1. BACKGROUND AND RATIONALE

Data Integration involves merging data present in heterogeneous data sources and providing the user with a unified view of the data. The need for data integration appears with increasing volume to share the existing data residing in different domains. Many researchers proposed several solutions to the problems in the integration of data from heterogeneous domains. The solutions were based on GLAV (global-and-local-as-view) mappings, Augmented mappings, Contextualized Linguistic Matching, XML Schema-based semantic integration etc [Chawathe S 1994].

There are many semantic integration ways to coordinate data that do not meet the real world requirements, which have well-defined relationships between the data sources. A few extensions to the already existing GLAV mapping system will express such fine-grained relationships. The most concerned problem in the data integration system is data coordination, which can be solved by using these GLAV Mappings. In most of the applications like bio-informatics, mining, business etc, multiple data sources need to coordinate data in such a way that changes or updates made to one data source should be reflected in other data sources. This data coordination helps the administrator or manager of the data source to keep track of the updates and see whether it is consistent with the latest data provided by the data sources from where the data is merged. There are some issues in the data coordination process, such as when sources of data describe related, but yet different types of elements or objects of the data source.

1.1. Existing Systems:

Combining heterogeneous data sources using a single query interface raised many issues. The adoption of databases led to the need to share existing repositories. Data warehousing is one
of the popular solution to this issue which extracts, transforms, and loads the information from heterogeneous sources into a single schema. There are problems with the data warehousing method which deal with the freshness of the data. When an update is made to the original or the existing data source, the data warehouse still contains the old data. Data warehousing sometimes include an additional step called a data mart, which involves aggregating the data and responding to user queries by merging and retrieving the appropriate data from the warehouse.

In data warehousing, problems arise in constructing data warehouses when the user has only a single query interface to the heterogeneous data sources and no access to the complete data. This problem has been solved by loosening the coupling between data. A mediated schema has been proposed and implemented which provided a uniform query interface over a mediated schema. This method is called “view-based query-answering”, which transformed a query into specialized queries over the original database. This has also been labeled as “Local-As-View”, where local refers to the local data sources. Another data integration model has a view over the sources called as “Global-As-View”, where global refers to the global schema [Headley K 2003].

Data Integration systems are defined as a triple (G, S, M). Here, G refers to global schema, S refers to a set of heterogeneous domains, and M refers to a mapping that involves mapping between the global schema and sources. The mapping M consists of assertions between Global queries and queried over the set of heterogeneous domains. When a query is posed over the data integration system, the mapping asserts the connection between the objects or elements in the global schema and the source schema.

Global-As-View (GAV) systems model the global database as a set of views over sources, S. The database of the source schema, S is comprised of a set of tuples for each and every
heterogeneous data sources called as source database. Similarly, the database of the G is called global database. The legality of mapping M is to satisfy the correspondence between global database and the source database [Fujun Zhu 2004].

Further, Query Processing becomes an easy operation due to the well-defined relations between G and S. The complexity lies in implementing the mediator and the data integration system so as to exactly retrieve the required data from the source databases. When there is an update or addition of new sources to the system, there is a need to update the mediator, thus Global-As-View appears to be a better choice when sources seem unlikely to change.

Local-As-View (LAV) is modeled as a set of views over global databases. Here, a mapping is related to each element of the source databases S to query over global databases G. There exists no well-defined relation between global and source databases. An advantage of using LAV is new sources can be added and updated with less work than in Global-As-View system. Query processing in data integration systems is expressed using conjunctive queries [DI Wiki 2006].

1.2. Disadvantages of Existing Systems:

Existing data integration systems cannot address such problems which have a small intersection between the schemas and lack a single domain. This is because there exists a very fine grained and complex relations between the values, rather than coarse grained relations between the tables in the databases. Such relationships are needed to be modified, such that they offer coarse grained relationships between the tables by using aggregation, arithmetic, conditional expressions etc. These issues exists in the data coordination process while coordinating and integrating data from heterogeneous domains, which can be solved by augmenting the already existing mapping formalisms using well defined GLAV mapping tables,
key etc. The semantic integration problem is another issue to be addressed in data integration from heterogeneous domains. Ontology-based Integration of data provided resolution of such problems with the use of ontologies that explicitly define schema and resolve semantic conflicts.

1.3. **XML Schema based Data Integration:**

XML-based data modeling follows the Resource Description Framework (RDF) which allows ontological reasoning. XML schema defines the structure of the XML documents and checks for all the tag elements and their positions. The figure illustrates a typical XML-based data integration scenario [Krueger I 2009].

![Figure 1.1 XML-based Data Integration Scenario](image_url)
XML-based Data Integration Scenario follows a few steps in order to address some issues faced in data integration.

1) Obtains all information of tables before mapping such as table name, field name, field type, length etc.
2) Checks for the head information and root element of the document
3) Maps each table into an element whose name is the table name
4) Constructs field mapping into the sub-elements of the tables and attributes of the table.
5) Performs data type mapping
6) Describes primary and foreign keys

1.4. **Advantages of using XML Schema for Data Integration:**

XML Schema for data integration has many advantages which are listed below.

a) Provides a solution to name conflict: XML schema uses two aspects: name conflict and field conflict that addresses the solution by comparing phase and setting up a uniform name in parent element/sub-element format among each group attribute/sub-attribute with respect to relation among them.

b) Provides a solution to structure conflicts: Entity conflict is addressed by the sub-tree set in the output schema.

c) Provides a solution to key conflict: If k1 and k2 are keys for two schemas, if k1 is a sub-set of k2 then select k2 as uniform primary key. Then, convert all elements of \{k2-k1\} into attribute and change describing the primary key in the output schema.
d) Provides a solution to numerical value conflict: Numerical value conflicts can be solved by the mapping condition offered that can support SQL functions like substring(), avg(), count() etc.

1.5. Challenges in Data Integration:

There are many challenges that should be addressed in data coordination of the existing methods. There are many conflicts in the mappings but a few flaws in the performance can be solved by a few enhanced proposals.

a) **Conflicting Mappings**: There could be the chance of conflicts, if one mapping constraint results in removing a tuple which is needed by another mapping constraint that wants to update.

b) **Performance**: There can be many mapping constraints involved depending on the size of the application. Scalability of as many mapping constraints as possible will improve the performance. This is another big challenger that is yet to be achieved.

c) **Creating Mappings**: Creating mappings for a big application is a tedious task when a complex relationship exists in the heterogeneous domains.

d) **Supporting multiple independently managed schemas**: S is a difficult and challenging task to expect the management of the schemas in a centralized way.

Making schema evolution and schema mappings easier is a challenging issue as it requires integrating legacy data in different formats and versions from various distinct domains.

1.6. Web Services:

The term “Web services” refers to the application programming interface or Web API. The Web services can be utilized via HTTP (hyper text transfer protocol). The services are executed
on the remote system depending on the request. Web services can make the application built in to a Web-application. They are found and used throughout the Web. Web service has an interface which is described in a machine understandable format i.e. WSDL (Web Service Descriptive Language). WSDL is used to describe and locate Web services and is based on XML (Extensible Markup Language). The clients will use the Web services by using SOAP (Simple Object Access Protocol) messages. SOAP is a communication protocol which is primarily designed to communicate between applications. It is both language and platform independent [Fujun Zhu 2004].

UDDI (Universal Description, Discovery and Integration) is also an important element in Web services. It is a directory for storing information about Web services. The clients can register and search the Web services through UDDI. The platform for a basic Web service is XML+HTTP. The function or a message present in an application can be published to the entire world by using the Web services. The coding and decoding of data is done by using XML and SOAP is used to transfer the data.

There are mainly 2 categories of Web services. They are big Web services and RESTful Web services. The primary purpose of the RESTful Web services is to manipulate the XML representation of Web resources by using a certain operations. The other services do not make use of XML, SOAP, WSDL, etc. The big Web services will expose an arbitrary set of operations. These types of Web services will follow the SOAP standards.

