

# Automated Web Service Inventory Management Software

*The Benefits of Client Driven Dynamically Generated Web Services*



Philip Pittle

Thesis Supervisor:  
Adam Belloum

Faculty of Science  
Universiteit van Amsterdam

A thesis submitted for the degree of *Master of Science (MSc)* in *Grid Computing*

2012 July

*The transition from science to e-Science is happening: a data deluge emerges from publicly-funded research facilities; a massive investment of public funds into the potential answer to the grand challenges of our times. This potential can only be realised by adding an interoperable data sharing, re-use and preservation layer to the emerging ecosystem of e-Infrastructures. The importance of this layer, on top of emerging connectivity and computational layers, has not yet been addressed coherently at ERA or global level.*

**- Opportunities for Data Exchange**

<http://cordis.europa.eu/fp7/ict/e-infrastructure/docs/ode.pdf>

*With a proper scientific e-infrastructure, researchers in different domains can collaborate on the same data set, finding new insights. They can share a data set easily across the globe, but also protect its integrity and ownership. They can use, re-use and combine data, increasing productivity. They can more easily solve today's Grand Challenges, such as climate change and energy supply. Indeed, they can engage in whole new forms of scientific inquiry, made possible by the unimaginable power of the e-infrastructure to find correlations, draw inferences and trade ideas and information at a scale we are only beginning to see.*

**- High Level Expert Group on Scientific Data**

<http://cordis.europa.eu/fp7/ict/e-infrastructure/docs/hlg-sdi-report.pdf>

## Acknowledgments

I would like to acknowledge the encouragement and support of those that helped me in completing not only this master's thesis but the entire Computer Science – Grid Computing master's program at the University of Amsterdam. I would like to thank my friends and family for encouraging my decision to study abroad and turning my dream into reality. I thank the excellent faculty at the University of Amsterdam for providing an exciting and enriching atmosphere for learning and personal growth. And I must additionally acknowledge the faculty for having the flexibility to custom tailor the program to best fit my needs and for helping to ensure I completed the program on time.

I thank my supervisor and professor, Dr. Adam S.Z. Belloum for his guidance, knowledge and wisdom. His patient encouragement greatly contributed to the quality of this thesis paper and his support gave me the confidence to challenge myself and pursue a topic of my choosing.

Finally, I must acknowledge my former colleagues at the Public Broadcasting Service (PBS). It was my time at PBS that first exposed me to the intricacies of the data sharing problem and the lack of adequate solutions available. And it was there that I first had the opportunity to explore the problem and made the decision to pursue its solution.

To everyone that has helped me on this two year journey, thank you. I could not have come this far without you.

## Abstract

Data sharing is integral to unlocking the full potential of scientific data; enabling multi-disciplinary researchers to collaborate and work together in ways not otherwise possible. And the growing adoption of e-science infrastructures opens new possibilities for data analysis as long as raw data is readily available. Yet, despite the recognized value of data sharing, modern technology infrastructures have yet to provide the scientific community with an easy-to-use data sharing solution. In order for a solution to effectively meet the needs of the scientific community and promote data sharing, three data sharing goals have been identified:

1. Ease the burden on data owners that want to share their data.
2. Enable data consumers to easily consume data in a format most conducive to their needs.
3. Support the integrated data schema so that data can be readily consumed by computation engines.

Web services have emerged as a standardized and interoperable data sharing mechanism. However, while several frameworks exist to ease implementation, utilizing them requires non-trivial technology expertise; proving that the frameworks alone are not sufficient to meet the data sharing goals. Further, operating a web service inventory in practice requires additional effort, specifically version management, and the technical know-how to deploy, operate and secure a web server.

This thesis introduces the Automated Web Service Inventory Management Software (AWSIMS) as a complete data sharing solution that expands upon the proven power of web services to meet the data sharing goals stated above. The design and architecture are presented to discuss how AWSIMS meets the data sharing goals. The application is then evaluated against three use cases to determine how AWSIMS meets data sharing requirements in real world scenarios. Finally, AWSIMS's applicability in the realms of bio-informatics and health care are discussed.

## Contents

Acknowledgments .....	3
Abstract.....	4
List of Figures.....	8
List of Tables .....	9
1 Introduction.....	11
1.1 Scientific Data in the Computer Age.....	14
1.2 e-Science Infrastructure .....	15
1.3 Data Sharing Goals.....	15
1.4 Web Services .....	18
1.4.1 SOAP .....	18
1.4.2 WSDL .....	19
1.4.3 UDDI.....	20
1.4.4 Web Service Limitations .....	20
1.5 Case Studies .....	21
1.5.1 Case Study: Small Research Team.....	21
1.5.2 Case Study: Intra-Department Sharing .....	22
1.5.3 Case Study: Large Multi-Disciplinary Research Team.....	22
1.6 Current Solutions.....	23
1.6.1 Web Service Based Solutions .....	24
1.7 Motivation and Goal.....	25
2 Automated Web Service Inventory Management Software.....	27
2.1 Design Goals .....	27
2.1.1 Querying Data and Schema.....	27
2.1.2 Enable Users to Design Web Services, Methods, and Return Types .....	27
2.1.3 Generate Standards Compliant Web Services .....	28

2.1.4	Web Service Versioning .....	28
2.1.5	Discoverability .....	28
2.1.6	Extensibility .....	29
2.1.7	Scalability and Performance .....	29
2.2	.NET Foundation .....	29
2.2.1	Customizing the ASP.NET Pipeline .....	31
2.3	Architecture .....	32
2.3.1	Data Sources .....	34
2.3.2	Client Driven Interface Generation .....	37
2.3.3	Web Service Compilation .....	40
2.3.4	Web Service Method Execution.....	42
2.3.5	Versioning.....	43
2.3.6	Discoverability.....	44
2.3	Platform Benefits.....	46
2.3.1	Support for Multiple Protocols.....	46
2.3.2	Performance .....	47
2.3.3	Upgradability .....	48
2.4	Deployment .....	48
2.4.1	AWSIMS Cloud.....	49
3	Analysis .....	51
3.1	Interview with the Bio-informatics Community.....	51
3.2	Real World Integration: VPH-Share.....	51
3.3	Case Studies .....	54
3.3.1	Case Study: Small Research Team.....	54
3.3.2	Case Study: Intra-Department Sharing .....	55
3.3.3	Case Study: Large Multi-Disciplinary Research Team.....	55
3.4	Further Discussion.....	56

3.4.1 Big Data .....	56
3.4.2 Semantic Web .....	57
4 Conclusion .....	58
4.1 Future Work .....	59
4.1.1 User Review and Feedback .....	59
4.1.2 CRUD Support .....	59
4.1.3 Security .....	60
References .....	62
Appendix A – A Quick User Guide to AWSIMS in the Cloud .....	67
A.1 Sharing Data .....	67
A.2 Consuming Data .....	69

## List of Figures

Figure 1 - Swine Flu Outbreaks in Several Geographic Regions .....	12
Figure 2 - Health Care Facilities' Overlapping Coverage Areas .....	12
Figure 3 - Diagnosis for Hospital 1.....	13
Figure 4 - Diagnosis for Hospital 2.....	13
Figure 5 - Aggregated Diagnoses for Hospital 1 and 2.....	14
Figure 6 - The Modern Data Sharing Problem .....	16
Figure 7 - ENFOS Email Based Workflow .....	23
Figure 8 - AWSIMS - The Data Sharing Solution.....	26
Figure 9 - IIS - ASP.NET Request Pipeline.....	30
Figure 10 - AWSIMS Web Service Server Pipeline .....	32
Figure 11 - AWSIMS Architecture Diagram .....	33
Figure 12 - Data Source Key Classes .....	35
Figure 13 - Screenshot of AWSIMS Client Showing the Web Service Method Creation Screen .....	38
Figure 14 - Query Mutator and Query Mutator Parameter Base Classes .....	40
Figure 15 - AWSIMS Web Service Definition Compilation Pipeline .....	41
Figure 16 - Sample Web Service Method Source Code .....	42
Figure 17 - Example Output from the Catalog Web Service .....	45
Figure 18 - AWSIMS Deployment Diagram .....	49
Figure 19 - LOBCDER Architecture .....	53



## **List of Tables**

Table 1 - List of Existing and Planned Data Sources.....	35
--	----



## 1 Introduction

The advancement and application of science is dependent on quality scientific data created through observation and calculation. However, for the full value of scientific data to be realized it must be shared and aggregated within the scientific community. When data resides in information silos accessible only to an individual scientist or scientific team, the data is prevented from being subjected to independent peer review in order to verify its quality. Teams working on similar hypothesize or overlapping areas must dedicate limited resources towards recreating data sets; hampering the overall pace of scientific progress. And, perhaps most importantly, the ability to analyze that data in a comprehensive manner is reduced and conclusions drawn from limited data sets can be inaccurate and inconclusive.

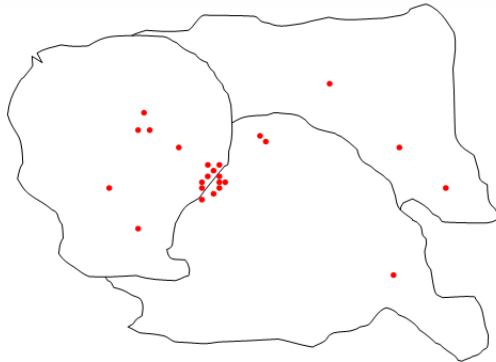
The field of epidemiology, specifically the study of severe food borne illness outbreaks is a perfect example justifying the need and value of the open sharing of scientific data. Local physicians and hospitals collect diagnosis data from patients, but as these medical establishments often have overlapping coverage areas, rarely can a single establishment determine if there are sufficient diagnosis's to qualify as a localized outbreak. Instead, responsibility typically falls to a government agency, such as the Dutch Ministry of Health, Welfare and Sport (VWS)<sup>1</sup>, to aggregate and analyze data from local physicians in order to spot regional and national trends.

The distributed data management solution, OGSA-DAI<sup>2</sup>, provide an excellent example of this within the context of epidemiology, from which I paraphrase. Within several geographical regions there have been several diagnoses of patients exhibiting swine flu symptoms. Figure 1 - Swine Flu Outbreaks in Several Geographic Regions, presents this, showing the home address of patients as red dots. From this view, it is easy to recognize that the majority of patients reside in small clusters, engendering the conclusion that the contagion might be localized and warranting investigation of that location.

---

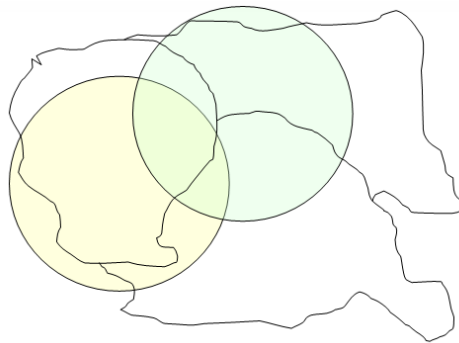
<sup>1</sup> <http://www.government.nl/ministries/vws>

<sup>2</sup> <http://www.ogsadai.org.uk/about/index.php>



**Figure 1 - Swine Flu Outbreaks in Several Geographic Regions<sup>3</sup>**

While the conclusion that the contagion is largely concentrated in one area is easy to deduce once the data has been aggregated, in many cases this is a non-trivial step. In our example region, there are multiple health care facilities with overlapping coverage areas, as shown in Figure 2 - Health Care Facilities' Overlapping Coverage Areas. Consequently, patients from the identified "hot-spot" may elect to go to different health care providers. Figure 3 - Diagnosis for Hospital 1 shows the diagnoses that one provider, Hospital 1 is aware while Figure 4 - Diagnosis for Hospital 2 shows the diagnoses that another provider, Hospital 2, is aware of.



**Figure 2 - Health Care Facilities' Overlapping Coverage Areas<sup>4</sup>**

---

<sup>3</sup> Original image <http://www.ogsadai.org.uk/about/index.php>

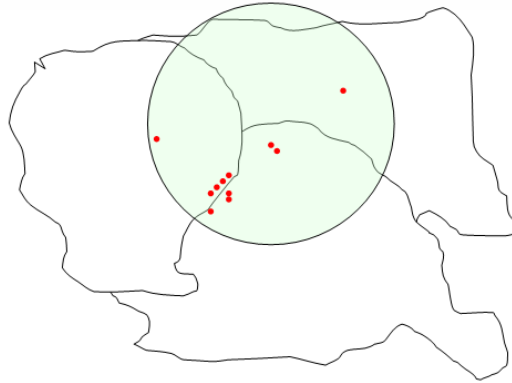


Figure 3 - Diagnosis for Hospital 1<sup>5</sup>

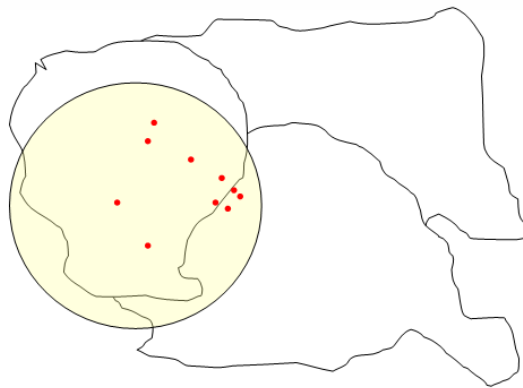


Figure 4 - Diagnosis for Hospital 2<sup>6</sup>

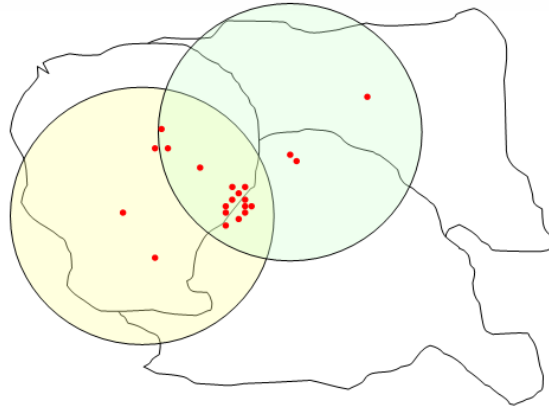
As the diagrams above illustrate, neither Hospital 1 nor Hospital 2 has enough information to make a definitive conclusion. Only when the two data sets are aggregated together, as in Figure 5 - Aggregated Diagnoses for Hospital 1 and 2, is it possible to reach a correct and informed conclusion.

---

<sup>4</sup> Original image <http://www.ogsadai.org.uk/about/index.php>

<sup>5</sup> Original image <http://www.ogsadai.org.uk/about/index.php>

<sup>6</sup> Original image <http://www.ogsadai.org.uk/about/index.php>



**Figure 5 - Aggregated Diagnoses for Hospital 1 and 2**

## **1.1 Scientific Data in the Computer Age**

The recent proliferation of computing equipment and networks and the ubiquitousness of computing in daily human activities has engendered an avalanche of data creation. While the increase of scientific data is unequivocally beneficial for science, its true potential cannot be unlocked without data sharing mechanisms that facilitate the aggregation and analysis of ever increasing quantities of data. But this rise in computer generated data has fundamentally changed data sharing requirements. The sheer volume of data being generated means that traditional human analysis is simply not feasible. Take the Large Hadron Collider (LHC) as an example. Project experiments produce roughly 15 petabytes of data annually [8] and with such high data volumes, human analysis of raw data is logistically impossible. While the LHC is certainly on the high end of data producing experiments, it is indicative of the way forward for modern science.

As experiments are producing more data than a human can analyze, scientists must rely on computational resources to perform the analysis. This in turn means that modern data sharing must be consumable by computation engines. The advent of the public Internet has indeed made huge troughs of information available; however, it has specialized in serving data summaries in formats easily consumed by humans. It has not been nearly as successful in sharing massive amounts of raw data in a format conducive to algorithmic and computational analysis. The scientific community has largely recognized the value of

data sharing for enabling scientific progress and the need to overcome the deficiencies found in current technology infrastructures [5][6][7].

