Overall gateway functionality is divided into three processing sections, namely criteria monitors, component selectors and gateway logic.

- i. **Criteria Monitors** - Criteria Monitors are the monitoring applications running on every component which keep track of certain information or monitor the respective components for some criteria. In a business environment every service might have different requirements. It can have business constraints like third party contracts, quality constraints like real time response or accuracy, geographical constraints, etc. Criteria Monitors observe for such criteria.

Criteria Monitors exhibit reactive as well as proactive behavior. They are self monitoring within the components and send back values to component selectors reactively after periodic interval of time.

- ii. **Component Selectors** - Component Selectors select the best component from each of the criteria monitors running on many distributed components. Thus one component selector selects a set of components which are best for that criterion. Effectively a number of component selectors for a particular service return many best sets of components to the Gateway.
- iii. **Gateway** – Gateway performs second level of filtering on the component set returned by component selectors. It selects best over the k-criteria observed by component selectors related with that application service.

### **5.3.2. Implementation Details**

In the current implementation, CoSMo supports best selection based on load balancing to illustrate the Gateway functionality. Every time Gateway receives a request it checks the local processor usage and decides on whether to look for other components or satisfy the request locally. It adds a little overhead of monitoring the CPU but that problem can be further mitigated by keeping averages for individual processor usage in memory over a period of time. Thus we don't need to benchmark CPU usage per request.

Once it decides to use a remote component, it invokes Component Selectors for that particular service and selects the best remote component from all components returned by Component Selectors.

### **5.3.3. Summary of Gateway Functionality**

- i. Rudimentary Gateway functionality implemented in the CoSMo system illustrates the feasibility of CoSMo system on a grid platform in near future.
  
- ii. Design decisions were taken and abstractions were added to make Gateway Logic as generic as possible which is sufficient to address most of the resource management issues. For example, mapping of application services to multiple component selectors, mapping of component selectors to criteria monitors which run on distributed components etc.

Let's try to summarize the technologies used in the CoSMo system in the next section.

## 5.4. Technologies used with Service Oriented Middleware

### 5.4.1. UDDI based communication within CoSMo

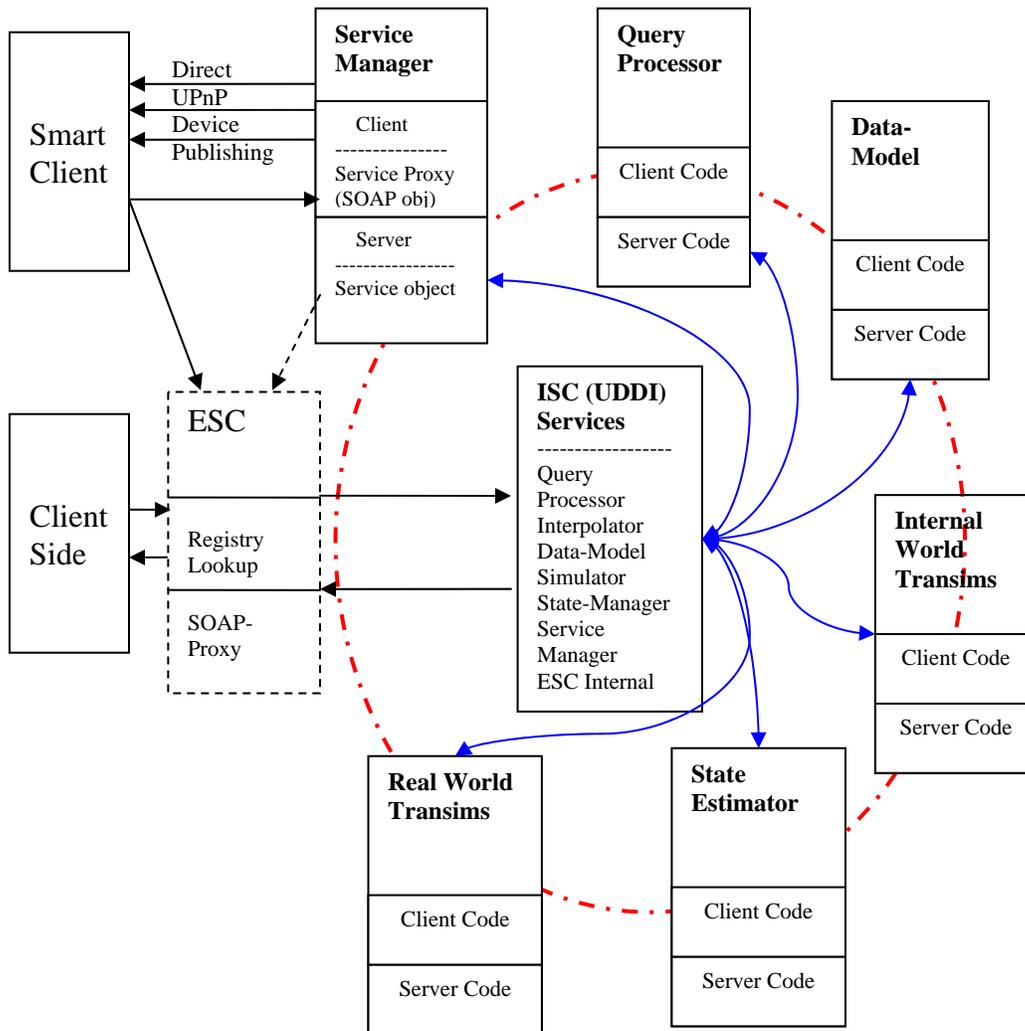
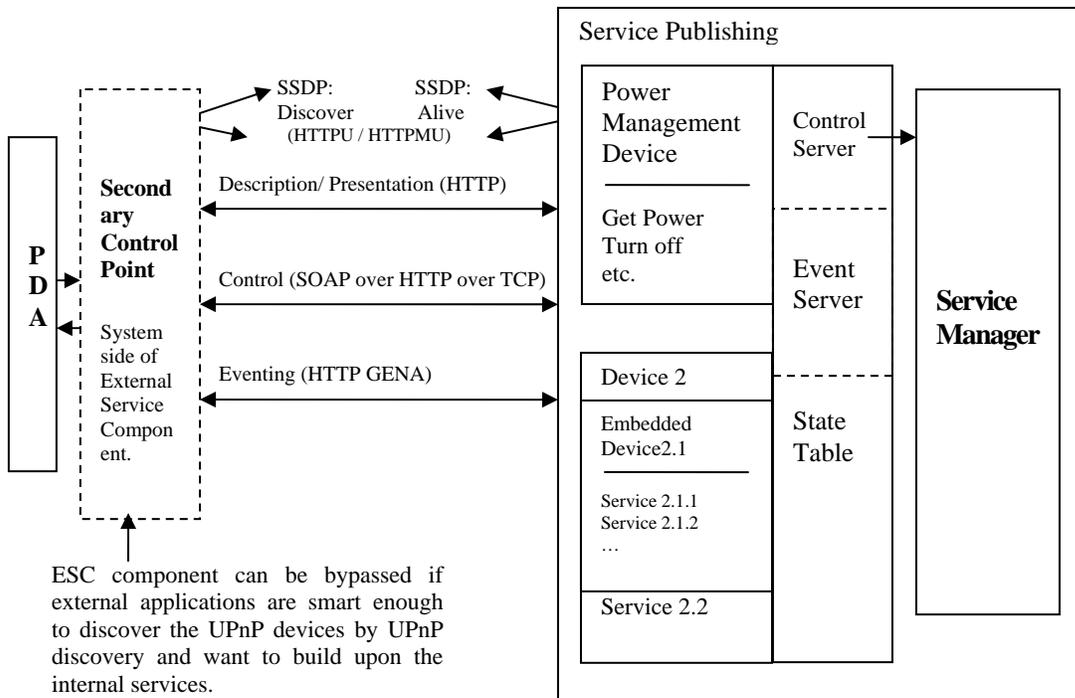


Figure 34: UDDI registry based Communication within CoSMo. The red circle within the diagram shows that entire communication infrastructure within the CoSMo system is centered on the UDDI registry. All components interact with the registry to find out about the remote component.