Basically, Web API can be considered as group of HTTP request messages and also contains the format of response message which is written in XML or JSON (JavaScript Object Notation), etc. Sometimes, the Web services can also be composite. In this case, the sub service
can be considered individually. Web services can be designed by a developer or programmer by using a top-down approach or a bottom-up approach. Bottom-up approach can be considered as a simpler approach compared to top-down approach.

There are many tools in a Web service which can be used in many ways. Most common usage styles are RPC (Remote Procedure Call), SOA (Service Oriented Architecture) and REST (Representational State Transfer). RPS was first of this kind. It is tightly coupled. The services are mapped directly to the method calls or language specific functions. OMG (Object Management Group), COBRA (Common Object Request Broker Architecture), DCOM (Distributed Component Object Model), etc almost provide the same functionality as RPC [Chirathamjaree C 2008].

In SOA, messages are the basic unit of communication rather than methods. It is loosely coupled. This kind of Web services can also be used to implement architecture according to Service-oriented architecture (SOA) concepts. REST focuses mostly on stateful resources rather than on methods or messages unlike RPC and SOA. REST (RESTful) architecture use WSDL to describe SOAP messaging over HTTP.
2. NARRATIVE

The goal of Data Integration of Ocean Observatory System is to extract the data from several ocean observatory systems. The application developed shows the user an integrated view of the ocean data in a well formatted manner. The implementation of data integration system at the mediator level thereby provides a unique interface to query several ocean observation systems using global schema where the tuples are expressible over sources.

The project is an extension to the previously accomplished one “Ocean Observatory Systems” which uses HTML parser to get the data from several sources. Here, the user sends HttpRequest/HttpClient to the Web data sources, the Http Response obtained is passed through Jericho HTML Parser. The parser filters the inappropriate data and sends the data required by the user.

Now-a-days, most of the documents on the Internet appear to be in XML. These documents lose advantage of being well-formatted when parsed using the HTML Parser. It would be better to use Web services to parse and thereby extract the data. The advantage of Web services is its simplicity. It is simple in almost all the Web sources. The XML data that is built and stored either statically or dynamically can be downloaded by including an HTTP library [Kedad Z 2005]. In this case, debugging is also easier because the Web services can be loaded easily in the Web browser and see the raw data.

HTTP Web services are programmatic ways of sending and receiving data using HTTP operations from remote services. GET can be used to get data from the server and POST can be used to send data to the server. The extraction of data using Web services can also be done using CORBA, RMI, EJBs and other technologies. Web services use standard XML and so they are
language independent and platform independent. The message transmission is done using HTTP and there is no problem with the internet traffic. The Web services are more adequate for loosely coupled systems. Here, client has no knowledge of Web service until it is invoked.

2.1. Ocean Data Observatory stations:

There are organizations that maintain stations for measuring different ocean parameters and atmospheric data. They exist along Gulf of Mexico, Florida and several other places. Each station maintains its own database in various forms. They have several data fields like salinity, time, direction, water temperature, etc that can be extracted [Krueger I 2009]. Each database contains attributes such as date, time, east, north, speed, direction, air temperature, atmospheric pressure, etc. There is a need for integrating these heterogeneous databases accordingly.

There are several different stations in which some data are related to the same stations, but at different locations. These are the following stations.

1) Texas Coastal Ocean Observation Network (TCOON)
2) Texas Automated Buoy System (TABS)
3) National Water Level Observing network (NWLO)
   a) NWLON Station locator
4) Texas National Estuarine Research Reserve (NERR)
   a) Weeks Bay (Alabama)
   b) Apalachicola Bay (Florida)
   c) Rookery Bay (Florida)
5) National Data Buoy Center (NDBC)
   a) Stations and locations information
b) Western Gulf of Mexico

6) Wave Current Surge Information System (WACVIS)
7) US Army Corps of Engineers (CoE) Wave Data Sites
8) Hydro meteorological Networks in the United States
9) Galveston Bay Estuary Program
10) Mississippi Department of Marine Resources Data
11) Mississippi Beach Monitoring Program
12) Florida Inshore Marine Monitoring and Assessment Program (IMAP)
13) Coastal Ocean Monitoring and Prediction System (COMPS)

2.2. **Global View over different schemas from different stations:**

The Web services are present for most of the above mentioned stations. The application calls the corresponding service or method present in the server and gets the required data. The application must first discover whether the station has Web services or not. If present, then it adds a reference for using the Web services. The invocation of a method is done by using the SOAP language. Here are the details of some of the stations and the methods they offer which can be invoked through the application.

1) **TCOON** (Texas Coastal Ocean Observation Network): TCOON is operated by the Conrad Blucher Institute for Surveying and Science (CBI) at Texas A&M University Corpus Christi. There are 26 stations within the network that provide meteorological observations. These stations are located along the gulf coast of Texas. The use of the TCOON data is to establish tidal datum’s in boundary determination, to provide good
quality water level information for water circulation and to prepare affected communities for events such as hurricanes and tropical storms. Information collected is even used for oil spill preparation and response, recreation, marine and navigation safety, oceanographic and environmental research, coastal engineering and construction, U.S Army Corps of Engineers dredging operations and management of Texas coastal waterways. To date, the TCOON observation has forty two stations, which includes seven full-time stations operated by National Ocean Service (NOS). [TCOON Website]

Most of the stations provide wind direction, wind speed, air temperature, water temperature salinity, pH, water current, dissolved oxygen data, etc. The various methods that are supported at TCOON Web service are as follows:

a) GetSiteInfo: This method takes site number as input and returns site’s metadata.

b) GetSiteObjectInfo: This method also takes site number as input and returns site’s metadata.

c) GetSites: It takes array of site numbers and returns site metadata for all the site numbers.

d) GetSitesXml: When an array of site numbers is given, it sends the site metadata for each one.

e) GetValues: The time series is returned in XML format when site code, variable code, start date, end date and authentication code are given.

f) GetValuesObject: A TimeSeriesResponseType object containing TCOON time series data is returned here. The input variables are site code, variable code, start date, end date and authentication code.

g) GetVariableInfo: When a variable code is given, variable’s name is returned.

h) GetVariableInfoObject: When a variable code is given, variable’s name is returned.
2) **TABS** (Texas Automated Buoy Systems): TABS is the only system in the country with a primary mission of ocean observation in the service of oil spills. It was built with a goal of ocean observation in an effort to improve quality early response in forecasted spills. Here, the instrumented buyos continuously measure current velocity about six feet below the surface. Satellite telephone is used to transmit data to the system. Initially there are 5 buyos but now they are 9 buyos. Atmospheric pressure data and data about waves are recorded.

3) **NWLon** (National Water Level Observational Network): National Oceanic and Atmospheric Administration (NOAA) and National Ocean Service's (NOS) are Center for Operational Oceanographic Products and Services (CO-OPS). To achieve the goals of NOAA, and maintain water level information, CO-OPS controls the National Water Level Program. NWLP (National Water Level Program) is a nationwide reference for water level datum, which acts a federal backbone for Integrated Sustained Ocean Observing System (IOOS), and is also a chief organization for observational program in NOS. NWLP is an end-to-end system; it has network systems at water-level stations, both long-term and short-term. The data derived by NWLP involving tide and water-level are primarily used for navigation and shoreline boundary.

NWLon is the basic component for NWLP. NWLon operates 175 long-term water-level stations located all over the USA, including island property and the Great Lakes. According to the rapid growth of national and local needs, NWLon stations have expanded and eventually became the basis for NOAA’s tide prediction products, and
plays a major role in determining the tidal data provided for all the short-term water-level stations. Technological advancements like using GOES satellite for sensors, data communication, and data collection, permits automation of real-time routine and event-driven acquisition. In this way, the data-collection platforms are able to measure oceanographic parameters like water levels and meteorological parameters. The development of the NWLON products and data plays a major role in NOAA Storm Surge Warning System and NOAA Tsunami Warning System. In the major ports and harbors, the NWLON stations support the Physical Oceanographic Real-Time Systems (PORTS). NWLON stations have standard configurations for primary data collection platforms, GOES satellite radios, etc. The end-to-end system, the vertical stability and keen observation on NWLON stations data is used to estimate seal-level trends for the Nation. NWLON stations provide water level data, meteorological and ancillary data, tide predictions, datums and real-time ports conditions. The various data values that can be obtained are air temperature, barometric pressure, conductivity, rain fall, relative humidity, water temperature, wind, visibility, water velocity, active water level, etc. There are also Web methods available in this system to get these field values.