## 1.2 e-Science Infrastructure

The maturation of the field of high performance computing has produced computing infrastructures that enable an application to access the resource equivalent of tens of thousands of conventional desktop computers. Known commonly as Cloud computing [11] and Grid computing [12], these systems offer scientists near unlimited computational processing power to perform analyses never before possible. This technology, when applied in conjunction with the Internet, with its collaborative nature and philosophy, form the heart of what has been dubbed e-Science [13].

E-Science is a response to the needs of modern science. With theoretical and observational experimentation generating ever larger and complex volumes of data, research objectives becoming increasingly multi-disciplinary and research teams being more often geographically dispersed, the technological infrastructures from even 10 years ago are no longer adequate. Beyond greatly increasing the computing power available to scientists, e-Science also seeks to provide tools to enhance collaboration and encourage data sharing. This is accomplished by a fundamental shift in design ideologies and principles aimed at bringing science technical infrastructure in line with the Internet age; e-Science is built upon a model of service oriented architecture.

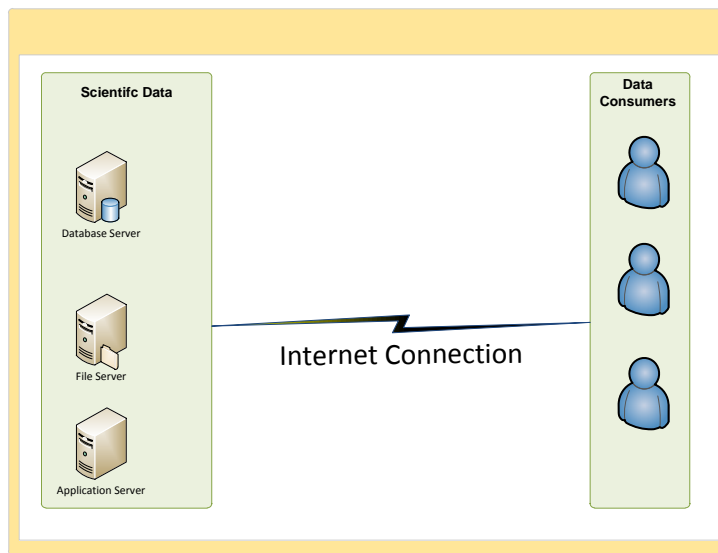
Service oriented architecture (SOA), as a design paradigm, is the loose coupling of autonomous interactive software agents [14] that operate as services. It provides for communication between agents over a network, such as the Internet, via a standardized interface framework [15], promoting interoperability and providing a strong foundation for computing across distributed hardware. Clients can invoke services, utilizing this standard communication channel, to send data and receive a response.

The successful adoption and utilization of e-Science requires scientists to make their data available via SOA communication services; fundamentally changing the nature of the data sharing problem. Scientific data must be made consumable by e-Science frameworks. By doing so, not only will scientists reap the benefits e-Science provides, they will intrinsically make their data available to the scientific community at large.

## 1.3 Data Sharing Goals

As technology evolves and science evolves to take full advantage of it, the value of data sharing persists; but the mechanisms and technical requirements of sharing change. The

proliferation of computing has resulted in the vast majority of scientific data being digitized and warehoused electronically. The rise of the Internet has effectively networked the vast majority of the scientific community and has become the transport medium of choice. And scientific data aggregation and analysis has become reliant on computer applications, most notably e-Science infrastructures. Thus, the sharing of scientific data is distilled to the problem of effectively sending data from one computer system to another, via the Internet, such that it can be appropriately consumed; as can be seen in Figure 6 - The Modern Data Sharing Problem.



**Figure 6 - The Modern Data Sharing Problem**

The service-oriented approach to performing distributed and computationally intensive scientific research is potentially very powerful. Yet widespread adoption by many scientists has been depressed, partly due to the technical difficulties involved in sharing data [16]. When these difficulties are overcome, researchers with widely different backgrounds - from the humanities and social sciences to the physical, biological and engineering sciences – can collaborate on the same set of data from different perspectives. Indeed, we begin to see what some have called a “fourth paradigm” [35] of science – beyond observation, theory and simulation, and into a new realm of exploration driven by mining new insights from vast, diverse data sets. [17]



In order to free data from being locked up in “data islands” [36] and help realize the full potential of data, this thesis identifies the following data sharing goals an application must meet in order to fulfill the data sharing needs of the modern scientific community:

1. Ease the burden on data owners that want to share their data.
2. Enable data consumers to easily consume data in a format most conducive to their needs.
3. Support the integrated data schema so that data can be readily consumed by computation engines.

For many researchers, the concept of research data sharing is a new one and they do not know how to share their data [5]. Technological challenges should not be a barrier for a data owner that has made the decision to share data. Additionally, data owners should not be overly burdened to adapt their data to meet the needs of data consumers, who may want the data to be provided in a certain format or the data sharing solution to support advanced features such as pagination and filtration. Lowering the level of effort required to both share data and maintain a repository will enable more data owners to share their data and keep it up-to-date. And by offering data owners a solution that provides performance and stability, they can be confident they are meeting the needs of their data consumers.

For a data consumer to receive the maximum value from shared data, they must be able to easily consume it using standardized protocols. Additionally, they need the data customized to a format that meets their needs and offers querying functionality that allows them to efficiently retrieve it. Together, this ensures data consumers can quickly and effectively make use of shared data repositories, combining and aggregating it as well as allowing them to quickly evolve their querying techniques.

In the age of modern computing infrastructures, it is essential that any data sharing solution supports computational discovery, processing and data retrieval. Complex analysis of data is no longer possible without the aid of computers and requiring a data consumer to manually derive data format and structure creates an onerous burden. Additionally, the sheer volume of data being generated annually makes it impossible for a data consumer to effectively find a relevant data repository, mandating that a data sharing solution supports discoverability. The onus is on the data owner to ensure that not only their data is made easily available, but that a data consumer can query the data repository to retrieve a description and structure of the data being stored.

## 1.4 Web Services

Web service technologies have emerged as a key tool for facilitating the structured sharing of machine readable data in distributed system architectures, capable of powering current and future e-Science infrastructures [16]. Web services adhere to SOA design principles defining a framework for defining and invoking autonomous and interoperable software agents. Consequently, it is a key technology for implementing modern data sharing solutions.

The World Wide Web Consortium (W3C)<sup>7</sup> is the standards organization responsible for defining and curating the official Web Services Architecture definition [18] from which an excerpt is presented below:

*A web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.*

Essentially, the web service framework is comprised into three primary areas: communication, description, and discovery [19]. Communication between software agents takes place using the Simple Object Access Protocol (SOAP). Web Service Description Language is used to provide a formal definition of a web service and its capabilities. Universal Description, Discovery and Integration (UDDI) provides a standard repository mechanism where clients can discover web services available in a given network. Each topic is explored further in the sections below.

### 1.4.1 SOAP

SOAP was originally developed by Microsoft<sup>8</sup> with the goal of creating a protocol for remote procedure calls (RPCs) that could operate on the existing Internet infrastructures. This meant it needed to be platform-independent, interoperable and text based. The designers opted to base the protocol in XML, the lingua franca for information and data encoding for platform independence and internationalization [19]. Additionally, because SOAP and XML are text based, SOAP can use existing internet transport protocols, such as HTTP.

---

<sup>7</sup> [www.w3.org](http://www.w3.org)

<sup>8</sup> [www.microsoft.com](http://www.microsoft.com)

The W3C elected to use SOAP as the communication platform of choice for the web services framework and now maintains the official protocol definition [21]. An excerpt from the specification is presented below:

*SOAP provides a simple and lightweight mechanism for exchanging structured and typed information between peers in a decentralized, distributed environment using XML. SOAP does not itself define any application semantics such as a programming model or implementation specific semantics; rather it defines a simple mechanism for expressing application semantics by providing a modular packaging model and encoding mechanisms for encoding data within modules. This allows SOAP to be used in a large variety of systems ranging from messaging systems to RPC.*

SOAP functions as essentially a message passing protocol, with SOAP messages (XML documents) passed between client and server. A SOAP message consists of three parts:

1. The SOAP envelope is the root of the XML file and defines an overall framework for expressing what is in a message; who should deal with it, and whether it is optional or mandatory.
2. An optional SOAP header allows the injection of specialized processing rules, including rules defining serialization and how recipients should process SOAP messages.
3. A SOAP body containing the serialized message. This can either be a RPC sent from a client or a payload containing a data set returned from a web service.

Using SOAP, web services have a robust mechanism for executing remote procedures to query data. And because SOAP messages and payloads are serialized into XML, a plain text data format, it can be transported over existing Internet technologies, most notably HTTP(S). Additionally, the standard fully embraces interoperability to ensure not only that a client on any platform can consume any web service, but web services developed on different technology stacks can communicate with one and another.

#### **1.4.2 WSDL**

For clients to understand the capabilities of a web service, properly format a RPC message, and correctly parse the response, the client must be able to retrieve a standardized description of the web service. The Web Service Description Language [37] is the web service solution to this problem. WSDL files describe a web service in two ways, the abstract implementation details and the concrete end points. The abstract

component of a WSDL file describes method signatures and type definitions; essentially how messages will be serialized. The concrete component of a WSDL file contains binding information that instructs clients on which communication protocol is supported, how to accomplish individual server interactions and an endpoint, such as a port number.

### 1.4.3 UDDI

Universal Description, Discovery and Integration (UDDI) provides a standardized mechanism for web services to be registered and queried from a repository. UDDI repository entries contain descriptive information about not only each web service and where it's located, but information about the author as well, such as business name. Querying and updating a repository is accomplished via SOAP messages, meaning the repository itself functions as a web service.

UDDI repositories add a powerful level of functionality to the web service framework. Because they support querying, a client can easily find a web service that offers the functionality or provides the data they need. Additionally, multiple vendors can implement similar web services, allowing clients to choose from a range of market options. Finally, the repository model supports service choreography protocols, such as Web Service Choreography Description Language (WS-CDL) [22], by providing a standard mechanism for resource discovery and advertisement.

### 1.4.4 Web Service Limitations

While there exists multiple web service frameworks [20] that abstract the complexities of communicating and implementing web services, these frameworks still impose implementers possess a non-trivial technological skill set. First and foremost, authoring a web service requires rudimentary programming skills, including code authoring, library linking, and source code compilation. Hosting a web service requires instantiating a web server, and while most modern operating systems include a basic web server, hosting a web service on a personal computer does not provide a highly-available platform. Consequently, working knowledge of deploying a web application to a dedicated server is required. And this list precludes the skills necessary to deploy a web service that follows industry best practices and meets non-functional requirements of high availability, performance and scalability.

Web services pose an additional barrier for a successful implementation: interfaces. A web service interface defines the methods and method signatures provided by a web service, including the structure of the returned data in WSDL format [18]. Web service interfaces are required to be defined at compile time and cannot be modified without a

web service being recompiled. Because of this, a web service author must collaborate with potential implementers and stakeholders in order to ensure all necessary data is made available in a format that can be readily consumed, before the web service can be authored and deployed. This back and forth negotiation adds negative time pressures, slowing down software development cycles, especially in rapidly changing environments [1].

Problems associated with interfaces do not end once a web service has been successfully deployed and accepted by all stakeholders. The schema of data provided by a web service is not inherently static and may evolve. This in turn requires web service interfaces be updated to expose the new schemas. As web service clients are developed with the assumption that web service schemas will not change [3], web service authors are burdened with creating backwards-compatible versions of their services in order to ensure old clients may continue to use them. Unfortunately, there is a lack of robust versioning support in relevant standards and tools [2] resulting in web service authors having to develop non-standardized ad-hoc solutions, thus increasing the man hour costs necessary to implement new features.

## 1.5 Case Studies

This thesis explores the barriers to successfully implementing web services described in the previous section; technical skill requirements cost of interface negotiation, and version management, in the context of three theoretical case studies. The case studies are first described in this section as well as the applicable web service limitations relevant to each case. Each case study is re-examined in section 3 Analysis to gauge how the problems preventing a successful web service implementation can be addressed.

### 1.5.1 Case Study: Small Research Team

The first case study is a small university research team performing biological experiments where the results are stored in a local database provided by the university. The team has two data sharing goals. First, they want to make their data available to be consumed by an e-Science Grid so that it can be used to drive large scale computations. Second, the team wants to publish a paper as an executable publication [49] and need their data to be publically available.

The research project is mature at this stage and there is little risk of the data schema changing. However, the database provided by the university does not allow public access and the team lacks the in-house technical skills required to build a data sharing solution themselves. Additionally, their budget does not afford hiring outside developer resources. In essence, the team does not have the technical capacity to share their data.

### **1.5.2 Case Study: Intra-Department Sharing**

The second case study is inspired by my personal experience in industry. A company is grouped into several departments. The internal solutions department has a small technical staff and is responsible for managing web applications for supporting industry partners. Through several years of operations they have amassed a large database of information on the company's partners and this data may have value to other departments within the company, specifically the customer solutions department, which is responsible for managing public facing web applications for customers. The two departments operate on differing technology stacks, one using Microsoft's .NET platform, the other using LAMP. Providing the customer solutions department with direct access to the internal solutions database will overburden the database administrator and is risky because of interoperability concerns. A conclusion is reached to implement a SOA solution utilizing web services.

While the internal solutions department does have a technical staff they do not have the time or budget to implement a solution from scratch. Additionally, the staff does not have experience implementing a highly available SOA application capable of powering a public facing web application, thus posing implementation risk. Further complicating the task, several other departments have also expressed interest in accessing the same data, should the project prove successful, but would have differing interface requirements. This poses potential versioning problems as the internal solutions department could be faced with supporting multiple stakeholder requirements. In essence the internal solutions team does not have the resources to develop an enterprise grade data sharing application that supports versioning.

### **1.5.3 Case Study: Large Multi-Disciplinary Research Team**

The third case study is a large multi-disciplinary research team working on a multi-phase project. The team stores research results in a database and routinely makes their data available to the public via web services. As the project has progressed it has been necessary to change and update their database schema to store data collected in new phases. For each schema change they have to author an updated version of applicable web services and as a result have a large inventory of versioned web services. The number of versions has become unwieldy to manage and the team is having to spend increasing resources to support it. Additionally, the team is unsure which versions of their web services are actively being used and which can be safely decommissioned. In essence, the team has a versioning problem resultant from supporting multiple evolutions of their web services.

## 1.6 Current Solutions

The Internet age has spawned a variety of data sharing solution. This section explores a sample of these technologies and evaluates them against the data sharing goals from section 1.3 Data Sharing Goals.

E-mail is perhaps the most widely adopted data sharing solution offered in the Internet age. Users commonly share photos with one another or collaborate on document authoring by attaching files to an email message and sending it directly to the recipient. Companies have even built workflows using e-mail to share data between parties as can be seen in Figure 7 - ENFOS Email Based Workflow. ENFOS<sup>9</sup> is an environment liability and risk management company that works with contracted field scientists to take local environment samples and then aggregates that data to provide their clients with information as to whether or not they are in compliance with industrial pollution legislation. Their reporting workflow requires field contractors to manually email raw data packages to a centralized report generation service, which in turn emails reports to clients [23].