The UDDI web service registry is the core of inter-component communication within CoSMo. Components interact with the registry and find remote web service locations. External Service Component and Service Manager are the two entry points in the system. ESC supports HTTP based communication. Service Manager supports UPnP based interaction. UPnP enabled smart devices can subscribe to the UPnP services or they can access UPnP Presentation URLs/ Web Pages.

### 5.4.2. UPnP Service Publishing within CoSMo



**Figure 35: UPnP Service Publishing within CoSMo. It shows the internals of UPnP service publishing.**

Service Manager exposes UPnP services on the intranet. UPnP enabled devices (called as Control Points []) access the system in two ways. It can directly use UPnP method invocations to interact with the system or it can fetch UPnP Presentation URLs to access the striped down functionality exposed over web pages. If a UPnP device has a presentation URL, the client can retrieve and load a page from this URL. Depending on

the capabilities of the presentation page and the UPnP device, the Control Point can control the device and view the status of the device. [41]

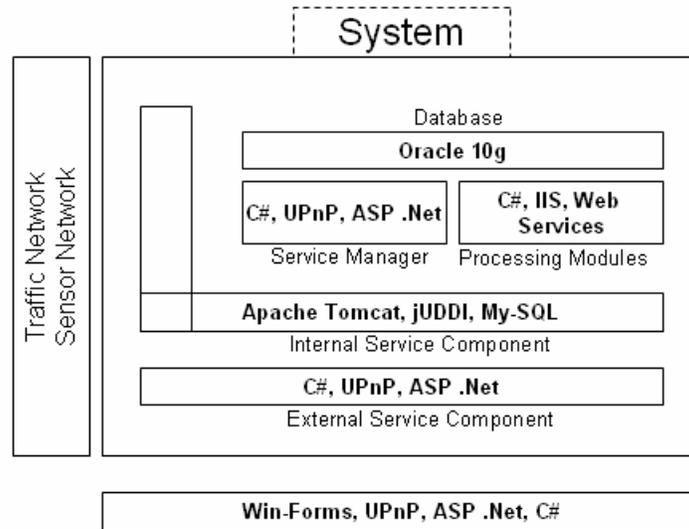
If the PDA is not able to understand UPnP but it has internet access, it still will be able to access the UPnP services provided by the system. This is achieved by providing a built in UPnP discovery component within the system.

### **5.4.3. Application Packages**

We are using C# .Net as our primary platform for development. Complete list of all the packages used in CoSMo is as follows. Apart from Google Map control, Thinktecture parser and implementation for specific standards everything has been built internally.

- i. C# -- Primary coding language
- ii. ASP .Net and Internet Information Services (IIS)
- iii. Win-Forms
- iv. Oracle 10g – System database
- v. Thinktecture dynamic web service library
- vi. Google Map Control – ASP.Net wrapper around Google Maps
- vii. My-SQL – Web registry database
- viii. Apache Tomcat – Application Server for handling UDDI requests
- ix. Java implementation of UDDI – jUDDI
- x. Referred Intel UPnP C# code

Next figure shows the places in CoSMo where technologies are present.



**Figure 36: CoSMo: Technology Overview. It shows all the application packages and technologies used in all the components.**

Interfacing of all of these technologies was a very challenging task. With a running prototype we showed the success achieved in integration of all of these technologies in our system.

#### **5.4.4. Summary of technologies used in CoSMo**

CoSMo is built over many state of the art technologies. Presence of all of these technologies into a single unified architecture is the significant advantage CoSMo has over most of the other systems. Integration of all of these technologies was an intriguing and significantly challenging task. A working CoSMo prototype based on these technologies demonstrates the success behind these efforts.

## 6. Design Decisions and Future work

In this chapter we describe the design decisions made and future work that can be done within the context of Service Components and Service Architecture. In the next chapter we will focus on future work for the entire CoSMo system. While describing the components and service architecture we will briefly repeat the individual components description again to make the description coherent.

### 6.1. External Service Component (ESC)

#### 6.1.1. Design Decisions

- i. Treated as the System Gateway or Entry point from clients.
- ii. Exposes web user interfaces for external clients.
- iii. We decided on having a secondary UPnP discovery component within the system. It adds redundancy within the system but provides a dynamic user interface to the system in which available UPnP services get removed or new services gets added on the fly. It gives end users an option of using limited functionality presentation web pages published by service providers, to control and use UPnP services, even if users device does not understand UPnP.
- iv. We also have a query request-response database which stores client specific information including the queries issued by the client and corresponding output. This approach adds to the storage requirements on the server side but it helps the system in keeping the user activity information. Such information can be used intelligently by the system such as making predictions from user's past preferences, finding current contexts using past queries, etc.

Such storage is also useful in case when periodic results need to be sent back to the clients. We took the decision of having the output ready for clients at External Service Component, but rather than the system making contact with the clients, clients should query those values periodically.

## **6.1.2. Future Work**

- i. Addition of dynamic user interfaces.
- ii. Improving UPnP Service Discovery capabilities.
- iii. Intelligent Data Mining over user activities to provide a richer user experience in the future.

## **6.2. Service Manager**

### **6.2.1. Design Decisions**

- i. Service Manager has been given a central role of service manager.
- ii. It creates the services and publishes them within the UDDI registry.
- iii. Service Manager has been given UPnP capabilities of dynamic UPnP device creation and publishing. The rationale behind having UPnP as another access mechanism is:
  - a. UPnP is light weight and supports zero configuration which is very useful from constrained devices (PDAs) point of view. These devices can dynamically join the network and start using the services over UPnP.
  - b. Because of UPnP, Internet Access becomes an accessory and not as a necessity for accessing the system. PDAs without Internet Access are able to talk to the system using UPnP.
- iv. Preliminary authentication is performed which helps in secure transfer of user passwords across the network.
- v. Within the current implementation one can combine External Service Component and Service Manager Components but we have decided to keep it separate. From a design point of view it is a much cleaner approach and in a broader deployment with multiple application domains both the modules perform logically separate and important tasks.

## **6.2.2. Future Work**

- i. Current implementation is limited to dynamic discovery of web service description from WSDL files only. Support can be extended to other description formats such as WSFL (Web Service Flow Language) [48], WSEL (Web Service End Point Language) [3]
- ii. Addition of intelligence in terms of dynamic service generation and semantic web capabilities [2]

## **6.3. Internal Service Component (ISC)**

### **6.3.1. Design Decisions**

- i. ISC stores web service specific information.
- ii. In the current design we assume a single centralized service repository. In future it can be migrated to a grid environment with gateways and multiple registries co-ordinating and co-operating with each other.
- iii. Entire communication mechanism is centered on the ISC. Every component needs to consult ISC before making a remote web service call.