4) **NDBC** (National Data Buoy Centers): This gives the locations of the National Data Buoy center Stations. There are a number of stations in National Data Buoy Center. To gain access to each of the stations, the user must click on the 5-character ID to get into the desired station.
5) COMPS (Coastal Ocean Monitoring and Prediction System): COMPS was developed by University of South Florida (USF). It provides real-time oceanographic and meteorological data. It measures wind speed, wind direction, wind gust, water temperature, relative humidity, salinity, specific conductivity, precipitation, air and sea surface temperatures, relative humidity, barometric pressure, short and long-wave radiation and precipitation. The data is measured by using various sensors. Some of the sensor types are Platinum resistance Thermometer, R. M. Young 5103 wind monitor, Sea Bird IM Microcat, Acoustic Doppler Current Profiler, Rotronic MP-101A, etc. A new module is added to incorporate Acoustic Doppler Current Profiler (ADCP), initially designed by Woods Hole Oceanographic Institution (WHOI), and then modified by USF Center for Ocean Technology (COT) for integration with both ASIMET, and a USF-COT designed data logger and telemetry system. This logging and telemetry system also handles the temperature and salinity provided by the inductively coupled MicroCat sensors. All data are both locally stored and telemetered hourly via GOES satellite. The logger-transmitter also has radio communications. This allows for remote communications with the controller for the purposes of setup, data download, and troubleshooting without having to board the buoy.

6) NERR (Texas National Estuarine Research Reserve): NERR is a system which comprises of 27 areas representing the biogeographic regions of the United States. The reserve system is a partnership program between the coastal states and National Oceanic and Atmospheric Administration. This system helps to investigate the physical and biological characteristics of nearly 1.3 million acres of coastal and estuarine habitats. The
reserve system and monitoring program mainly includes addressing coastal management issues through coordinated estuarine research, collecting information necessary for improved understanding and management of estuarine areas, ensuring a stable environment.

7) **WACVIS** (Wave Current Surge Information System): WACVIS is an ocean observation and forecasting system. This system was developed and is maintained by Coastal Studies Institute at Louisiana State University. WACVIS was mainly developed to provide information about waves including wave period, height, surge, direction, near surface current speed and direction of propagation. WACVIS also provides information about water level, visibility, humidity, currents, temperature, turbidity and salinity. The entire information is made online for multiple uses. The data present in the WACVIS can be downloaded, retrieved, queried and analyzed.

### 2.3. System Requirements:

This project is implemented using C# ASP.NET Web application. Microsoft .NET is a software development platform based on virtual machine based architecture. .NET was designed from scratch to support programming language independent of application development. The entire .NET programs are independent of any particular operating system and physical hardware machinery. Data integration is done from legitimate Web sources, which maintain the ocean data along Gulf of Mexico and Florida. It is always advantageous to integrate heterogeneous Web sources as the information becomes more powerful, when organizations share their data.
2.4. **Web Service Creation for extracting XML documents:**

Remote data access with .Net is easy by using Web Services and DataSets used as parameters. .Net can handle all the details of persisting and restoring a full DataSet object over a Web Service including the ability to merge changes into the database providing a powerful tool for building remote data services over the Web. Figure 2.1 illustrates how a Web service is used to extract the XML documents as data sets [Zhang Feng 2009].

![Remote Data Access with .Net Web Services](image)

Figure 2.1 A Scenario of Creating Web Service to Extract XML Documents
2.5. Sensor Observation Service (SOS):

OGC’s Sensor Web Enablement (SWE) activity is being executed through the OGC Web Services (OWS) initiatives, which establishes the interfaces, agreements and protocols that will enable a Sensor Web. The applications and services will be able to access sensors of all types over the Web through the Sensor Web.

A Sensor Observation Service (SOS) is a Web service that provides an API for managing deployed sensors. SOS even retrieves the sensor data from the deployed sensors. The information is retrieved from the in-situ sensors (e.g., water monitoring) or dynamic sensors (e.g., satellite imaging). The measurements and observations made from these sensor systems provide most of the geospatial data by volume. These measurements are used in geospatial systems. The general scenario for in-situ sensors is illustrated in Figure 2.2. Here, deployed sensors ($S_n$) of various types are categorized into several constellations ($C_n$) that are then accessed through Web service like SOS.

![Figure 2.2 General Case of In-Situ Sensors](image-url)

Figure 2.2 General Case of In-Situ Sensors
The Sensor observation service (SOS) Web service has three mandatory and core operations. They are \textbf{GetObservation}, \textbf{DescribeSensor}, and \textbf{GetCapabilities}. The \textbf{GetObservation} operation provides the user with access to sensor observations and measurements data. This is done through a spatio-temporal query which is filtered by phenomena. The \textbf{DescribeSensor} operation retrieves full information about the deployed sensors. This operation even makes those measurements and the platforms that carry the deployed sensors. The \textbf{GetCapabilities} operation provides the medium to access Sensor observation service metadata. Apart from these mandatory operations, there are several optional, nonmandatory operations. There are two operations to support transactions, RegisterSensor and InsertObservation. There are six more operations which are non-mandatory operations provided by Sensor observation service, but are used only when necessary.

When Sensor observation service is used in combination with other OGC specifications, it provides a wide range of interoperable capability for discovering and interrogating individual sensors. It can even bind to the sensors and interrogate the sensor platforms, networked constellations of sensors. These sensors can be in real-time or simulated environments.

\subsection*{2.5.1. Operations for the Sensor Data Consumer:}

A sensor data consumer is interested in accessing and obtaining sensor measurements and observations from one or more sensors, that is from more than one ocean observatory systems. The consumer might solve this problem from either a sensor-centric or an observation-centric point of view. A sensor-centric point of view should be used if the user already know the existence of the deployed sensors and wants to find observations and measurements for those
sensors. An observation-centric point of view should be used if the consumer wants to see sensor data from a particular geographic area. Then, capture particular phenomena of that sensor but the user is not aware of any particular sensors a-priori.

The consumer performs service discovery in either case using a catalog service in order to find SOS service instances. These service instances provide the desired sensor observations. The consumer after consulting the catalog directly obtains observations from services or can perform discovery at service level or can get metadata of sensors before getting sensor observations. Service-level discovery plays a role in invoking the GetCapabilities operation to return information about the offerings that are available from each service. After extracting the sensor system identifiers out of each observation offerings detailed sensor metadata is obtained by invoking the DescribeSensor operation.

Figure 2.3 shows the sensor data consumer in an operational context with OGC catalogs for services discovery and SOS service instances with observation offerings and observations. An instance of a service can talk directly to the sensors or can be a proxy for other services as Indicated in the diagram. Services can be structured into complex topologies by using aggregation and other techniques. These topologies are transparent to the data consumer. The consumer only needs to deal with service interfaces and registries [SOS pdf 2006].

Figure 2.4 shows the flow chart of how the consumer retrieves the data in an operational mode. The user can first discover all the services the Web server provides using the catalog service. The user then discovers the observation of sensors and the metadata related to it. Finally uses the GetObservation operation to access and retrieve the sensor data of the ocean observatory systems using the parameters required in a Request/Response manner.
Figure 2.3 Sensor Data Consumer in Operational Context

Figure 2.4 Sensor Data Consumer Flow Chart
2.5.2. Mandatory Operations of SOS:

Operations defined for a Sensor Observation Service fall into four categories. They are

1. Core
2. Enhanced
3. Transactional and
4. Entire.

The mandatory core operations are GetCapabilities, DescribeSensor and GetObservation. The entire procedures of SOS implement all of these operations (core, enhanced and transactional).

The implementation and use of SOS operations is specified in accordance with the HyperText Transfer Protocol (HTTP) Distributed Computing Platform (DCP). The latest versions may apply to other DCPs.