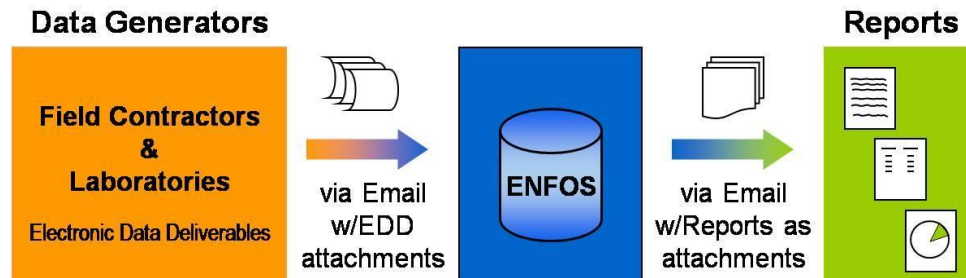


Figure 7 - ENFOS Email Based Workflow<sup>10</sup>

While e-mail is decisively simple to use for data owners, it presents a significant barrier to data consumer; it lacks a query mechanism. Data consumers must send an e-mail to the data owner requesting data and as a human must reply, response times can easily take days at best. In the worst case, requests for data are easily ignored or denied [24], proving email to be ineffective data sharing solution.

<sup>9</sup> [www.enfos.com](http://www.enfos.com)

<sup>10</sup> Original image <http://www.enfos.com/www/productsSolutionsLab.php>

Another current data sharing solution, GridFTP, is a data transfer protocol designed for transferring files to and from the grid as well as between different grid implementations. GridFTP is based on the traditional FTP protocol and is therefore geared towards only working with files; it can't be used to transfer data from databases. Consequently, GridFTP does not support sharing data schema, one of the data sharing goals identified in section 1.3 Data Sharing Goals.

Several of the largest collaborative research projects decide against using existing technology directly and instead create their own custom data sharing solution. The Large Hadron Collider project is one such example and the team has created a project specific solution called the Worldwide LHC Computing Grid (WLCG). The WLCG is a highly customized grid platform combining resources from hundreds of world wide data centers to store and process the 15 Petabytes of data generated annually; making the data available to more than 8,000 physicists around the world [26]. While the volumes of data generated, as well as, the budget available for project such as the LHC, allow them to develop their own unique data sharing solution, this option isn't available to the vast majority of research projects.

#### **1.6.1 Web Service Based Solutions**

Web services offer an ideal framework for creating a data sharing solution and research has been done to both expand its usefulness as well as reduce the technological complexities required for implementation. One of the more mature projects is OGSA-DAI, which describes itself as a distributed data access and management solution. OGSA-DAI supports a wide range of data sources including databases, files and web services and can be accessed directly on the web or from grid or cloud infrastructures. While OGSA-DAI supports the data sharing goals, supporting data consumers as well as sharing schema data, standing up a working OGSA-DAI instance is non-trivial task. [25] describes the manual installation process for OGSA-DAI requiring the setup and configuration of a Tomcat web server instance, a MySQL database instance, and several classpath and bash configurations. In the end, this makes OGSA-DAI a very challenging tool for a non-technical team of scientists, or a technical team short on resources to implement.

Escalante et al have researched and developed a tool, SW4BD, to automatically generate web services for database management systems (DBMS) [27]. Their SW4BD tool excels in allowing users to stand up web services that query databases without needing to write the web services themselves or even have a basic understanding of the web service protocol. This strategy meets all of the data sharing goals allowing data owners to easily



share their data, enabling data consumers an easy mechanism for consumption and supports the sharing of data schemas. However, the problem with this approach is it is primarily intended for facilitating querying across heterogeneous DBMS platforms. Users are required to enter the SQL commands a generated web service is to execute, creating technical requirements upon users as well as opening up serious security concerns. A malicious user could, for example, use techniques similar to SQL injection attacks to delete data. Consequently, this makes the SW4BD platform less than ideal for use as data sharing platform in a public or non-trusted environment.

## **1.7 Motivation and Goal**

There is a clear demand for a robust data sharing solution that lowers the barrier to entry for non-technical users as well as that supports integrating with modern e-Science infrastructures. Web services have been shown as an ideal framework, but proper implementation places unreasonable demands on data owners. The goal of this thesis is to develop an application that, similar to SW4BD [27], generates web service wrappers to underlying data sources without requiring data owners or data consumers to write either web service or data access code. In essence, the goal is to transform Figure 6 - The Modern Data Sharing Problem into Figure 8 - AWSIMS - The Data Sharing Solution.

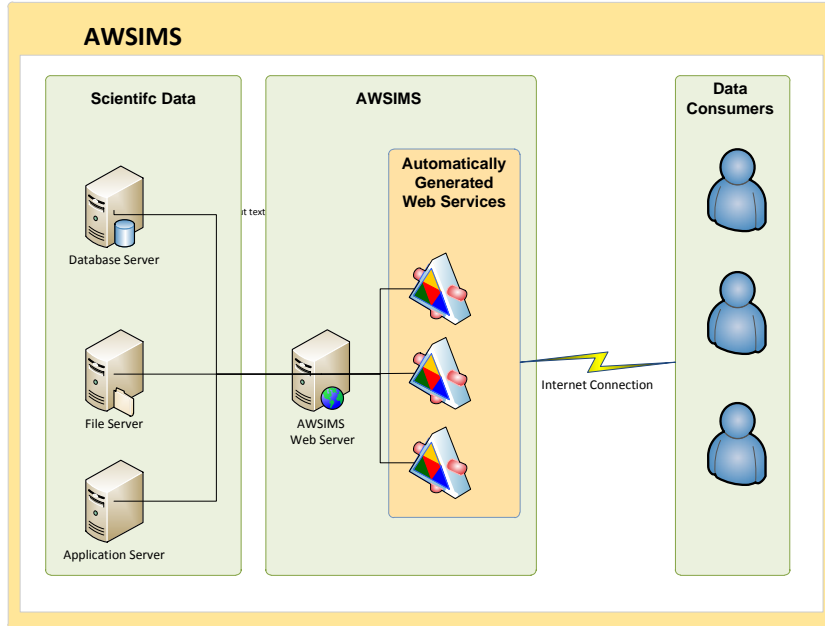


Figure 8 - AWSIMS - The Data Sharing Solution

## 2 Automated Web Service Inventory Management Software

### 2.1 Design Goals

The overarching goal of Automated Web Service Inventory Management Software (AWSIMS) is to expand the capabilities of web services into an enterprise class [30] solution that meets the data sharing goals outlined in section 1.3 Data Sharing Goals. AWSIMS should enable data owners to specify a data source they would like to share and then enable data consumers to generate web services based on the available data sources. In order to accomplish these top level goals, the following design goals were developed for AWSIMS:

- Support querying both data and schema from a variety of data sources including databases and spreadsheets.
- Enable data consumers to design web services, web service methods, and return types.
- Generate standards compliant web services.
- Support web service versioning.
- Support Discoverability.
- Support Extensibility.
- Provide Scalability and Performance required for highly available web service servers.

Each goal is discussed further in the sections below.

#### 2.1.1 Querying Data and Schema

In order to automate the creation and execution of web services, AWSIMS must be able to query both the data and schema of a variety of data sources. When a user specifies a data source they would like to share they should not need to specify the schema, AWSIMS should parse the schema directly from the data source. This is necessary to ease the burden on data owners.

#### 2.1.2 Enable Users to Design Web Services, Methods, and Return Types

This design goal ensures that data consumers can easily consume data in a format conducive to their needs without burdening the data owner. Data consumers should be able to specify the name and namespace of any web service and the name of any web service method that they create. When they create a web service method they should be able to map it to an existing data source and then customize the return type by specifying

the name and the fields it will return. Additionally, they should be able to indicate they would like advanced features to modify the returned results, such as sorting, filtering, or pagination.

### **2.1.3 Generate Standards Compliant Web Services**

Web services generated by AWSIMS should be fully standards compliant, including WSDL definitions, calling conventions, and serialization. This is a requirement for AWSIMS to operate as an interoperable data sharing solution enabling data to be consumed by computation engines. Additionally, AWSIMS should have the ability to support other standards, such as Java Script Object Notation (JSON)<sup>11</sup> to further promote data sharing and support as many data consumers as possible.

### **2.1.4 Web Service Versioning**

AWSIMS should allow data consumers to modify existing web services and create new versions of a web service. Data consumers should be able to decide which version of a web service they wish to invoke without requiring them to implement modifications to web service convention or standards. Data owners should be informed of the number of web services and web service methods relying on a particular data source so they can know if modifying the data source could break a web service. This goal reduces the burden of managing a web service inventory for data owners as well as empowering data consumers to change the format of the data they consume to match their needs as the evolve.

### **2.1.5 Discoverability**

AWSIMS should generate web services that advertise their functionality by generating valid and standards compliant WSDL files, such that a consuming application can query the full capabilities of each web service. Additionally, AWSIMS should offer a standardized catalog web service that lists the following information for each web service that has been generated:

1. Web service name and namespace
2. Web service WSDL Url
3. Name of every method
4. Invocation Url for every method

This information allows all generated web services to be fully discoverable and supports computer applications querying AWSIMS for the services and data provided.

---

<sup>11</sup> <http://www.json.org/>

### 2.1.6 Extensibility

AWSIMS should support modular additions that provide enhanced functionality and maintenance releases. Adding additional features should not require users to perform any maintenance on any data sources, web services, or web service methods that have been defined. This goal ensures AWSIMS has architectural designs in place to ensure it can expand iteratively to offer enhanced data sharing features; future-proofing the application.

### 2.1.7 Scalability and Performance

AWSIMS should be able to effectively scale vertically and horizontally in order to meet demand, mitigate hardware failure risk and maintain high performance. While the performance of generated web services will be dependent on the performance of the underlying data source, web services should implement caching to improve performance under high loads. These non-functional requirements ensure AWSIMS can function optimally in real-world scenarios and ensure AWSIMS can appropriately meet the needs of data consumers.

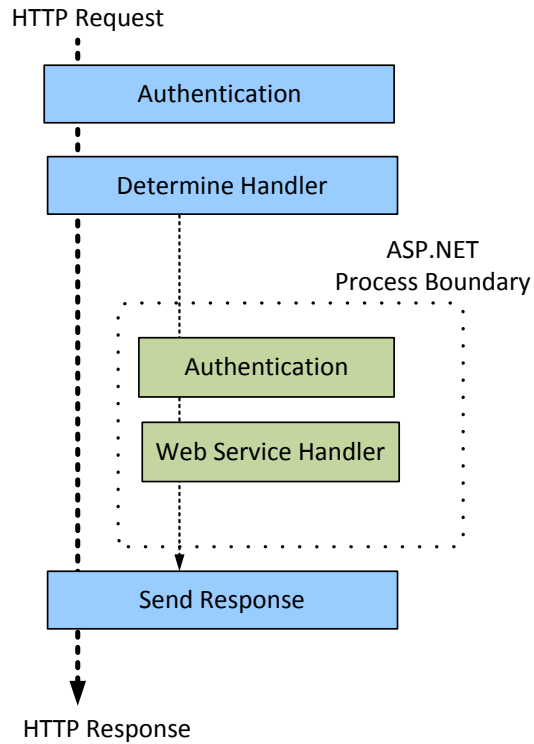
## 2.2 .NET Foundation

Several frameworks were evaluated during the design of AWSIMS with the goal of selecting a framework that provided as much core web service functionality as possible. The criteria included support for run-time web service generation and compilation, automatic WSDL generation and xml type serialization. After the evaluation period Microsoft .NET was selected as the framework of choice to serve as the architectural foundation for AWSIMS as it met the goals listed above and was the framework the author was most familiar with.

The Microsoft .NET technology stack utilizes the Microsoft Internet Information Server (IIS) web server and ASP.NET web and web service libraries. The http client request pipeline, as diagramed in Figure 9 - IIS - ASP.NET Request Pipeline, begins with IIS receiving a client request such as:

```
http://Example/HelloWorld.asmx/HelloWorld
```

IIS performs some initial inspection including authentication and authorization checks. The mime-type is parsed from the request URL and indicates the request should be handled by ASP.NET. IIS spins up an instance of the ASP.NET common language runtime (CLR) and passed control to ASP.NET for further processing [31].



**Figure 9 - IIS - ASP.NET Request Pipeline**

Once control passes to the CLR, ASP.NET launches its own pipeline. The request is mapped to a web application based on the URL and the pipeline loads all applicable configuration files. The request is then authenticated and authorized by ASP.NET<sup>12</sup> and ASP.NET instantiates the appropriate `IHttpHandler` to service a response [32]. In a default configuration, ASP.NET instantiates a `WebServiceHandlerFactory` [33], which examines the request URL and `Http Verb`, maps the request to a web service definition file, compiles the web service definition, and instantiates a handler that executes the compiled web service and writes a response to the client.

---

<sup>12</sup> ASP.NET authentication and authorization is done in addition to authentication and authorization performed by IIS.

The default ASP.NET pipeline expects a web service request to map to a physical web service definition file by requiring the URL to be of the format:

```
<http | https>://<path>/<to>/<file>/<class filename>.asmx/<method>
```

The example in **Error! Reference source not found.** would be mapped to:

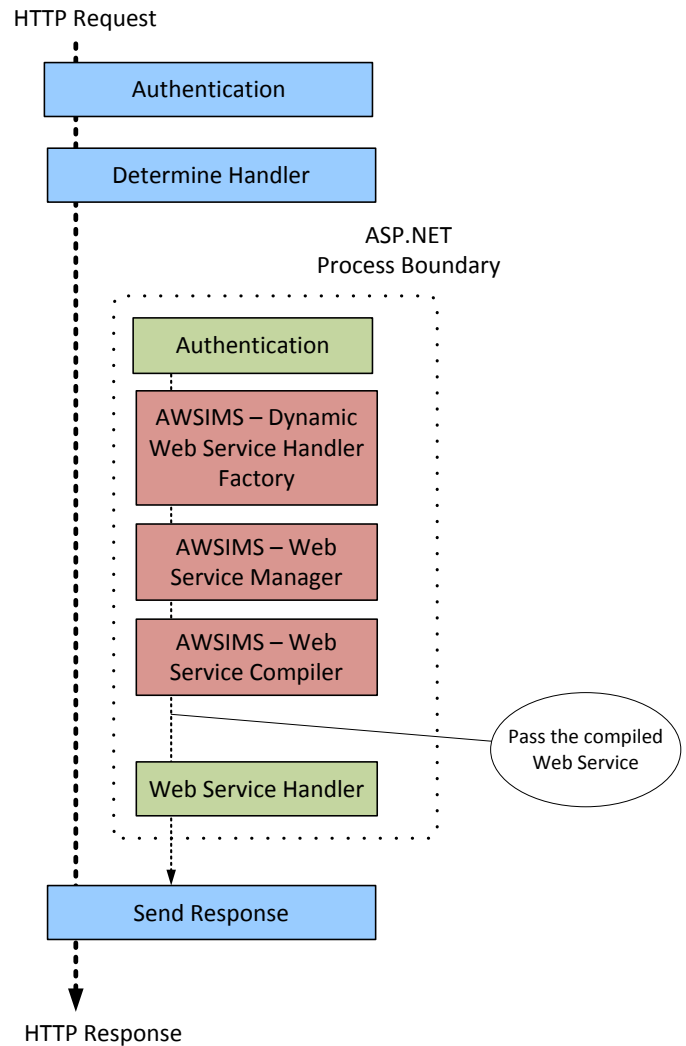
```
%Web Application Root%/Example/HelloWorld.asmx
```

Consequentially, the default ASP.NET web service pipeline has an implicit requirement that every web service be backed by a physical class definition file. This creates the restrictions that every client be served the same web service, and as each web service definition explicitly states the return type for every web service method, each client is returned the same return type. Herein, lays the root of the versioning problem intrinsic to the ASP.NET web service pipeline. Because the run time does not allow the intelligent routing of requests to different web service definitions, a web service author is required to change the physical path and filename of a web service in order to implement a new version.

The physical file mapping requirement additionally prevents the ASP.NET web service pipeline from supporting authorization or alternative filtering of a method's return type. So if a web service method has a return type comprised of three properties, X, Y and Z, there is no mechanism to indicate that property Y should only be served to clients that have passed an authorization check and property Z should only be returned at the client's request, as indicated by the presence of a Query String parameter.