### **6.3.2. Future Work**

- i. Support for latest Version 3 UDDI registry.
- ii. Deployment on a Grid environment.

## **6.4. Five Layered Service Architecture and Generic Component Design**

### **6.4.1. Design Decisions**

Every component in CoSMo can act as a service requestor as well as a service provider. A CoSMo component has a place for user interfaces, validations, web services, business processing logic and data access manager. All these functionalities are separate and can

be divided into layers. They perfectly map with the 5-Tier well known architecture used in the industry for a long time. Division of each component into five logical sections and application service flow through these five layers gives rise to an interesting mesh like structure.

Functional division of a component is as follows:

- i. **User Interfaces section** - User Interfaces residing within the system and client applications communicating over UPnP and Web Services. It maps to Application UI layer or Presentation Layer in the well known 5-Tier architecture.
- ii. **Client side processing section** - We assigned the task of initial validations, context specific information, client side catching, web service discovery and invocation mechanism functionalities etc in this module. A component need not implement all these functionalities, a small subset of these can suffice for a particular application. It maps to Application UI Process layer in the well known 5-Tier architecture.
- iii. **Web Service section** - Actual web service implementation resides over here. It maps to Web Services layer in the well known 5-Tier architecture.
- iv. **Server processing section** - We assigned the task of server side validations, server side catching and business processing of the request happens at this section. This section maps to Business Layer in the well known 5-Tier architecture.
- v. **Database access section** - We assigned the task of providing a uniform way of accessing data to this section. It abstracts the components from database specific details. This section maps to Data Access Layer in the well known 5-Tier architecture.

All of this generality within the component design minimizes the changes required as per the component role and application domain it belongs to.

## **6.4.2. Future Work**

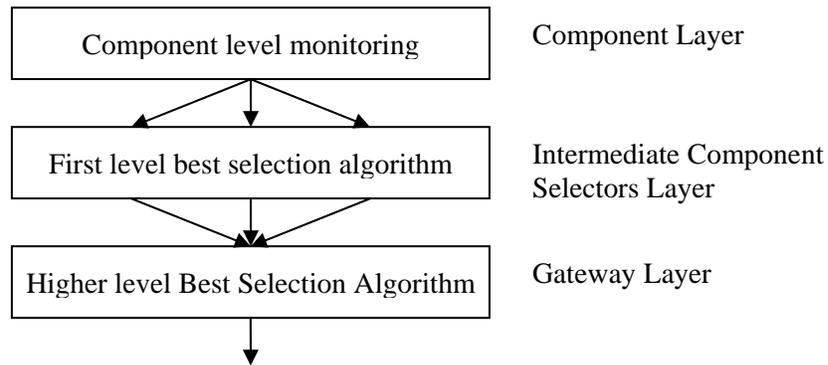
- i. In the current implementation all the modules are functional and functionality is tested. But in some components application flow is not adhering to the layered architecture proposed because of time constraints.
- ii. We would like to test our application in a real environment and quantify the advantages gained by using 5-Tier architecture in terms of performance and future maintainability. These parameters are opposite in nature because as we increase the abstraction and add new layers, system performance slowly degrades. This is compensated by the flexibility and maintainability of such systems in future.

## **6.5. Gateway Functionality**

### **6.5.1. Design Decisions**

We decided to implement a Gateway functionality to illustrate the feasibility of CoSMo on to a grid environment. In a normal Gateway mechanism, Gateway interacts with the components and finds out the best component to satisfy the query. CoSMo gateway works on the similar approach, but we have slightly modified it such that CoSMo gateway contains gateway logic in addition to full fledged computation logic corresponding to the component. And gateway tries to optimize the performance by trying to satisfy requests locally as much as possible.

Our Gateway logic is further divided into three layers



**Figure 37: Summary of Gateway Logic. It shows division of gateway logic into three layers, namely, Component Layer, Component Selectors and Gateway layer.**

- i. **Component Layer** - Distributed components reside here. Proactive and Reactive behavioral patterns are demonstrated by the use of local web services on each of the components. Components monitor themselves for particular workload, business constraints or quality parameters and keep such values ready. These ready made values are proactively polled from component selectors.
- ii. **Component Selector Layer** - Once the criteria monitors sends back the local benchmarked values, component selector chooses the best set of components for a particular criterion from the available pool of components. Various criteria such as CPU usage, business and location constraints, quality specifications for the required service, etc., can be observed.
- iii. **Gateway Layer** - Gateway module gets the list of all the best components for all the criteria. Then best selection across all the criteria is done by the algorithm which returns the best component from the entire pool.

Such three layered structuring, wherein processing and filtering happens at every layer is capable of addressing most of the application specific needs from variety of domains. We

believe this approach is generic in nature because it is designed by keeping most of the current and future business requirements in mind.

## 6.5.2. Future Work

- i. Current implementation is limited to load monitoring on individual components. In future it should incorporate business constraints, quality parameters and other service specific criteria.
- ii. Testing on a real grid platform.

## 6.6. Inter-Component and Intra-Component Communication Design

### 6.6.1. Design Decisions

#### Fully Asynchronous Communication

CoSMo's message passing is based on a fully asynchronous communication infrastructure. This fully asynchronous communication functionality is one of the biggest strengths of the CoSMo system. The decision to make CoSMo fully asynchronous has increased the complexity of the coding, but we believe these efforts will pay off when we actually deploy our system into the real world.

#### Sections based on Query Processing Logic

Apart from the division based on functionality, generic component is further divided into three sections depending upon the query processing logic they perform.

These sections are Web Service section, Shared section, Private section.

- i. **Web Service Section** - All query requests are handled by web service section first. The number of simultaneous requests served depends upon the system settings and capabilities. Licensing conditions also put restrictions on this number. Once the query is received, it is unwrapped and sent to the shared section.
- ii. **Shared section** - This section keeps information which is shared by all the queries. We have a fully asynchronous service paradigm which makes it

necessary to maintain the state as well as retrieve query specific information once the call returns. State information along with the specific references for a single query are stored and retrieved from shared section.

- iii. **Private Section** - Private section is important as depending upon the service and application domain the system needs to do query specific manipulations and query specific state information storage. This information is stored in the private section. Specific access handles for these private sections are paired with query identification and stored in the shared section. These access handles are useful in retrieving query specific information in asynchronous paradigm.

### **Motivation for separate sections**

The rationale behind having web service section and shared section as different modules is, it is a cleaner approach to have these functionalities separate and it has further advantages as we can decouple the request receiving logic from request processing logic. As soon as request is passed to the shared section web service resources for that call can be freed resulting into increased system throughput.

Separation between shared section and private section is intuitive and done to minimize the changes required to the service infrastructure per new functionality in the system or even application of this design into a completely new domain.

From all the above design decisions one can see that the only things that need to be changed in case of a new application domain and services are some part of processing in the shared and private section of the component. All other functionalities and mechanisms within the architecture remains the same. Generality achieved with such a design is the biggest advantage of the CoSMo system.