2.5.2.1. GetObservation:

To retrieve the observation data in the form of Observation and Measurement specification, this GetObservation method is defined to query systems. The system first receives a GetObservation request, then a SOS either satisfy the request or returns an exception report. All the sensor systems of all platforms of ocean observatory systems are queried by by GetObservation operation and the observation data of the sensors is retrieved. The response data is obtained as below in Request/Response the if an SOS satisfies the request.
Request:

A GetObservation message contains one or more elements which constraints the observations that is retrieved from Sensor Observation Service. The attributes `<service>` and `<version>` are mandatory for GetObservation query element. The `<version>` element is mandatory and must correspond to the specific service interface version. It can be negotiated between the service and client during the service binding process. The `<service>` element is also mandatory and it explicitly specifies the service type. For SOS interface, a fixed value of SOS should be used.

Figure 2.5 Request Format of GetObservation
Response:

The response for GetObservation request will be combined within Observation Features, Observation Collections and Observation Arrays, and within Discrete Observation Coverages. The SOS interface is optimized in order to access the observations and associated information.

2.5.2.2. DescribeSensor:

One of the ways to obtain the metadata is to retrieve it from a service catalog. The metadata describes the characteristics of a sensor, sensor network or sensor constellation. The details provided in a service catalog may only contain high-level information. The information is about the types of observables, locations, contact information, etc. Due to the large amounts of sensor related metadata, the Sensor observation service specifies some describe operations. These operations provide highest level of detail about the platforms and sensors associated with an SOS. This operation is defined to request sensor metadata [SOS pdf 2006].

Moreover, the response to a GetCapabilities request will also provide a list of sensors associated with an SOS. These sensors are devices for the measurement of physical quantities for analyze the data. There are two languages which provide a detailed definition of a sensor. They are Sensor Model Language (SensorML) and Transducer Markup Language (TML) for in-situ and remote sensors specification. The DescribeSensor operation is used to obtain detailed information of sensors. The sensor characteristics are encoded in either SensorML or TML. The sensor characteristics include lists and definitions of observations and measurements that are supported by the sensor. The SensorML and TML definitions will detail the response model for the DescribeSensor operation.
Request:

Figure 2.6 Illustrates the SOS DescribeSensor

Response:

A SensorML document is obtained which describes the sensor system.
2.5.2.3. GetCapabilities:

The method of GetCapabilities makes the client to retrieve all the service metadata that is associated to a particular service instance. No "request" parameter is included in this XML schema encoding. This is because the element name itself specifies the specific operation. This base type can be extended later for each specific service and also extra service attributes can be added.

Request:

The information about elements that make up an OGC OWS service type definition can be obtained from Service information Model. The request conforms to the OWS Common definition. Annex B provides an example of a SOS GetCapabilities document.

Response:

An XML encoded document is obtained as a response to GetCapabilities operation. Clients are provided with service metadata about specific service instance. This also included metadata about the tightly-coupled data that is served. Even though the updateSequence parameter is not implemented by the server, the server returns complete document regarding Capabilities but without the updateSequence parameter. When the updateSequence parameter is implemented by server and the GetCapabilities operation request included the updateSequence parameter with current value, this element is returned by server with only the "updateSequence" and "version" attributes. If not this case, all other optional elements shall be included or not depending on Sections parameter's actual value which is present in the GetCapabilities operation request.
2.5.3. **Sensor data consumer flow:**

Figure 2.7 is a sequence diagram that shows a sensor data consumer discovering two SOS instances from a CS-W catalog by using the GetRecords operation. The consumer then performs service-level discovery on each service instance by requesting the capabilities document and inspecting the observation offerings. The consumer invokes the DescribeSensor operation to retrieve detailed sensor metadata in SensorML for sensors advertised in the observation offerings of the two services. Finally, the consumer calls the GetObservation operation to actually retrieve the observations from both service instances [SOS pdf 2006].

When the DescribeSensor operation is invoked by the consumer, all the observation offerings are scrutinized and the required offerings are filtered and called using the Http Request. For example, GetFeatureOfInterest returns a featureOfInterest that is advertised in one of the observation offerings of the SOS capabilities document.

The GetCapabilities operation is called by the user for retrieving service metadata about a specific service instance. The SOS sends the Capabilities response back to the consumer of the ocean data as shown in Figure 2.7. After capturing the service metadata about a specific service the user sends the GetObservation request by calling the operation to the SOS. Finally, SOS sends back Observations and Measurements data (O&M data) to the user as a response. This data will have all the observations of the sensors which capture the required ocean data the user wants for this application to display it in the application GUI. Figure 2.7 shows the complete flow of how the consumer captures and retrieves the sensor data.
Figure 2.7 Sensor Data Consumer Sequence Diagram
3. SYSTEM DESIGN

The project is implemented using Visual C# Asp.net Web Application. The various data sources for the project would be existing on multiple servers each maintained by one of the organizations. Each organization would keep large databases for its own use, but it is often the case that the information in one organization can be useful to another and vice versa. Therefore, if two organizations could share their data, it means their information power would more increase and benefit each other. This project would deal with some Web data sources that belong to Texas Coastal Ocean Observation Network (TCOON), Coastal Ocean Monitoring and Prediction System (COMPS), Dauphin Island Sea Lab (DISL), Central Gulf Ocean Observatory System (CenGOOS) and other institutions. If the user wants to compare data from both TCOON and COMPS, it is very complicated. In the traditional method, the user has to retrieve same category data from these Websites individually at a point of time. The situation becomes hard if the user wants to analyze huge data. But as the project provides a mediator based interface where the global view of all the data from different Web data sources is known, it collects from the required data from the remote Web data sources and the results are displayed at the mediator interface.

Here, the author introduces the global-as-view schema and how the data is retrieved using Web services. It uses the data that is obtained locally from each observatory system and integrates it in to one. In the latter part, the author introduces development environment of the project, then analysis, design, and implementation phases keeping software engineering point of view. The application actually integrates data from many observatory sources. Since all the
systems used are at different places, the user can see only one systems data at a particular point of time.

3.1. Development Environment:

The internet is a real extension of a home PC, where computing is desktop centric, here it would be net centric. Services on one site will be integrating with services on another site, by which a true experience will be provided. This ‘experience’ is what .NET is, as main resources like software as well as storage. The application project is developed in Visual C# .Net Web Application.

Microsoft .NET is a software development platform based on virtual machine based architecture. .NET was designed from scratch to support programming language independent of application development. The entire .NET programs are independent of any particular operating system and physical hardware machinery. They can run on any physical machine running any operating system containing the implementation of .NET Framework. The core component of the .NET framework is its Common Language Runtime (CLR), which provides the abstraction of execution environment (Physical machine and Operating System) and manages the overall execution of any of the .NET based program [Programmer 2004]. The Web service used for the project is Sensor Observation Service (SOS).

3.1.1 Data Retrieval Phase:

The data source to be retrieved is located on different Web servers maintained individually by each ocean observatory stations. A user sends the Http Request/HttpClient to the
Web server that provides the Web service and the Http Response from the Web data server is passed to the consumer of the data and which the user have interest in displaying in the data grid of GUI. The Web service used to retrieve the data is Sensor Observation Service (SOS).

The retrieved data is in the format of XML, which is further parsed into ADO.NET DataTable. The process of parsing is done by DataSet ReadXML feature. Each node in the XML is converted into a table in the DataSet and the name of the node is given to the table as table name. Within the Dataset the tables that are used are ‘Fields’ and ‘DataArray’. Fields table has all the fields’ names returned by the Web service and the DataArray has the result values.

The field names and values are copied into a new DataTable respectively. Then the new DataTable is binded to ASP.NET GridView. GridView can be customized in the way we display the data. Data can be sorted, filtered and can add paging in case of larger amount of data.

![Figure 3.1 An Illustration of Data Retrieval Phase](image-url)
Figure 3.1 illustrates the flowchart of the data retrieval phase. The user sends a Common Http Request to the Web sites of the ocean observatory systems, which provides a Web service (SOS). The Web service replies back with a response in the form of XML documents. The ocean data is hence in the XML format, which needs to be parsed.