### 2.2.1 Customizing the ASP.NET Pipeline

AWSIMS overcomes these difficulties, yet retains the proven capabilities of the ASP.NET pipeline, by hijacking the `WebServiceHandlerFactory`. The AWSIMS Web Service Server (WSS) overrides the default `IHttpHandler` (i.e. `WebServiceHandlerFactory`) and provides a custom mapping for determining and compiling the web service definition that should be instantiated in response to a client request. Once a web service definition is selected and compiled, the WSS passes the compiled type back to the `WebServiceHandlerFactory`, resuming the default pipeline and serializing the result set into a proper SOAP message. The WSS pipeline is displayed in Figure 10 - AWSIMS Web Service Server Pipeline.



**Figure 10 - AWSIMS Web Service Server Pipeline**

### 2.3 Architecture

AWSIMS’s architecture is conceptually divided amongst several components based on intended functionality. Component interaction is displayed in Figure 11 - AWSIMS Architecture Diagram and each component is described in the list below.



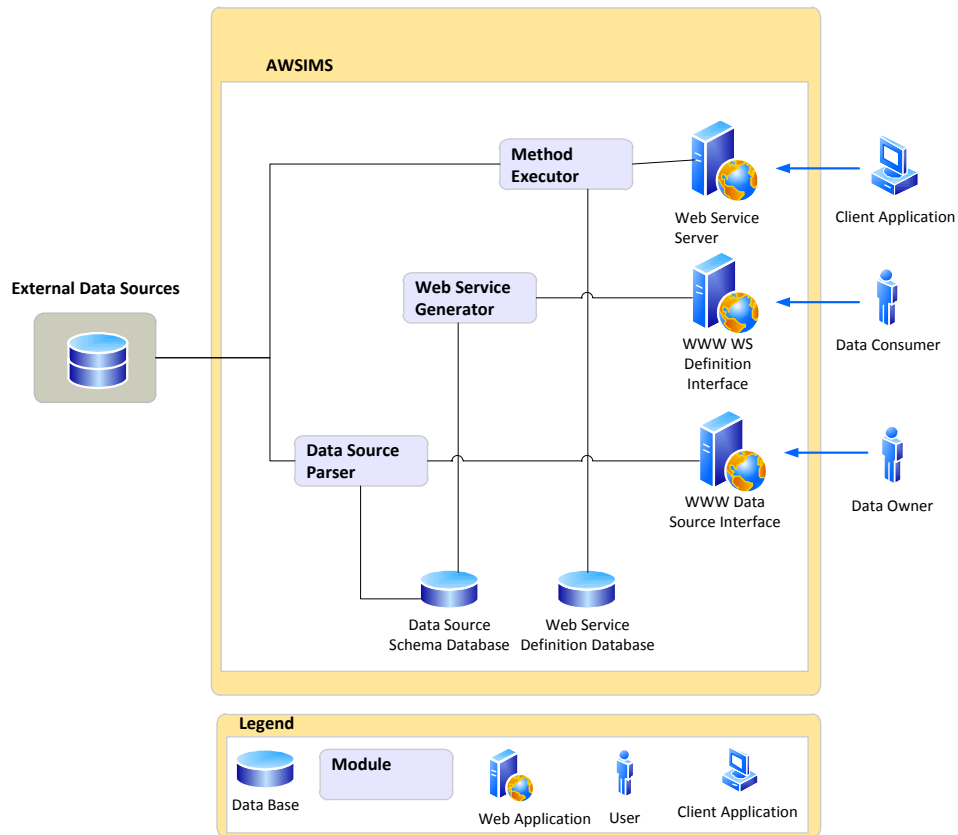


Figure 11 - AWSIMS Architecture Diagram

- **Web Service Server (WSS)** - Responsible for handling and processing all web service requests including WSDL generation, return type serialization and SOAP message formatting.
- **Web Service Definition Interface** - Web front end that provides an interface for data consumers to define and modify web service and web service method definitions.
- **Data Source Interface** - Web front end that provides an interface for data owners to define data sources and generate schema.
- **Method Executor** - Invoked by the WSS to instantiate and populate a return type. Responsible for retrieving and compiling and executing a web service method definition.

- **Web Service Generator** - Invoked by the Web Service Definition Interface to generate a properly formed web service definition and web service method definition based on user input.
- **Data Source Parser** - Invoked by the Web Service Definition Interface to query a user specified data source and generate properly formed data source schema definitions.
- **Data Source Schema Database** - Persistent storage repository for data source schema definitions.
- **Web Service Definition Database** - Persistent storage repository for web service definitions and web service method definitions.

The proceeding sections detail key architectural design issues that enable the components listed above to successfully interact to enable users to define data sources and web service definitions and allow AWSIMS to compile and execute web service methods.

### 2.3.1 Data Sources

In order for AWSIMS to provide access to data via a web service method, the method must be mapped to a data source definition created within the application. A data source definition's initial responsibility is to detail how to connect and extract a data source's schema. The schema is a collection of one or more schema columns and is conceptually analogous with a database schema. Each schema column must, at minimum, contain a name and a data type, as this information will be used by the web service method to define the fields in the return type.

The AWSIMS data source model is designed to maximize the number of supported data sources by utilizing a modular approach. The goal is to create a standardized interface so that new data sources can easily be added and will integrate seamlessly into the existing application. This allows schema parsing to be unified with the end result that parsing system can be reused by multiple data sources. Additionally, it allows the web service definition process to work with any data source through this standard interface.

Data sources are created by authoring classes that inherit from the base classes `SchemaDataSource`, `SchemaDefinition`, `DataSourceParser`, and optionally `SchemaColumn`. Class diagrams for these classes are shown in Figure 12 - Data Source Key Classes. This strategy provides a standardized interface for web service definitions to map to any data source regardless of its underlying implementation.

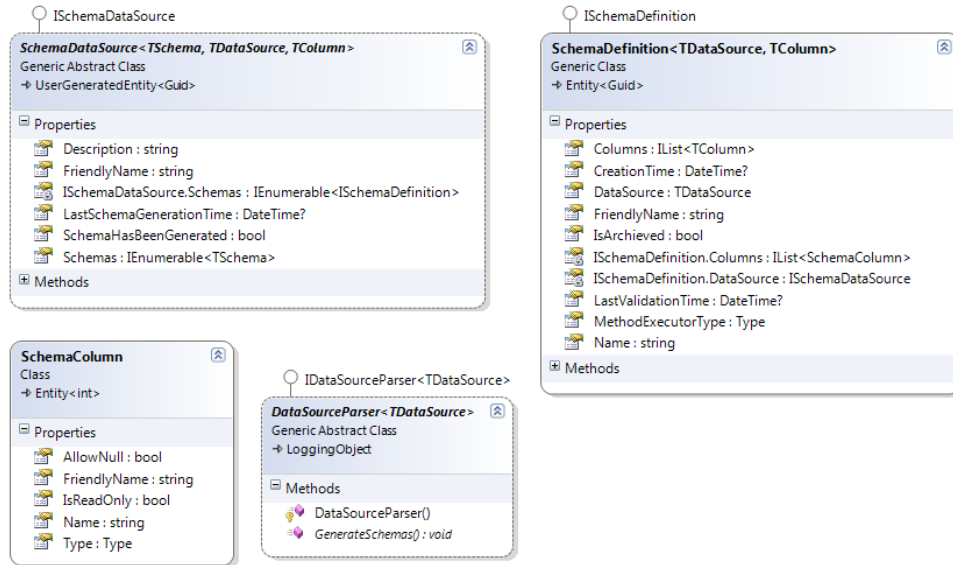


Figure 12 - Data Source Key Classes

The module approach to data source definitions takes on a special case for file system based data sources. AWSIMS provides a specialized infrastructure for file system data sources such that the data source is only responsible for implementing IO (input/output) functions and is not responsible for directly generating schema. Instead a collection of File Parsers examine each file in the data source and generate the schema. This mechanism allows AWSIMS to support a file of a given type independent of the file system and provides robust support for web based file systems such as DropBox<sup>13</sup> and the VPH-Share project [40].

A full list of data sources provided by AWSIMS is listed in Table 1 - List of Existing and Planned Data Sources. The table additionally lists data sources that have yet to be implemented but are planned for a future release.

Table 1 - List of Existing and Planned Data Sources

Data Source	Description
SQL Database	Connects to a Microsoft SQL Database and provides read only access to Tables and Views.

<sup>13</sup> www.dropbox.com

Local File	Serves a single file that is located on a logical drive the server can access. This includes physical and network drives. File support is limited by available File Parsers.
Local Directory	Serves all files (non-recursively) that are located in a defined directory on a logical drive the server can access. This includes physical and network drives. Data Source supports optionally specifying a regular expression white list filter against file names. File support is limited by available File Parsers.
DropBox Directory	Serves all files (non-recursively) that are located in a DropBox cloud storage folder. Data Source supports optionally specifying a regular expression white list filter against file names. File support is limited by available File Parsers.
WebDAV Directory	Serves all files (non-recursively) that are located in a defined WebDAV folder. Data Source supports optionally specifying a regular expression white list filter against file names. File support is limited by available File Parsers. This Data Source was created to provide support for the VPH-Share project <sup>14</sup> [40].
ODBC Database (Planned)	Connects to any database engine that provides a .NET compatible Open Database Connectivity (ODBC) driver. Data Source provides read only access to Tables and Views.
.NET Assembly (Planned)	Imports an existing external .NET Assembly and provides wrappers for methods meeting the following requirements: <ul style="list-style-type: none"> <li>• Method is public</li> <li>• Method return type is serializable</li> <li>• Method parameters are serializable</li> <li>• Method has a public parameterless constructor</li> </ul>
JAVA Assembly (Planned)	Imports an existing external Java Assembly and provides wrappers for methods meeting the following requirements: <ul style="list-style-type: none"> <li>• Method is public</li> <li>• Method return type is serializable</li> <li>• Method parameters are serializable</li> <li>• Method has a public parameterless constructor</li> </ul>

<sup>14</sup> Additional information on the integration of VPH-Share and AWSIMS is presented in section 3.2 Real World Integration: VPH-Share

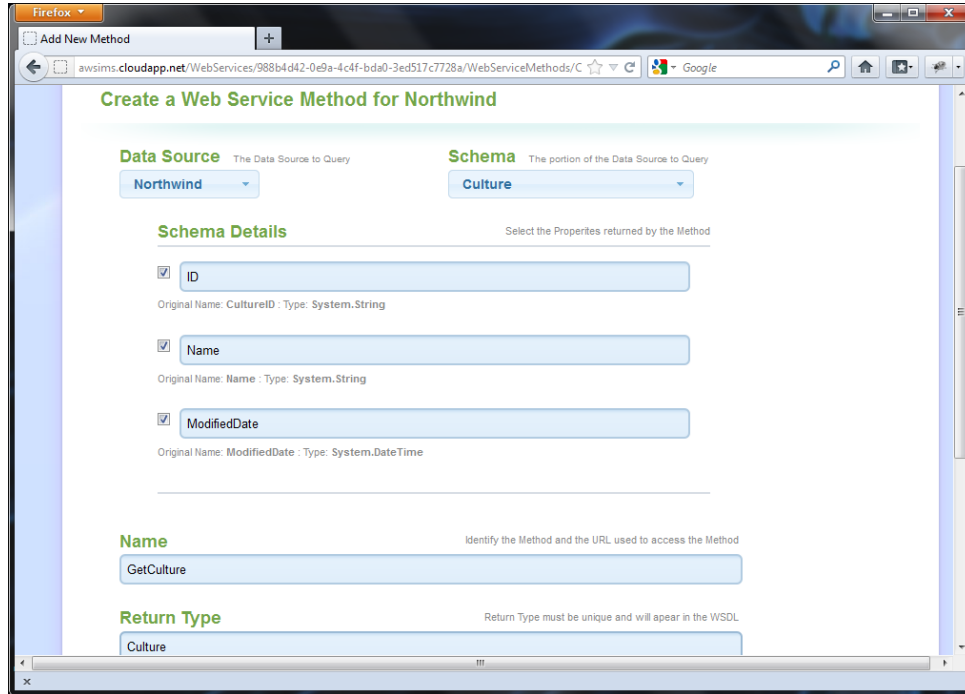
COM Object (Planned)	Imports an existing external COM Object <sup>15</sup> and provides wrappers for methods meeting the following requirements: <ul style="list-style-type: none"> <li>• Method is public</li> <li>• Method return type is serializable</li> <li>• Method parameters are serializable</li> <li>• Method has a public parameterless constructor</li> </ul>
-------------------------	---

### 2.3.2 Client Driven Interface Generation

The key to absolving data owners from the burden of negotiating web service method interfaces with data consumers is empowering data consumers to independently define the interfaces they need. The AWSIMS client tool is responsible for not only providing this functionality but ensuring that the process is as simplistic as possible for the data consumer. Using it, a potential data consumer can define their own web service(s), specifying a name and namespace, and can then populate each web service with methods. The data consumer is required to map to an existing data source and schema, but can customize the return type, specifying the name and which fields should be returned. Figure 13 - Screenshot of AWSIMS Client Showing the Web Service Method Creation Screen shows the simplicity and ease of defining a web service method.

---

<sup>15</sup> <http://www.microsoft.com/com/default.msp>



**Figure 13 - Screenshot of AWSIMS Client Showing the Web Service Method Creation Screen**

Beyond simply defining web service methods, the AWSIMS client tool offers a data consumer the ability to customize the method's query behavior using a feature called Query Mutators. By default, when a method is executed AWSIMS will return all data present in the mapped data source schema in the order it is stored. This corresponds to a `SELECT *` from a database data source or a sequential file read from a file system data source. Query Mutators override this default behavior by allowing data consumers to customize the returned result set.

When presenting a data set to a user it is common to provide sorting and filtering controls. Additionally, in order to minimize load times and not overwhelm users, data is paginated so that it can be sent in manageable subsets. Supporting this functionality is computationally expensive and is therefore expected to be provided server side. If the data source is a web service, this means the web service must support this functionality. Within AWSIMS, adding this functionality to a web service method is the job of Query Mutators.

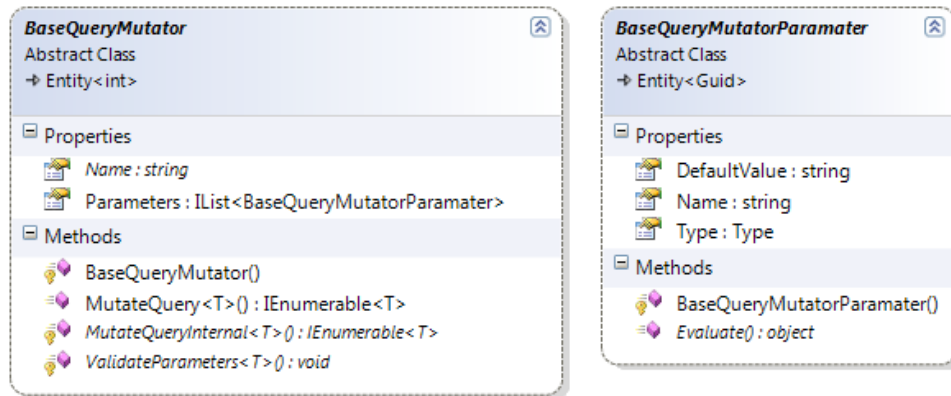
When defining a web service method, the data consumer can add multiple Query Mutators to perform data manipulation tasks such as filtering, sorting and pagination. Query Mutators are executed in order to provide the correct behavior. For example, if a method is to both sort and then limit the number of results to the first 100, the sorting Query Mutator is applied first followed by the limiter Query Mutator.