### **6.6.2. Future Work**

Benchmarking of Current Service Architecture verses other methods considering alternative methods, e.g., synchronous vs. asynchronous web service calls, N-Tier

architectures, performance gains obtained via service proxies, caching techniques and distributed grid components etc.

## **7. Conclusion and Future Work**

### **7.1. Conclusion and Summary**

CoSMo architecture was designed for situational awareness and monitoring of vehicular traffic in urban transportation system. It demonstrates a novel unified architecture which incorporates number of architectural paradigms. Its end to end representation incorporates cognitively inspired architecture which extends from model driven and data centric paradigms on the modeling side and service oriented paradigm on the delivery and deployment side.

CoSMo's unified architecture has cognitively inspired model to internally represent static and dynamic model of external world, i.e., sensor and transportation network, and a service oriented architecture to facilitate and represent interplay between client side, internal processing components and external world comprising of sensor and transportation network. Other strengths of CoSMo include a generic architecture which can be put into use in many different application domains, fully asynchronous communication infrastructure and state of the art technology integration. CoSMo prototype is capable of traffic monitoring, storing traffic data, supporting different types of queries and predicting future traffic conditions as well. It supports sensor network management, service management and shows intelligence in terms of adaptive behavior, balance between accuracy and cost and use of dynamic representation of the real world.

### **7.2. Future Work**

CoSMo covers broad research areas touching many aspects of cognitive, model driven, data driven and service oriented architectures. It assumes existence of large number of wireless sensors, which monitor equally complex system representing transportation network. The goal of current implementation was to cover as much breadth as possible and leave place holders for incorporating efficient logic in future.

In the following section we conclude with the directions for the future work:

### **7.2.1. Addition of efficient algorithms**

From the standpoint of efficiency and adaptation, a number of parts of the architecture need more efficient algorithms. Examples include:

- i. Calculation of confidence in a query output depends upon many factors. Some of them are the values returned by number of sensors observing a particular attribute, number of attributes satisfied within a single query, interpolation performed by state estimator on the values, inherent error in the sensed data and none the less the faulty sensors itself. All these factors make quality and confidence calculation of the final output a tedious task. We need better algorithms to address these factors.
- ii. We need better gateway algorithm for selecting best component out of all the available components.

### **7.2.2. Support for Hybrid queries**

Currently we have not implemented Hybrid queries, which show interaction between Data Model, Simulator and Sensor networks. But architecture is designed such that in future hybrid queries can be easily incorporated. Hybrid queries are advantageous for detection of an event and assertion of that event based on verification. This functionality will improve the confidence in detected events.

### **7.2.3. Extension to the Grid environment**

CoSMo has rudimentary gateway functionality implemented to illustrate the feasibility of CoSMo architecture on a grid environment. In future, we would like to make CoSMo more scalable, flexible and widely usable by extending it to run on a real environment over a real grid infrastructure.

### **7.2.4. Interfacing with Hardware sensors and Simulator**

In the current implementation, CoSMo simulates hardware sensor and uses a static feed from simulator. Replacing software sensors with hardware sensors and interfacing with

real simulator to get dynamic updates are the next steps which will make CoSMo usable in a real world with full functionality.

### **7.2.5. Performance Evaluation**

We could not quantify the CoSMo's performance results because of lack of infrastructure and time constraints. It is a major task which needs to be done in future.

### **7.2.6. Creation of Secure Infrastructure**

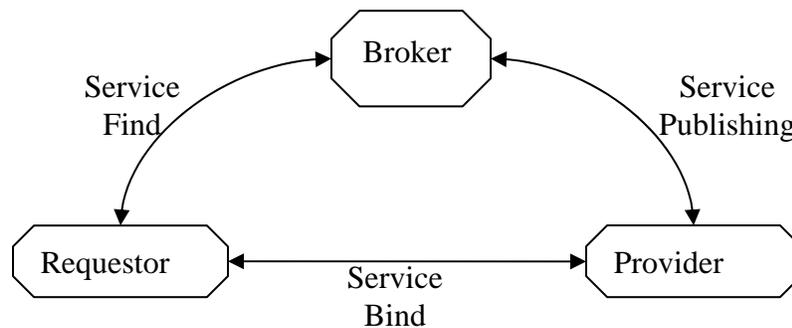
Creation of a secure infrastructure is another challenging task remaining within the current implementation.

## 8. Appendix

### 8.1. Web Services

#### 8.1.1. Web Service Architecture

Let us look at the web service architecture. It has three basic operations publish, discover, bind and three basic entities provider, broker and requestor [36].



**Figure 38: Web Service Publish, Discover and Bind Operations.**

Following are the important points which need to be taken care of when using web services: [36]

- i. First a business analyst analyses the expected input and output from the service. Then it creates the web service with necessary web-methods.
- ii. Web Server is setup which waits for the web service requests.
- iii. Web service is published in one of the web service registries so that its location and functionality can be discovered.
- iv. Service registry and service locations are made accessible from internet or intranet as per the requirement.

Web service entries need to be removed when the web service is discontinued.

### 8.1.2. Web Services benefits

Use of the Web Services architecture provides the following benefits [21] [37]:

- i. **Interoperability** – Web Services are made interoperable by minimizing the requirements for common understanding. Web Service Description and a protocol for message passing over the network are the only things required for understanding and access of services between a provider and a requester. With the common description and black box functionality web services can be truly language and platform independent. By limiting what is absolutely required, web services can be implemented using a large number of different underlying infrastructures.
- ii. **Encapsulation** – Web service users need no be aware of how the service is implemented, what they need is just a web service description language and a protocol to communicate.
- iii. **Loose coupling** – Interaction with the web services over the Internet via messages yields higher decoupling between the modules. Problems related with centralized system and single points of failure are mitigated to a great extent.
- iv. **Just-in-time integration** – Web services can be dynamically found and bound at run time. A service requester describes the service capabilities, queries the service repositories via interface provided by service broker. He gets back the web service description document and service access information via which he can further explore the service or directly bind to it with appropriate input.
- v. **Scalability** – Different web services with different implementations but the same functionality are common from different service providers.
- vi. **Extensibility** – Services can be optimized, extended with add on functionality without having to worry about their side-effects if the input and output is kept similar.
- vii. **Ubiquitous in nature** - HTTP and XML-SOAP are the two well known technologies used in web services. Any device supporting these technologies can bind to published web services or serve others its services.

- viii. **Legacy support** - By specifying a web service wrapper around legacy applications we can take advantage of legacy components.
- ix. **Industry Support** – Lot of industry support in terms of development and research is available for web services.
- x. **Peer-to-peer and Web services** – Integration of peer-to-peer and web services creates an interesting paradigm. Imagine a futuristic global economy in which all the consumers have a local web service, a list of items they want to sell and buy and software agents. Once consumer programs its own software agent that he is looking for some item, software agent goes places to places, talks to web services provided by other consumers and returns a list with top ten other customers which are willing to sell the product with least price. In this case transactions are not involving any centralized service servers; it's a consumer to consumer co-operation and evolution model.

### 8.1.3. Web Services Challenges

Though web services are the current happening thing they have many challenges to overcome. Some of them are pointed out in next section [37] [27].