The parsed data will be in some different tables in a data set and the result will be combination of two different data tables. Finally, the ocean data is displayed in the data grid of the GUI. This process of data retrieval involves a simple Http request/response method and a parser to convert the achieve compatibility with the XML documents. The parsing is handled by .NET framework feature dataset ReadXML. This phase gives a succession to the Design phase for the project.

3.1.2. Design phase:

Figure 3.2 shows the design phase of the project. The ocean observatory systems namely, TCOON, DISL, COMPS and MOTE are the Web sources that are being integrated in the project. Each data source has its own local schema and at the intermediate level the mediator has global schema. There are mappings between the global schema and the local schemas. The data sources offer SOS Web service through which the application can retrieve the ocean data and display it in the GUI of the application. The end users at the mediator level query for either water level or meteorological observations and the resulting data can be seen at the data grid of GUI and can be used further for scientific analysis and research purposes.
3.1.3. Implementation Phase:

This phase illustrates the execution of the plan, design, specifications and model. The implemented project would require an idea about the schema of all the Web data sources. The queries may range from simple to complex; the application should pull the data whatever the Web service of that particular ocean observatory system provides.

There are two URL’s to get the data. One to get the capabilities and other to get the observations as shown below:
**GetCapabilities:**

http://<host>/<oostethys_sos.cgi||oostethys_server.asp>?request=GetCapabilities&service=sos&version=1.0.0

**GetObservation:**

http://<host>/<oostethys_sos.cgi||oostethys_server.asp>?request=GetObservation&service=SOS&version=1.0.0&responseFormat=text/xml;subtype='om/1.0'&offering=<local name of offering>&observedProperty=<property>&procedure=<urn of offering>

To get the capabilities of specific observatory, replace the “http://<host>” in GetCapabilities link with the respective observatory link. Get capabilities provides the list of stations and offerings that observatory is offering. Respective list are binded to the station and offering dropdown lists.

To get the observation of specific station, replace the “http://<host>” in GetObservation link with the respective observatory link and replace “<local name of offering>” with station name and “<property>” with offering and “<urn of offering>” with stations respective urn. After forming the URL call the URL which returns the XML document.

The flow of execution of the application is pretty simple and is explained as follows. All the major steps are considered and minor ones are excluded in the explanation. During the start of program execution, only the various observatory sources are displayed to the user. All the grid views and map are visible only when the data necessary fields are selected and search button is clicked by user. When the user selects a station in the dropdown list, then the function lstbxObsevatory_SelectedIndexChanged() is executed.

The application screen has three dropdowns to select. First the Observatory, second the Stations and third the Offerings. Observatory dropdown has all the items during the page load.
Based on the selection of the observatory, stations and offerings are populated accordingly. Only one selection is allowed on the observatory, where as multiple selections are allowed on station and offerings.

Figure 3.3 Flowchart Indicating the Implementation Process
All the station ids and various offerings related to the particular stations are extracted and displayed in the corresponding dropdown lists. The user can select a single station/multiple stations and single offering/multiple offerings at a time. The user then click the submit button. The field validations are also performed using the jscript file (with the help of Ajax toolkit). This helps in checking the input whether it is valid or not at the client side itself before sending the invalid input to the server thereby reducing the burden on the server.

When the user selects a valid input from the dropdown lists, then only the query is sent to the required data sources. When the submit button is clicked, then a combination consisting of observatory system, station, observed property is sent to the necessary database at a time by calling PerformDataBind() function. If there are multiple selections in stations and offerings, then for each selection in station, loop through each selected offering and follow above mentioned process.

A dynamic URL is actually created on the fly for each observatory system by appending the station id and observed property values. The data which we get is in XML format. Now, using the function GetDataSetByObservatory() and by using the .NET feature (Dataset), the data in XML format is converted to a dataset for display. GetDataTableFromDataSet() function converts the data into a tabular format which will be in rows. These rows are then added to the main table in the function AddDataToMainTable() which is ready for display. The data now is displayed on the screen in a GridView which is easily readable.

Apart from this, a map is also displayed on the screen. The map used is obtained from Google and it indicated the locations of observatory systems. The map has all the details such as
latitude, longitude and marks the exact locations of observatory systems by using pointers provided on the map. When the user clicks on a pointer, more detailed information is displayed. This information is same as the information present in the GridView which is obtained for particular stations and observed properties. Some extra features include displaying the data obtained in the GridView either in ascending or descending order. The order can be ordered with respect to station numbers or observed property.

The snapshot below shows the basic page of the DIOOS application which retrieves data from the SOS Web services provided by the ocean observatory stations.
When a user selects observatory station as DISL and stations bsc, disl, mbla and mhpa. Then selects all the offerings the data is displayed in a grid format showing 10 values in a single page, continued by next values in other pages. Sorting has been done to make it user convenient, so that the values are sorted by ascending or descending order of either station names or observed property. The snapshot below shows all the values of the data that the user queried for.
Figure 3.5 A Snapshot Showing the Results of the Query with Values Sorted as per Station Name
3.2. Detailed Implementation of the Application:

**Step 1:** The user has to launch the application selecting the C# .Net Web Application in Visual Studio.

**Step 2:** Open the project and the application loads, build the application project.
Step 3: Once the building is successful, then debug the application project.

Step 4: The application launches a new Internet Explorer with GUI part.

Step 5: The user selects all the required attributes from the GUI part and click Search button.

Step 6: The application internally sends multiple http requests to the selected observatory Web service.

Step 7: The results XML's are converted into DataTables and presented to user in GridView.

Step 8: Upon receiving the XML response from each individual station, code removes the unnecessary data from the DataSet.

Step 9: Most of the times the retrieved data is displayed in the GridView part of the GUI. Sometimes it is possible that the http request returns empty data, in such cases the content in the grid view is left empty.

In this way, dynamically data is retrieved from the remote Web data sources.
4. EVALUATION AND RESULTS

4.1. Evaluation:

Testing is very important for any project. Testing involves testing the application for what it specified and what it is delivering. It verifies the implementation and performance of the project.

Testing provides information about the quality of the service or a product. It is a process of validation and verification that an application meets the technical requirements that guides its design and development. The application should work as expected and can be implemented with the same characteristics.

Validation and Verification checks that the application meets all specifications and fulfills its intended purpose. Validation checks are done on the application in many scenarios. 3 cases are illustrated below and sorting of values retrieved is done to make the application user convenient.

When the user selects an observatory station and does not select stations with in that observatory, the application validates the user by giving a selection message.

4.2. Results:

Case 1: Test case for validation. In this test case, when a user clicks on a search button without selecting any observatory it displays a “please select observatory” message. This case hence provides a validation for selecting precise fields in order.
Figure 4.1. Snapshot Showing Result Validation in the Application

**Case 2:** Test case for validation. When a user clicks search button after selecting an observatory, the application will ask the user to select stations within that observatory before proceeding. This will validate the user to select the fields correctly.
Case 3: Test case for validation. When a user clicks search button after selecting station and observatory, the application will ask user to select the offerings. This will validate the user to select the fields correctly.
Figure 4.3 Snapshot Showing Validation in the Application

**Case 4:** This case verifies whether the data retrieved by the application is same as the data which the Web service of a particular ocean observatory system provides. For example, consider a query that retrieves dissolved oxygen offering from BSCA station of DISL ocean observatory system. When the user queries the offering from the application it displays the data as shown in the figure below.
Figure 4.4 A Snapshot Showing Dissolved Oxygen Value = 10.34 from BSCA Station of DISL

Now, send a request to SOS using the GetObservation URL to the DISL Web service. The link is,

http://gcoos.disl.org/cgi-bin/oostethys_sos.cgi/oostethys_server.asp?request=GetObservation&service=SOS&version=1.0.0&responseFormat=text/xml;subtype='om/1.0'&offering=bsca&observedProperty=dissolved_oxygen&procedure=urn:disl.org:source.mooring#bsca.
It will respond back with the values of dissolved oxygen in the BSCA station of DISL. If the value of dissolved oxygen from the SOS response is same as the value retrieved by the application dynamically, the verification is successful. The figure below shows the value of dissolved oxygen retrieved from the Web service of DISL.