As many of the Query Mutators are parameter driven, sorting for example, needs to know which column to sort on and in which direction, AWSIMS provides Query Mutator Parameters. Query Mutator Parameters provide a mechanism for data consumers to define the data needed or how the web service client will provide this data. For example, a data consumer may define a web service method where the sorting column is defined in a constant but the sort direction is provided by a custom query string parameter.

Following the design goal of extensibility, the Query Mutator and Query Mutator Parameters architecture follows the Strategy and Strategy Factory patterns [41]. These Gang of Four pattern create a standardized interface to allow zero or more Query Mutators to modify the result set and greatly simplifies their use in the method execution pipeline<sup>16</sup>. Each Query Mutator and Query Mutator Parameter inherits from a respective base class, shown in Figure 14 - Query Mutator and Query Mutator Parameter Base Classes, unifying behavior, ensuring interoperability and providing a simplified hook for future Query Mutators and Query Mutator Parameters to plugin to the existing application infrastructure.

---

<sup>16</sup> See section 2.3.4 Web Service Method Execution for more information on the method execution pipeline.



**Figure 14 - Query Mutator and Query Mutator Parameter Base Classes**

The net result of the AWSIMS client tool is data consumers are given a powerful set of features that enable them to design and customize web services to meet their individual specifications. They can easily modify their web service methods, modifying return type fields and, via Query Mutators, the result set itself. Modifications are realized immediately, allowing data consumers greater flexibility in implanting their software lifecycles. And all of this can be accomplished with zero input and effort on the part of data owners.

### 2.3.3 Web Service Compilation

Once a data owner has defined a data source and a data consumer has used the AWSIMS client interface to create a corresponding web service method definition, the web service must be compiled before any requests can be served by the Web Service Server. When a web service or web service method is defined, AWSIMS automatically generates the source code needed for execution. The source code for the web service is comprised of three chunks generated at two different times. The web service class is generated when a data consumer defines a new web service and is comprised of a simple namespace and class declaration with the class decorated with the necessary attributes for the ASP.NET Web Service framework to recognize the web service correctly. The source code for the web service method and its return type are generated when a data consumer creates and edits a web service method definition. Whenever source code is generated, it is additionally analyzed and any external dependencies are noted so that they can be passed to the compiler during compilation.



Compilation is performed Just-In-Time<sup>17</sup> by the AWSIMS compilation pipeline. When the AWSIMS WSS intercepts a request from the ASP.NET pipeline, it invokes the AWSIMS compilation pipeline via a call to the `IWebServiceManager` interface. The pipeline examines the request Url and client identity maps the request to a web service definition. Once the web service definition is mapped, the pipeline checks the local assembly store to see if a previously compiled assembly is available and if so the assembly is loaded and the requested web service type is returned. Otherwise, the web service definition is compiled, saved in the local assembly store and the requested type is then returned. This process is shown in Figure 15 - AWSIMS Web Service Definition Compilation Pipeline.

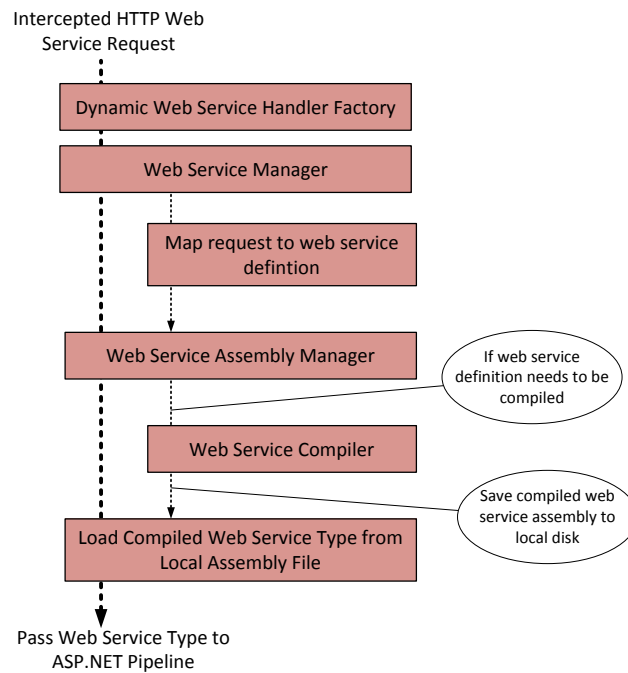


Figure 15 - AWSIMS Web Service Definition Compilation Pipeline

<sup>17</sup> [http://en.wikipedia.org/wiki/Just\\_In\\_Time\\_compilation](http://en.wikipedia.org/wiki/Just_In_Time_compilation)

The AWSIMS compilation pipeline relies on the wrapper classes for the C# compiler<sup>18</sup> provided by the .NET framework. When a web service definition is flagged for compilation, the source code for the web service, each method, and each method's return type are aggregated as to produce a single assembly. Dependencies are also aggregated into a unique list, as the compiler will fail if an external assembly is referenced more than once. This process allows a user generated web service definition to be properly compiled and fully served by the ASP.NET Web Service pipeline facilitating both WSDL generation and method execution.

### 2.3.4 Web Service Method Execution

When a web service method definition is created or updated, AWSIMS generates the source code that will be executed by the ASP.NET Web Service pipeline. The source code is responsible for returning a single object or a collection of objects that can be serialized by the ASP.NET Web Service pipeline and returned to the web service caller. The source code, as shown in Figure 16 - Sample Web Service Method Source Code, has several key features necessary to properly integrate with the ASP.NET Web Service pipeline as well as to meet AWSIMS design goals.

```
1: [System.Web.Services.WebMethod]
2: public System.Collections.Generic.List<GeneratedReturnType.Category2> GetCategories()
3: {
4:     var kernel = AWSIMS.WSS.NinjectKernelFactory.DefaultKernel;
5:     return kernel.Get<AWSIMS.Core.WebService.MethodExecutor.SqlMethodExecutor>()
6:         .Execute<GeneratedReturnType.Category2>(
7:             new System.Guid("c5bfea33-ea7e-413b-81c0-92330c76004c"),
8:             new System.Web.HttpContextWrapper(System.Web.HttpContext.Current));
9: }
```

Figure 16 - Sample Web Service Method Source Code

First and foremost, the web service method must be decorated with the `WebMethod` attribute and must be marked `public`. Failure to do either of these will result in the ASP.NET Web Service pipeline throwing a run-time exception. The return type of all AWSIMS web service methods is a generic type of `List`<sup>19</sup>, which will be serialized as an array. This prevents AWSIMS from needing to inspect a data source to determine if it will return a single item or a collection and has little impact on callers.

---

<sup>18</sup> <http://msdn.microsoft.com/en-us/library/microsoft.csharp.csharpcodeprovider.aspx>

<sup>19</sup> <http://msdn.microsoft.com/en-us/library/6sh2ey19.aspx>

The body of the web service method is designed so that no data processing is performed within the generated source code. Instead, the method offloads all processing to a `IMethodExecutor` class specialized to handling a certain type of data source. The `IMethodExecutor` interface exposes a single generic method, `Execute`, which expects the return type and ID of the web service method. With this information the method executor can query the schema powering the web service method and populate a collection of strongly typed return types. Additionally, the `IMethodExecutor` interface provides all child classes a standardized engine for incorporating the Query Mutators discussed in section 2.3.2 Client Driven Interface Generation.

AWSIMS makes heavy use of the Dependency Injection pattern<sup>20</sup>, specifically the Ninject framework<sup>21</sup>. The pattern requires objects list their dependencies, in the case of Ninject, via a constructor. Object instantiation is relegated to a kernel, which is configured with a type mapping that enables the kernel to inject dependencies into an instantiated object, via its constructor. AWSIMS web service methods make use of Dependency Injection when instantiating a `IMethodExecutor`. A kernel instance is provided by the static class `AWSIMS.WSS.NinjectKernelFactory`, which is responsible for providing Ninject with sufficient type mappings to enable it to instantiate all known method executors.

The Dependency Injection pattern offers AWSIMS WSS significant extensibility. Because the web service method code isn't responsible for instantiating its method executor, additional dependencies could be added to a method executor and as long as the `NinjectKernelFactory` is updated, the changes could be deployed to the hosting server and existing web service method source code would still be compatible and would not need to be regenerated. Additionally, these changes could be incrementally staged across a server farm and all servers in the farm could still serve the same web service methods. Ultimately, this pattern ensures the longevity of AWSIMS web service methods, ensuring that once they are generated, they will not need to be regenerated, even in the face of platform upgrades.

### 2.3.5 Versioning

Several web service versioning techniques have been proposed by the computer science community. Some focus on modifying the SOAP header [2] while others offer version specific WSDL [37] or namespaces [9] [38] [39]. However, these techniques place the onus of authoring and maintaining multiple web service versions on the data owner.

---

<sup>20</sup> [http://en.wikipedia.org/wiki/Dependency\\_injection](http://en.wikipedia.org/wiki/Dependency_injection)

<sup>21</sup> <http://www.ninject.org/>

Instead, AWSIMS proposes client specific versioning. As data consumers can define and maintain their own web service methods, they effectively manage their own specific versions.

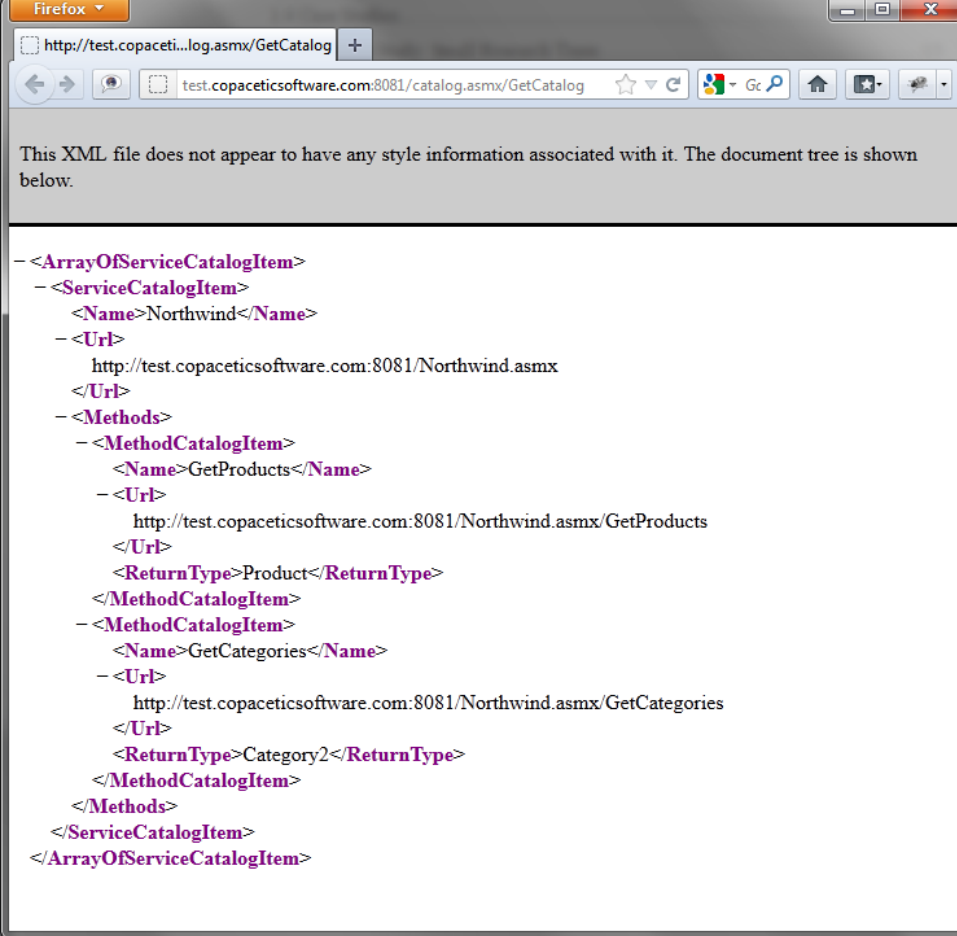
AWSIMS additionally proposes providing authorization based versioning as well. At present, when a data owner defines a data source in AWSIMS and generates the data source's schema, they are given the option to hide a schema column to prevent it from being mapped to a web service method. This mechanism can be extended so that a data owner could restrict access to schema column to a particular set of clients or clients that are a member of a security group. This security restriction would prevent an underprivileged data consumer from creating a web service method that referenced the restricted set of columns. However, a privileged data consumer could create such a web service method and mark the web service public, enabling any client to call it. When handling such a call, the AWSIMS Web Service Server would be responsible for automatically versioning the method return type so that the client would receive only the field they have permissions to access.

The capabilities listed above provide a rich versioning tool set, but are inadequate at addressing breaking changes made to an underlying data source. If a data owner removes a column from a database, for example, any web service method that depends on that column will no longer be able to fully populate the return type, and while method execution will not explicitly fail, the return type will contain a null or empty value for that schema column. While AWSIMS does not have the power for preventing data owners from making such changes, it does offer data owners the ability to see how many web service methods depend on a particular schema column. Additionally, this functionality is planned to be extended to allow data owners to see the number of clients consuming web service methods dependent on a schema, the frequency clients execute the methods, and offer the data owners the ability to inform the clients that the column is to be depreciated. The goal is to provide data owners with the information they need to correctly gauge the impact depreciating or changing a data source will have on the clients using the system.

### **2.3.6 Discoverability**

AWSIMS gets individual web service discoverability largely for free. Because it hijacks the existing ASP.NET web service pipeline, WSDL files for compiled web service definitions are generated automatically. Providing a listing of available web services, on the other hand, is not provided by the ASP.NET web service pipeline and had to be manually implemented. AWSIMS offers the Catalog web service which provides a listing of all web services, WSDL Urls, web service methods and web service methods

invocation Urls. Example XML output returned by invoking the Catalog web service's GetCatalog method, including the structure and format of the serialized data, is shown in Figure 17 - Example Output from the Catalog Web Service.



```
-<ArrayOfServiceCatalogItem>
- <ServiceCatalogItem>
  <Name>Northwind</Name>
  - <Url>
    http://test.copaceticsoftware.com:8081/Northwind.asmx
  </Url>
  - <Methods>
  - <MethodCatalogItem>
    <Name>GetProducts</Name>
    - <Url>
      http://test.copaceticsoftware.com:8081/Northwind.asmx/GetProducts
    </Url>
    <ReturnType>Product</ReturnType>
  </MethodCatalogItem>
  - <MethodCatalogItem>
    <Name>GetCategories</Name>
    - <Url>
      http://test.copaceticsoftware.com:8081/Northwind.asmx/GetCategories
    </Url>
    <ReturnType>Category2</ReturnType>
  </MethodCatalogItem>
  </Methods>
</ServiceCatalogItem>
</ArrayOfServiceCatalogItem>
```

Figure 17 - Example Output from the Catalog Web Service

## 2.3 Platform Benefits

Relegating web service management to a separate dedicated application intrinsically offers several additional benefits over an ad-hoc solution. The AWSIMS application is developed to offer several advanced performance features that could easily be neglected in an ad-hoc solution due to a combination of lack of technical expertise and time or budget constraints. AWSIMS can also implement multiple data sharing protocols to ensure maximum interoperability support. Finally, AWSIMS can receive iterative application updates to add new features and functionalities resulting in a future-proof data sharing solution. Each of these benefits is further discussed in the following sections.

### 2.3.1 Support for Multiple Protocols

While web services have emerged as the de facto standard for facilitating data sharing across networks and in particular the internet, the technology is not without limitations. Because of its reliance on XML and use of specialized SOAP headers and envelopes, web services are verbose and have large payloads. Additionally, because the results are returned in XML, web service method return types must be serializable, which places limits on the design of objects, preventing, for example, objects containing circular references.