- i. **Discovery** – For services to be of any good they need to be discovered. If a service can not be discovered it is as good as not having that service.
- ii. **Reliability** – One need to be sure that service provider is trustworthy and the service infrastructure is reliable. When some of the servers go offline, service provider should take care of providing alternative locations where all the services residing on those servers could be located.
- iii. **Scalability** – Load balancing is a big issue in web services
- iv. **Security** – Most of the times web services are accessible from internet. Hence its security is a key concern. Is the service requestor having proper access level to access that service, is anyone eavesdropping over web services communication, what level of authentication is required for the current web service such as just

service level authentication is valid or a further granular role based method access should be provided are some of the questions those need to be answered.

- v. **Transactions** – Transaction support is not matured yet but improving with time.
- vi. **Manageability** – With businesses expanding and offering more and more services, web service management becomes a crucial aspect in successful running of the business.
- vii. **Accountability** – Who will be accountable for problems related with published web services, how the user should get charged such as per service, per use, per month or life time membership are some of the related issues.
- viii. **Testing** – With many web services may be from different businesses is a common collaborative model observed it becomes tricky from testing point of view. How can one assure quality, response time and correct functioning of each and individual module across the heterogeneous network of web services.

#### **8.1.4. Technologies for Web Services and Web Services Stack**

- i. **XML** - Extensible Markup Language is a platform independent data standard for describing, storing and transporting messages over the internet.
- ii. **SOAP** - Simple Object Access Protocol is protocol based on XML to invoke services. It supports both synchronous and asynchronous calls and it can be carried over normal HTTP making it widely used over the internet.
- iii. **WSDL** - Web Services Definition Language is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information.[24]
- iv. **UDDI** - Universal Description, Discovery and Integration is a registry for registering and finding Web Services. We will go in depth over this in the next section.

UDDI (Universal Description Discovery and Integration)
WSDL (Web Services Description Language)
SOAP (Simple Object Access Protocol)
XML (Extensible Markup Language)
HTTP/ HTTPS etc

Figure 39: Web Services Stack

## 8.2. Universal Plug and Play (UPNP)

### 8.2.1. Motivation

We are using Universal Plug and Play architecture for service publishing along with the centralized web service registry. It adds redundancy and replication of services within the system. It has many advantages over UDDI registry. UPnP has strong competition from another middleware architecture called “Jini” – java based solution from Sun Microsystems.

There are some efforts made in the direction of Interoperability between these two technologies [12]. We have chosen UPnP over Jini due to following advantages.

Let us look at some of the advantages provided with UPnP [53] [64] [4].

- i. UPnP defines architecture for pervasive peer to peer computing.
- ii. It is designed by keeping ad-hoc networks in mind – a type of local area network where devices join the network dynamically and there is no central authority managing the network configuration. Generally devices are attached with the network for small duration of time.

- iii. Fairly lightweight and supports existing protocols such as SOAP, HTTP, TCP, UDP, etc.
- iv. It supports zero configurations, i.e., a device can dynamically join a network, obtain an IP address, publish its capabilities and learn about the presence and capabilities of other devices automatically.
- v. UPnP is a distributed, open network architecture, defined by the protocols used; it is independent of any particular operating system, programming language, or physical medium.
- vi. Many devices have started supporting UPnP (nowadays cell phones are also UPnP enabled) and it has a very strong industry support.

Because of all these advantages in an application like traffic monitoring via wireless sensor network, UPnP provides excellent means for service publication, discovery and invocation for road side smart devices. These smart devices do not need internet connectivity or access to service registry to know about the capabilities of the system. They can directly be a part of intranet and discover all the necessary information.

## 8.2.2. UPnP Basics

UPnP Device, Services and Control Points are the three main components in UPnP. [4]

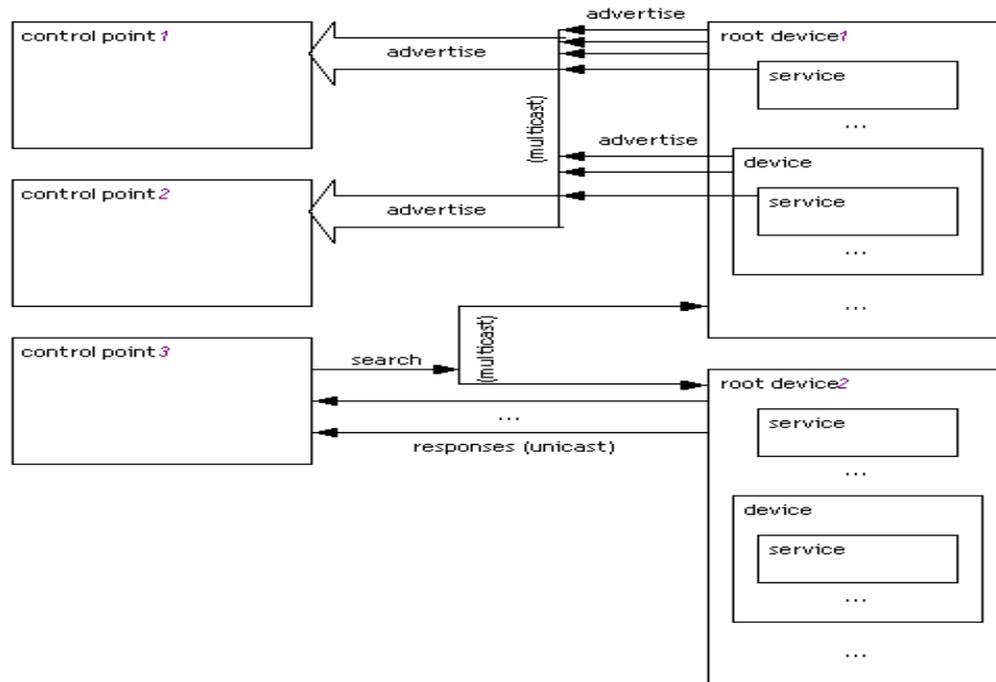


Figure 40: UPnP: Interaction between Control Point and Services

### UPnP Device

A UPnP device is a container of services and nested devices. Different categories of UPnP devices are associated with different sets of services and embedded devices. All device specific information is captured in an XML device description document. In addition to the set of services, the device description also lists the properties such as device name, version, etc, associated with the device [6]. UPnP devices publish their services on the internet and wait for the requests.

### UPnP Services [6]

The smallest unit of control in a UPnP network is a Service. A service exposes actions and models its state with state variables. Similar to the device description, this

information is part of an XML service description standardized by the UPnP forum. Device description file contains the URL for this Service Description.

Each service in a UPnP device consists of a state table, a control server and an event server. The state table models the state of the service through state variables and updates them when the state changes. The control server receives action requests, executes them, updates the state table and returns responses. The event server publishes events to interested subscribers anytime the state of the service changes.

### **Control Point [6]**

A control point in a UPnP network is a controller capable of discovering and controlling other devices. After discovery, a control point could:

Retrieve the device description and get a list of associated services.