Figure 4.5 A Snapshot Showing the Data of Dissolved Oxygen=10.34 Using the SOS Response Message
This test case hence verifies the application showing that retrieving the data using the application and manually sending the request to SOS server and getting back the response will yield same results.
5. FUTURE WORK

Future research includes addition of new ocean observatory stations to the existing ones in the DIOOS project. Currently, data from DISL, CenGOOS, COMPS, TCOON and MOTE have been integrated into the application. The application even includes stations from all these ocean observatory stations. There is a good scope for extending the project for other ocean observatory stations for which SOS is still in the implementation stage. One of such station is Texas Automated Buoy System (TABS). The domain being very huge has a good opportunity for a wealth of problems, such as how to preserve data consistency, how to replicate data the data for research purposes.

Although this application is just a prototypic implementation of SOS Web service, adding new Web data sources is easier. This application is very useful as retrieved data has already been pulled and displayed in the data grid of GUI, the user can retrieve data from the stored archive. The SOS Web service not only saves the bandwidth but also avoids the response time in retrieving the same data from distant offshore Web data sources.

This project can be further extended by enhancing the functionality of the application. The data from COMPS and TCOON takes a very long time to display data and sometimes they don’t display any. Such problems can be only resolved from the server side and in future once the server becomes efficient the application can be efficient too.

Currently, data from the stations of DISL, CenGOOS, COMPS, TCOON, MOTE have been integrated into the application. There is a good scope for extending the project for other ocean observatory stations which offer SOS Web service (Example NOAA/NDBC, GCOOS, etc). This
project can be further extended by enhancing the functionality of the application (extra parameters like date range).

The look and feel of the application can also be enhanced by incorporating extra features in the application. The parameter values can be shown in many different ways using data customization to improve the feasibility of the application.
6. CONCLUSION

The idea of integrating ocean observatory systems provides a unique interface for the user to search for the ocean data of ocean observatory systems that provide SOS Web service. This application emphasizes not only an ad-hoc, distributed Web data sources, but it provides a easily decentralized, scalable and extensible environment.

The application retrieves the data from the SOS response message and using .NET framework puts the data into the grid and display the results in a grid view as well as in a map pointing to the station locations.

The data integrated from the ocean observatory stations will provide user with the required ocean data in a well defined manner. Also, the resulting application is scalable enough if the number of Web sources to be integrated increases.

BIBLIOGRAPHY AND REFERENCES


APPENDIX A

A.1. Code for DIOOS class file:

```csharp
using System;
```
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;
using System.Text;
using System.Xml;
using System.IO;
using Subgurim.Controles;
using System.Collections.Generic;

namespace pavani
{
    public partial class _Default : System.Web.UI.Page
    {
        DataTable UrnOfStation;
        DataTable dtFinalTable;
        DataRow drMain;
        List<string> SelectedStationList = new List<string>();
        string mapToolTip;

        public String gvSortDirection
        {
            get { return ViewState["SortDirection"] as String ?? "ASC"; }
            set { ViewState["SortDirection"] = value; }
        }

        public String gvSortExpression
        {
            get { return ViewState["SortExpression"] as String ?? ""; }
            set { ViewState["SortExpression"] = value; }
        }

        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                GMap1.Visible = false;
                DataColumn dcTime = new DataColumn();
                dcTime.Caption = "Time";
                DataColumn dcDepth = new DataColumn();
                dcTime.Caption = "Depth";
                DataColumn ObservedProperty = new DataColumn();
                dcTime.Caption = "Property";
                DataColumn PropertyValue = new DataColumn();
                dcTime.Caption = "Value";
                imgBtnSubmit.onClientClick = "return ValidateSubmit('" +
                lblObservatory.ClientID + "," + hdnStationSelected.ClientID + "," +
                hdnOfferingSelected.ClientID + ")";
                lstbxStation.Attributes.Add("onclick", "OnStationLstClick('" +
                lstbxStation.ClientID + ")");
            }
        }
    }
}
lstbxOffering.Attributes.Add("onClick", "OnOfferingLstClick('" + lstbxOffering.ClientID + ")");
}
dtFinalTable = new DataTable();
dtFinalTable.Columns.Add("Observatory");
dtFinalTable.Columns.Add("Station");
dtFinalTable.Columns.Add("Observed Property");
dtFinalTable.Columns.Add("Observed Property Value");

//lstbxStation.Attributes.Add("onClick", "OnResonLstClick('" +
//lstbxStation.ClientID + "," + lblStation.ClientID + ")");
//lstbxOffering.Attributes.Add("onClick", "OnResonLstClick('" +
//lstbxOffering.ClientID + "," + lblOffering.ClientID + ")");

private DataTable GetDataTableFromDataSet(DataSet ds)
{
    DataTable dtTable = new DataTable();
dtTable.Columns.Add("Fields");
dtTable.Columns.Add("Values");
    if (ds.Tables.Count > 2)
    {
        string values =
            ds.Tables["DataArray"].Rows[0]["values"].ToString();
        string[] split = values.Split(',');
        ds.Tables["field"].Columns.Add("Values");
        for (int i = 0; i < ds.Tables["field"].Rows.Count; i++)
        {
            DataRow dr = dtTable.NewRow();
            dr["Fields"] = ds.Tables["field"].Rows[i]["name"];
            if (ds.Tables["field"].Rows.Count != split.Length)
                dr["Values"] = "";
            else
            {
                dr["Values"] = split[i];
                dtTable.Rows.Add(dr);
            }
        }
        return dtTable;
    }
    else
    {
        return GetEmptyDataTable();
    }
}

private DataSet GetDataSetByObservatory(string url)
{
    string sUrl = url;
    StringBuilder oBuilder = new StringBuilder();
    StringWriter oStringWriter = new StringWriter(oBuilder);
    XmlTextReader oXmlReader = new XmlTextReader(sUrl);
    XmlTextWriter oXmlWriter = new XmlTextWriter(oStringWriter);
    while (oXmlReader.Read())
    {
        oXmlWriter.WriteNode(oXmlReader, true);
    }
    oXmlReader.Close();
oXmlWriter.Close();
    DataSet ds = new DataSet();
try
{
    StringReader sr = new StringReader(oBuilder.ToString());
    ds.ReadXml(sr);
}
catch (Exception ex)
{
    //MessageBox.Show(ex.ToString());
}
return ds;

private DataTable GetEmptyDataTable()
{
    DataTable emptyTable = new DataTable();
    return emptyTable;
}

private DataSet GetCapabilitiesByCenter(string CenterID)
{
    string url = CenterID+"/oostethys_sos.cgi|oostethys_server.asp>?request=GetCapabilities&service=sos&version=1.0.0";
    DataSet dsInputValues = GetDataSetByObservatory(url);
    return dsInputValues;
}

protected void lstbxObsevatory_SelectedIndexChanged(object sender, EventArgs e)
{
    GMap1.Visible = false;
    gvTcoon.Visible = false;
    lblgridHeader.Visible = false;
    String selectedValue = lstbxObsevatory.SelectedItem.Value;
    lblObservatory.Text = selectedValue;
    hdnOfferingSelected.Value = "";
    hdnStationSelected.Value = "";
    switch (selectedValue)
    {
        case "Select":
            break;
        case "TCOON":
            DataSet TcoonCapabilities =
            GetCapabilitiesByCenter("http://lighthouse.tamu.edu/sos/oostethys_sos.cgi")
            ;
            SetStationDropDownValues(TcoonCapabilities);
            break;
        case "DISL":
            DataSet DISLCapabilities =
            GetCapabilitiesByCenter("http://gcoos.disl.org/cgi-bin/oostethys_sos.cgi");
            SetStationDropDownValues(DISLCapabilities);
            break;
        case "COMPS":
            break;
    }
DataSet COMPSCapabilities = GetCapabilitiesByCenter("http://compsdev1.marine.usf.edu/cgi-bin/sos/v1.0/oostethys_sos.cgi");
SetStationDropDownValues(COMPCapabilities);

break;

case "CenGOOS":
DataSet CenGoosCapabilities = GetCapabilitiesByCenter("http://www.cengoos.org/cgi-bin/oostethys_sos.cgi");
SetStationDropDownValues(CenGoosCapabilities);
break;

case "MOTE":
DataSet MoteCapabilities = GetCapabilitiesByCenter("http://coolcomms.mote.org/cgi-bin/sos/oostethys_sos.cgi");
SetStationDropDownValues(MoteCapabilities);
break;
}

protected void SetStationDropDownValues(DataSet CapabilityDataSet)
{
    DataRow[] drStations = CapabilityDataSet.Tables["Value"].Select("AllowedValues_Id='2'");
    DataRow[] drOffering = CapabilityDataSet.Tables["Value"].Select("AllowedValues_Id='3'");
    DataRow[] drStationLinks = CapabilityDataSet.Tables["Value"].Select("AllowedValues_Id='4'");