The limits of web services have given rise to alternative data sharing protocols. JavaScript Object Notation (JSON)<sup>22</sup> is designed for web applications and mobile clients that need to minimize transport payloads and uses a custom serialization process that use name/value pairs and doesn't include object metadata. Both Java and .NET have custom protocols that use binary serialization and transport enabling any object graph to be transported but are not interoperable. These protocols all essentially serialize an object and distribute it over a known channel.

Because AWSIMS already has the infrastructure to query data sources and populate return objects, adding support for additional protocols only involves using a different serialization engine. While AWSIMS currently only supports web services, support for JSON is planned for a later release. When implemented, data consumers will have the ability to specify which protocols they would like their web service to implement as well as how they would like to alter the Url for each protocol by indicating if the protocol should be specified in the web service extension or path. For example a JSON Url could have the format `http://server/webService.json/method` or `http://server/json/webService.asmx/method/`.

---

<sup>22</sup> <http://www.json.org/>

### 2.3.2 Performance

AWSIMS functions analogously to an abstraction layer and consequently incurs a slight performance penalty versus a hard coded web service. For every request AWSIMS must load a web service definition from the database and method execution requires further database calls to load the web service method definition and return type definition. Additionally, the return type is instantiated and populated using reflection<sup>23</sup> which incurs a performance penalty. This performance degradation can be largely mitigated using caching techniques to reduce or completely remove the number of round trips to the database for frequently called web services.

Despite this initial overhead AWSIMS, as an application, offers data owners several advanced caching features designed to improve performance over an ad-hoc web service solution. Because the bulk of a web service method's execution time is expected to occur during the querying of a data source, caching the results of data source queries can significantly decrease execution time. Using the .NET framework's built-in cache libraries<sup>24</sup>, AWSIMS builds upon an established and proven caching infrastructure.

While full implementation has been relegated to a later development phase, AWSIMS plans to offer a two staged data source caching solution. All data source queries will be initially cached to ensure maximum utilization and the .NET framework automatically evicts least access items when memory limitations are reached. Queries will be initially cached for a default time interval in the range of ten minutes. Once a query has expired a data source specific dependency will be checked to determine if the query is stale and needs to be re-executed. .NET provides an out-of-box dependency manager for SQL database, and file system data sources can use the file's last modified time stamp. Data owners have the ability to modify the default time interval each query is cached. Increasing the cache time can reduce unnecessary 'staleness' checks for data sources that rarely change. Conversely, reducing the cache time to zero effectively disables the first caching tier causing a 'staleness' check to be performed during every query and ensuring that stale data is never served to a client.

Because a data source query is cached, rather than the full XML web service method response, multiple web service methods can share the same cached query. This strategy reduces the cost of storing redundant data but requires additional processing to populate the return object, run the Query Mutator pipeline and serialize the return object. In a

---

<sup>23</sup> [http://msdn.microsoft.com/en-us/library/f7ykdhsy\(v=vs.100\).aspx](http://msdn.microsoft.com/en-us/library/f7ykdhsy(v=vs.100).aspx)

<sup>24</sup> [http://msdn.microsoft.com/en-us/library/ms178597\(v=vs.100\).aspx](http://msdn.microsoft.com/en-us/library/ms178597(v=vs.100).aspx)

deployment where performance is critical the cost of caching a full response may be worth the increased memory requirements and would engender a performance boost, however, experimentation is necessary to determine the realized benefit of caching at this level.

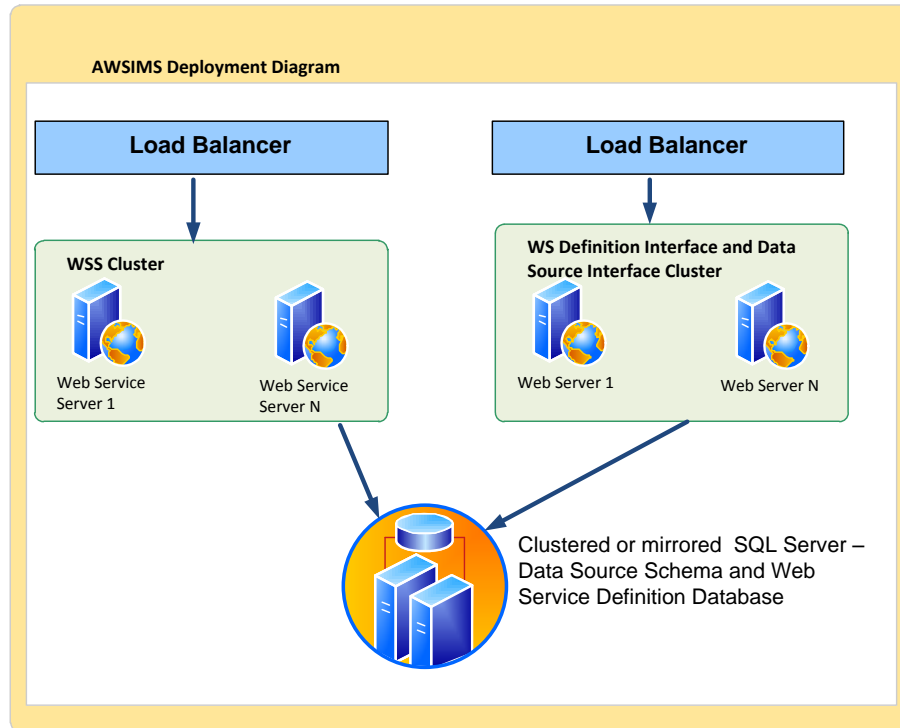
### **2.3.3 Upgradability**

Arguably the most powerful benefit the AWSIMS application offers over a traditional web service application is that the AWSIMS architecture is designed to support non-obtrusive application upgrades. As discussed in section 2.3.4 Web Service Method Execution, the web service method execution pipeline minimizes source code and web service method definition dependencies. This allows tremendous flexibility for upgrades to the AWSIMS application to add additional features, bug fixes, and performance enhancements without requiring users to perform any upgrade steps on their actual web service definitions.

## **2.4 Deployment**

AWSIMS must be deployed to a server running Microsoft's Internet Information Server (IIS) and Microsoft SQL Server to function correctly. While the entire AWSIMS application could be deployed to a single server, AWSIMS supports horizontal scaling to provide increased performance and redundancy. The Web Service Server (WSS) and both the Web Service Definition Interface (WSDI) and Data Source Interface (DSI) are stateless, allowing multiple IIS instances to be used in conjunction with a rudimentary load balancer. Microsoft SQL Server has native support for server clustering enabling multiple database servers to be added. Additionally, the AWSIMS WSS can be deployed separately from the WSDI and DSI for targeted scaling, as it is expected the majority of traffic for a default AWSIMS deployment will be web service invocation requests handled by the WSS. A recommended deployment diagram is shown in Figure 18 - AWSIMS Deployment Diagram.





**Figure 18 - AWSIMS Deployment Diagram**

This deployment strategy enables a range of benefits. Each primary component can be mirrored, providing hardware redundancy and protecting against hardware failure. This additionally allows targeted horizontal scaling for performance. If a bottleneck is discovered in one piece of the application, additional role specific hardware can be added to increase available throughput and ensure response times remain low. Additionally, this strategy supports server maintenance as individual machines can be taken offline, serviced and brought back online without interrupting operations.

#### **2.4.1 AWSIMS Cloud**

As correctly deploying and configuring public facing web and database servers are non-trivial tasks requiring both technical knowledge and hardware resources, mandating AWSIMS be deployed on-site limits its potential user base and does not fully support the goal of allowing data owners to easily share their data. To overcome this limitation,

AWSIMS has been fully deployed to the Microsoft Azure<sup>25</sup> cloud platform, enabling data owners to share their data immediately, without requiring them to configure a web server or install the application. The AWSIMS Cloud is deployed as shown in Figure 18 - AWSIMS Deployment Diagram, offering the same benefits of a properly deployed on-site installation. The Microsoft Azure platform delivers the same scalability and redundancy ensuring AWSIMS is highly-available and can maintain acceptable performance levels during high usage.

The cloud version of AWSIMS, AWSIMS Cloud, functions much the same way as a local deployment with a few minor drawbacks. Specifically, the cloud version has limited data source support. As AWSIMS in the cloud cannot access a data owner's local disk or network file systems, these data sources are unavailable. However, as AWSIMS supports reading from cloud storage, data owners can easily use the DropBox data source to generate web services from files. Additionally, as it is uncommon for organizations to allow public access to their databases, AWSIMS will not be able to consume local database content. The Microsoft Azure platform does support cloud hosting applications securely reading local database instances [10], however, this functionality has not yet been developed for AWSIMS, but is planned for a later release.

---

<sup>25</sup> <http://www.windowsazure.com>

### 3 Analysis

A fully functional AWSIMS prototype<sup>26</sup> was built as part of this thesis. I performed a demo of the prototype for a member of the Bio-informatics community and elicited feedback regarding its potential usefulness to future projects. The feedback is presented in section 3.1 Interview with the Bio-informatics Community. I additionally worked with members of the VPH-Share [40] project to discuss how AWSIMS could meet their data sharing requirements. Full details are provided in section 3.2 Real World Integration: VPH-Share. The AWSIMS prototype was analyzed against the case studies presented in section 1.5 Case Studies and the resulting discussion is given in section 3.3 Case Studies. Finally, section 3.4 Further Discussion explores additional topics relevant to the success of using AWSIMS as a data sharing solution in real world circumstances.

#### 3.1 Interview with the Bio-informatics Community

I presented a demo of the AWSIMS prototype to Dr. Marco Roos<sup>27</sup>, a biosemantics and e-science researcher at the Human Genetics department of Leiden University Medical Centre<sup>28</sup> (LUMC) and the Informatics Institute<sup>29</sup> of the Faculty of Science at the University of Amsterdam. Dr. Roos agreed that AWSIMS had the potential of benefiting his research team, especially if the interface could be further simplified for non-technical users and the application was able to self-discover database data sources instead of requiring users to supply AWSIMS with a connection string. Additionally, AWSIMS could provide significant value in dealing with legacy systems, such as the Nuclear Protein Database (NPD) [42]. When the NPD project was first launched, it was developed in house and was not required to expose data via web services. After the project went live, the team was requested to add a web service interface [43] and the project was completed over several months by a graduate student completing his master's thesis. AWSIMS could have potentially created a comparable web service offering without requiring a large development effort.

#### 3.2 Real World Integration: VPH-Share

The Virtual Physiological Human: Sharing for Healthcare – a Research Environment (VPH-Share) is an EU consortium project tasked with building an application that

---

<sup>26</sup> The prototype is publically available at [www.awsims.com](http://www.awsims.com). A usage guide is presented in Appendix A – A Quick User Guide to AWSIMS in the Cloud

<sup>27</sup> <http://www.biosemantics.org/index.php?page=dr>

<sup>28</sup> <http://www.lumc.nl/home/>

<sup>29</sup> <http://www.science.uva.nl/ii/home.cfm>

facilitates the sharing of clinical and research data and tools within the medical field [40]. The VPH-Share project plan [44] defines a two-fold strategy data sharing strategy. First, existing data must be federated by a data management platform in order to aggregate data into a single virtual file system. Second, the virtual file system must be exposed via data services and the data services must support querying source data that is contained in relational databases or flat files such as CSV files.

I interviewed a member of the VPH-Share team, PhD student Spiros Koulouzis<sup>30</sup>, who is working on creating Large Object Cloud Data Storage Federation (LOBCDER) [48], the federated data management platform for VPH-Share. LOBCDER unifies a collection of storage frameworks and providers and has a WebDAV [45] interface for file system operations. The LOBCDER application architecture is shown in Figure 19 - LOBCDER Architecture.

---

<sup>30</sup> <http://staff.science.uva.nl/~skoulouz/pmwiki/index.php>

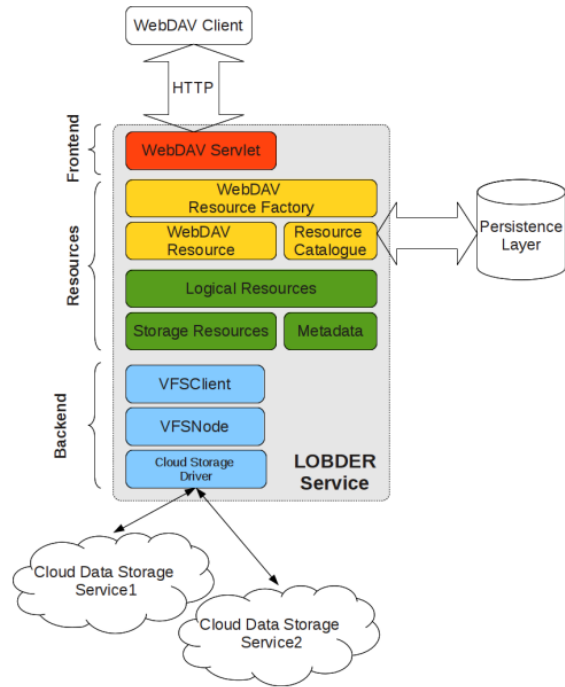


Figure 19 - LOBCDER Architecture<sup>31</sup>

Because WebDAV offers functionality similar to a traditional file system, AWSIMS could theoretically act as a front end for the LOBCDER system, creating web service wrappers for exposed files. Consequently, a WebDAV data source was authored for AWSIMS enabling AWSIMS to successfully interface with an early prototype of LOBCDER. A basic CSV file was added to a cloud data storage exposed by LOBCDER and then AWSIMS was used to create a web service exposing it for querying, proving that AWSIMS can meet the VPH-Share project goals of offering data services to query the virtual file system. The end result of the integration between AWSIMS and the VPH-Share and LOBCDER project highlights the demand for web service data sharing solutions and the utility of AWSIMS.

<sup>31</sup> Original image source: [48]

### 3.3 Case Studies

The following sections analyze AWSIMS in the context of the case studies presented in 1.5 Case Studies, and explore how AWSIMS functions as a data sharing solution.

#### 3.3.1 Case Study: Small Research Team

The small research team wants to make the scientific data currently in their database easily consumable by an e-Science grid and the greater scientific community. They do not have the in house technical knowledge or budget to hire an IT staff in order to build a custom solution. Without AWSIMS they could run a few queries against their database and save the results into a series of CSV data files. They could set up a FTP server and use that to share the CSV data files with the scientific community. And by using technology such as GridFTP they could push data to their e-Science infrastructure.

This non-AWSIMS solution has several drawbacks. As FTP doesn't support database queries, they have to run a query and save it to a file in order to be served by the FTP server. As soon as the database is updated the FTP server contains stale data and will need to be updated with a fresh set of CSV files; a time consuming process. Another drawback is using GridFTP to push data to their e-Science infrastructure. Depending on the infrastructure, the scientists may not have permanent storage and will need to either push the files to the infrastructure every time, or invest the time into learning and writing scripts to automate the file movements.

Using the AWSIMS prototype offers the team a data sharing solution with none of these drawbacks. By deploying a local instance of AWSIMS and configuring it to find their database, AWSIMS will create web service wrappers for their data. The web services provide real time data access and will pick up changes made to the underlying data base. The web service Urls can be shared with the scientific community in order to provide them access to the team's data. The web services can also be used to get the data up to the e-Science infrastructure. The web service can either be queried manually, using curl [28], or by integrating the query into a workflow.

While using AWSIMS offers significant benefits, the proposed implementation does require the team to setup a local AWSIMS instance. This requires the team to procure hardware for a web server, execute the AWSIMS installer and then configure the firewall to ensure public access. This process will be significantly eased with the planned addition of the local-to-Cloud database connector that will enable the AWSIMS Cloud version to read local databases. Once this feature has been added, the team will be able to use the Cloud version of AWSIMS and will not need to deploy a local instance. Instead, they will

only need to configure the local-to-Cloud database connector to ensure their data can be read properly by AWSIMS Cloud.