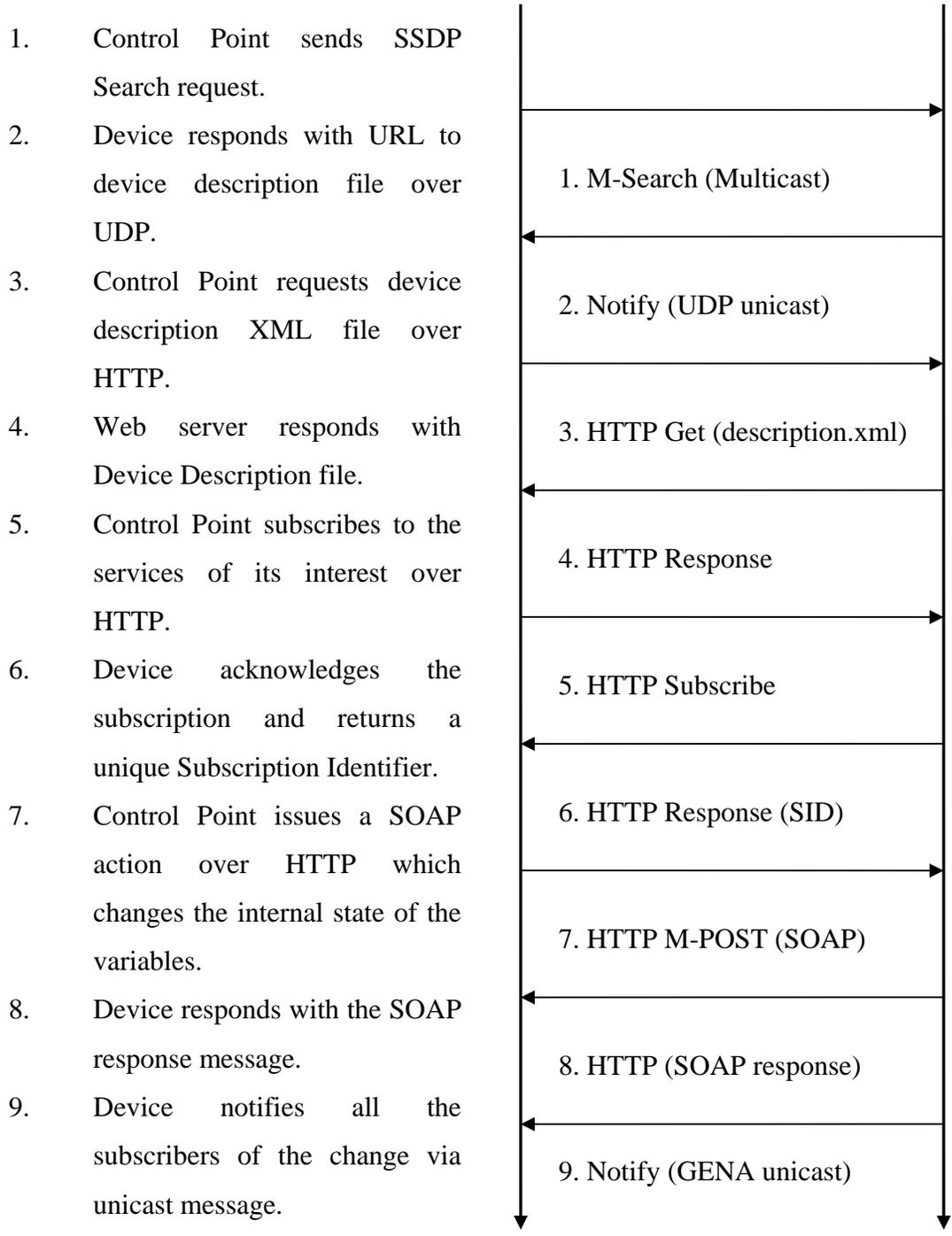
Retrieve service descriptions for interesting services.

Invoke actions to control the service.

Subscribe to the service's event source. Anytime the state of the service changes, the event server will send an event to the control point.

### **8.2.3. UPnP Flow Diagram**

Following Service-Control Point interaction figure outlines the sequence of messages exchanged between a UPnP device and Control Point. [22]



**Figure 41: UPNP Control Flow Diagram**

## **8.3. Universal Description Discovery and Integration (UDDI)**

We are using UDDI as our centralized service repository. Detailed information about the Service Names, Description, Type, Calling and Access semantics for each service are published here.

### **8.3.1. Motivation**

UDDI adds redundancy and replication of services within the system. It mitigates the problem of end point failures. It provides a standardized way for storing and retrieving the service information. Once our efforts with web services passes the prototype and piloting stages and we have multiple groups within our organization developing web services, the situation becomes too complex for developers to find out about services. In such situations we need a standard mechanism to advertise and discover services. Storage of service information in a hierarchical classified format helps in efficient retrieval of the service information.

The key advantage to using UDDI is that it is an industry standard, and therefore most tools support it. Most web services development tools provide UDDI browsers and registration wizards. Many of the web services management (WSM) products also integrate with UDDI. UDDI has become a key component of the web service provisioning process. Many companies use UDDI in the SOA governance process. This summarizes the motivation behind going for UDDI registry node.

Let us briefly go through UDDI basics and then understand the little tweaking we have done to incorporate the Gateway and Performance Monitors logic.

### 8.3.2. UDDI Basics

Following are the four basic types in the UDDI standard [20]:

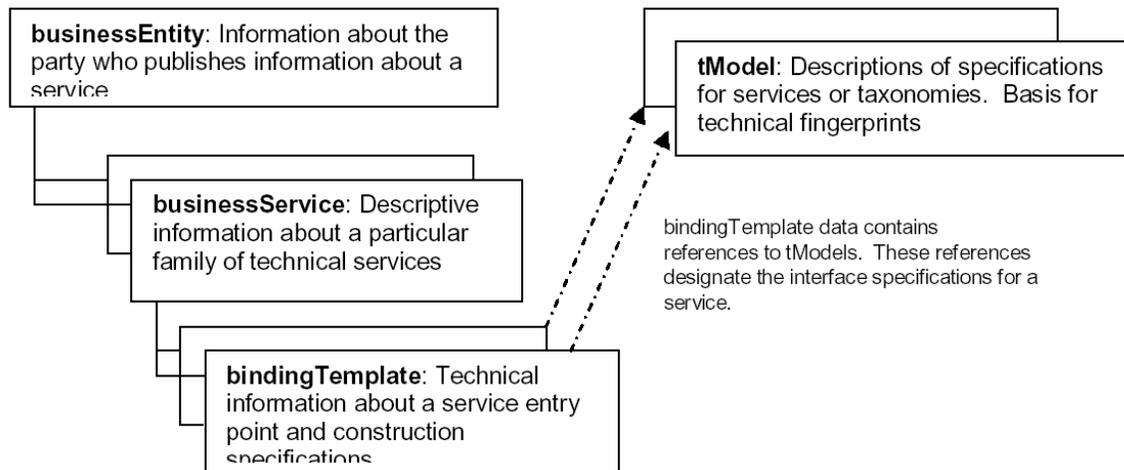


Figure 42: UDDI Basic Data Types

**businessEntity**: This structure represents all the information about a business. It publishes descriptive information about the entity as well as the services that it offers. It is the top-level data structure that accommodates holding descriptive information about a business. Service descriptions and technical information are expressed within a business entity by a containment relationship [20]. A few of the substructures in Business Entity are Name, Business Key, Name, Description, Contacts, and Business Services, etc.