    DataTable dtOffering = drOffering.CopyToDataTable();
    DataTable dtStation = drStations.CopyToDataTable();
    UrnOfStation = drStationLinks.CopyToDataTable();

    lstbxOffering.DataSource = dtOffering;
    lstbxOffering.DataTextField = "Value_Text";
    lstbxOffering.DataValueField = "Value_Text";
    lstbxOffering.DataBind();

    lstbxStation.DataSource = dtStation;
    lstbxStation.DataTextField = "Value_Text";
    lstbxStation.DataValueField = "Value_Text";
    lstbxStation.DataBind();
}

protected void imgBtnSubmit_Click(object sender, ImageClickEventArgs e)
{
    List<string> stationList = new List<string>();
    List<string> offerlingList = new List<string>();

    string selectedValue = lstbxObsevatory.SelectedItem.Value;

    //string selectedStation = lstbxStation.SelectedItem.Value;
    foreach (ListItem listItem in lstbxStation.Items)
    {
        if (listItem.Selected == true)
string selectedProperty = lstbxOffering.SelectedItem.Value;
foreach (ListItem listItem in lstbxOffering.Items)
{
    if (listItem.Selected == true)
    {
        offerlingList.Add(listItem.Text);
    }
}
for (int i = 0; i < stationList.Count; i++)
{
    for (int j = 0; j < offerlingList.Count; j++)
    {
        PerformDataBind(selectedValue, stationList[i], offerlingList[j]);
    }
}
gvTcoon.DataSource = dtFinalTable;
gvTcoon.DataBind();
gvTcoon.Visible = true;
GMap1.Visible = true;
lblgridHeader.Visible = true;
Session["FinalTable"] = dtFinalTable;
GMap1.addControl(new
GControl(GControl.preBuilt.GOverviewMapControl));
GMap1.addControl(new
GControl(GControl.preBuilt.LargeMapControl));
//string Lat =
dtFinalTable.Select("Fields='latitude'").ElementAt(0)["Values"].ToString();
//string Long =
dtFinalTable.Select("Fields='longitude'").ElementAt(0)["Values"].ToString();
//GetGoogleMap(Lat, Long);
}
private void GetGoogleMap(String Lat, string Long, string MapToolTip)
{
    if (Lat != "" && Long != "")
    {
        GMap1.Visible = true;
        GMarker marker = new GMarker(new
GLatLng(Convert.ToDouble(Lat), Convert.ToDouble(Long)));
        GInfoWindow window = new GInfoWindow(marker, "<center><b>" +
MapToolTip + "</b></center>", true);
        GMap1.addInfoWindow(window);
    }
}
private void PerformDataBind(string ObservatoryName, string selectedStation, String selectedProperty)
{
    string url =
"/oostethys_sos.cgi||oostethys_server.asp?request=GetObservation&service=SO"
switch (ObservatoryName)
{
    case "Select":
        break;
    case "TCOON":
        url = url.Replace("<local name of offering>",
            selectedStation);
        url = url.Replace("<property>", selectedProperty);
        url = url.Replace("<urn of offering>",
            "http://lighthouse.tamucc.edu/overview/" + selectedStation);
        url = 
            "http://lighthouse.tamucc.edu/sos/oostethys_sos.cgi" + url;
        DataSet TcoonDataSet = GetDataSetByObservatory(url);
        DataTable dtTcoon =
        GetDataTableFromDataSet(TcoonDataSet);
        AddDataToMainTable(dtTcoon, selectedStation,
            ObservatoryName, selectedProperty);
        lblgridHeader.Text = "TCOON Data";
        break;
    case "DISL":
        url = url.Replace("<local name of offering>",
            selectedStation);
        url = url.Replace("<property>", selectedProperty);
        url = url.Replace("<urn of offering>",
            "urn:disl.org:source.mooring#" + selectedStation);
        url = 
            "http://gcoos.disl.org/cgi-bin/oostethys_sos.cgi" + url;
        DataSet DislDataSet = GetDataSetByObservatory(url);
        DataTable dtDisl = GetDataTableFromDataSet(DislDataSet);
        AddDataToMainTable(dtDisl, selectedStation,
            ObservatoryName, selectedProperty);
        lblgridHeader.Text = "DISL Data";
        break;
    case "COMPS":
        url = url.Replace("<local name of offering>",
            selectedStation);
        url = url.Replace("<property>", selectedProperty);
        url = url.Replace("<urn of offering>",
            "urn:marine.usf.edu:source.mooring#" + selectedStation);
        url = "http://compsdev1.marine.usf.edu/cgi-bin/sos/v1.0/oostethys_sos.cgi" + url;
        DataSet CompsDataSet = GetDataSetByObservatory(url);
        DataTable dtComps =
        GetDataTableFromDataSet(CompsDataSet);
        AddDataToMainTable(dtComps, selectedStation,
            ObservatoryName, selectedProperty);
        lblgridHeader.Text = "COMPS Data";
        break;
    case "CenGOOS":
        break;
}
url = url.Replace("<local name of offering>", selectedStation);
url = url.Replace("<property>", selectedProperty);
url = url.Replace("<urn of offering>", "urn:cengoos.org:source.moooring#" + selectedStation);
url = "http://www.cengoos.org/cgi-bin/oostethys_sos.cgi" + url;
DataSet CengoosDataSet = GetDataSetByObservatory(url);
DataTable dtCenGoos = GetDataTableFromDataSet(CengoosDataSet);
AddDataToMainTable(dtCenGoos, selectedStation, ObservatoryName, selectedProperty);
lblgridHeader.Text = "CenGOOS Data";
break;
case "MOTE":
url = url.Replace("<local name of offering>", selectedStation);
url = url.Replace("<property>", selectedProperty);
url = url.Replace("<urn of offering>", "" + selectedStation);
url = "http://coolcomms.mote.org/cgi-bin/sos/oostethys_sos.cgi" + url;
DataSet MoteDataSet = GetDataSetByObservatory(url);
DataTable dtMote = GetDataTableFromDataSet(MoteDataSet);
AddDataToMainTable(dtMote, selectedStation, ObservatoryName, selectedProperty);
lblgridHeader.Text = "MOTE Data";
break;
}
private void AddDataToMainTable(DataTable dtTable, string SelectedStation, string ObservatoryName, string SelectedProperty)
{
    if (dtTable.Rows.Count > 0)
    {
        drMain = dtFinalTable.NewRow();
        drMain["Station"] = SelectedStation;
        drMain["Observed Property"] = SelectedProperty;
        drMain["Observed Property Value"] = dtTable.Select("Fields='ObservedProperty1'").ElementAt(0)["Values"].ToString();
        drMain["Observatory"] = ObservatoryName;
        dtFinalTable.Rows.Add(drMain);
        if (!SelectedStationList.Contains(SelectedStation))
        {
            mapToolTip = "Observatory: " + ObservatoryName + "<br />
Station: " + SelectedStation + "<br />" + SelectedProperty + ": " + dtTable.Select("Fields='ObservedProperty1'").ElementAt(0)["Values"].ToString();
            SelectedStationList.Add(SelectedStation);
        }
        else
        {
            mapToolTip = mapToolTip + "<br />" + SelectedProperty + ": " +
        