### **3.3.2 Case Study: Intra-Department Sharing**

The internal solutions department is struggling to architect a SOA data sharing solution that will be agile enough to meet the evolving needs of a growing group of stakeholders. The department employs a small team of software developers but they lack the experience and knowledge to build and deploy an enterprise grade SOA application. Additionally, their technical staff has other projects they are responsible for and don't have the resources available to constantly field stakeholder change requests.

Without AWSIMS they would be forced to, in the common engineering vernacular, 'wing it'<sup>32</sup>. The development team would need to build and deploy a prototype application and then interactively release updates to appease each new stakeholder and new stakeholder requirement or change request. Due to scheduling pressures, it is unlikely the team will be able to deliver a robust solution, having to sacrifice quality for a quick delivery. This solution does not scale and maintenance costs will increase as new versions saddle the application with technical debt [29]. The developers are further left with a data dependency problem as they may not be able to easily discern which versions of their applications are active and which portions of the database each version depends on. This adds significant risk that introducing a database schema change could break their web service solution.

Implementing AWSIMS removes the onus on the department's developers to build and maintain a custom solution. By installing a local instance and adding their databases as data sources, their job is done. Other departments can use the AWSIMS web application to define and version their own custom interfaces and can easily modify them as their own application life cycle dictates. AWSIMS offers usage reporting so the internal solutions department can easily check database dependencies and deduce the impact a database change will have on their web service offering. And AWSIMS offers them performance enhancements and the ability to scale to ensure web service performance metrics match stakeholder expectations.

### **3.3.3 Case Study: Large Multi-Disciplinary Research Team**

The large multi-disciplinary research team has a versioning problem. They need to share the data in their database with scientists from several different backgrounds that are

---

<sup>32</sup> <http://idioms.thefreedictionary.com/wing+it>

interested in different aspects of the data. And as the project progresses the database schema is updated, requiring the authoring of new versions of old web services while still needing to support legacy versions. They have the technical staff to write and maintain their web service inventory but the versioning problem is quickly becoming overwhelming.

Without AWSIMS the research team needs to implement and maintain a custom solution. Every time the underlying database schema needs to be changed, the team needs to survey their current web service offering to ensure the change will not break an existing web service. Once the schema is changed, a new set of web services need to be authored to ensure clients can get the latest version and client applications will need to be updated to use the latest version URLs and then likely recompiled. While the team could use a web service version management strategy, such as the Chain of Adapters [9] pattern, to mitigate some of the version management development costs, they are still required to invest an increasing amount of development effort for every schema change.

Implementing AWSIMS significantly reduces the team's web service version management problem. AWSIMS allows consumers to manage their own versions, independent of the team, offloading the effort to the data consumers and allowing the data consumers to use the latest version of the database schema immediately without having to wait for the team's developers to author a new version. Additionally, if the data consumer is not interested in the new data provided by the schema change they can elect to continue using their current version.

If the research team needs to implement a schema change that has the potential to break existing web services they can use AWSIMS to easily see which web services would be affected and whether the web services are actively being used. That knowledge can be used to power and outreach effort to communicate with the impacted data consumers and schedule an 'end-of-life' date. Ultimately, AWSIMS empowers the team to make a more informed decision as to the impact of any database schema changes.

## **3.4 Further Discussion**

### **3.4.1 Big Data**

One of the potentially limiting factors to AWSIMS applicability is its ability to handle large data sets. Cutting edge research projects are producing data sets on the order of terabytes and even petabytes and are requiring data storage solutions and data sharing



solutions that can cope with such large data sets. Technologies such as hadoop<sup>33</sup>, HBase<sup>34</sup> and NoSQL<sup>35</sup> are emerging as solutions capable of meeting the data storage needs of such high data volume projects. While not fully tested during the prototyping of AWSIMS, the AWSIMS architecture can theoretically support these new data stores as long as they provide APIs to query both data and schema. However, serving large data sets via web services presents additional technological challenges in that web services, or more specifically the http protocol, is not designed for transmitting large data payloads. Independent research [46] has investigated mechanisms for overcoming these limitations, which AWSIMS could implement. At present though, AWSIMS supports large data via pagination, requiring consumers to make repeated calls requesting manageable chunks of data.

### 3.4.2 Semantic Web

The concept of the semantic web has been generated interest since it was first coined by Tim Berneres-Lee in 2001 [47]; promising a mechanism for standardizing data formats for the Internet. Semantic webs have become popular within the scientific community for providing the technology to create a controlled vocabulary, or ontology, for scientific disciplines. The benefit of semantic webs is improved data integration amongst multiple scientific content repositories and simplified query interfaces that more closely models natural language when compared to traditional internet search.

Semantic webs work by defining an ontology which contains the controlled vocabulary for a particular domain and data repository schema are then mapped to the ontology in order to aggregate that repository into the semantic web. While support for semantic web is not among AWSIMS primary design goal, because AWSIMS allows data consumers to customize web service method return types (as discussed in section 2.3.2 Client Driven Interface Generation), including setting field names, a data consumer could use AWSIMS to expose a web service method that conforms to a given semantic web ontology. However, this is a tedious solution for the data consumer and would likely need to be improved in a future version in order for it to be considered practical.

---

<sup>33</sup> <http://hadoop.apache.org/>

<sup>34</sup> <http://hbase.apache.org/>

<sup>35</sup> <http://en.wikipedia.org/wiki/NoSQL>

## 4 Conclusion

Data sharing is integral to unlocking the full potential of scientific data; enabling multi-disciplinary researchers to collaborate and work together in ways not otherwise possible. And the growing adoption of e-science infrastructures, built on SOA principles, opens new possibilities for data analysis as long as raw data is readily available. Yet, despite the recognized value of data sharing, modern technology infrastructures have yet to provide an easy-to-use data sharing solution. In order to provide such a data sharing solution, three data sharing goals have been identified:

1. Ease the burden on data owners that want to share their data.
2. Enable data consumers to easily consume data in a format most conducive to their needs.
3. Support the integrated data schema so that data can be readily consumed by computation engines.

Web services have emerged as a standardized and interoperable data sharing mechanism and several frameworks exist to ease implementation. However, utilizing them requires non-trivial technology expertise; proving that the frameworks alone are not sufficient to meet the data sharing goals. Further, operating a web service inventory in practice requires additional effort, specifically version management.

The Automated Web Service Inventory Management Software application has been designed and built to harness the power of web services while simultaneously lowering the barrier of entry such that the data sharing goals of ease-of-use are met. AWSIMS allows data owners to specify their data and data consumers to define their own web services. The net result is neither party needs to write any code and version management is greatly simplified; AWSIMS offers an ideal data sharing solution.

AWSIMS has been analyzed in the context of three case studies, each one showing how implementing AWSIMS adds value as a data sharing solution versus the current technology. AWSIMS offers a small non-technical team the means to easily share their data and make it available to e-science infrastructures, it helps manage multiple competing stakeholder requirements, and provides an ideal version management solution. AWSIMS has been shown to have value for the bio-informatics community, such as creating web service wrappers for legacy database, facilitating data integration and aggregation. AWSIMS has been proven to be able to integrate with cutting edge projects like VPH-Share, providing a data sharing solution for cloud based file systems.

In essence, AWSIMS highlights the advantages of a dedicated data sharing application to meet the needs of modern science. Simplifying data sharing allows scientists, who would not have otherwise been able, to make their research data available to the scientific community unlocking the full power of their research and promoting collaboration. However, despite what AWSIMS has achieved there is still much work that can be done to improve its capacity for data sharing. A discussion on future work to enhance AWSIMS is presented in the next section, section 4.1 Future Work.

## 4.1 Future Work

While the benefits of AWSIMS as a data sharing solution have been shown there is still room for improvement. As AWSIMS has not yet been tested in a real world application, doing so would offer tremendous value as scientists will certainly have requirements and scenarios that have not been thought of during development. AWSIMS currently only supports read-only operations and expanding the application to include full CRUD<sup>36</sup> support would expand its value as a data sharing solution, enabling web service consumers to contribute additional data. Finally, the AWSIMS security model is currently rather limited and expanding it to enable data owners to allow finer grained control over which parties can access which parts of their data; empowering data owners with a greater sense of control over their data and encouraging them to share non-public data.

### 4.1.1 User Review and Feedback

User feedback is valuable for any application and AWSIMS is no exception. Having users navigate and use the application's web interface would highlight flaws in the navigation model and give users a chance to comment on recommended improvements to streamline processes. Deployment in a live research project would generate usage statistics to gauge performance bottlenecks and areas for improvement. Finally, having an open dialogue with users is the ultimate forum to determine if AWSIMS meets the data sharing needs of real-world research projects and is an avenue for discussing what product enhancements can be made to improve the data sharing experience.

### 4.1.2 CRUD Support

AWSIMS currently only allows generated web services read-only access to data sources. Expanding AWSIMS capabilities to support create, update and delete (CRUD) operations would enhance its data sharing capabilities. Web service clients would be able to create interfaces enabling them to contribute to a data source by adding, modifying or deleting content. Additionally, e-Science infrastructures would be able to write results back

---

<sup>36</sup> [http://en.wikipedia.org/wiki/Create,\\_read,\\_update\\_and\\_delete](http://en.wikipedia.org/wiki/Create,_read,_update_and_delete)

directly to a database as part of a workflow, making it easier for scientists to view results and perform analysis.

Adding CRUD support would not be a trivial undertaking. Care must be given to ensure data owners recognize the potential risks of allowing web service clients to modify their data. Appropriate controls would need to be set in place to ensure data owners can have finely grained control to indicate what parts of the data can be modified and by whom. Additional auditing would likely need to be implemented as well so that data owners can track data modifications.

#### 4.1.3 Security

The current AWSIMS security model is extremely basic and needs to be expanded to ensure AWSIMS can be securely used in a wide range of operating circumstances. Security development needs to focus on two areas:

1. Providing data owners with greater tools to restrict which users can access which portions of their data, ie Authorization.
2. Authenticating web service clients so that applicable security restrictions can be enforced.

By empowering data owners the ability to restrict access to certain schema columns or data sources allows them to share their data with the widest audience possible while still enabling them to protect the integrity and privacy of their data. The greatest foreseen challenge is designing a system to enable data owners to easily manage access control lists (ACLs) without being overburdened. Data consumers could be organized into groups and data owners could then provide restrictions based on group. Additionally, group membership could be manually controlled or based on registered email. For example, all data consumers who have an UvA email could be grouped together.

In order for AWSIMS to enforce restrictions put in place by data owners web service clients will have to be authenticated. ASP.NET and IIS support a variety of authentication mechanisms for web services including basic authentication, where a username and password are sent with each request, and forms authentication, where clients are required to authenticate and are then given a security token that will be sent with each request. AWSIMS can make use of this built-in support and could support different authentication mechanisms based on circumstances or data owner specification. However, further research is required to ensure the most appropriate authentication mechanisms are available for AWSIMS.



## References

1. Hartrum, Thomas C., "Automated Code Generation Tools for Collaboration Systems," Proceedings of the 2007 International Symposium on Collaboration Technologies and Systems (CTS 2007), May 21-25, 2007, Orlando, FL, pp. 183-190
2. Frank, D.; Linh Lam; Liana Fong; Ru Fang; Khangaonkar, M., "Using an Interface Proxy to Host Versioned Web Services," Services Computing, IEEE International Conference on, pp. 325-332, 2008 IEEE International Conference on Services Computing Vol. 2, 2008
3. (2012, May) Microsoft .NET Documentation. "Service Versioning" [Online] Available: <http://msdn.microsoft.com/en-us/library/ms731060.aspx>
4. (2012, May) Microsoft .NET Documentation. " Best Practices: Data Contract Versioning" [Online] Available: <http://msdn.microsoft.com/en-us/library/ms733832.aspx>
5. European Commission Information Society and Media. "Opportunities for Data Exchange – Ten Tales of Drivers and Barriers in Data Sharing" [http://www.libereurope.eu/sites/default/files/ODE\\_10Stories\\_0.pdf](http://www.libereurope.eu/sites/default/files/ODE_10Stories_0.pdf)
6. European Commission Information Society and Media. "Opportunities for Data Exchange – Research Infrastructure" <http://cordis.europa.eu/fp7/ict/e-infrastructure/docs/ode.pdf>
7. European Commission Information Society and Media. "Riding the Wave – How Europe can Gain from the Rising Tide of Scientific Data". <http://cordis.europa.eu/fp7/ict/e-infrastructure/docs/hlg-sdi-report.pdf>
8. (2012, May) CERN web site. [Online]. Available: <http://public.web.cern.ch/public/en/lhc/Computing-en.html>
9. Piotr Kaminski, Hausi Muller, and Marin Litoiu. 2006. A design for adaptive web service evolution. In Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems (SEAMS '06). ACM, New York, NY, USA, 86-92.
10. (2012, June) Microsoft. NET Documentation "Synchronizing SQL Azure". [Online]. Available: [http://msdn.microsoft.com/en-us/library/ff928514\(SQL.110\).aspx](http://msdn.microsoft.com/en-us/library/ff928514(SQL.110).aspx)
11. Lizhe Wang; Jie Tao; Kunze, M.; Castellanos, A.C.; Kramer, D.; Karl, W.;; "Scientific Cloud Computing: Early Definition and Experience," *High Performance Computing and Communications, 2008. HPCC '08. 10th IEEE International Conference on* , vol., no., pp.825-830, 25-27 Sept. 2008

12. Shengxian Luo; Xiaochuan Peng; Shengbo Fan; Peiyu Zhang; , "Study on Computing Grid Distributed Middleware and Its Application," *Information Technology and Applications, 2009. IFITA '09. International Forum on* , vol.3, no., pp.441-445, 15-17 May 2009
13. J. Yaylor. (2012, June) Defining e-science. [Online]. Available: <http://www.nesc.ac.uk/nesc/define.html>
14. H. He, "What is service-oriented architecture" September 2003. [Online]. Available: <http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html>
15. K. Channabasavaiah, K. Holley, and J. Edward M. Tuggle, "Migrating to a service-oriented architecture." December 2003. [Online]. Available: <http://www.ibm.com/developerworks/library/ws-migratesoa/>
16. J. D. Blower, A. B. Harrison, and K. Haines. 2006. *Styx Grid Services: Lightweight middleware for efficient scientific workflows*. Sci. Program. 14, 3,4 (December 2006), 209-216.
17. (2012, May) High Level Expert Group on Scientific Data. "*Riding the Wave - How Europe can Gain from the Rising Tide of Scientific Data*" [Online]. Available: <http://cordis.europa.eu/fp7/ict/e-infrastructure/docs/hlg-sdi-report.pdf>
18. D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, and D. O. C. Ferris. W3C. "Web Services Architecture," February 2004. [Online]. Available: <http://www.w3.org/TR/ws-arch/>
19. Curbera, F.; Duftler, M.; Khalaf, R.; Nagy, W.; Mukhi, N.; Weerawarana, S.; , "*Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI*," Internet Computing, IEEE , vol.6, no.2, pp.86-93, March-April 2002
20. Turner, M.; Budgen, D.; Brereton, P.; , "Turning software into a service," *Computer* , vol.36, no.10, pp. 38- 44, Oct. 2003
21. M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. F. Nielsen, A. Karmarkar, and Y. Lafon, "*SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*," April 2007. [Online]. Available: <http://www.w3.org/TR/soap12-part1/>
22. N. Kavantzaz, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon "*Web Service Choreography Description Language Version 1*," November 2005. [Online] Available: <http://www.w3.org/TR/ws-cdl-10/>