- i. **businessService** - These structures each represent a logical service classification. Each service structure is the logical child of a single Business Entity structure. Various substructures within this are Name, Business Key, Service Key, Binding Template and Category Bag, etc. [20]
- ii. **bindingTemplates** - These structures provide support for determining a technical entry point and it is a lightweight facility for describing unique technical characteristics of a given implementation. Each Binding Template structure has a

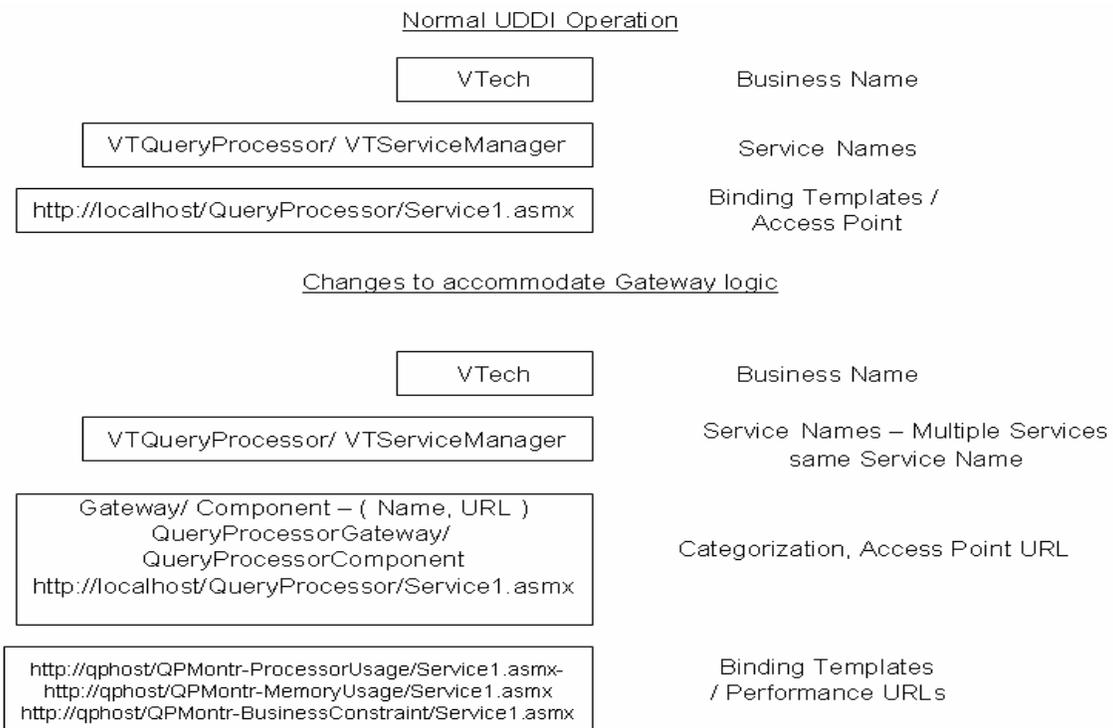
single logical Business Service parent, which in turn has a single logical Business Entity parent. Some of the substructures are Service Key, Binding Key and Access Point, etc. [20]

- iii. **tModel** - The tModel is an abstraction for a technical specification of a service type; it organizes the service type's information and makes it accessible in the registry database.

tModel serves as service interface, categorization tool and it can be used as Namespaces

### 8.3.3. UDDI support in CoSMo

UDDI provides four basic data types as described above. In normal UDDI service mapping, a single business contains multiple services; each service is different and has multiple access points. Thus there is no categorization within the same service. Each service name is associated with just one type of functionality end point.



**Figure 43: UDDI Support in the CoSMo system. It shows the change made to service storage to accommodate the gateway logic.**

UDDI specification has in built redundancy in terms of service classification. tModels and Service Categories are used to categorize the services. We are taking advantage of this fact for incorporation of Gateway and Component Selector logic. With the changed requirements we have one service component acting as a Gateway or Processing Module and that service can have different criteria for selection of best component out of the available pool.

To accommodate this change without changing the table schemas we are creating multiple services and categorizing them as Gateways or Components. We still had problems with storing the URL's which monitor a component for processor usage or memory usage, etc. That problem was resolved by pulling the access point information from Binding Template to Service Category table key\_value field. Hence (key\_name, key\_value) pair in Service category becomes (category name, access point) for example (QueryProcessorGateway, <http://qphost/QPGatewayServiceec.asmx>).

Now each of these rows in Service Category table corresponds to a unique binding template. We store all the URLs which monitor the component in Binding Template for that Service. Thus we were able to store all the information without any underlying changes to the structure. Additional classification can be added using tModels.

## 9. References

- [1] "OMG Model-Driven Architecture Home Page," <http://www.omg.org/mda/index.htm>.
- [2] "Semantic Web Services Initiative," <http://www.swsi.org>.
- [3] "Services Endpoint Language (WSEL)," [http://www.service-architecture.com/web-services/articles/web\\_services\\_endpoint\\_language\\_wsel.html](http://www.service-architecture.com/web-services/articles/web_services_endpoint_language_wsel.html).
- [4] "UPnP™ Device Architecture," [http://www.upnp.org/download/UPnPDA10\\_20000613.htm](http://www.upnp.org/download/UPnPDA10_20000613.htm).
- [5] "Scaling the N-Tier Architecture: Solaris™ Infrastructure Products and Architecture." CA: Sun Microsystems, 2000.
- [6] "Understanding Universal Plug and Play - White Paper," Microsoft Corporation, 2000.
- [7] "Developing a Software Engineering View," The Open Group, 2002.
- [8] "Intel Development Tools for Implementing UPnP Devices," WinHEC, 2003.
- [9] "Transims Overview," <http://ndssl.vbi.vt.edu/transims.html>, 2004.
- [10] "Introduction to UDDI: Important Features and Functional Concepts," Oasis Technical Paper October 2004.
- [11] J. Albus and A. Barbera, "RCS: A Cognitive Architecture for Intelligent multi-agent systems." Intelligent Systems Division Manufacturing Engineering Laboratory, National Institute of Standards and Technology.
- [12] J. Allard, V. Chinta, S. Gundala, and G. Richard-III, "Jini Meets UPnP: An Architecture for Jini/UPnP Interoperability," in Symposium on Applications and the Internet, 2003.
- [13] R. Archer, C. Lebiere, W. Warwick, and D. Schunk, "Integration of Task Network and Cognitive Models to Support System Design," in Collaborative Technology Alliances Conference 2003 Advanced Decision Architectures. College Park, MD, 2003.

- [14] D. Barry, "Web Services and Service-Oriented Architectures," Morgan Kaufmann, 2003.
- [15] D. Benjamin, D. Lyons, and D. Lonsdale, "ADAPT: A Cognitive Architecture for Robotics," 2004.
- [16] S. Berka, D. Boyce, B. Janson, and D. Lee, "Dynamic User-Optimal Route Choice Modeling of a Large-Scale Traffic Network," 58, February 1997.
- [17] P. Biswas and S. Phoha, "A Middleware-driven Architecture for Information Dissemination in Distributed Sensor Networks," ISSNIP, 2004.
- [18] J. Blumenthal, M. Handy, F. Golatowski, M. Haase, and D. Timmermann, "Wireless Sensor Networks - New Challenges in Software Engineering."
- [19] J. Borges and M. Levene, "Data Mining of User Navigation Patterns," WEBKDD, 2003.
- [20] T. Boubez, M. H. C. Kurt, J. Rodriguez, and D. Rogers, "UDDI Data Structure Reference V1.0 - UDDI Published Specification," June 2002.
- [21] K. Channabasavaiah and K. Holley, "Migrating to a service-oriented architecture – part 2," IBM Software Group, Dec 2003.
- [22] I. Chen and S. Midkiff, "Middleware: Peer-to-Peer Computing," pp. Lecture 5.
- [23] R. Cheung, "Adaptive Middleware Infrastructure for Mobile Computing," in ACM. Hong Kong Polytechnic University, 2005.
- [24] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Services Description Language (WSDL) 1.1," Microsoft and IBM Research, 2001.
- [25] X. Chu and R. Buyya, "Service Oriented Sensor Web," Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia.
- [26] M. Connor, C. MacKenzie, B. Watson, M. Cowan, and E. Chase, "Service Oriented Architecture - White Paper," Adobe Systems 2005.
- [27] A. Danylyszyn, "Facing the Challenges of Web Services BPM," <http://webservices.sys-con.com/read/39876.htm>.