    }
string lat = dtTable.Select("Fields='latitude'").ElementAt(0)["Values"].ToString();
string lon = dtTable.Select("Fields='longitude'").ElementAt(0)["Values"].ToString();
GetGoogleMap(lat, lon, mapToolTip);
}
}

protected void gvTcoon_Sorting(object sender, GridViewSortEventArgs e)
{
    if (e.SortExpression != null)
    {
        if (gvSortExpression == e.SortExpression)
            gvSortDirection = GetSortDirection();
        else
            gvSortDirection = "ASC";
        gvSortExpression = e.SortExpression;
        DataView dataview = new DataView((DataTable)Session["FinalTable"]);
        string sortDir = GetSortDirection();
        ViewState["SortDirection"] = sortDir;
        dataview.Sort = e.SortExpression + " " + sortDir;
        gvTcoon.DataSource = dataview;
        gvTcoon.DataBind();
    }
}

protected void gvTcoon_PageIndexChanging(object sender, GridViewPageEventArgs e)
{
    gvTcoon.PageIndex = e.NewPageIndex;
    gvTcoon.DataSource = (DataTable)Session["FinalTable"];    
    gvTcoon.DataBind();
}

private String GetSortDirection()
{
    String newSortDirection = String.Empty;
    switch (gvSortDirection)
    {
        case "DESC":
            newSortDirection = "ASC";
            break;
        case "ASC":
            newSortDirection = "DESC";
            break;
    }
    return newSortDirection;
}
A.2. Code for DIOOS Design page:

```csharp
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="pavani._Default" %>

<%@ Register Assembly="GMaps" Namespace="Subgurim.Controles" TagPrefix="cc1" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit" TagPrefix="asp" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Data Integration of Ocean Observatory Systems using Web services</title>
    <script src="Jscript.js" type="text/javascript"></script>
    <style type="text/css">
      .style2
      { width: 179px; }
      .style3
      { width: 133px; }
    </style>
  </head>
  <body style="height:100%">
    <form id="form1" runat="server">
      <table style="width: 80%;background-color: #99CC66;height:81px" align="center">
        <tr class="gradient ffffff 000000 vertical" style="width:100%;height:100%;filter:progid:DXImageTransform.Microsoft.Gradient(GradientType=0, EndColorStr='#000000', StartColorStr='#ffffff');" >
          <td colspan="4" align="center">
            <a style="font-size:larger;font-family:Arial Black;font-style:italic;color:White">Data Integration of Ocean Observatory Systems using Web services</a>
          </td>
        </tr>
      </table>
    </form>
  </body>
</html>
```
<tr>
    <td align="center">
        <asp:ImageButton ID="imgBtnSubmit" runat="server" ImageUrl="/Images/bluebutton_search.gif" onclick="imgBtnSubmit_Click" />
    </td>
    </tr>
</table>

---

```html
<asp:Image ID="Image1" runat="server" ImageUrl="Images/Untitled-1.jpg" />
</td>
</tr>
</table>

<table style="width: 80%;background-color: #99CC66;height:81px" align="center">
<tr valign="middle">
    <td class="style2" >
        <asp:Label ID="lblObservatoryText" runat="server" ForeColor="White" Text="Observatory : " />
        <asp:Label ID="lblObservatory" runat="server" Text="Select" Width="75"></asp:Label>
        <asp:DropDownExtender ID="ddExtObservatoryExt" TargetControlID="lblObservatory" DropDownControlID="lstbxObservatory" runat="server" />
    </td>
    <td class="style3" >
        <asp:Label ID="lblStationText" runat="server" ForeColor="White" Text="Station : " />
        <asp:Label ID="lblStation" runat="server" Text="Select" Width="75"></asp:Label>
        <asp:DropDownExtender ID="drpextStation" TargetControlID="lblStation" DropDownControlID="lstbxStation" runat="server" />
    </td>
    <td style="width:150px;">
        <asp:Label ID="lblOfferingText" runat="server" ForeColor="White" Text="Offering : " />
        <asp:Label ID="lblOffering" runat="server" Text="Select" Width="75"></asp:Label>
        <asp:DropDownExtender ID="drpextOffering" TargetControlID="lblOffering" DropDownControlID="lstbxOffering" runat="server" />
    </td>
    <td >
        <asp:ImageButton ID="imgBtnSubmit" runat="server" ImageUrl="/Images/bluebutton_search.gif" onclick="imgBtnSubmit_Click" />
    </td>
</tr>
</table>
```
A.3. Code for DIOOS Validation:

```javascript
var DDE;
var DDE1;
function pageLoad() {
    if ($find('ddExtObservatoryExt') != null) {
        $find('ddExtObservatoryExt')._dropWrapperHoverBehavior_onhover();
        $find('ddExtObservatoryExt').unhover = VisibleMe;
    }
    if ($find('drpextStation') != null) {
        $find('drpextStation')._dropWrapperHoverBehavior_onhover();
        $find('drpextStation').unhover = VisibleMe;
    }
    if ($find('drpextOffering') != null) {
        $find('drpextOffering')._dropWrapperHoverBehavior_onhover();
        $find('drpextOffering').unhover = VisibleMe;
    }
    DDE = $find('drpextOffering');
    DDE._dropWrapperHoverBehavior_onhover();
    if (DDE._dropDownControl) {
        $common.removeHandlers(DDE._dropDownControl,
        DDE._dropDownControl$delegates);
    }
    DDE._dropDownControl$delegates = {
        click: Function.createDelegate(DDE, ShowMe),
        contextmenu: Function.createDelegate(DDE,
        DDE._dropDownControl_oncontextmenu)
    }
    $addHandlers(DDE._dropDownControl, DDE._dropDownControl$delegates);
}
DDE1 = $find('drpextStation');
DDE1._dropWrapperHoverBehavior_onhover();
if (DDE1._dropDownControl) {
    $common.removeHandlers(DDE1._dropDownControl,
    DDE1._dropDownControl$delegates);
}
DDE1._dropDownControl$delegates = {
    click: Function.createDelegate(DDE1, ShowMe1),
    contextmenu: Function.createDelegate(DDE1,
    DDE1._dropDownControl_oncontextmenu)
}
$addHandlers(DDE1._dropDownControl, DDE1._dropDownControl$delegates);
```
function ShowMe() {
    DDE._wasClicked = true;
}

function ShowMe1() {
    DDE1._wasClicked = true;
}

function VisibleMe() {
    $find('ddExtObservatoryExt')._dropWrapperHoverBehavior_onhover();
    $find('drpextStation')._dropWrapperHoverBehavior_onhover();
    $find('drpextOffering')._dropWrapperHoverBehavior_onhover();
}

function OnResonLstClick(listView, lblID) {
    var combo1 = document.getElementById(listView);
    var lblLstReason = document.getElementById(lblID);
    //var val = combo1.options[combo1.selectedIndex].text;
    for (var i = 0; i < combo1.length; i++) {
        if(combo1.options[i].selected == true)
            lblLstReason.innerHTML = combo1.options[i].innerHTML;
    }
    //    var cmbText = combo1.options[combo1.selectedIndex].text;
    //    if (combo1.options[combo1.selectedIndex].text.length > 12) {
    //        lblLstReason.innerHTML = cmbText.substring(0, 12)
    //    } else {
    //        lblLstReason.innerHTML = cmbText;
    //    }
    // lblLstReason.title = combo1.options[combo1.selectedIndex].text;
}

function ValidateSubmit(lblObservatory, lblStation, lblOffering) {
    var Observatory = document.getElementById(lblObservatory);
    var Station = document.getElementById(lblStation);
    var Offering = document.getElementById(lblOffering);
    var msg = '';

    if (Observatory.innerHTML == "Select" & msg == '') {
        msg = 'Please Select Observatory';
    } else if (Station.value!="True") {
        msg = 'Please Select Station';
    } else if (Offering.value != "True") {
        msg = 'Please Select Offering '
    }
if (msg != '') {
    alert(msg);
    return false;
} else {
    return true;
}

function OnStationLstClick(lstbxStationID) {
    var hdnStationField = document.getElementById('hdnStationSelected');
    hdnStationField.value = "True";
}

function OnOfferingLstClick(lstbxOfferingID) {
    var hdnOfferingField = document.getElementById('hdnOfferingSelected');
    hdnOfferingField.value = "True";