23. (2012, June) ENFOS web site. "Web-Based Collaboration and Workflow". [Online]. Available: <http://www.enfos.com/www/productsSolutionsLab.php>
24. Savage CJ, Vickers AJ (2009) Empirical Study of Data Sharing by Authors Publishing in PLoS Journals. PLoS ONE 4(9)
25. (2012, June) Angus Macdonald. "Installing OGSA-DAI 3.1 (A Rough Guide)". [Online]. Available: <http://blogs.cs.st-andrews.ac.uk/angus/2009/03/installing-ogsa-dai/>
26. (2012, June) Worldwide LHC Computing Grid website. [Online]. Available: <http://lcg-archive.web.cern.ch/lcg-archive/public/>
27. Escalante, L.G.; Castillo, A.B.; Ocana, L.B.; , "Automatic generation and publication of Web services for the access and integration of distributed data sources," *Computer Science, 2005. ENC 2005. Sixth Mexican International Conference on* , vol., no., pp. 96-103, 26-30 Sept. 2005
28. (2012, June) cURL Manual Page. [Online]. Available: <http://curl.haxx.se/docs/manpage.html>
29. Nanette Brown, Yuanfang Cai, Yuepu Guo, Rick Kazman, Miryung Kim, Philippe Kruchten, Erin Lim, Alan MacCormack, Robert Nord, Ipek Ozkaya, Raghvinder Sangwan, Carolyn Seaman, Kevin Sullivan, and Nico Zazworka. 2010. Managing technical debt in software-reliant systems. In *Proceedings of the FSE/SDP workshop on Future of software engineering research (FoSER '10)*. ACM, New York, NY, USA, 47-52.
30. (2012, June) Definition of 'enterprise class'. PC Magazine. [Online]. Available: [http://www.pcmag.com/encyclopedia\\_term/0,1237,t=enterprise+class&i=42639,00.asp](http://www.pcmag.com/encyclopedia_term/0,1237,t=enterprise+class&i=42639,00.asp)
31. Mike Volodarsky. IIS Team. "ASP.NET Integration with IIS 7," May 2010. [Online]. Available: <http://learn.iis.net/page.aspx/243/aspnet-integration-with-iis/>
32. (2012, June) IHttpHandler Interface. Microsoft. [Online]. Available: [http://msdn.microsoft.com/en-us/library/system.web.ihttphandler\(v=vs.100\).aspx](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler(v=vs.100).aspx)
33. (2012, June) WebServiceHandlerFactory Class. Microsoft. [Online]. Available: <http://msdn.microsoft.com/en-us/library/system.web.services.protocols.webservicehandlerfactory.aspx>
34. (2012, June) Managing, Tuning, and Configuring Application Pools in IIS 7.0. Microsoft. [Online]. Available: <http://technet.microsoft.com/en-us/library/cc745955.aspx>



35. Hey T, Stewart T, Tolle K: The Fourth Paradigm Data-Intensive Scientific Discovery. Microsoft Research 2009.
36. Fujun Zhu, Mark Turner, Ioannis Kotsiopoulos, Keith Bennett, Michelle Russell, David Budgen, Pearl Brereton, John Keane, Paul Layzell, Michael Rigby, and Jie Xu. 2004. Dynamic Data Integration Using Web Services. In *Proceedings of the IEEE International Conference on Web Services (ICWS '04)*. IEEE Computer Society, Washington, DC, USA, 262
37. E. Christensen, F. Curbera, G. Meredith, S. Weerawarna. "Web Services Description Language (WSDL) 1.1," March 2001. [Online] Available: <http://www.w3.org/TR/wsdl>
38. J. Evdemon, "Principles of Service Design: Service Versioning", <http://msdn.microsoft.com/en-us/library/ms954726.aspx>, 2005.
39. OASIS WSDM-MOWS 1.0 specification, Draft. [http://www.oasis-open.org/committees/download.php/5664/wd-wsdm-mows\\_versioning\\_change\\_2.23.04a.doc](http://www.oasis-open.org/committees/download.php/5664/wd-wsdm-mows_versioning_change_2.23.04a.doc)
40. (June, 2012). VPH-Share Project web site. [Online] Available: <http://www.vph-share.org>
41. C. Larman. "Applying UML and Patterns". Prentice Hall. 2005.
42. (June, 2012). The Nuclear Protein Database web site. [Online] Available: <http://npd.hgu.mrc.ac.uk/user/>
43. (June, 2012). Accessing NPD content via web services. [Online] Available: <http://npd.hgu.mrc.ac.uk/user/services>
44. (June, 2012). VPH-Share Analysis of the State of the Art; Work Package Definition. [Online] Available: [https://www.biomedtown.org/biomed\\_town/vphshare/reception/public\\_repository/deliverables/VPH-Share\\_D2.1\\_1v5.pdf?action=download](https://www.biomedtown.org/biomed_town/vphshare/reception/public_repository/deliverables/VPH-Share_D2.1_1v5.pdf?action=download)
45. E. James Whitehead, Jr. and Yaron Y. Goland. 1999. WebDAV: a network protocol for remote collaborative authoring on the Web. In *Proceedings of the sixth conference on European Conference on Computer Supported Cooperative Work (ECSCW'99)*. Kluwer Academic Publishers, Norwell, MA, USA, 291-310.

46. Koulouzis, S.; Cushing, R.; Karasavvas, K.A.; Belloum, A.; Bubak, M.; , "Enabling Web Services to Consume and Produce Large Datasets," *Internet Computing, IEEE* , vol.16, no.1, pp.52-60, Jan.-Feb. 2012
47. Berners-Lee, T., James H., Or L. The Sematic Web, "*Scientific American Magazine*", May 2001.
48. S. Koulouzis, R. Cushing, A. Belloum, M. Bubak. Large Object Cloud Data Storage Federation. Submitted to 8<sup>th</sup> IEEE International Conference on eScience 2012.
49. R. Strijkers, R. Cushing, D. Vasyunin, C. de Laat, A. Belloum, R. Meijer, Toward Executable Scientific Publications, *Procedia Computer Science*, Volume 4, 2011, Pages 707-715.

## Appendix A – A Quick User Guide to AWSIMS in the Cloud

This section shows the AWSIMS web site, available at [www.awsims.com](http://www.awsims.com), in action. You can sign up for an account and follow along. If you register with an UvA email address, your account will be instantly approved and you'll just need to respond to a confirmation email. Otherwise, your account will need to be approved by an Administrator.

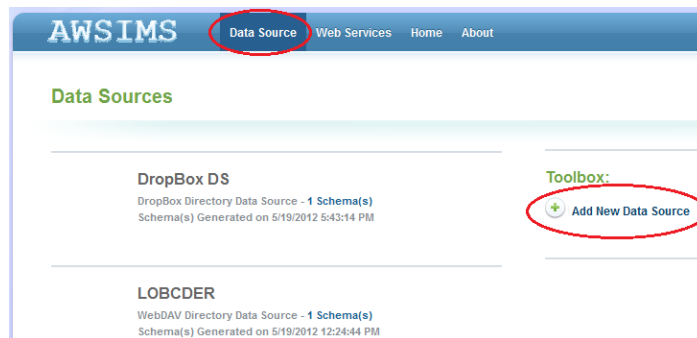
### A.1 Sharing Data

Adam Scientists want to use AWSIMS to make some data he has in a spreadsheet available to Bob Scientist in real time. To do so, he follows the following steps:

1. Adam Scientist logs into AWSIMS web site.



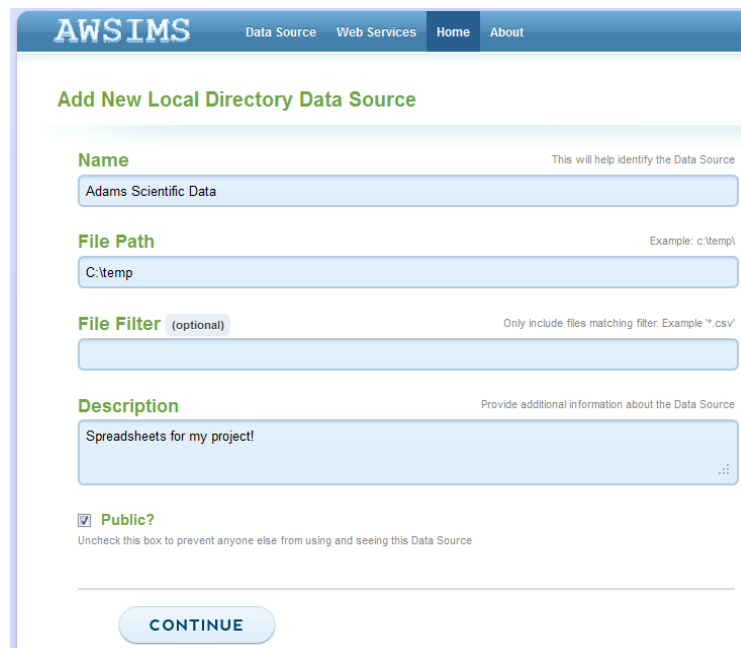
2. Adam Scientist navigates to the *Data Sources* section and clicks *Add New Data Source*.



- Adam Scientist selects the type of Data Source, in his case he wants to share a File System Directory.



- Adam Scientist fills in the necessary form.



**Name** This will help identify the Data Source  
Adams Scientific Data

**File Path** Example: c:\temp\  
C:\temp

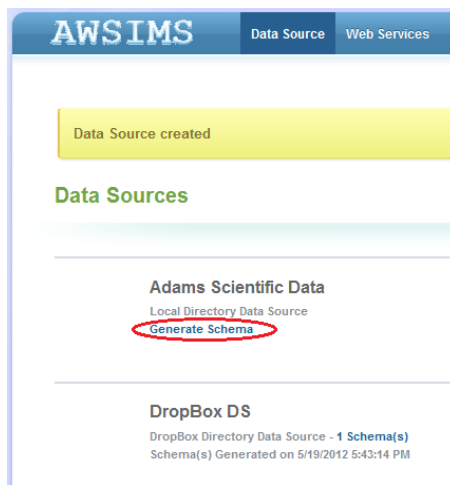
**File Filter** (optional) Only include files matching filter. Example \*.csv

**Description** Provide additional information about the Data Source  
Spreadsheets for my project!

**Public?**  
Uncheck this box to prevent anyone else from using and seeing this Data Source

**CONTINUE**

- Adam Scientist's Data Source has been created, and he clicks *Generate Schema* to import the Data



- At this point Adam can see that his spreadsheet has been imported and his job is done.

#### Schema(s) for Adams Scientific Data

<p><b>example.csv</b>          Last Validated On: 5/23/2012 10:32:40 AM          MethodExectuor: CSVMethodExecutor</p> <p><b>Columns:</b>          N - System.String          M - System.String          Iter - System.String          Threads - System.String          Time (s) - System.String          FLOP/s - System.String          Speed Up vs PTH-1 - System.String          Speed Up vs N-1 - System.String</p> <p><b>Actions:</b>  <a href="#">Edit Schema Columns</a>  <a href="#">Archive</a></p> <p>▲  <a href="#">LESS DETAILS</a></p>	<p><b>Toolbox:</b>  <a href="#">ReGenerate Schema</a></p> <p><b>Schema(s) Details:</b>          Schema(s) Count: 1 ( 0 Archived )          Created Date: 5/23/2012 10:32:40 AM          Last Validated On: 5/23/2012 10:32:40 AM</p> <p><b>Stats:</b>          Data Source is used by: 0 Web Service(s) and 0 Method(s)</p>
--	---

## A.2 Consuming Data

Bob Scientist wants to use Adam Scientist's data via a web service. He'll use AWSIMS to create the web service.

1. Bob Scientist logs into the AWSIMS web site.

Enter your AWSIMS login to continue

**User Name**  
Bob Scientist

**Password**  
••••••

**CONTINUE**

[Forgot your password?](#)

2. Bob Scientist navigates to *Web Services* and clicks *Add New Web Service*

**Web Services**

**Drop\_Box\_Web\_Service**  
0 Method(s)

**Northwind**  
2 Method(s)

**Toolbox:**  
[+ Add New Web Service](#)

3. Bob Scientist creates a new Web Service for himself.

The screenshot shows the 'Add Web Service' form in the AWSIMS application. The navigation bar at the top includes 'AWSIMS', 'Data Source', 'Web Services', 'Home', and 'About'. The form has a title 'Add Web Service' and contains the following fields and options:

- Name:** A text input field containing 'Bobs Service'.
- Namespace:** A text input field containing 'Example'.
- Public?:** A checked checkbox with the label 'Public?'. Below it, a note reads: 'Uncheck this box to prevent anyone else from using and seeing this Web Service'.

At the bottom of the form is a 'CONTINUE' button.

4. Bob Scientist adds a method to his web service

The screenshot shows the 'Web Services' page in the AWSIMS application. The navigation bar at the top includes 'AWSIMS', 'Data Source', 'Web Services', 'Home', and 'About'. A yellow banner at the top of the main content area reads 'Web Service created'. Below this is the 'Web Services' section, which displays a card for 'Bobs\_Service'.

The card for 'Bobs\_Service' shows '0 Method(s)' and a list of actions:

- Actions:**
  - Add Method** (circled in red)
  - Edit Web Service
  - Delete Web Service
- LESS DETAILS

5. Bob Scientist fills out the necessary form indicating that he wants to use the data source Adam Scientist created.

AWSIMS Data Source Web Services Home About Welcome Bob Scientist! Log Off

### Create a Web Service Method for Bobs\_Service

**Name** Identify the Method and the URL used to access the Method

**Data Source** The Data Source to Query

**Schema** The portion of the Data Source to Query

**Schema Details** Select the Properties returned by the Method

**N**   
Original Name: N : Type: System.String

**M**   
Original Name: M : Type: System.String

- Bob Scientist's method has been created and he can click the *Invoke* button to preview the data.

AWSIMS Data Source Web Services Home About

Web Service Method created

### GeneratedReturnType.AdamsData Adams\_Scientific\_Data() Details

**Actions**  
[Edit](#)

**WSDLUrl**  
[http://localhost:38863/Bobs\\_Service.asmx?WSDL](http://localhost:38863/Bobs_Service.asmx?WSDL)

**ExecutionHarnessURL**  
[http://localhost:38863/Bobs\\_Service.asmx?op=Adams\\_Scientific\\_Data](http://localhost:38863/Bobs_Service.asmx?op=Adams_Scientific_Data)

- The XML data is previewed for Bob Scientist. This data stream can be easily consumed by a software application, a Grid workflow, or even an Excel spreadsheet!



localhost:38863/Bobs\_Service.a ☆ ▼ ↻ Gc 🔍 🏠 🖨️ 🌟

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
-<ArrayOfAdamsData>
  -<AdamsData>
    <N>100_pj</N>
    <M>100</M>
    <Iter>50000</Iter>
    <Threads>Seq</Threads>
    <Time>2.850577</Time>
    <FLOP>2.104844E+09</FLOP>
    <Speed_Up_vs_PTH/>
    <Speed_Up_vs_N1/>
  </AdamsData>
  -<AdamsData>
    <N>100</N>
    <M>100</M>
    <Iter>50000</Iter>
    <Threads>PTH - 1</Threads>
    <Time>5.710776</Time>
    <FLOP>1.050628E+09</FLOP>
    <Speed_Up_vs_PTH/>
    <Speed_Up_vs_N1/>
  </AdamsData>
  -<AdamsData>
    <N>100</N>
    <M>100</M>
    <Iter>50000</Iter>
    <Threads>PTH - 2</Threads>
    <Time>3.974436</Time>
    <FLOP>1.509623E+09</FLOP>
    <Speed_Up_vs_PTH>143.69%</Speed_Up_vs_PTH>
    <Speed_Up_vs_N1>0.44</Speed_Up_vs_N1>
  </AdamsData>
  -<AdamsData>
    <N>100</N>
    <M>100</M>
    <Iter>50000</Iter>
    <Threads>PTH - 3</Threads>
    <Time>3.019312</Time>
```