- [28] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong, "Model-Driven Data Acquisition in Sensor Networks," in 30th VLDB Toronto, Canada 2004.
- [29] P. Desnoyers, D. Ganesan, H. Li, and P. Shenoy, "PRESTO: A Predictive Storage Architecture for Sensor Networks," in Tenth Workshop on Hot Topics in Operating Systems (HotOS X). Santa Fe, New Mexico, June 2005.
- [30] M. El-Ramly and E. Stroulia, "Mining System-User Interaction Logs for Interaction Patterns," MSR, 2004.
- [31] M. Esler, J. Hightower, T. Anderson, and G. Borriello, "Next Century Challenges: Data-Centric Networking for Invisible Computing," in MobiCom, 1999.
- [32] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next Century Challenges: Scalable Coordination in Sensor Networks," in MobiCom. Seattle, 1999.
- [33] T. Gardner, "An Introduction to Web Services," IBM United Kingdom Laboratories, 2001.
- [34] S. Gleave and J. Swanson, "The Dynamic Urban Model: Transport and Urban Development," Department of Environment, Transport and the Regions 1999.
- [35] F. Golatowski, J. Blumenthal, M. Handy, M. Haase, H. Burchardt, and D. Timmermann, "Service-Oriented Software Architecture for Sensor Networks," [http://www-md.e-technik.uni-rostock.de/veroeff/Service-Oriented Software Architecture for Sensor Networks.pdf](http://www-md.e-technik.uni-rostock.de/veroeff/Service-Oriented_Software_Architecture_for_Sensor_Networks.pdf).
- [36] K. Gottschalk, "Web Services architecture overview." <http://www-128.ibm.com/developerworks/web/library/w-ovr/>; IBM, September 2000.
- [37] K. Govindarajan and A. Banerji, "HP Web Services Architecture Overview," <http://www.w3.org/2001/03/WSWS-popa/paper36/>, 2001.
- [38] R. Grimm, J. Davis, B. Hendrickson, E. Lemar, A. MacBeth, S. Swanson, T. Anderson, B. Bershad, G. B. S. Gribble, and D. Wetherall, "Systems Directions for Pervasive Computing." University of Washington, 2001.

- [39] Y. Gsottberger, X. Shi, T. Sturm, H. Linde, and E. Naroska, "Sindrion: A Prototype System for Low-Power Wireless Control Networks," MASS, 2004.
- [40] S. Hadim and N. Mohamed, "Middleware: Middleware Challenges and Approaches for Wireless Sensor Networks," in IEEE Distributed Systems Online. Stevens Institute of Technology, 2006.
- [41] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks," in Sixth Annual International Conference MobiCom. Boston, MA, 2000.
- [42] M. Jeronimo, "It Just Works: UPnP in the Digital Home," The Journal of Spontaneous Networking, 2004.
- [43] J. Joines, R. Barton, K. Kang, and P. Fishwick, "Traffic Flow Simulation using CORSIM," in Winter Simulation Conference, 2000.
- [44] B. Kokinov, "The Context-Sensitive Cognitive Architecture DUAL." Institute of Mathematics Bulgarian Academy of Sciences, 1994.
- [45] P. Kropf and B. Chaib-draa, "Resource Management in Complex Socio-Technical Systems: A Multiagent Co-ordination Framework."
- [46] F. Lewis, "Wireless Sensor Networks," in Smart Environments: Technologies, Protocols, and Applications. New York, 2004.
- [47] R. Lewis, "Cognitive Theory, SOAR," Department of Computer Information Science the Ohio State University October 1999.
- [48] F. Leymann, "Web Services Flow Language (WSFL) 1.0," IBM, May 2001.
- [49] W. Li, "A Service Oriented SIP Infrastructure for Adaptive and Context-Aware Wireless Services," in ACM. Department of Computer and Systems Sciences Royal Institute of Technology, 2003.
- [50] K. Lorincz, D. Malan, T. Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, M. Welsh, and S. Moulton, "Sensor Networks for Emergency Response: Challenges and Opportunities," in IEEE. Boston, 2004.

- [51] M. Marathe, "Routing in Very Large Multi-Modal Time Dependent Networks: Theory and Practice," presented at Algorithmic Methods and Models for Optimization of Railways - ATMOS, Spain, 2002.
- [52] C. Marin, E. Adele, M. Desertot, and E. Adele, "Sensor Bean : A Component Platform for Sensorbased Services," MPAC, 2005.
- [53] J. Newmarch, "Jini for Home Middleware," Dec 2004.
- [54] D. Nickull, "Service Oriented Architecture - White Paper," Adobe Systems, 2005.
- [55] OMG-Team, "Model-Driven Architecture: A Technical Perspective," in <ftp://ftp.omg.org/pub/docs/ab/01-02-01.pdf>.
- [56] E. Ort, "Article: Service-Oriented Architecture and Web Services: Concepts, Technologies, and Tools," April 2005.
- [57] M. Pallos, "Service-Oriented Architecture: A Primer," eAI, 2001.
- [58] S. Park, A. Savvides, and M. Srivastava, "SensorSim: a simulation framework for sensor networks," 3rd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems, pp. 104-111, 2000.
- [59] K. Phalak, "A Cognitively Inspired Architecture for Wireless Sensor Networks: A Model driven approach for data integration in a Traffic monitoring system," May 2006.
- [60] J. Poole, "Model-Driven Architecture: Vision, Standards and Emerging Technologies," ECOOP, 2001.
- [61] R. Ramanathan and J. Redi, "A brief overview of ad hoc networks: Challenges and Directions," in IEEE Communications Magazine: BBN Technologies, May 2002.
- [62] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu, "Data-Centric Storage in Sensornets with GHT, A Geographic Hash Table," in SIGCOMM, February 2002.
- [63] K. Romer, O. Kasten, and F. Mattern, "Middleware Challenges for Wireless Sensor Networks," 2002.

- [64] H. Song, D. Kim, K. Lee, and J. Sung, "UPnP-Based Sensor Network Management Architecture," Real-time and Embedded Systems Lab information and Communications University.
- [65] K. Terfloth, "Driving Forces Behind Middleware Concepts for Wireless Sensor Networks," in RealWSN: Computer Systems and Telematics Group, 2005.
- [66] E. Yoneki, "Evolution of Ubiquitous Computing with Sensor Networks in Urban Environments," in UbiComp. University of Cambridge Computer Laboratory, 2005.